修 士 論 文

生成語彙論に基づく名詞句「AのB」の意味解釈

北陸先端科学技術大学院大学
情報科学研究科

植村 将人

2004年3月

修 士 論 文

生成語彙論に基づく名詞句「AのB」の意味解釈

指導教官　島津 明 教授

審査委員主査　島津 明 教授
審査委員　東条 敏 教授
審査委員　白井 清昭 助教授

北陸先端科学技術大学院大学
情報科学研究科

210007 植村 将人

提出年月: 2004 年 2 月

概 要

日本語の名詞句には「A の B」という表現が多々現れる。名詞句「A の B」の表現は多様である。表現によっては多くの曖昧性がある。助詞「の」で二つの名詞 A と B とが結ばれた表現のため、それらの意味関係は陽には表現されず、多くの曖昧性を持っている。「A の B」の意味を導くために従来の研究では、名詞を辞書により解析したり、用例あるいは国語辞典を用いて曖昧性意味を解析するなどの方法がある。従来の研究ではどちらかというと個別に対処していたというきらいがある。

本研究では、語彙項目を意味毎に与えない生成語彙論 [5] に基づいて名詞の語彙項目を与え、それらの語彙項目が「A の B」の意味を導く方法を示す。生成語彙論では、数え上げによって語彙を与えず、対象となる名詞が持ちうる意味タイプを一つの語彙項目内に記述し、それらを関連付け、語彙項目同士を生成的に組み合わせることで意味を導く。生成語彙論による語彙の記述を選択した理由は複数の意味タイプを一つの語彙項目内に記述し名詞の曖昧性を捉え、さらにそれらの項目を組み合わせることで「A の B」の意味的曖昧性を表現できるという考えによる。

「A の B」の既存研究には [2]、[3]、[6], などがある。[2] は生成語彙論による「A の B」の意味解析を行っている。それは語彙項目を名詞毎に与え、名詞を事物名詞、事象名詞、関係名詞とに分け、B が事物名詞であるとき、B が事象名詞であるとき、B が関係名詞であるときの三つの場合にわけ、A と B との語彙項目を組み合わせることで解析を行っている。[3] では国語辞典の説明文を利用し「A の B」の解析を行っている。例えば、「ラグビーのコーチ」という名詞句を解析するとき、「コーチ」の説明文に「何らかのスポーツで技術を教える人」とあったとき、「ラグビー」と説明文中にある「スポーツ」、「技術」とによって類似度を算出し類似度が高い方に「ラグビー」を当てはめることによって意味を導いている。それに加え格フレーム辞書を使った格解析と統合しより信頼性の高い結果を「A の B」の意味解析結果としている。また、[6] では A、B に素性を与え、「A の B」を意味関係により五分類し、その分類と素性を使って「A の B」の意味的関係を推定することで意味解析を行っている。

[2] では「A の B」の解析を B が事物名詞、事象名詞、関係名詞であるときの各場合に分けて行っているが、本研究では名詞を分けずに一つの解析方法によって語彙項目同士を合成する。さらに、シソーラスによる語彙項目の継承を行い、語彙項目を合成することで「A の B」の意味解釈を導く。シソーラスとしては、IPAL[4] を利用した。

語彙項目の合成方法は、A に与えられた意味タイプと B が持つ述語表記で要求される意味タイプとのマッチの成否によって合成を行っている。例えば「私の絨毯」では、まず合成を行うために「私」の意味タイプと「絨毯」が要求するタイプを調べる。この場合「私」は human で、「絨毯」は要求するタイプがないため合成失敗となる。つぎに B の継承を行う。「絨毯」は con_product(concrete と product を合成したタイプ) という意味タイプを持ち、そのタイプの語彙項目を継承する。そして継承した項目が要求しているタイプを調べると、述語 make の引数が human を要求し「私」の意味タイプとのタイプマッチが成功し、「絨毯」が要求したタイプの代わりに「私」の語彙項目を入れ合成結果とする。その合成結果をリストに加え、次は B の語彙継承を行う。継承の後また、同じように合成し、合成結果をリストに加える。もし A にあたる名詞が複数のタイプを持っていたときは、タイプごとに順に B との合成を行う。また、このとき B が要求すタイプがイベントタイプであるときは A よりそのイベントタイプを持つ述語表記を抜き出し、述語表記にある引数の情報とともに B の項目内に追加したものを合成結果とする。

上記の解析法により、シソーラス中の主な名詞について、基本的な場合を導けることをプログラムを作成して確かめた。

目 次

第1章　はじめに

1.1　目的

名詞には多様な意味を持つものがある。しかし我々が日常の会話でその多様な意味の内どの意味で使われているのか困惑することは少ない。それは単語の組み合わせや、話している内容などによって意味が選択されるからである。つまり、語と語の意味が影響し合い、全体の意味が決まってくる。本研究ではこの語と語が影響し合い、全体の意味が導かれることに関する研究を行う。

日本語には名詞句「A の B」が多く見られる。この「A の B」という表現は出現度が高く、かつ自然言語処理において意味解析が困難とされている。それは助詞「の」で名詞 A と名詞 B とが繋がれただけで、助詞「の」が「A の B」全体の意味を特定するために何の働きもないからである。したがって、A と B との意味的関係から「A の B」全体の意味を導かなければならない。たとえば、「私の本」では「私が書いた本」、「私が読んだ本」などがあり、また、「家の本」という表現のときは、「家にある本」、「家についての情報が載っている本」などの意味が考えられ、A、B がどのような名詞であるかで、「A の B」が取り得る意味が変わってくる。

本研究では、システムに名詞の静的な意味情報を与え、それらの情報を使い単語の組み合わせによって意味が選択されることについて調べる。意味情報の枠組みとしては、生成語彙論に準拠し、単語の組み合わせによって意味が選択されるということを扱うために名詞句「A の B」という表現を使う。そして、その名詞句が A と B との関係から、どのような意味を取るかを出力するシステムを構築する。

生成語彙論は、従来の数え上げによって語彙情報を与えるのではなく、複数の面から詳細に語彙情報を記述し、さらに、それらの情報を生成的に操作することで意味解釈を導く方法を提案している。ここでの数え上げとは、例えば、名詞「りんご」は食物、植物などの側面をもち、その側面ごとに語彙情報を与えることである。

また、「A の B」は予め与えた意味情報では、その名詞句の意味を導き出せないときがある。それは、文脈に応じて「の」の解釈が予め与えた情報以上の意味情報を必要とし、

その情報から解釈を導く必要があるからである。そのために文脈に応じて情報を新たに取り出し補完する方法が必要となる。

本論文では、2 章で既存研究での名詞句「A の B」の意味を解析するために、どのようなアプローチがあったかを述べる。３章では本研究で用いた生成語彙論について述べる。4 章では、「A の B」の意味解析法について、扱う意味素性や生成語彙論による語彙項目をどのように組み合わせるかを述べる。5 章では、名詞句「A の B」の例の意味解釈が 4 章の合成方法を使って、どのようにして導かれるかについて説明する。

1.2 背景

自然言語処理において名詞句「A の B」の意味を解析することは困難とされている。それは、助詞「の」が「A の B」の意味を決定するのに役に立たず、名詞の意味や文脈等によって「A の B」の意味を導く必要があり、また、曖昧性を持つ語や、文脈等の影響により新しく出てくる意味解釈があるからである。

そのような「A の B」の意味解析における研究には、[2]、[3]、[6] などがある。[2] では生成語彙論による語彙情報を与え、名詞を三分類して三種類の解析方法によって「A の B」の意味解析を行い、検討点として慣用的表現や文脈情報により可能となる「A の B」の意味解釈を挙げている。[3] では、国語辞典の語釈文（例えば、名詞「コーチ」: 何らかのスポーツで技術を教える人）を利用した解析法と、格フレームを用いた解析法の二つを統合し「A の B」の意味解釈を導いている。検討点として、語釈文に意味的役割を表す記述がないときと、「A の B」が隠喩的であるときが類似度が十分に上がらず解析が困難であることを挙げている。[6] では名詞に素性を与え、その素性を使って「A の B」を意味的に五分類、その分類と素性を利用することで「A の B」の意味解析を行っている。検討点として、文脈情報が必要な場合、より詳しい知識かヒューリスティックスが必要な場合、類推などの推論が必要な場合を挙げている。

第2章　名詞句「AのB」の既存研究

名詞句「A の B」の研究に、

(1) 語彙意味情報に基づく日本語名詞句の意味解析-「A の B」を例に- [2]

(2) 名詞句の意味解析：国語辞典に基づく新しいアプローチ [3]

(3) 助詞「の」が結ぶ意味関係の解析 [6]

(4) 教師なし学習による日本語名詞句の言い換え [1]

などがある。(1) の研究では、名詞の使われ方を分析することより、「家」、「自動車」などの存在物を示す事物名詞、「会議」、「調査」などの事象を示す事象名詞、「前」、「兄」などの存在物間の関係や事象間の関係を示す関係名詞の三つに名詞を分類している。また、「AのB」に現れる名詞Aと名詞BでAを補部、Bを主辞として、補部と主辞との関係として以下の二つで捉えている。

- 補部が主辞の修飾語
- 補部が主辞の補語

補部が数量詞、形容詞相当語句、時間を指示する名詞であるとき、補部は主辞の修飾語で、主辞の項は統語情報による決定でき、名詞 A、B の意味的関係は計算する必要はなく、補部が主辞の補語の場合または、補部が主辞が修飾語であるが主辞の項を決定できないときが、統語情報だけでは意味解釈が困難な場合ある。この研究では語彙項目内で補部のタイプが主辞のデフォルト項のタイプと同一クラスのとき、補部は主辞の補語で、異なるクラスであれば主辞の修飾語としている。

解析手法としては、タイプ付きの単一化で行われているが、主辞の名詞の性質から単一化される項が異なることから、主辞が事物名詞、事象名詞、関係名詞であるときに分けて意味解析方法を分けている。また、解析結果として 1448 文中 774 例あった「A の B」を無作為に取り出した 100 例に対し解析を行い、100 例中 75 例が妥当な解釈を得ることが

できたとしている。ここでの妥当な解釈とは人間が妥当な文脈が考えられるような意味解釈がされた場合の解釈である。さらに補部が「私」、「兄」のように人間を指示したとき、A が B を所有しているという意味解析を行うようにしたとき、100 例中 89 例が妥当な解釈を得ることが出来たとしている。また、妥当な解釈が導けなかったものに「ウドの大木」、「赤の他人」ような慣用的表現があったとある。

(2) の研究では国語辞典の語釈文を利用して「A の B」の意味を解析している。語釈文は形態素解析と日本語構文解析と格解析を行い、名詞の意味役割を見つけるために利用される。国語辞典を利用した解析方法は、例えば、「ラグビーのコーチ」を解析するとき、「コーチ」の語釈文に‘ 何らかのスポーツで技術を教える人 ’とあった場合、この語釈文にある‘ スポーツ ’が持つ（NTT 意味素性辞書による）意味素性 SPORT、‘ 技術 ’が持つ意味素性 METHOD とラグビーが持つ意味素性 SPORT との間で類似度を算出する。この場合、‘ スポーツ ’と‘ ラグビー ’が類似度１、となり、‘ 技術 ’と‘ ラグビー ’が類似度 0.21 となり、類似度が高い方が選ばれ、解析結果は‘ ラグビーで技術をを教える人 ’となる。また、この解析に加え、格フレームを使った解析も行い二つの解析結果を統合し信頼性がより高い方を「A の B」の意味解析結果としている。実験は EDR 辞書、IPA 辞書と「A の B」に関する文献より 300 の「A の B」句を集め解析を行っている。、平均で一つの句に 1.1 の関係が与えられ、少なくとも一つの適切な関係が見つけ出された割合が 81%、全ての可能な適切な関係が見つけ出された割合と一つも適切でない関係が見つけ出された割合が 73%であったとしている。また、適切でない解析結果が出た原因として国語辞典の語釈文に意味役割となる記述がないときがあることと、句が隠喩的であり、類似度が十分に高くならないことを挙げている。

(3) の研究では、語が持ちうる情報 (意味) を明示的に示す手段としての素性を使い、A、B それぞれの語に素性のリストを辞書情報として与える。そして「A の B」の意味関係分類と決定のために主素性、依存素性、機能素性の三つの基準を設けている。

- 主素性　語の素性のうち、その語の主要な意味特徴を示す素性
  例、「イス」: thing,「イヌ」: animate,etc

- 依存素性　語の素性のうち、他の素性との意味的依存関係を示す素性
  依存関係を示す素性と依存対象を示す素性の対により表す。
  例、「日本人」: 属している (belong-to) 国 (nation) から [belogn-to nation] を依存素性として持つ。

- 機能素性　他の語との結合の仕方や、結合における役割を示す素性

素性名とその値を示す素性の対で表す。
例、「人間」:動作に対し、動作主 (agent) という格役割を持ちうることより [role agent] を機能素性として持つ。

加えて素性間の上位下位の階層関係を定義、さらに「A の B」の意味関係を、(a)A あるいは B の主素性、機能素性、(b)A あるいは B の依存素性により定義し (a) により大きく五分類し、さらに (b) の情報を使って小分類している。
大分類（五種類）

(1) B が述語相当語で、A はその述語の格要素である場合（太郎の結婚）（意味関係が格関係のとき）

(2) B が、後続の語に対し、A を基点にして格的な役割を示す場合（ビルの前）（意味関係が場所等の指定のとき）

(3) B が A の属性である場合（バラの色）（意味関係が属性の指定のとき）

(4) A が述語相当語で、B はその述語の格要素である場合（散歩の人）（意味関係が格関係のとき）

(5) A が B の一種の属性値であるとみなせる場合（弁護士の太郎）（意味関係が数量等による限定のとき）

そして「A の B」の意味関係の解析に A あるいは B の主素性を使い、それらの分類の内、なりうる意味関係を予測する。それからその予測を利用し素性ユニフィケーションを使って具体的な意味関係を求めている。解析実験では、一つの「A の B」について、複数個の意味関係が解析可能ならそれらを出力するようにし、そのうち一つでも適切な関係があれば正解とした場合、992 例について正しい意味関係を解析するがことができ、延べで出力された解析結果は 1365 個でそのうち約 96%が正しい結果であったとしている。また、出力されるべき関係が出なかった場合が５個あり、その検討点として、文脈情報が必要な場合、より詳しい知識かヒューリスティックスが必要な場合、類推などの推論が必要な場合を挙げている。

最近はコーパスと機械学習を用いた研究が盛んになってきている。そのような中で、(4) では「A の B」を自動的に言い換える研究を行っている。将来、このような研究と関係付けることも考えられる。

第3章　生成語彙論

生成語彙論 [5] は言語表現の意味をシステムが導くために使われる意味的情報をどのような形式で与えるか、また、その情報をどのように組合わせ、表現全体の意味を導くかについての手法が提案されている。生成語彙論では従来の数え上げによって名詞のタイプ毎に語彙項目を与えるのではなく、そのタイプ同士を項目内で述語表記を使って関係付けることで、タイプを一つの項目内にまとめる項目の記述を行っている。例えば、「銀行」は文脈によって組織ととられたり、場所や建物ととられたりする場合があるが、従来の研究では組織などに個別に語彙項目を与えている。これは組織や場所の間にある共通点を認識することを不可能にする。生成語彙論では「銀行」においての組織や場所などのタイプを一つの項目内に記述することによって、組織や場所が同時に参照されるときに組織や場所間にある関係を認識できるようにしている。また、名詞の意味を形式、構成、目的、主体の四つの側面に分け、さらに、タイプ強制、共合成、選択束縛の三つの操作を使って生成的に表現の意味を導く方法を提案している。

例えば、従来の語彙を使った意味解析では以下の文は困難となる。

I begin the book.

この文は字面だけでは意味は導けないが、生成語彙論では上述の語彙項目内の情報と操作を使うことで次のような文として意味解析が可能としている。

I begin to read the book.

この例では、「book」の項目内に「book」の目的として read という述語を持っているおり、「begin」がその read を要求していることから導けるとしている。

3.1　語彙項目について

生成語彙論による意味の記述は、項構造、事象構造、特質構造、語彙継承構造の４要素からなる。特質構造では語の述語関係を記述し、その述語関係で使われる引数のタイプ等を項構造と事象構造で記述する。また、語彙継承構造は語と語の関係を階層的に表すための要素である。生成語彙論による語彙項目の記述例を以下に示す。

$$\left[\begin{array}{lcl} car & & \\ \text{項構造} & = & \left[\ \text{項}1 = x : vehicle\ \right] \\ \text{特質構造} & = & \left[\begin{array}{lcl} \text{形式役割} & = & x \\ \text{目的役割} & = & drive(e,y,x) \\ \text{主体役割} & = & create(e,z,x) \end{array}\right] \end{array}\right]$$

また、四つの構造ついての説明は以下の通りである。

1 **項構造 (argument structure)**: 述語表現で扱われる引数と引数のタイプを記述

2 **事象構造 (event structure)**: 述語表現で扱われる事象についてのタイプと引数を記述

3 **特質構造 (qualia structure)**: 形式役割 (formal role)、構成役割 (constitutive role)、目的役割 (telic role)、主体役割 (agentive role) の４つに分けて、語の意味を表現

4 **語彙継承構造 (lexical inheritance structure)**: 語と語との間にある継承関係を表示

3.2 項構造

項構造で扱われる‘ 項 ’には、必須項 (true argument)、暗黙項 (default argument)、影項 (shadow argument)、付加詞 (true adjuncts) がある。以下にそれぞれの項について説明する。

- **必須項**: 特質構造における述語表現での必須格
- **暗黙項**: 特質構造における述語表現での任意格
- **影項**: 語彙項目に意味的に組み込まれているパラメータ
- **付加詞**: 論理表現を修飾するが、状況的な解釈の一部であり、特定の語彙項目の意味表現に関連づけられないパラメータ

3.3 事象構造

事象のタイプとして、状態 (state)、過程 (process)、変化 (transition) の 3 つを扱い、特質構造での述語表現で事象を扱うときに記述する。また、記号 $<_\propto$ 、$\circ_\propto$ 、$< \circ_\propto$ を使って事象と事象との時間的前後関係や中心となる事象 (主辞事象) を表記する。

- $<_\propto$: 時間的に重複なしの前後関係
- $\circ_\propto$: 時間的に重複
- $< \circ_\propto$: 時間的重複ありの前後関係

記述例としては次の通りである。

$$
\left[\begin{array}{ll}
develop & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}\,1 = x : animate \\ \text{項}\,2 = y : artifact \\ \text{デフォルト項}\,1 = z : artifact \end{array}\right] \\
\text{事象構造} & = \left[\begin{array}{l} \text{事象}\,1 = e1 : process \\ \text{事象}\,2 = e2 : state \\ Restr =< \circ_\propto \\ Head = e1 \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & exist(e2, y) \\ \text{主体役割} & = & build\_act(e1, x, z) \end{array}\right]
\end{array}\right]
$$

3.4 特質構造

特質構造では、意味情報を四つの役割に分けて記述する。以下で四つの役割について説明する。

- **形式役割**: 上位概念内での対象概念を区別するもの
- **構成役割**: 対象概念とその構成要素や適した部分の関係
- **目的役割**: 対象概念の目的と機能
- **主体役割**: 対象概念の発生や起源に含まれる要因

3.4.1 形式役割

形式役割では、上位概念を使って、語のタイプ等の静的な属性を表記する。語を表すタイプとしては、単一のタイプで表されるときと、複数のタイプ (dotted type) で表されるときがある。

- シンプルタイプ：引数の sortal typing に相当する形式役割の値
- 複合タイプ：異なるタイプ間にある関係を定義する形式役割の値

シンプルタイプを示す名詞に対する語彙項目の簡単な記述形式を以下に示す。

$$\left[\begin{array}{ll} \alpha & \\ \text{項構造} & = \left[\ \text{項}\,1 = x : type\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ x\ \right] \end{array}\right]$$

また、名詞「man」に対する語彙項目の記述例を以下に示す。

$$\left[\begin{array}{ll} man & \\ \text{項構造} & = \left[\ \text{項}\,1 = x : human\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ x \\ \text{構成役割} & = \ male(x) \end{array}\right] \end{array}\right]$$

さらに、複合タイプの場合は α が複数のタイプを持ち、それらのタイプを形式役割で関係付けて記述する。語彙項目の記述形式は以下の通りである。

$$\left[\begin{array}{ll} \alpha & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}\,1 = x : type1 \\ \text{項}\,2 = y : type2 \end{array}\right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ relation(y, x)\ \right] \end{array}\right]$$

また、information と physical_object をタイプとして持つ名詞「book」に対する語彙項目の記述例を以下に示す。

$$
\left[\begin{array}{ll}
book & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : information \\ \text{項}2 = y : physical\_object \\ \text{デフォルト項}1 = w : human \\ \text{デフォルト項}2 = v : human \end{array}\right] \\
\text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \\ \text{事象}2 = e2 : transition \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & hold(y, x) \\ \text{目的役割} & = & read(e1, w, x.y) \\ \text{主体役割} & = & write(e2, v, x.y) \end{array}\right]
\end{array}\right]
$$

3.4.2 構成役割

対象概念と対象概念の構成要素や部分要素との関連を記述する。名詞「hand」を使って語彙項目の記述例を示す。

$$
\left[\begin{array}{ll}
hand & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : limb \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & x \\ \text{構成役割} & = & part\_of(x, y : body) \end{array}\right]
\end{array}\right]
$$

この語彙項目は構成役割で「hand」が‘ body ’となるものの一部であることを関係付けている。

3.4.3 目的役割

目的役割の‘ 目的 ’には、人が対象概念に直接的に行なうものと対象概念によってなされるものとの二つがある。前者を直接目的 (direct telic)、後者を用途目的 (purpose telic) と呼ぶ。直接役割と用途目的の語彙項目の記述例を以下に示す。

直接役割：名詞「beer」を例に

$$\left[\begin{array}{ll} beer & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : liquid \\ \text{デフォルト項}1 = w : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = x \\ \text{目的役割} & = drink(e1, w, x) \end{array}\right] \end{array}\right]$$

この語彙項目では、目的役割で beer を human が drink するものであることを示している。

用途役割：名詞「knife」を例に

$$\left[\begin{array}{ll} knife & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : tool \\ \text{デフォルト項}1 = y : physical\_object \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = x \\ \text{目的役割} & = cut(e1, x, y) \end{array}\right] \end{array}\right]$$

この語彙項目では、目的役割で knife が physical_object を cut するものであることを示している。

3.4.4 主体役割

対象概念の発生や起源に関する関連付けを行い記述する。名詞「car」を使って語彙項目の記述例を示す。

$$\left[\begin{array}{ll} car & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : vehicle \\ \text{デフォルト項}1 = w : human \\ \text{デフォルト項}2 = v : human \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = x \\ \text{目的役割} & = drive(e1, w, x) \\ \text{主体役割} & = create(e2, v, x) \end{array}\right] \end{array}\right]$$

これは主体役割で human が「car」を create することを示している。

3.5 語彙継承構造

語彙継承構造では生成語彙論による記述で扱われるタイプの階層を定義する。生成語彙論ではこの階層が特質構造の役割ごとに定義される。これは例えば、「辞書は本である」と is-a 関係により捉えた場合、「本」の語彙項目から全ての情報を継承すると問題が生じる。その問題とは「本」の目的としては‘ 読む ’ということが考えられるが、「辞書」が「本」の情報を継承することによって辞書は‘ 引く ’ものであるのに辞書を‘ 読む ’ものとなってしまうことである。このような多重継承の問題を役割ごとに継承することで回避している。

第4章　生成語彙論に基づく名詞句「Aの B」の意味解析法

4.1　主なカテゴリに対する語彙記述

個々の名詞に語彙記述を与えるのではなく、基本的に、名詞に対応するカテゴリに語彙記述を与える。

4.1.1　シソーラス

本研究では、IPAL[4] の意味素性をシソーラスとして利用している。以下に扱うシソーラスの階層構造を示す。

動物の領域
(1) animate(ANI)
　　(1-1) animal(ANL)
　　(1-1-1) human(HUM)

具体物の領域
(2) concrete(CON)
　(2-1) automata(AUT)
　(2-2) edible(EDI)
　(2-3) liquid(LIQ)
　(2-3) pasty(PAS)
　(2-4) solid(SOL)
　(2-5) con_ product

場所の領域

(3) top

(3-1) locus(LOC)

(3-2) interior(INT)

(3-3) organization(ORG)

(3-4) network(NET)

(3-5) space(SPA)

出来事および動作/作用の領域

(4) top

(4-1) phenomenon(PHE)

(4-2) natural entitiy(NAT)

(4-3) plant(PLA)

(4-4) gaseous(GAS)

(4-5) element(ELM)

(4-6) potency(POT)

(4-7) activity(ACT)

(4-8) event(EVE)

(4-9) appointment(APO)

(4-10) product(PRO)

(4-11) resultant(RES)

(4-12) process(PRC)

抽象の領域

(5) top

(5-1) price(PRI)

(5-2) measure unit(MEA)

(5-3) social bonds(SOC)

(5-4) gradable(GRA)

(5-5) attribute(ATT)

(5-6) reciprocal(REC)

(5-7) personality(PER)

(5-8) mind(MIN)

(5-9) manner(MAN)

(5-10) forms(FOR)

(5-11) evaluation(EVA)

(5-12) currency(CUR)

(5-13) duration(DUR)

(5-14) distance(DIS)

(5-15) item(ITM)

(5-16) ratio(RAT)

(5-17) quantity(QUA)

(5-18) value(VAL)

(5-19) state(STA)

(5-20) role(ROL)

(5-21) relational terms(REL)

(5-22) direction(DIR)

(5-23) phases(PHA)

(5-24) reference point(REF)

(5-25) norms(NOR)

(5-26) subfield(FLD)

(5-27) information(INF)

(5-28) inclination(INC)

(5-29) quality(QAL)

(5-30) proposition(PRP)

(5-31) stage(STG)

(5-32) appearance(APP)

(5-33) unit(UNT)

(5-34) point in time(PIT)

(5-35) time(TIM)

(5-36) ordinal(ORD)

(5-37) name(NAM)

(5-38) entity(ENT)

(5-39) congregation(GAT)

(5-40) kind(KND)

(5-41) abstract(QUA)

この階層の適切なタイプで該当する機能に語の属性・性質を記述する。[1]

4.1.2 各カテゴリーに対する生成語彙論による記述

(1) animate ヒトを含む動物で、文脈情報などからその指示対象を特定することが可能と考えられるもの。(例：父親、子供、生き物など)

- animate は top を上位概念とする。
  形式役割に animate を表す引数を使って表現。
  formal = animate(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} animate & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ animate(x)\ \right] \end{array}\right]$$

(1-1) animal ヒト以外の動物で、文脈情報などからその指示対象を特定することが可能と考えられるもの。(例：犬、牛、トラなど)

- animal は animate を上位概念とする。
  形式役割に animal を表す引数を使って表現。
  formal = animal(x:animate)
  以下に全体の語彙記述を与える。

[1]形式役割は生成語彙論では x などの変数が記述されるだけだが、本研究では関数を使い '本 (x)' のような表現を記入する。また、生成語彙論ではタイプの階層が一部しか書かれていないため本研究では IPAL basic noun の意味素性をタイプとして扱う。

$$\left[\begin{array}{ll} animal & \\ \text{項構造} & = \left[\ \text{項}1 = x : animate\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ animal(x)\ \right] \end{array}\right]$$

(2-2) human(人間) ヒトで、特定可能なもの。(例：彼、太郎、先生、父親など)

- human は animal を上位概念とする。
  形式役割に human を表す引数を使って表現。
  formal = human(x:animal)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} human & \\ \text{項構造} & = \left[\ \text{項}1 = x : animal\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ human(x)\ \right] \end{array}\right]$$

(2) concrete 物質としての側面をもち、視覚を始めとする五感で認識可能なもの。(手紙、頭、桜など)

- concrete は top を上位概念とする。
- concrete は、ヒトが所有することができる。
  述語 possess をつかって目的役割で表現、
  telic = possess(e:state,y:human,x:concrete)

全体の語彙記述を以下に与える。

$$\left[\begin{array}{ll} concrete & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{項}2 = y : locus \\ \text{デフォルト項}1 = z : human \end{array}\right] \\ \text{事象構造} & = \left[\ \text{事象}1 = e1 : state\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ concrete \\ \text{目的役割} & = \ possess(e1, z, x) \end{array}\right] \end{array}\right]$$

(2-1) automaton 乗り物や機械などのうち自立的な処理機能をもつもの。(例：車、コンピュータ、電子レンジなど)

- automaton は concrete を上位概念とする。
  形式役割に concrete を表す引数を使って表現。
  formal = automaton(x:concrete)

- automaton は人が操作することを目的とする。
  述語 operate を使って、目的役割で表現。
  telic = operate(e:process,y:human,x:automaton) 以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} automaton & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}\,1 = x : con\_product \\ \text{項}\,2 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}\,1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ automaton(x) \\ \text{目的役割} & = \ operate(e1, y, x) \end{array}\right] \end{array}\right]$$

(2-2) edible ヒトが日常的に食用とするもの。（例：料理、ラーメン、味噌汁など）

- edible は concrete を上位概念とする。
  形式役割に concrete を表す引数を使って表現。
  formal = edible(x:concrete)

- edible は人が食べるもの。
  述語 eat を使って、目的役割で表現、
  telic = eat(process,x:human,y:foods)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} edible & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}\,1 = x : concrete \\ \text{項}\,2 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}\,1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ edible(x) \\ \text{目的役割} & = \ eat(e1, y, x) \end{array}\right] \end{array}\right]$$

(2-3) liquid 液体。（例：ジュース、スープ、雨水など）

- liquid は concrete を上位概念とする。
  形式役割に concrete を表す引数を使って表現。
  formal = liquid(x:concrete)

- liquid は凍る。述語 freeze を使って、形式役割で表現、formal = freeze(e:transition,x:liquid)

- liquid は飲むことができる。
  述語 drink を使って、目的役割で表現、
  telic = drink(e:process,y:human,x:liquid) 以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} liquid & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : concrete \\ \text{項}2 = y : animate \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : transition \\ \text{事象}2 = e2 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = liquid(x), freeze(e1, x) \\ \text{目的役割} & = drink(e2, y, x) \end{array}\right] \end{array}\right]$$

(2-4) pasty(粘着性のもの)　粘着性のもの。（例：糊、ポタージュ、クリームなど）

- pasty は concrete を上位概念とする。
  形式役割に concrete を表す引数を使って表現。
  formal = pasty(x:concrete)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} pasty & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : concrete \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = pasty(x) \end{array}\right] \end{array}\right]$$

(2-5) solid　固体。

- solid は concrete を上位概念とする。
  形式役割に solid を表す引数を使って表現。
  formal = solid(x:concrete)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} solid & \\ \text{項構造} & = \left[\ \text{項}1 = x : concrete\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ solid(x)\ \right] \end{array}\right]$$

(3-1) locus ヒトが活動する場所。(例：学校、病院、日本など)

- locus は concrete が存在する場所または、event が行われる場所を表す。
  述語 at を使って形式役割で表現。
  formal = in(y:concrete.event,x:top)

- interior は、内部からの出現を示す。
  述語 go_ out を使って形式役割で表現、
  formal = go_out(e:process,y:concrete,x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} locus & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : concrete.event \end{array}\right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ at(y,x)\ \right] \end{array}\right]$$

(3-2) interior 具体的に指し示すことができるもののうち、立方体の五方または、六方に囲まれているもの、あるいは平面体の三辺または、四辺を囲まれているもの。建物や乗り物の内部。身体部位のうち、口や鼻などの開口部をもつもの。(例：箱、ビル、車、耳など)

- interior は内部の存在を示す。
  述語 in を使って形式役割で表現、
  formal = in(y:concrete,x:interior)

- interior は、内部からの出現を示す。
  述語 go_ out を使って形式役割で表現、
  formal = go_out(e:process,y:concrete,x:interior)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{lcl} interior & & \\ 項構造 & = & \left[\begin{array}{l} 項1 = x : \top \\ 項2 = y : concrete \\ 項3 = z : concrete \end{array}\right] \\ 事象構造 & = & \left[\ 事象1 = e1 : process\ \right] \\ 特質構造 & = & \left[\ 形式役割 \ = \ in(y,x), go\_out(e1,z,x)\ \right] \end{array}\right]$$

(3-3) organization ヒトがそこに所属することで成り立つ組織や活動母体。(例：学校、日本、店など)

- organization は幾人かのヒト、または組織で構成される集団。
  述語 member_of を使って構成役割で表現、
  constitutive = member_of(x:human,y:organization)

- organization は組織に属すことでヒトは役割 (role) を持つ。
  述語 post を使って構成役割で表現。
  constitutive = post(x:human,y:role)

- organizaiton は、人または、組織に組織されて出来る。
  述語 organize を使って主体役割で表現。
  agentive = organize(e:process,x:human,y:organization)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{lcl} organization & & \\ 項構造 & = & \left[\begin{array}{l} 項1 = x : \top \\ デフォルト項1 = y : human \\ デフォルト項2 = z : role \end{array}\right] \\ 事象構造 & = & \left[\ 事象1 = e1 : process\ \right] \\ 特質構造 & = & \left[\begin{array}{l} 形式役割 \ = \ organization(x) \\ 構成役割 \ = \ member\_of(y,x), post(y,z) \\ 主体役割 \ = \ organize(e1,y,x) \end{array}\right] \end{array}\right]$$

(3-4) network 交通網、公共エネルギー、生命保険などネットワークとしての側面を

持つが、それを利用することに力点があり、たとえそこに加入あるいは利用可能でもメンバーには何の役割も振られず活動母体とは認めにくいもの。（例：電話、鉄道、メーリングテストなど）

- network は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  formal = network(x:top)

$$\left[\begin{array}{ll} network & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ network(x) \end{array}\right] \end{array}\right]$$

(3-5) space 場所を表す概念のうち locus、interior、organization、network のいずれにも特定できないもの。論理・思考・作品空間を含む。（例：世界、地獄、腹、漫画など）

- space は情報（information）に対する空間を表す。
  述語 for を使って形式役割で表現。
  formal = for(x:top,y:information)

$$\left[\begin{array}{ll} network & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : information \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ for(x,y) \end{array}\right] \end{array}\right]$$

(4-1) phenomenon 自然現象、天候現象、生理現象のうち、その成立/変化/消滅に関するもの、あるいはその程度か結果に着目したもの。ヒトの利用するエネルギーのうち実態の曖昧なもの。natural_entity、plant、gaseous、element、potency 以外の自然現象。（例：地震、光、電気、呼吸、しもやけ、ひげなど）

- phenomenon は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  formal = phenomenon(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} phenomenon & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ phenomenon(x)\ \right] \end{array}\right]$$

(4-2) natural_entity 自然現象のうち存在・移動に関するもの、あるいはその位置に着目されているもの。(例：台風、地球、高気圧など)

- natural_entity はある場所へと動く。
  述語 move を使って、形式役割で表現。
  formal = move(x:natural_entity,y:locus)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} natural\_entity & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : locus \end{array}\right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ natural\_entity(x), move(x,y)\ \right] \end{array}\right]$$

(4-3) plant 生えたり、実ったりする主体。主に、植物、果実など。(例：木、桜、みかんなど)

- plant は成長する。
  述語 grow を使って、主体役割で表現、
  agentive = grow(e:transition,x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} natural\_entity & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \quad \text{事象構造} & = \left[\ \text{事象}1 = x : transition\ \right] \\ \text{特質構造} & = \left[\ \text{主体役割} \ = \ grow(e1, x)\ \right] \end{array}\right]$$

(4-4) gaseous 気体。(例：ガス、空気、息、煙草など)

- gaseous は吸うことができる。
  述語 inhale を使って、形式役割で表現、
  inhale(e:process,y:human,x:gaseous)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} gaseous & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top \text{デフォルト項}1 = y : human\ \right] \\ \text{事象構造} & = \left[\ \text{事象}1 = e1 : process\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = gaseous(x) \\ \text{目的役割} & = inhale(e1, y, x) \end{array}\right] \end{array}\right]$$

(4-5)element　それぞれの専門分野で存在が認められているもの。（例：ウィルス、たんぱく質、鉄など）

- element は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  formal = element(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} element & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ element(x)\ \right] \end{array}\right]$$

(4-6) potency　動物の身体部位の機能。（例：胃、心臓、内蔵など）

- potency は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  constitutive = part_of(x:top,y:animate)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} potency & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : animate \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = potency(x) \\ \text{構成役割} & = part\_of(x, y) \end{array}\right] \end{array}\right]$$

(4-7) activity　ヒトの意志的に行う動作 (動詞のうち、肯定命令が可能なもの)。(例：テニス、料理、要求など) 概念毎に語彙の記述を行う必要がある。
(4-8) event　ある時間ある場所でヒトが集まって計画的に行われる催しもの。(例：ウィルス、たんぱく質、鉄など)

- event はある時間 (point_in_time)、ある場所 (locus) で。
  述語 occur を使って、主体役割で表現。
  主体役割 = occur(x:top,point_in_time.locus)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} event & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{項}2 = y : point\_in\_time.locus \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}\ 1 = e1 : process\right] \\ \text{特質構造} & = \left[\text{主体役割} \ = \ occur(e1, x, y)\right] \end{array}\right]$$

(4-9) appointment　あらかじめ決められた予定に従って行われる行動。(例：計画、電車、学校など)

- appointment は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  formal = appointment(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} appointment & \\ \text{項構造} & = \left[\text{項}1 = x : \top\right] \\ \text{特質構造} & = \left[\text{形式役割} \ = \ appointment(x)\right] \end{array}\right]$$

(4-10) product　ヒトの生産あるいは創作活動の結果生じるモノ、あるいは情報を担った作品。(例：ごはん、ビル、手紙など)

- product はヒトの手によって作られるもの。
  述語 make を使って、主体役割で表現。
  agentive = make(e:process,y:human,x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} product & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = product(x) \\ \text{主体役割} & = make(e1, y, x) \end{array}\right] \end{array}\right]$$

(4-11) resultant ある行為や変化の後に生じる状態あるいはその評価。(例:結果、損失、到達など)

- resultant は何かによって引き起こされるもの。
  述語 occur を使って主体役割で表現。
  主体役割=occur(e:process,y:top,x:resultant) 以下に語彙項目を示す。

$$\left[\begin{array}{ll} resultant & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : \top \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = resultant(x) \\ \text{主体役割} & = occur(e1, y, x) \end{array}\right] \end{array}\right]$$

(4-12) process 動きや変化のうち、phenomenon、natural entity、plant、gaseous、element、potency、activity、event、appointment、resultant、process のいずれにも特定できないもの。(例:回転、消化、遅刻、人生など)

- process は top を上位概念とする。
  形式役割に top を表す引数を使って表現。
  formal = process(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} process & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{l} \text{形式役割} \quad = \quad element(x) \end{array}\right] \end{array}\right]$$

(5-1) price お金を表す概念。(例:金、値段、資金など)

- price はお金を表し、お金の金額等を表す意味素性 currency と関係付けられる。
  述語 cost,value_of を使って形式役割で表現。
  主体役割=cost(y:⊤,x:price),value_of(z:currency,x)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} price & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : abstract \\ \text{項}2 = y : \top \\ \text{項}3 = z : currency \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; cost(y,x), value\_of(z,x)\right] \end{array}\right]$$

(5-2) measure_unit 計測できる尺度。(例:背、面積、数、重量など)

- measure_unit は計測できる尺度を表し、尺度の数量や程度等を表す意味素性 duration/distance/item/ratio/quantity と関係付けられる。
  述語 have,value_of を使って形式役割で表現。
  formal = have(y:⊤,x:price), value_of(z:duration.distance.item.ratio.quantity,x)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} measure\_unit & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : abstract \\ \text{項}2 = y : \top \\ \text{項}3 = z : duration.distance.item.ratio.quantity \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; have(y,x), value\_of(z,x)\right] \end{array}\right]$$

(5-3) social bonds 二者関係を表す概念のうち、程度に着目した述語や二者関係の成立/消滅に着目した述語に従属するもの。(例:つながり、縁、格差など) 概念毎に語彙を記述する必要がある。

(5-4) gradable 計ることができないが、程度に着目した述語に従属するもののうち、social bonds 以外で、「高い/低い/大きい/小さい/広い/狭い/強い/弱い/多い/少ない/重い/軽い/深い/浅い/濃い/薄い」に従属するもの。また、そこから推論可能な範囲にある評価を表す述語に従属するもの。(例:地位、心、腰、影響など) 高い/低いなどの概念毎に語彙を記述する必要がある。

(5-5) attribute
程度に着目した述語に従属するもののうち、price、measure unit、social bonds、gradable 以外のもの。(例:傾斜、決意、非常識、消耗など）名詞によってどのような述語をとるかが異なるので概念毎に語彙を記述する。

(5-6) reciprocal 二者間の関係を表す概念のうち、socical bonds ではなく、評価に着目した述語に従属するもの。また、二者関係の成立/消滅に関する述語に従属するもの。(例:仲、相性、間など) 概念毎に語彙の記述を行う必要がある。

(5-7) personality 性格や性質を表す概念。(例:気立て、人当たり、気性など)

- personality は生物 (human) の性格や性質を表す。
  述語 nature_of を使って構成役割で表現。
  構成役割=nature_of(x:top,y:animate)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} personality & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}2 = y : animate \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ personality(x) \\ \text{構成役割} & = \ nature\_of(x,y) \end{array}\right] \end{array}\right]$$

(5-8) mind 動物の感性や知能。(例:良心、感覚、神経など)

- mind は生物 (animate) の感性や知能を表す。
  述語 nature_of を使って構成役割で表現。
  構成役割=nature_of(x:mind,y:animate)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} mind & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}2 = y : animate \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ mind(x) \\ \text{構成役割} & = \ nature\_of(x,y) \end{array}\right] \end{array}\right]$$

(5-8) manner ヒトなどの能力や傾向。(例:考え方、行動、働きぶりなど)

- manner は生物 (animate) と組織 (organization) の能力や傾向を表す。
  述語 nature_of を使って構成役割で表現。
  構成役割=nature_of(x:top,y:animate.organization)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} manner & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}2 = y : animate.organization \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & manner(x) \\ \text{構成役割} & = & nature\_of(x,y) \end{array}\right] \end{array}\right]$$

(5-10) forms　評価や具体的な属性に着目した述語に従属するもののうち、reciprocal,personality,mind,manner 以外で、「良い/悪い」に従属するもの。名詞によってとりうる述語が異なるため個別に語彙を記述する必要がある。

(5-11) evaluation　評価に着目した述語に従属するものうち、reciprocal、personality、mind、manner、forms 以外のもの。（例：経済、事態、舌など）名詞によってとりうる述語が異なるため個別に語彙を記述する必要がある。

(5-12) currency　金額を表す値。(例:1000 円、一万円、100 万ドルなど)

- currency は数字を使って、金額を表す。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数')
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} currency & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & currency(x) \\ \text{構成役割} & = & number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-13) duration　時間、期間、年齢 (時間的な延長) を表わす値。(例:三ヶ月、三回、二十歳など)

- duration は時間的長さを表す。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数') 以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} duration & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = duration(x) \\ \text{構成役割} & = number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-14) distance 距離、長さ (空間的な延長) を表わす値。(例:五キロメートル、1 センチメートルなど)

- distance は空間的長さを表す。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数')
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} distance & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = distance(x) \\ \text{構成役割} & = number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-15) item ヒトやモノの数を表す値。(例:三百人、六頭、十枚など)

- item はモノの数を表わす。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数')
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} item & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = item(x) \\ \text{構成役割} & = number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-16) ratio 割合や率を表わす値。(例:50 パーセント、3 割、120km/h など)

- ratio は割合などの値を表わす。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数') 以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} ratio & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = ratio(x) \\ \text{構成役割} & = number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-17) quantity 回数や量を表わす値。duration、distance、item、ratio 以外の数値。(例:五回、千件、10 キログラムなど)

- quantity は回数や量を表わす。
  述語 number を使って構成役割で表現。
  constitutive = number(x:top, 数')
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} quantity & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = quantity(x) \\ \text{構成役割} & = number(x, \text{数}') \end{array}\right] \end{array}\right]$$

(5-18) value 形や色を表わす段階性のない値。(例:赤、四角、クリームなど) 赤、四角など個別な意味を持つので名詞毎に語彙を記述する必要がある。

(5-19) value ある一つの状態 (いわゆる形容動詞語幹が状態副詞的な連用修飾として働く場合)。ある状態からある状態へ移行する境界としての状態 (状態あるいは変化の度合いを計る物差しになる場合)。(例:安易、極端、定数など) 個別な意味を持ちまとめることが困難なので名詞毎に語彙を記述する必要がある。

(5-20) role 役職名などの社会的な役割。(例:議長、患者、大工など)

- role はヒトの役職を表わす。また役職につくには組織 (organization) に属することになる
  述語 post、member を使って形式役割で表現。
  formal = post(y:human,x:top),member(y,z:organization)
  以下に全体の語彙項目を示す。

$$\left[\begin{array}{ll} role & \\ & \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : human \\ \text{デフォルト項}2 = z : organization \end{array}\right] \\ \text{項構造} \quad = & \\ \text{特質構造} \quad = & \left[\text{形式役割} \quad = \quad post(y,x), member(y,z)\right] \end{array}\right]$$

(5-21) relation terms 親族あるいは交友関係。(例:娘、いとこ、家庭など) 娘、いとこなどの概念に応じた述語を使って、語彙を記述する必要がある。

(5-22) direction 方向。相対的な位置や進行方向を持つベクトルを表わす。(例:北、上、前など) ある場所 (locus) からを基点とした位置、方向を表わすために意味素性の項目内に情報をまとめることは困難となる。ある場所を表す述語と、その場所からの方向を表す述語を使って概念毎に語彙を記述する必要がある。

(5-23) phases 話など、内容を持つ一続きのものや、記号や物などの列・並びや、幾つかの出来事の連続によって構成される一つの出来事などについての時間的順序・位置的順序。(例:最初、前、2 番目など) ある基点を表す述語と、その基点からの時間的、位置的関係を表す述語を使って概念毎に語彙を記述する必要がある。

(5-24) reference point ある基準点によって相対的に決まる対象。(現実世界ではどの素性に当てはまるかわからないもの、あるいは animate、concrete、space、process、abstract の中で複数の名詞句が該当するもの)(例:反対、すべて、以下など) ある対象を基準にすることによって決まる概念であるので、概念ごとに語彙を記述する必要がある。

(5-25) norms 法則、規則や方法。(例:法律、原理、標準など)

- norms は上位概念として top を持つ。
  形式役割に top を表す引数を使って表現。
  formal = norms(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} norms & \\ \text{項構造} \quad = & \left[\text{項}1 = x : \top\right] \\ \text{特質構造} \quad = & \left[\text{形式役割} \quad = \quad norms(x)\right] \end{array}\right]$$

(5-26) subfield 学術分野、スポーツや芸術などの分野。(例:科学、哲学、経営など)

- subfield は上位概念として top を持つ。
  形式役割に top を表す引数を使って表現。
  formal = subfield(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} subfield & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ subfield(x)\ \right] \end{array}\right]$$

(5-27) information 情報の持つ内容。(例:手紙、せりふ、予想など)

- information は上位概念として top をもつ。
  形式役割に top を表す引数を使って表現。
  formal = information(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} information & \\ \text{項構造} & = \left[\ \text{項}1 = x : \top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ information(x)\ \right] \end{array}\right]$$

(5-28) inclination ヒトがある対象に対して持つ心理的傾向。(例:関心、警戒心、意外など)

- inclination はヒトがもつもの。
  述語 have を使って形式役割で表現。
  formal = have(e:state,y:human,x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} inclination & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\ \text{事象}1 = e1 : state\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ inclination(x), have(e1, y, x)\ \right] \end{array}\right]$$

(5-29) quality ある基準や標準よりも大きい数値。(例:厚さ、重さ、高さなど)

- quality は何らかの物体 (concrete) がもつもの。
  述語 have を使って構成役割で表現。
  constitutive = have(y:concrete,x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} quality & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{項}2 = y : concrete \text{ デフォルト項}1 = y : human \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \quad = \quad quality(x), have(e1, y, x)\right] \end{array}\right]$$

(5-30) proposition ある値や程度が大きいこと、あるいは小さいこと。(例:薄さ、荒さ、軽さなど) 薄さ、荒さと結び付く概念がそれぞれ違うため、概念ごとに語彙を記述し、それぞれの結ぶつきを表す必要がある。

(5-31) stage ある一時的な状態で、state、resultant、quality でないもの。(例:緊張、姿勢、バランスなど)

- stage は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = stage(x:top)

以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} stage & \\ \text{項構造} & = \left[\text{項}1 = x : top\right] \\ \text{特質構造} & = \left[\text{形式役割} \quad = \quad stage(x)\right] \end{array}\right]$$

(5-31) appearance 視覚チャンネルから主として得られる外的な印象。(例:輪郭、柄、色、服装など)

- appearance は top を上位概念としてもつ。
  形式役割に top を表す変数を使って表現。
  formal = appearance(x)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} appearace & \\ \text{項構造} & = \left[\ \text{項}1 = x : top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ appearance(x)\ \right] \end{array}\right]$$

(5-32) unit 通貨や単位。(例:円、ドル、リラなど)

- unit は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = unit(x)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} unit & \\ \text{項構造} & = \left[\ \text{項}1 = x : top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ unit(x)\ \right] \end{array}\right]$$

(5-33) point in time 不定語「いつ」で質問された場合に、年月日/時刻/時間区分を特定して答えることができるもの。(例:今年、初日、運動会など)

- point in time は top を上位概念としてもつ。
  形式役割に top を表す変数を使って表現。
  formal = point_in_time(x)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} point\_in\_time & \\ \text{項構造} & = \left[\ \text{項}1 = x : top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ point\_in\_time(x)\ \right] \end{array}\right]$$

(5-35) time 時間に関する概念のうち、point in time でも phases でも measure unit でもないもの。出来事の間の順序関係を表す名詞句や、時間的な幅を指す名詞句。(例:時間、タイミング、頭など)

- time は top を上位概念としてもつ。
  形式役割に top を表す変数を使って表現。
  formal = time(x)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} time & \\ \text{項構造} & = \left[\ \text{項}1 = x : top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ time(x)\ \right] \end{array}\right]$$

(5-36) ordinal 順序や順番などを表わすもの。(例:スケジュール、予定、方向など)

- time は top を上位概念としてもつ。
  形式役割に top を表す変数を使って表現。
  formal = ordinal(x)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} ordinal & \\ \text{項構造} & = \left[\ \text{項}1 = x : top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ ordinal(x)\ \right] \end{array}\right]$$

(5-37) name ヒトやモノの名前。(例:花子、太郎、辺 AB など)

- name はもの (concrete) が持つことができる。
  述語 name を使って形式役割で表現。
  formal = name(x:concrete, 名前')
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} name & \\ \text{項構造} & = \left[\ \text{項}1 = x : concrete\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ name(x, \text{名前}')\ \right] \end{array}\right]$$

(5-38) entity 文字や単語などのメタ言語、記号や数学的な対象、音声面に焦点が合わせられる名詞句。指事語、不定語、専門用語など。(例:単語、変数 a、どれ、集合など)

- entity は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = entity(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} entity & \\ \text{項構造} & = \left[\ \text{項}1 = x{:}\top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ entity(x)\ \right] \end{array}\right]$$

(5-39) congregation 複数の動物、モノ、組織。典型的に「多い」「集まる」などに従属する名詞句や、漠然と human や organization を指す名詞句、human や organization の条件に合致しない名詞句、locus と並列する名詞句で、そこに存在する複数の人を表わす述語に従属する名詞句、状態副詞のように機能する「X デ〜する」という構文に現れる名詞句。(例:父兄、グループ、委員など)

- congregation は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = congregation(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} congregation & \\ \text{項構造} & = \left[\ \text{項}1 = x{:}\top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ congregation(x)\ \right] \end{array}\right]$$

(5-40) kind 種や類をあらわす総称的なもの。role 以外の役割 (例:人類、高級品、柱など)

- kind は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = kind(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} kind & \\ \text{項構造} & = \left[\ \text{項}1 = x{:}\top\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ kind(x)\ \right] \end{array}\right]$$

(5-41) abstract 素性 (5-1) から (5-40) までのいずれにも特定できないもの。(例:不可能、特徴、価値など)

- astract は top を上位概念とする。
  形式役割に top を表す変数を使って表現。
  formal = abstract(x:top)
  以下に全体の語彙記述を与える。

$$\left[\begin{array}{ll} abstract & \\ \text{項構造} & = \left[\text{項}1 = x : \top\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; abstract(x)\right] \end{array}\right]$$

4.2 語彙記述の合成方法

A と B の語彙記述を合成することで「A の B」の意味構造を生成する。語彙項目を以下のように合成していく。
B の語彙項目で、特質構造にある述語引数 x に対して、

1 x のタイプが、A のタイプとマッチ。

- B の項構造にある x のタイプのところに A の語彙項目を代入

2 x のタイプが事象であるとき、

- A の特質構造にある述語が事象を表すとき、その述語を B の語彙項目に追加。

以下にアルゴリズムを書く。

function A_no_B(A, B) **returns** result_list
  intermidiate_result ← A_no_B1(A, B)
  if intermediate_result
    then push(intermediate_result, result_list)
  intermediate_result ← A_no_B1(B, A)
  if intermediate_result
    then push(intermediate_result, result_list)

function A_no_B1(x, y) **returns** intermediate_result
  if 名詞 x のタイプと, y の語彙情報にあるタイプがマッチ

then intermediate_result

← y でマッチしたタイプに x を代入した y

else if x が形式役割が 2 変数以上を引数をもつ述語

then intermediate_result

← x の形式役割にある述語を取り出し、

y の形式役割の述語として、またその取り出した述語の引数

のタイプも y の項構造, 事象構造に加えた y

else if x の項構造の変数がイベント変数

then x の項構造にあるイベント変数 v を引数として持つ述語を

x より取り出す

if v のタイプと y の語彙項目にある述語の引数がもつ

タイプとがマッチ

then intermediate_result

← 取り出した述語とその述語内のタイプ等を y に加え,

変数の辻褄を合わせた y

else

y のタイプの上位タイプが持つ語彙項目を y として継承し,

A_no_B1(x, y)

例を通して、「A の B」の解析法を説明していく。

4.3 解析例：名詞句「私の本」

まず、「私」は意味素性として human を持ち、形式役割に述語表記として私（x:human）を持つ。よって、「私」の語彙項目は以下のようになる。

$$\left[\begin{array}{l} \text{私} \\ \text{項構造} \quad = \quad \left[\ \text{項}\,1 = x : human\ \right] \\ \text{特質構造} \quad = \quad \left[\ \text{形式役割} \quad = \quad \text{私}\,(x)\ \right] \end{array}\right]$$

また、「本」については、「本」は情報を‘ 含み ’、目的として人が‘ 読む ’ことと、‘ 書く ’ことよって生じることから、形式役割に include(y:concrete,x:information)、目的役割

に read(e1:process,v:human,x)、主体役割に write(e2:transition,w:human,x) を持つ。よって、「本」の語彙項目は以下のようになる。

$$
\left[\begin{array}{ll}
\text{本} & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : information \\ \text{項}2 = y : concrete \\ \text{デフォルト項}1 = v : human \\ \text{デフォルト項}2 = w : human \end{array}\right] \\
\text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \\ \text{事象}2 = e2 : transtion \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & hon(x \cdot y), include(y, x) \\ \text{目的役割} & = & read(e1, v, x) \\ \text{主体役割} & = & write(e2, w, x) \end{array}\right]
\end{array}\right]
$$

次にこの二つの項目を合成して「私の本」の語彙項目を導く。その方法を示すと、まず、「私」の意味素性は形式役割で示されているように x から human であることがわかる。次に「本」の目的役割と主体役割にある述語の引数から「本」が human を要求していることがわかる。このとき「私」の意味素性 human と「本」が要求している human とがタイプマッチする。タイプマッチが成功したことにより「私」の語彙項目を、「本」にある human を要求した引数についての情報が載っている項構造の human のところに代入する。この結果を「私の本」の語彙項目とする。以下に結果となる「私の本」の語彙項目を示す。

$$
\left[\begin{array}{ll}
\text{私の本} & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : information \\ \text{項}2 = y : concrete \\ \text{デフォルト項}1 = v : \text{私} \\ \text{デフォルト項}2 = w : human \end{array}\right] \\
\text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \\ \text{事象}2 = e2 : transition \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & hon(x \cdot y), include(y, x) \\ \text{目的役割} & = & read(e1, v, x) \\ \text{主体役割} & = & write(e2, w, x) \end{array}\right]
\end{array}\right]
$$

しかし、ここで問題が出る。「本」には要求している human が二つある。上の結果では、「私が読んだ本」という意味的関係は導けるが「私が書いた本」という意味的関係は導けていない。この結果ではまだ足りないことになる。次に要求している素性が二つ以上あるときの合成について述べる。

要求している意味素性が二つ以上ある場合は単純に順に一つずつ合成し、その結果を出力する。つまり、「本」の引数 v に関して「私」を合成し、また新しく「本」の引数 w と合成、そしてその結果をそれぞれ出力する。その結果、上述の結果に加え以下の様な「私の本」の語彙項目ができる。

$$
\left[\begin{array}{ll}
\text{私の本} & \\
\text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : information \\ \text{項}2 = y : concrete \\ \text{デフォルト項}1 = v : human \\ \text{デフォルト項}2 = w : \text{私} \end{array}\right] \\
\text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \\ \text{事象}2 = e2 : transition \end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll} \text{形式役割} & = & hon(x \cdot y), include(y, x) \\ \text{目的役割} & = & read(e1, v, x) \\ \text{主体役割} & = & write(e2, w, x) \end{array}\right]
\end{array}\right]
$$

この語彙項目は「私が書いた本」という意味的関係を導く。

次に「私の絨毯」の語彙項目を導く。「絨毯」の語彙項目は、絨毯の素性が concrete と product の二つである。この二つの素性を合わせて concrete・product と表す。この concrete・product は concrete と product 二つの素性を持つことを示す。この素性を使うと「絨毯」の語彙項目は以下のようになる。

$$\left[\begin{array}{ll} \text{絨毯} & \\ \text{項構造} & = \left[\ \text{項}1 = x : concrete \cdot product\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ \text{絨毯}\,(x)\ \right] \end{array}\right]$$

この項目を使い「私」との合成を行うと、「絨毯」が要求している素性がないために合成は失敗する。しかし、「私の絨毯」の意味解釈として「私が作った絨毯」、「私が所有している絨毯」などが考えられる。ここで、これらの解釈を導くために語彙の継承を行って意味解釈を導きたい。まず、「絨毯」の素性は concrete と product で、この二つは「絨毯」の上位概念である。これら上位概念の語彙項目を利用する。concrete と product の語彙項目は以下のようになる。

$$\left[\begin{array}{ll} concrete & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\ \text{事象}1 = e1 : state\ \right] \\ \text{特質構造} & = \left[\begin{array}{l} \text{形式役割} \ = \ concrete(x) \\ \text{目的役割} \ = \ possess(e1, y, x) \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} product & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\ \text{事象}1 = e1 : process\ \right] \\ \text{特質構造} & = \left[\begin{array}{l} \text{形式役割} \ = \ product(x) \\ \text{主体役割} \ = \ make(e1, y, x) \end{array}\right] \end{array}\right]$$

次に「私」と‘ concrete ’、「私」と‘ product ’の合成を行う。
concrete、product 共に素性 human を要求しているので、「私」との合成結果はそれぞれ

以下のようになる。
「私」と‘ concrete ’の合成結果

$$\left[\begin{array}{ll} \text{私の絨毯} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : top \\ \text{デフォルト項}1 = y : \text{私} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = concrete(x) \\ \text{目的役割} & = possess(e1, y, x) \end{array}\right] \end{array}\right]$$

「私」と‘ product ’の合成結果

$$\left[\begin{array}{ll} \text{私の絨毯} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = product(x) \\ \text{主体役割} & = make(e1, y, x) \end{array}\right] \end{array}\right]$$

また、‘ concrete ’と‘ product ’の継承を行うと、上位概念はともに top であるのでこれ以上継承は行わない。よって、「私の絨毯」の語彙項目は以上の二つとなり、これれらより「私の絨毯」の意味解釈「私が所有している絨毯」、「私が作った絨毯」が導ける。

4.4 解析例：名詞句「本の楽しさ」

「本の楽しさ」についての解析方法について述べる。まず、「楽しさ」の語彙項目は以下のようになる。

$$\left[\begin{array}{ll} \text{楽しさ} & \\ \text{項構造} & = \left[\text{デフォルト項}1 = e2\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : process \cdot state \cdot transition \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \quad = \quad enjoy\_act(e1, e2)\right] \end{array}\right]$$

「本」と「楽しさ」との語彙項目を合成し、「本の楽しさ」の語彙項目を導く。合成処理を行うと、まず「楽しさ」の形式役割にある述語 enjoy_act がイベントタイプを要求する。イベントタイプが要求されたときは「A の B」の A にあたる「本」の特質構造よりその要求されたイベントタイプを持つ述語を取り出し、「楽しさ」の語彙項目に追加する。この場合は「本」の語彙項目から read、write を取り出して、「楽しさ」の語彙項目にそれぞれの述語と述語引数のタイプ情報等を追加した二つの語彙項目が合成結果となる。合成結果は以下に示す。

- 「本」と「楽しさ」の合成結果 1

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = v : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = read(e2, v, x \cdot y) \end{array}\right] \end{array}\right]$$

- 「本」と「楽しさ」の合成結果 2

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = v : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : transition \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = write(e2, v, x \cdot y) \end{array}\right] \end{array}\right]$$

これらの合成結果より、「読むことを楽しむ」、「書くことを楽しむ」という意味解釈を導ける。しかし、これらの語彙項目からは何を読んだり、書いたりすることを楽しむのかが解らないので「本」の語彙項目にある hon(x・y) という関数を取り出し、read、write の隣にそれぞれ追加する。hon(x・y) を追加した合成結果を以下に示す。

- 「本」と「楽しさ」の合成結果 1

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = v : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = read(e2, v, x \cdot y), hon(x \cdot y) \end{array}\right] \end{array}\right]$$

- 「本」と「楽しさ」の合成結果 2

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = v : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : transition \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = write(e2, v, x \cdot y), hon(x \cdot y) \end{array}\right] \end{array}\right]$$

上二つの語彙項目より、改めて「本を読むことを楽しむ」、「本を書くことを楽しむ」という意味解釈が導ける。さらに、「本」の上位概念 concrete より語彙継承を行い、継承した語彙項目と「楽しさ」を合成することで、「本を所有することを楽しむ」という意味解釈が新たに導かれ、「本の楽しさ」の意味解釈は合計三つの意味解釈が導かれる。「本を所有することを楽しむ」に関する語彙項目を以下に示す。

- 「concrete」と「楽しさ」の合成結果

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : state \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = possess(e2, y, x), concrete(x) \end{array}\right] \end{array}\right]$$

第5章　実装・考察

5.1　プログラム

プログラムは Allegro Common Lisp を用いて関数 約 800 行、語彙項目約 170 行を作成した。また、語彙項目は IPAL にある意味素性 68 項目のうち、53 項目を記述、23 項目を実装した。

解釈可能な範囲としては、島津 [6] の分類に従って見ると、A または B が述語である場合、B が場所等である場合が可能。B が「高さ」等の属性については可能だが未記述である。その他の一般的な場合、「私の絨毯」、「太郎の制服」、「公園の銅像」など 10 例について動作を確認した。以下に主な関数を挙げる。

- 入力と出力を行う関数 AnoB
- B を関数として A の語彙項目と B の語彙項目とで、「A の B」の意味記述を合成し、「A の B」の語彙項目を導く関数 AnoB1
- A を関数として A の語彙項目と B 野語彙項目とで、「A の B」の意味記述を合成し、「A の B」の語彙項目を導く関数 BnoA
- A の意味素性と B の意味素性とが同じときに A の形式役割にある関数を B の項目内に追加する関数 AnoB-for-same-type
- 項目より述語を取り出したときに、変数の統合をはかる関数 combine_variable
- 語彙項目内の変数は x、y など一定の変数を使っている。語彙項目間での変数の違いをだすために、変数を個別のものに変更する関数 change
- 語彙の継承を行う関数 inherit_upper_noun

主な処理について説明する。

- 「A の B」で B を関数として用いる場合
  「A の B」において B の特質構造にある述語の引数が要求する意味タイプと A の意味タイプがマッチするとき、A の項目を B の項構造での要求した引数のタイプ情報があるところに代入する。

- 「A の B」で A を関数として用いる場合
  「A の B」において A の特質構造にある述語の引数が要求する意味タイプと B の意味タイプがマッチするとき、その述語と述語内の引数に関する情報を A の項目より抜き出し、B の項目に追加。

5.2 実験例

「A の B」を解析する方法を実装し、「A の B」の意味解釈を導いた結果を示す。

1. 「私の絨毯」

$$\left[\begin{array}{ll} \text{私} & \\ \text{項構造} & = \left[\ \text{項}\,1 = x : human\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ \text{私}\,(x)\ \right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{絨毯} & \\ \text{項構造} & = \left[\ \text{項}\,1 = x : concrete \cdot product\ \right] \\ \text{特質構造} & = \left[\ \text{形式役割} \ = \ \text{絨毯}\,(x)\ \right] \end{array}\right]$$

「私」と「絨毯」との語彙項目を合成することにより以下のような語彙項目が「私の絨毯」として導かれる。

$$\left[\begin{array}{ll} \text{私の絨毯} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}\,1 = x : \top \\ \text{項}\,2 = y : \text{私} \end{array}\right] \\ \text{事象構造} & = \left[\ \text{事象}\,1 = e1 : state\ \right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ product(x) \\ \text{主体役割} & = \ make(e1, y, x) \end{array}\right] \end{array}\right]$$

上記の項目より、「私の絨毯」は「私がつくる絨毯」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{私の絨毯} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{項}2 = y : \text{私} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = concrete(x) \\ \text{主体役割} & = possess(e1, y, x) \end{array}\right] \end{array}\right]$$

上記の項目より、「私の絨毯」は「私が所有する絨毯」という意味解釈が導ける。

2. 「太郎の制服」

$$\left[\begin{array}{ll} \text{太郎} & \\ \text{項構造} & = \left[\text{項}1 = x : human\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = tarou(x) \\ \text{構成役割} & = name(x, \text{"太郎"}) \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{制服} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : concrete \cdot product \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process \cdot state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = seifuku(x) \\ \text{目的役割} & = wear(e1, y, x) \end{array}\right] \end{array}\right]$$

「太郎」と「制服」との語彙項目を合成することにより以下のような語彙項目が「太郎の制服」として導かれる。

$$\left[\begin{array}{ll} \text{太郎の制服} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : concrete \cdot product \\ \text{デフォルト項}1 = y : \text{太郎} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process \cdot state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = seifuku(x) \\ \text{目的役割} & = wear(e1, y, x) \end{array}\right] \end{array}\right]$$

上記の項目より、「太郎の制服」は「太郎が着る制服」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{太郎の制服} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : \text{太郎} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ concrete(x) \\ \text{目的役割} & = \ possess(e1, y, x) \end{array}\right] \end{array}\right]$$

上記の項目より、「太郎の制服」は「太郎が所有する制服」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{太郎の制服} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : \top \\ \text{デフォルト項}1 = y : \text{太郎} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ product(x) \\ \text{目的役割} & = \ make(e1, y, x) \end{array}\right] \end{array}\right]$$

上記の項目より、「太郎の制服」は「太郎がつくる制服」という意味解釈が導ける。

3. 「制服の太郎」

「制服」と「太郎」との語彙項目を合成することにより、以下のような「制服の太郎」の語彙項目が導かれる。

$$\left[\begin{array}{ll} \text{制服の太郎} & \\ \text{項構造} & = \left[\text{項}1 = x : concrete\right] \\ \text{特質構造} & = \left[\text{形式役割} \ = \ animate(x), seifuku(x)\right] \end{array}\right]$$

上記の項目より、「制服の太郎」は「制服である太郎」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{制服の太郎} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : human \\ \text{デフォルト項}1 = y : \text{太郎} \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process \cdot state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \ tarou(x) \\ \text{構成役割} & = \ name(x, \text{"太郎"}) \\ \text{目的役割} & = \ wear(e1, x, y), seifuku(y) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の太郎」は「制服を着ている太郎」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{制服の太郎} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : human \\ \text{デフォルト項}1 = y : top \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : process\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = tarou(x) \\ \text{構成役割} & = name(x, \text{"太郎"}) \\ \text{目的役割} & = make(e1, x, y), product(y) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の太郎」は「制服を作る太郎」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{制服の太郎} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : human \\ \text{デフォルト項}1 = y : \top \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : state\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = tarou(x) \\ \text{構成役割} & = name(x, \text{"太郎"}) \\ \text{目的役割} & = possess(e1, x, y), concrete(y) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の太郎」は「制服を所有している太郎」という意味解釈が導ける。

4. 「公園の銅像」

$$\left[\begin{array}{ll} \text{公園} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : locus \\ \text{デフォルト項}1 = y : concrete \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; kouen(x), at(y, x)\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{銅像} & \\ \text{項構造} & = \left[\text{項}1 = x : concrete \cdot product\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; douzou(x)\right] \end{array}\right]$$

「公園」と「銅像」との語彙項目をごうせいすることにより、以下のような「公園の銅像」の語彙項目が導かれる。

$$\left[\begin{array}{ll} \text{公園の銅像} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : concrete \cdot product \\ \text{項}2 = y : locus \end{array}\right] \\ \text{事象構造} & = \left[\text{事象}1 = e1 : state\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; douzou(x), at(x,y), kouen(y)\right] \end{array}\right]$$

上記の項目より、「公園の銅像」は「公園にある銅像」という意味解釈が導ける。

5. 「本の楽しさ」

$$\left[\begin{array}{ll} \text{本} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = x : information \\ \text{項}2 = y : concrete \\ \text{デフォルト項}1 = v : human \\ \text{デフォルト項}2 = w : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : process \\ \text{事象}2 = e2 : transition \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \; hon(x \cdot y), hold(y,x) \\ \text{目的役割} & = \; read(e1, v, x \cdot y) \\ \text{主体役割} & = \; write(e2, w, x \cdot y) \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{楽しさ} & \\ \text{項構造} & = \left[\text{項}1 = e2\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : process \cdot state \cdot transition \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; enjoy\_act(e1, e2)\right] \end{array}\right]$$

「本」と「楽しさ」との語彙項目を合成することにより、以下のような「本の楽しさ」の語彙項目が導かれる。

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = v : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : process \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = read(e2, v, x \cdot y), hon(x \cdot y) \end{array}\right] \end{array}\right]$$

上記の項目より、「本の楽しさ」は「本を読む楽しさ」という意味解釈が導ける。

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : information \\ \text{項}3 = y : concrete \\ \text{デフォルト項}1 = w : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : transition \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = write(e2, w, x \cdot y), hon(x \cdot y) \end{array}\right] \end{array}\right]$$

上記の項目より、「本の楽しさ」は「本を書く楽しさ」という意味解釈が導ける。また、「本 (concrete の語彙項目を継承)」と「楽しさ」との語彙項目を合成することにより、以下のような「本の楽しさ」の語彙項目が導かれる。

$$\left[\begin{array}{ll} \text{本の楽しさ} & \\ \text{項構造} & = \left[\begin{array}{l} \text{項}1 = e2 \\ \text{項}2 = x : top \\ \text{デフォルト項}1 = y : human \end{array}\right] \\ \text{事象構造} & = \left[\begin{array}{l} \text{事象}1 = e1 : state \\ \text{事象}2 = e2 : state \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = enjoy\_act(e1, e2) \\ \text{目的役割} & = possess(e2, y, x), hon(x) \end{array}\right] \end{array}\right]$$

上記の項目より、「本の楽しさ」は「本を所有する楽しさ」という意味解釈が導ける。

6.「制服の前」

$$\left[\begin{array}{ll} \text{前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}2 = y : pro \cdot sta \cdot tra \cdot concrete \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; \text{前}\,(x, y)\right] \end{array}\right]$$

「制服」と「前」との語彙項目を合成することにより、以下のような「制服」の語彙項目が導かれる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : \text{制服} \\ \text{デフォルト項}2 = y : pro \cdot sta \cdot tra \cdot concrete \end{array}\right] \\ \text{特質構造} & = \left[\text{形式役割} \;=\; \text{前}\,(x, y)\right] \end{array}\right]$$

上記の項目より、「制服の前」は「制服がある前」という意味解釈になる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : sta \\ \text{デフォルト項}2 = y : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}3 = z : human \\ \text{項}1 = v : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{l} \text{形式役割} \;=\; \text{前}\,(x, y) \\ \text{目的役割} \;=\; possess(x, z, v), concrete(v) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の前」は「制服を所有する前」という意味解釈になる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \\ \text{デフォルト項}2 = y : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}3 = z : human \\ \text{項}1 = v : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{l} \text{形式役割} \;=\; \text{前}\,(x, y) \\ \text{目的役割} \;=\; possess(x, z, v), product(v) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の前」は「制服をつくる前」という意味解釈になる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \cdot sta \\ \text{デフォルト項}2 = y : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}3 = z : human \\ \text{項}1 = v : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \text{前}\,(x, y) \\ \text{目的役割} & = wear(x, z, v), seifuku(v) \end{array}\right] \end{array}\right]$$

上記の項目より、「制服の前」は「制服を着る前」という意味解釈になる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}2 = y : \text{制服} \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \text{前}\,(x, y) \end{array}\right] \end{array}\right]$$

上記の項目は「制服の前」が「何かの前にある制服」という解釈になる。しかし、この解釈は「'何か' の前の制服」という表現になされると解釈である考えから不適切な解釈となる。以下三つの解釈も同様に不適切となる。

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}2 = y : sta \\ \text{デフォルト項}3 = z : human \\ \text{項}1 = v : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \text{前}\,(x, y) \\ \text{目的役割} & = possess(y, z, v), concrete(v) \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{ll} \text{制服の前} & \\ \text{項構造} & = \left[\begin{array}{l} \text{デフォルト項}1 = x : pro \cdot sta \cdot tra \cdot concrete \\ \text{デフォルト項}2 = y : pro \\ \text{デフォルト項}3 = z : human \\ \text{項}1 = v : \top \end{array}\right] \\ \text{特質構造} & = \left[\begin{array}{ll} \text{形式役割} & = \text{前}\,(x, y) \\ \text{目的役割} & = make(y, z, v), product(v) \end{array}\right] \end{array}\right]$$

$$
\left[\begin{array}{ll}
\text{制服の前} & \\
\text{項構造} & = \left[\begin{array}{l}
\text{デフォルト項}\,1 = x : pro \cdot sta \cdot tra \cdot concrete \\
\text{デフォルト項}\,2 = y : pro \cdot sta \\
\text{デフォルト項}\,3 = z : human \\
\text{項}\,1 = v : \top
\end{array}\right] \\
\text{特質構造} & = \left[\begin{array}{lll}
\text{形式役割} & = & \text{前}\,(x, y) \\
\text{目的役割} & = & wear(y, z, v), seifuku(v)
\end{array}\right]
\end{array}\right]
$$

5.3 考察

A またはB が述語や属性の基本的な場合は扱いやすいが、一般的な場合の特質構造の記述は難しい。また、一般に文脈を考慮する必要がある。例えば、本研究での解析では「人の前」という表現の解釈は「人」がいる場所の「前」という意味解釈が導かれるが、「前の人」という表現は文脈等から基準となる点に関する情報を得ることで成り立つ表現であるから、「前」にいる「人」という解釈になる。その基点となる部分は「彼の前の人」ような表現や文脈などから得ると解釈する。

さらに、「学校の制服」などの表現を「学校で使われる制服」と解釈するなら、場所格を付加する処理が必要となるが、今回は実装していない。

「鉄の男」、「赤の他人」などの慣用的な表現や擬人的な表現の意味解釈は導くことができなかった。これらの名詞句をヒューリスティックや推論等を用いる必要がある。

また、特質構造内にある意味情報に関して、概念ごとにどれくらいの述語等を与えれば良いのか、どういった述語を与えれば良いのかなどをさらに考察する必要がある。

謝辞

本研究を進めるにあたり、多大なご支援、ご指導を頂いた島津明教授に深くお礼申し上げます。さらに、貴重なご助言を頂いた白井清昭助教授並びに山田寛康助手に厚く感謝致します。

参考文献

[1] Kentaro Torisawa, A Nearly Unsupervised Learning Method for Automatic Paraphrasing of Japanese Noun Phrases, Proceedings of the Workshop on Automatic Paraphrasing,pp. 63-72, Dec, 2001

[2] 菊池, 語彙意味情報に基づく日本語名詞句の意味解析-「A の B」を例に, 中京大 修士論文 1999.

[3] S.Kurohashi, Y.Sakai, Semantic Analysis of Noun Phrases : A New Approach to Dictionary-Based Understanding, ACL 99, pp.481-488.

[4] 情報処理振興事業協会技術センター [編], 計算機用日本語基本名詞辞書 IPAL(Basic Nouns), 情報処理振興事業協会, 1997.

[5] J.Pustejovsky, The Generative Lexicon, The MIT Press 1995.

[6] 島津, 内藤, 野村, 助詞「の」が結ぶ名詞の意味関係の解析, 計量国語学, 第 15 巻 7 号, 1986.

付録

以下に作成したプログラムを付録としてのせる。

```
\small
(defmacro arg-st-arg (x)
  `(first , x))
(defmacro arg-var (x)
  `(second , x))
(defmacro arg-type (x)
  `(third , x))
(defconstant top nil)

(defstruct lexical-entry phone arg-st event-st qualia-st)
(defstruct semantic-structure phone arg-st event-st qualia-st)

;; 合成結果を出力
(defun AnoB (A_noun B_noun)
  (let ((baseB (AnoB1 A_noun B_noun))
        (baseA (BnoA A_noun B_noun)))
    (format t "Base: ~a~%~a no ~a~%" B_noun A_noun B_noun)
    (dolist (B baseB)
      (cond ((first B)
             (format t "~a~%~%" (change-of-phone A_noun B_noun (first B))))
            ))
    (format t "~%Base: ~a~%~a no ~a~%" A_noun A_noun B_noun)
    (dolist (A baseA)
      (cond (A
             (format t "~a~%~%" (change-of-phone A_noun B_noun A))
             ))
      )
    ))
```

```
(defun change-of-phone (A B B-with-A)
  (let ((phone (semantic-structure-phone B-with-A)))
    (setf (semantic-structure-phone B-with-A)
  (concatenate 'string
       (string A) "_no_" (string B) "(" (string phone)")")))
  B-with-A)

;; 合成結果をリストに保持
(defun AnoB1 (A_noun B_noun)
  (let ((A1 (get A_noun 'lexical-entry))
        (B1 (list (get B_noun 'lexical-entry)))
        (lexicon_list nil))
    (loop
     (cond ((or (top? B1) (null B1))
    (return lexicon_list))
   (t
    (setf lexicon_list (append lexicon_list (AnoB2 A1 B1)))))
     (setf B1 (inherit_upper-noun B1))
     )
    ))

(defun AnoB2 (A-noun B-noun-list)
  (let ((A-noun-list (expansion-dotted-type (list A-noun)))
        (new-B-noun-list nil))
    (dolist (A A-noun-list)
      (let* ((new-A-noun (change (cp-lexical-entry A)))
             (type-lists (arg-st-types new-A-noun)))
        (setf new-B-noun-list
      (append (AnoB-for-same-type
       (noun-type new-A-noun)
       B-noun-list
       new-A-noun)
```

```
      new-B-noun-list))

        (dolist (A-type type-lists)
          (let ((new-B-noun (AnoB3 A-type B-noun-list new-A-noun)))
            (if new-B-noun
                (setf new-B-noun-list (append new-B-noun new-B-noun-list))
              ))
          )
        ))
    new-B-noun-list
    ))

;; 最初の合成で、A のタイプと B のタイプが同じときの合成
(defun AnoB-for-same-type (A-arg-type_list B-nouns A-noun)
  (let ((B-with-A-list nil))
    (dolist (B-noun B-nouns)
      (let* ((newB-noun (change (cp-lexical-entry B-noun)))
             (B (AnoB-for-same-type1 A-arg-type_list newB-noun A-noun)))
        (if B
            (push B B-with-A-list)
          ))
      )
    B-with-A-list
    ))

;; 項構造より (arg x human) の形を取り出す (デフォルト項などは取り出さない)
(defun noun-type (lexicon_lists)
  (let ((type-list nil))
    (if (listp lexicon_lists)
        (dolist (lexicon lexicon_lists)
          (let ((argments (semantic-structure-arg-st lexicon)))
            (dolist (arg argments)
```

```
              (if (and (< 2 (length arg))  (eq 'arg (arg-st-arg arg)))
                  (push arg type-list)
))))
      (let ((argments (semantic-structure-arg-st lexicon_lists)))
        (dolist (arg argments)
          (if (and (< 2 (length arg))  (eq 'arg (arg-st-arg arg)))
              (push arg type-list)
            ))))
    type-list
    ))

(defun AnoB-for-same-type1 (arg-type_list B-noun A-noun)
  (let ((B-arg-st (semantic-structure-arg-st B-noun))
        (B_with_A_list nil))
    (dolist (A-arg arg-type_list)
      (dolist (B-arg B-arg-st)
        (cond ((and (member 'arg B-arg)
                    (eq (arg-type A-arg) (arg-type B-arg)))
               (let ((cp-B-noun (cp-semantic-structure B-noun)))
                 (push (add-to-Blexicon
(arg-var A-arg)
(arg-var B-arg)
A-noun cp-B-noun)
                       B_with_A_list))))
        ))
    B_with_A_list
    ))

(defun add-to-Blexicon (select-variable base-B-variable A-noun B-noun)
  (let* ((A-formal (qualia-search 'formal (semantic-structure-qualia-st
   A-noun)))
         (A-kansuu (qualia-search select-variable (rest A-formal)))
```

```
          (B-qualia-st (semantic-structure-qualia-st B-noun))
          (B-formal (qualia-search 'formal B-qualia-st)))
    (cond (A-kansuu
           (setf (semantic-structure-qualia-st B-noun)
             (delete-insert 'formal
    (append B-formal (list A-kansuu))
    B-qualia-st))
           (combine-variable base-B-variable select-variable B-noun)))
    ))

(defun arg-st-types (noun_lists)
  (let ((type-list nil))
    (if (listp noun_lists)
        (dolist (noun noun_lists)
          (let ((argments (semantic-structure-arg-st noun)))
            (dolist (arg argments)
              (if (and (< 2 (length arg))  (eq 'arg (arg-st-arg arg)))
                  (push (arg-type arg) type-list)
))))
      (let ((argments (semantic-structure-arg-st noun_lists)))
        (dolist (arg argments)
          (if (and (< 2 (length arg))  (eq 'arg (arg-st-arg arg)))
              (push (arg-type arg) type-list)
            ))))
    type-list
    ))

;;A と B を合成
(defun AnoB3 (A-type B-nouns A-noun)
  (let ((B-with-A-list nil))
    (dolist (B-noun B-nouns)
      (let* ((newB-noun (change (cp-lexical-entry B-noun)))
```

```
            (B (match_A-formal-arg-type_B-arg-st A-type newB-noun A-noun)))
        (if B
            (setf B-with-A-list (append B B-with-A-list))
          ))
      )
    B-with-A-list
    ))

;;; 形式役割の引数を調べないで、arg に注目しタイプを取り出し合成
(defun match_A-formal-arg-type_B-arg-st (A-formal-arg-type B-noun A-noun)
  (let ((B-arg-st (semantic-structure-arg-st B-noun))
        (B_with_A_list nil))
    (dolist (B-arg B-arg-st)
      (cond ((and (member 'defarg B-arg)
                  (type-match A-formal-arg-type (arg-type B-arg)))
             (let ((cp-B-noun (cp-lexical-entry B-noun)))
               (push (list (composition (arg-var B-arg) A-noun cp-B-noun))
     B_with_A_list)))
            ((and (member 'arg B-arg) (member 'information B-arg))
             (let ((cp-B-noun (cp-lexical-entry B-noun)))
               (push (composition (arg-var B-arg) A-noun cp-B-noun)
     B_with_A_list))))
      )
    B_with_A_list
    ))

(defun composition (key A-noun B-noun)
  (let* ((arg-st-list (semantic-structure-arg-st B-noun))
         (target-list (qualia-search key arg-st-list))
         (new-lists nil))
    (dolist (list-x arg-st-list)
      (cond ((eq-list target-list list-x)
```

```
             (setf (arg-type target-list) A-noun)
             (setf (arg-st-arg target-list) 'arg)
             (push target-list new-lists))
            (t (push list-x new-lists)))
      )
    (setf (semantic-structure-arg-st B-noun) (reverse new-lists))
    B-noun
    ))

(defun BnoA (A_noun B_noun)
  (let ((B1 (get B_noun 'lexical-entry))
        (A1 (list (get A_noun 'lexical-entry)))
        (lexicon_list nil))
    (loop
      (cond ((or (top? A1) (null A1))
             (return lexicon_list))
            (t
             (setf lexicon_list (append lexicon_list (BnoA1 A1 B1)))
             (if (listp A1)
                 (dolist (one-of-A1 A1)
                   (setf lexicon_list
 (append lexicon_list
 (composition-for-event-variable1
  one-of-A1
  (list B1))))
                   )
               (setf lexicon_list
     (append lexicon_list
     (composition-for-event-variable1
      A1
      (list B1))))))
            )
```

```
      (setf A1 (inherit_upper-noun A1))
      )
    ))

(defun BnoA1 (A-noun-list B-noun)
  (let ((B-with-A nil))
    (dolist (A-noun A-noun-list)
      (let* ((newA-noun (change (cp-lexical-entry A-noun)))
             (newB-noun (change (cp-lexical-entry B-noun)))
             (A-arg-st (semantic-structure-arg-st newA-noun))
             (B-arg-st (semantic-structure-arg-st newB-noun)))
        (dolist (A-arg A-arg-st)
          (dolist (B-arg B-arg-st)
            (cond ((eq (arg-st-arg A-arg) 'defarg)
                   (cond ((and (eq (arg-st-arg B-arg) 'arg)
                               (type-match (arg-type B-arg)
   (arg-type A-arg)))
  (push (insert-predicate A-arg
  newA-noun
  B-arg
  newB-noun)
B-with-A)))
                  (if (dot-type-request-event-type? (arg-type B-arg))
                      (let ((cp-B-noun (cp-semantic-structure newB-noun)))
                        (push (event-type-for-dot-type B-arg
newA-noun
cp-B-noun)
                              B-with-A)))
                  )
                 )
            ))
        ))
```

```
    B-with-A
    ))

;; 項構造内でのイベントタイプはpro、sta、traと表記
(defun dot-type-request-event-type? (dot-type)
  (let ((decomposed-type-list (take-dotted-type-apart dot-type)))
    (dolist (one-type decomposed-type-list)
      (if (member one-type '(pro sta tra))
          (return t)))
    ))

(defun event-type-for-dot-type (dot-type-in-B-lexicon A-lexicon B-lexicon)
  (let ((decomposed-type-list
 (take-dotted-type-apart (arg-type dot-type-in-B-lexicon)))
        (A-event-st (semantic-structure-event-st A-lexicon))
        (B_with_A nil))
    (dolist (one-type decomposed-type-list)
      (cond ((eq one-type 'pro)
             (dolist (one-event A-event-st)
               (if (type-match 'process (arg-type one-event))
                   (return (setf B_with_A
                                 (insert-predicate one-event
                                                   A-lexicon
                                                   dot-type-in-B-lexicon
                                                   (cp-semantic-structure
B-lexicon))))
                 )))
            ((eq one-type 'sta)
             (dolist (one-event A-event-st)
               (if (type-match 'state (arg-type one-event))
                   (return
    (setf B_with_A
```

```
  (insert-predicate one-event
    A-lexicon
    dot-type-in-B-lexicon
    (cp-semantic-structure B-lexicon))))
                )))
           ((eq one-type 'tra)
            (dolist (one-event A-event-st)
              (if (type-match 'transition (arg-type one-event))
                  (return
    (setf B_with_A
  (insert-predicate one-event
    A-lexicon
    dot-type-in-B-lexicon
    (cp-semantic-structure B-lexicon))))
                ))
            )))
    B_with_A
    ))

(defun insert-predicate (A-arg A-noun B-arg-var B-noun)
  (let* ((A-var (arg-var A-arg))
         (B-event-st (semantic-structure-event-st B-noun))
         (A-qualia-st (semantic-structure-qualia-st A-noun))
         (B-qualia-st (semantic-structure-qualia-st B-noun))
         (A-predicate-role-list (qualia-search A-var A-qualia-st))
         (A-predicate-role (first A-predicate-role-list))
         (A-predicate (qualia-search A-var A-predicate-role-list))
         (A-role-in-B-qualia (qualia-search A-predicate-role B-qualia-st))
         (A-formal-role (qualia-search 'formal A-qualia-st))
         (formal-role-in-B-qualia (qualia-search 'formal B-qualia-st))
         (B-with-A nil))
```

```
   (dolist (A-predicate-var A-predicate)
     (let ((A-argment (qualia-search A-predicate-var
     (semantic-structure-arg-st A-noun)))
           (A-event (qualia-search A-predicate-var
   (semantic-structure-event-st A-noun)))
           (B-arg-st (semantic-structure-arg-st B-noun)))
       (cond ((search "*" (string A-predicate-var))
              (let ((var-list (take-dotted-type-apart A-predicate-var)))
                (dolist (var (reverse var-list))
                  (unless (list-search var (semantic-structure-arg-st
    B-noun))
                    (setf (semantic-structure-arg-st B-noun)
                      (append (semantic-structure-arg-st B-noun)
                              (list (qualia-search
                                     var
                                     (semantic-structure-arg-st A-noun))))))
                  )
                ))
             ((null (eq A-predicate-var (arg-var A-arg)))
              (if A-argment
                  (setf (semantic-structure-arg-st B-noun)
(append B-arg-st (list A-argment))))
              (if A-event
                  (setf (semantic-structure-event-st B-noun)
(append B-event-st (list A-event)))))
     )
       ))

   (let ((B-arg-st (semantic-structure-arg-st B-noun)))
     (setf (semantic-structure-arg-st B-noun)
           (delete-insert (arg-var B-arg-var)
                          (list (arg-st-arg B-arg-var)
```

```
                                (arg-var B-arg-var)
                                (shift-type-for-event-type
  (arg-type A-arg)))
                           B-arg-st))
      )

    (if A-role-in-B-qualia
        (if (list-search (first (second A-formal-role))
                         (semantic-structure-qualia-st B-noun))
            (setf (semantic-structure-qualia-st B-noun)
  (delete-insert A-predicate-role
 (append A-role-in-B-qualia
 (list A-predicate))
 B-qualia-st))
          (setf (semantic-structure-qualia-st B-noun)
(delete-insert A-predicate-role
       (append A-role-in-B-qualia
       (list A-predicate)
       (list (second A-formal-role)))
       B-qualia-st)))
      (if (list-search (first (second A-formal-role))
                       (semantic-structure-qualia-st B-noun))
          (setf (semantic-structure-qualia-st B-noun)
(append B-qualia-st
(list (list A-predicate-role A-predicate))))
        (setf (semantic-structure-qualia-st B-noun)
      (append B-qualia-st (list (list A-predicate-role A-predicate
      (second A-formal-role)))))))

    (combine-variable (arg-var B-arg-var) A-var B-noun)
    ))
```

```
;; イベントタイプprocess、state、transitionをpro、sta、traに置き換え
(defun shift-type-for-event-type (dot-type)
  (let ((decomposed-type-list (take-dotted-type-apart dot-type))
        (new-type nil))
    (dolist (one-type decomposed-type-list)
      (cond ((eq one-type 'process)
             (if (null new-type)
                 (setf new-type 'pro)
               (setf new-type
     (intern (concatenate 'string "PRO" "*"
  (string new-type))))))
            ((eq one-type 'state)
             (if (null new-type)
                 (setf new-type 'sta)
               (setf new-type
                 (intern (concatenate 'string "STA" "*"
      (string new-type))))))
    ((eq one-type 'transition)
             (if (null new-type)
                 (setf new-type 'tra)
               (setf new-type
     (intern (concatenate 'string "TRA" "*"
  (string new-type))))))
           ))
    new-type
    ))

(defun extract-list (key qualia-st)
  (let ((extract-qualia-st nil))
    (dolist (role-list qualia-st)
      (setf extract-qualia-st
    (append extract-qualia-st (list (first role-list))))
```

```
      (dolist (role role-list)
        (cond ((listp role)
               (if (member key role)
                   (setf extract-qualia-st
 (append extract-qualia-st (list role)))))
              ))
      )
    extract-qualia-st
    ))

;; 上位タイプとのマッチ
(defun type-match (A-type B-type)
  (let ((A-type-list nil))
    (cond ((null A-type)
           nil)
          ((symbolp A-type)
           (cond ((eq A-type B-type)
                  t)
                 ((search "*" (string B-type))
                  (if (matching-types A-type
      (take-dotted-type-apart B-type))
                      t
                    (type-match (get A-type 'isa) B-type))
                  )
                 ((search "*" (string A-type))
                  (type-match  (take-dotted-type-apart A-type) B-type))
                 (t
                  (type-match (get A-type 'isa) B-type))
 ))
          ((listp A-type)
           (cond ((matching-types B-type A-type)
                  t)
```

```
                (t (dolist (A A-type)
                     (let ((A-isa (get A 'isa)))
                       (if (listp A-isa)
                           (dolist (upper-Atype A-isa)
                             (cond ((search "*" (string upper-Atype))
                                    (push
    (take-dotted-type-apart upper-Atype)
    A-type-list))
                                   (t
                                    (push upper-Atype A-type-list)))
                             )
                         (cond ((search "*" (string A-isa))
                                (push (take-dotted-type-apart A-isa)
     A-type-list))
                               (t
                                (push A-isa A-type-list))
                               )
                         )
                       )))
                (type-match A-type-list B-type))
           ))
    ))

;; ドットタイプを分解した後のタイプマッチ
(defun matching-types (symbol-or-listA listB)
  (let ((match-type nil))
  (if (symbolp symbol-or-listA)
      (dolist (x listB)
        (if (eq symbol-or-listA x)
            (setf match-type symbol-or-listA)))
    (dolist (x symbol-or-listA)
      (dolist (y listB)
```

```
        (if (eq x y)
            (setf match-type x)
          ))))
    match-type
    ))

;; ドットタイプのタイプマッチをするときに、ドットタイプを分解
(defun take-dotted-type-apart (dotted-type)
  (let ((type-list nil))
    (loop
     (let* ((string-dotted-type (string dotted-type))
    (position (search "*" string-dotted-type)))
       (cond (position
      (push (intern (subseq string-dotted-type 0 position))
    type-list)
      (setf dotted-type (subseq string-dotted-type
(incf position))))
     ((null position)
      (push (intern string-dotted-type) type-list)
      (return type-list)))
       ))
    ))

;; 名詞 A の上位概念を継承
(defun inherit_upper-noun (B-noun-list)
  (let* ((B-nouns (expansion-dotted-type B-noun-list))
         (types_of_B-noun (arg-st-types B-nouns))
         (upper_noun nil))
    (dolist (B_type types_of_B-noun)
      (let ((B_lexicon (get B_type 'lexical-entry)))
        (if (and B_lexicon (not (member-phone B_lexicon upper_noun)))
    (push B_lexicon upper_noun))
```

```
        ))
    upper_noun
    ))

;; ドットタイプを一つのタイプ毎に項目を展開
(defun expansion-dotted-type (lexicon-list)
  (let ((new-lexicon-list nil))
    (dolist (lexicon lexicon-list)
      (let ((arg-st-lists (semantic-structure-arg-st lexicon)))
        (dolist (argment arg-st-lists)
          (cond ((eq (arg-st-arg argment)'arg)
                 (setf type-list (append (take-dotted-type-apart
  (arg-type argment))))
                 (dolist (one-type type-list)
                   (let ((new-lexicon (cp-semantic-structure lexicon)))
                     (setf (semantic-structure-arg-st new-lexicon)
   (delete-insert (arg-var argment)
  (list (arg-st-arg argment)
(arg-var argment)
one-type)
  arg-st-lists))
                     (push new-lexicon new-lexicon-list)))))
         )))
    new-lexicon-list
    ))

(defun formal-is-relation? (A-noun)
  (let ((A-formal-role (first (semantic-structure-qualia-st A-noun))))
    (unless (= (length (second A-formal-role)) 2)
      t)
    ))
```

```
;; 項構造より指定された変数リスト内の変数の項リストを取り出して、
;; 新たに項構造のリストを構成
(defun extract-arg-st (predicate lexicon)
  (let ((variable-list (rest predicate))
        (arg-st (semantic-structure-arg-st lexicon))
        (new-arg-st nil))
    (dolist (var variable-list)
      (let ((one-argment (qualia-search var arg-st)))
        (if one-argment
            (push one-argment new-arg-st))
))
    (reverse new-arg-st)
    ))

;; 項構造内の変数がイベント変数であるときの操作
(defun composition-for-event-variable1 (A-lexicon B-lexicon-list)
  (let ((newA-lexicon (change (cp-lexical-entry A-lexicon)))
        (results nil))
    (dolist (B-lexicon B-lexicon-list)
      (let* ((newB-lexicon (change (cp-lexical-entry B-lexicon)))
             (B-with-A (composition-for-event-variable2 newA-lexicon
newB-lexicon)))
        (if B-with-A
            (setf results
              (append B-with-A results))))
      )
    results
    ))

;;B 内での操作、B の項構造が event タイプを要求しているときで、
;; そのタイプと同じタイプが B の事象構造内にあるとき、
;; → A から要求したイベントタイプを持つ引数と述語を取り出し
```

;;B の項目内にそれらを追加。

```
(defun composition-for-event-variable2 (A-lexicon B-lexicon)
  (let ((event-var-list (A-event-arglist A-lexicon B-lexicon))
        (A-qualia-st-list (semantic-structure-qualia-st A-lexicon))
        (newB-lexicon-list nil))
    (dolist (event-var (second event-var-list))
      (push
       (composition-for-event-variable3 event-var
A-lexicon
(first event-var-list)
(cp-semantic-structure
 B-lexicon))
       newB-lexicon-list)
      )
    newB-lexicon-list
    ))

(defun composition-for-event-variable3 (A-event A-noun request-B-event
B-noun)
  (let* ((B-event-st (semantic-structure-event-st B-noun))
         (A-qualia-st (semantic-structure-qualia-st A-noun))
         (B-qualia-st (semantic-structure-qualia-st B-noun))
         (A-predicate-role-list (qualia-search A-event A-qualia-st))
         (A-predicate-role (first A-predicate-role-list))
         (A-predicate (qualia-search A-event A-predicate-role-list))
         (A-role-in-B-qualia (qualia-search A-predicate-role B-qualia-st))
         (A-formal-role (qualia-search 'formal A-qualia-st))
         (formal-role-in-B-qualia (qualia-search 'formal B-qualia-st))
         (B-with-A nil))

    (dolist (A-predicate-var A-predicate)
      (let ((A-arg-var (qualia-search A-predicate-var
```

```
      (semantic-structure-arg-st A-noun)))
            (A-event-var (qualia-search A-predicate-var
(semantic-structure-event-st A-noun)))
            (B-arg-st (semantic-structure-arg-st B-noun)))
        (cond ((search "*" (string A-predicate-var))
               (let ((var-list (take-dotted-type-apart A-predicate-var)))
                 (dolist (var (reverse var-list))
                   (unless (list-search var (semantic-structure-arg-st
     B-noun))
                     (setf (semantic-structure-arg-st B-noun)
                       (append (semantic-structure-arg-st B-noun)
                               (list (qualia-search
                                      var
                                      (semantic-structure-arg-st
       A-noun))))))
                   )
                 ))
              ((eq A-predicate-var A-event)
               (setf (semantic-structure-event-st B-noun)
                 (delete-insert request-B-event
                     (list A-event-var) B-event-st)))
              (t
               (if A-arg-var
                   (setf (semantic-structure-arg-st B-noun)
                     (append B-arg-st (list A-arg-var))))
               (if A-event-var
                   (setf (semantic-structure-event-st B-noun)
                     (append B-event-st (list A-event-var)))))
               )
        ))

    (if A-role-in-B-qualia
```

```
        (if (list-search (first (second A-formal-role))
                         (semantic-structure-qualia-st B-noun))
            (setf (semantic-structure-qualia-st B-noun)
              (delete-insert A-predicate-role
                             (append A-role-in-B-qualia
                                     (list A-predicate)) B-qualia-st))
          (setf (semantic-structure-qualia-st B-noun)
            (delete-insert A-predicate-role
                           (append A-role-in-B-qualia
                                   (list A-predicate)
   (list (second A-formal-role)))
                           B-qualia-st)))
      (if (list-search (first (second A-formal-role))
                       (semantic-structure-qualia-st B-noun))
          (setf (semantic-structure-qualia-st B-noun)
(append B-qualia-st
(list (list A-predicate-role A-predicate))))
        (setf (semantic-structure-qualia-st B-noun)
      (append B-qualia-st (list (list A-predicate-role A-predicate
      (second A-formal-role)))))))
    (combine-variable request-B-event A-event B-noun)
    ))

;;B の項構造が event タイプを要求しているとき、
;; → A からその event タイプに応じた event タイプを持つ引数を返す
(defun A-event-arglist (A-lexicon B-lexicon)
  (let* ((B-event-st-list (semantic-structure-event-st B-lexicon))
         (B-event (qualia-search (arg-var-e? B-lexicon) B-event-st-list))
         (B-event-var (arg-var B-event))
         (B-event-type (arg-type B-event))
         (A-event-lists (semantic-structure-event-st A-lexicon))
         (B-var-list nil))
```

```
    (if B-event-type
        (dolist (A-event-list A-event-lists)
          (let ((A-event-type (arg-type A-event-list))
                (A-event-var (arg-var A-event-list)))
            (if (or (eq A-event-type B-event-type)
    (type-match B-event-type A-event-type))
                (and (combine-variable A-event-var B-event-var A-lexicon)
                     (push A-event-var B-var-list)))
            )))
    (list B-event-var B-var-list)
    ))

;; 項構造で変数がイベント変数のときその変数を返す。
(defun arg-var-e? (lexicon)
  (let ((arg-st-list (semantic-structure-arg-st lexicon)))
    (dolist (arg-list arg-st-list)
      (let ((var (arg-var arg-list)))
        (if (and (symbolp var) (equal (char (symbol-name var) 0) #\E))
            (return var))))))


(defun combine-variable (B-var A-var A-lexicon)
  (let ((A-arg-st-lists (semantic-structure-arg-st A-lexicon))
        (A-event-st-lists (semantic-structure-event-st A-lexicon))
        (A-qualia-st-lists (semantic-structure-qualia-st A-lexicon)))
    (setf (semantic-structure-arg-st A-lexicon)
  (subst B-var A-var A-arg-st-lists))
    (setf (semantic-structure-event-st A-lexicon)
  (subst B-var A-var A-event-st-lists))
    (setf (semantic-structure-qualia-st A-lexicon)
  (subst B-var A-var A-qualia-st-lists))
    A-lexicon
```

```
    ))

(defun top? (noun)
  (if (= (length noun) 1)
      (let ((argment (lexical-entry-arg-st (first noun))))
        (if (eq top argment)
            t))))

;; 構造体 lexical-entry を semantic-structure としてコピー
(defun cp-lexical-entry (structure)
  (let ((copy-arg-st (cp-list-L2 (lexical-entry-arg-st structure)))
        (copy-event-st (cp-list-L2 (lexical-entry-event-st structure)))
        (copy-qualia-st (cp-list-L3 (lexical-entry-qualia-st structure))))
    (make-semantic-structure :phone (lexical-entry-phone structure)
                             :arg-st copy-arg-st
                             :event-st copy-event-st
                             :qualia-st copy-qualia-st)
    ))

(defun cp-semantic-structure (structure)
  (let ((copy-arg-st (cp-list-L2 (semantic-structure-arg-st structure)))
        (copy-event-st (cp-list-L2 (semantic-structure-event-st structure)))
        (copy-qualia-st (cp-list-L3 (semantic-structure-qualia-st
     structure))))
    (make-semantic-structure :phone (semantic-structure-phone structure)
                             :arg-st copy-arg-st
                             :event-st copy-event-st
                             :qualia-st copy-qualia-st)
    ))

;;list をコピー
(defun cp-list-L2 (lists)
```

```
  (let ((temp-list nil))
    (if (atom lists)
        lists
      (dolist (x lists)
        (if (atom x)
            (push x temp-list)
          (push (copy-list x) temp-list))))
      (reverse temp-list)
     ))

(defun cp-list-L3 (lists)
  (let ((temp-list nil))
    (dolist (x lists)
      (if (listp x)
          (push (cp-list-L2 x) temp-list)
        (push x temp-list)))
    (reverse temp-list)
    ))

;; 木構造より key を root とする部分木を取り出す
(defun list-search (key tree)
  (cond ((null tree) nil)
 ((listp tree)
  (if (eq key (car tree))
      tree
    (or (list-search key (car tree))
        (list-search key (cdr tree))))))
  )

;;list 内に key をもつとき真を返す。
(defun list-search-for-qualia-search (key list)
  (cond ((null list) nil)
```

```
        ((and (symbolp list) (eq key list)) t)
        ((listp list)
         (or (list-search-for-qualia-search key (car list))
             (list-search-for-qualia-search key (cdr list)))))
  )

;;(list (list )) 内で key を持つ list を取り出す。
(defun qualia-search (key list)
  (dolist (x list)
    (let ((match (list-search-for-qualia-search key x)))
      (if match
          (return x))))
  )

;; 等しいリストのとき真を返す
(defun eq-list (key list-x)
  (cond  ((and (symbolp key) (symbolp list-x))
          (if (eq key list-x)
              t))
        ((eq (first key) (first list-x))
         (eq-list (rest key) (rest list-x)))
        (t nil))
  )

;; リスト内より key を持つリストを取り除く
(defun delete-key (key lists)
  (let ((target-list (qualia-search key lists))
        (new-lists nil))
    (dolist (list-x lists)
      (unless (eq-list target-list list-x)
          (push list-x new-lists))
      )
```

```
    (reverse new-lists)
    ))

(defun delete-insert (delete-list-key insert-list lists)
  (let ((target-list (qualia-search delete-list-key lists))
        (new-lists nil))
    (dolist (list-x lists)
      (if (eq-list target-list list-x)
          (push insert-list new-lists)
        (push list-x new-lists))
      )
    (reverse new-lists)
    ))

(defun member-phone (A-noun B-noun-list)
  (let ((A-name (lexical-entry-phone A-noun)))
    (dolist (B-noun B-noun-list)
      (let ((B-name (lexical-entry-phone B-noun)))
        (if (eq A-name B-name)
            (return t))))))

;; 項目内の変数名を変更する。
(defun change (noun)
  (change1 "X" noun)
  (change1 "E" noun)
  noun
  )

(defun change1 (variable noun)
  (let ((argments (semantic-structure-arg-st noun))
        (event-argments (semantic-structure-event-st noun))
        (qualia-lists (semantic-structure-qualia-st noun)))
```

```
 (cond ((string= variable "X")
        (let* ((argments (semantic-structure-arg-st noun))
               (noun-qualia (semantic-structure-qualia-st noun))
               (noun-type (second (first noun-qualia)))
               (var-list (take-dotted-type-apart (second noun-type)))
               (dot-var nil)
               (new-dot-var nil)
               (new-dot-var-list nil))
          (dolist (arg argments)
            (let ((var (arg-var arg)))
              (unless (eq (char (symbol-name var) 0) #\E)
                (let ((new-var (intern (gensym variable))))
                  (if (member var var-list)
                      (push (list var new-var) new-dot-var-list))
                  (dolist (role qualia-lists)
                    (dolist (predicate (rest role))
                      (let ((temp (list-search var predicate)))
                        (if temp
                            (setf (first temp) new-var))
                        ))
                    )
                  (setf (arg-var arg) new-var)))
              ))
          (dolist (new-var (reverse new-dot-var-list))
            (if dot-var
                (setf dot-var (intern (concatenate 'string
                                (string dot-var) "*"
(string (first new-var)))))
              (setf dot-var (first new-var)))
            (if new-dot-var
                (setf new-dot-var
```

```
 (intern
  (concatenate 'string
       (string new-dot-var)
       "*"
       (string (second new-var)))))
                 (setf new-dot-var (second new-var)))
               )
             (combine-variable new-dot-var dot-var noun)
             ))
          ((string= variable "E")
           (dolist (arg event-argments)
             (let ((new-var (intern (gensym variable)))
                   (var (arg-var arg)))
               (dolist (role qualia-lists)
                 (dolist (predicate (rest role))
                   (let ((temp-event (list-search var argments))
                         (temp-arg (list-search var predicate)))
                     (if temp-arg
                         (setf (first temp-arg) new-var))
                     (if temp-event
                         (setf (first temp-event) new-var))
                     )))
               (setf (arg-var arg) new-var)
               )))
          )
    ))

(putprop 'top (make-lexical-entry
        :phone 'top
        :arg-st top)
  'lexical-entry)
```

```
(putprop 'animate (make-lexical-entry
     :phone 'animate
     :arg-st '((arg x concrete))
     :qualia-st '((formal (animate x))))
  'lexical-entry)
(putprop 'animate 'concrete 'isa)

(putprop 'animal (make-lexical-entry
     :phone 'animal
     :arg-st '((arg x animate))
     :qualia-st '((formal (animal x))))
  'lexical-entry)
(putprop 'animal 'animate 'isa)

(putprop 'human (make-lexical-entry
                 :phone 'human
                 :arg-st '((arg x animal))
                 :qualia-st '((formal (human x))))
         'lexical-entry)
(putprop 'human 'animal 'isa)

(putprop 'concrete (make-lexical-entry
                    :phone 'concrete
                    :arg-st '((arg x top)
                              (defarg y human))
                    :event-st '((event1 e1 state))
                    :qualia-st '((formal (concrete x))
                                 (telic (possess e1 y x))))
         'lexical-entry)
(putprop 'concrete '(locus top) 'isa)

(putprop 'automaton (make-lexical-entry
```

```
                    :phone 'automaton
                    :arg-st '((arg x concrete*product)
                              (defarg y human))
                    :event-st '((event e1 process))
                    :qualia-st '((formal (automaton x))
                                 (telic (operate e1 y x))))
         'lexical-entry)
(putprop 'automaton 'concrete*product 'isa)

(putprop 'locus (make-lexical-entry
                 :phone 'locus
                 :arg-st '((arg x top))
                 :qualia-st '((formal (locus x))))
         'lexical-entry)
(putprop 'locus 'top 'isa)

(putprop 'interior (make-lexical-entry
                    :phone 'interior
                    :arg-st '((arg x top)
                              (defarg y concrete))
                    :qualia-st '((formal (interior x) (in y x))))
         'lexical-entry)
(putprop 'interior 'top 'isa)

(putprop 'organization (make-lexical-entry
                        :phone 'organization
                        :arg-st '((arg x top)
                                  (defarg y human)
                                  (defarg z role))
                        :qualia-st '((formal (organization x))
                                     (constitutive (post y z) (member-of y
x))))
```

```
         'lexical-entry)
(putprop 'organization 'top 'isa)

(putprop 'product (make-lexical-entry
                   :phone 'product
                   :arg-st '((arg x top)
                             (defarg y human))
                   :event-st '((event1 e1 process))
                   :qualia-st '((formal (product x))
                                (agentive (make e1 y x))))
         'lexical-entry )
(putprop 'product 'top 'isa)

(putprop 'abstract (make-lexical-entry
                    :phone 'abstract
                    :arg-st '((arg x top)
                              (defarg y human))
                    :event-st '((event1 e1 state))
                    :qualia-st '((formal (abstract x))))
         'lexical-entry)
(putprop 'abstract 'top 'isa)

(putprop 'watashi (make-lexical-entry
                   :phone 'watashi
                   :arg-st '((arg x human))
                   :qualia-st '((formal (watashi x))))
         'lexical-entry)

(putprop 'otoko (make-lexical-entry
                 :phone 'otoko
                 :arg-st '((arg x human))
                 :qualia-st '((formal (otoko x))
```

```
                          (constructive (male x))))
        'lexical-entry)

(putprop 'tarou (make-lexical-entry
                 :phone 'tarou
                 :arg-st '((arg x human))
                 :qualia-st '((formal (tarou x))
                              (constructive (name x "tarou"))))
        'lexical-entry)

(putprop 'seifuku (make-lexical-entry
                   :phone 'seifuku
                   :arg-st '((arg x concrete*product)
                             (defarg y human))
                   :event-st '((event e process*state))
                   :qualia-st '((formal (seifuku x))
                                (telic (wear e y x))))
        'lexical-entry)

(putprop 'douzou (make-lexical-entry
                  :phone 'douzou
                  :arg-st '((arg x concrete*product))
                  :qualia-st '((formal (douzou x))))
        'lexical-entry)

(putprop 'kabe (make-lexical-entry
                :phone 'kabe
                :arg-st '((arg x concrete*product))
                :qualia-st '((formal (kabe x))))
        'lexical-entry)

(putprop 'jyuutan (make-lexical-entry
```

```
                :phone 'jyuutan
                :arg-st '((arg x concrete*product))
                :qualia-st '((formal (jyuutan x))))
          'lexical-entry)

(putprop 'building (make-lexical-entry
                    :phone 'building
                    :arg-st '((arg x interior*concrete*product))
                    :qualia-st '((formal (building x))))
          'lexical-entry)

(putprop 'kouen (make-lexical-entry
                 :phone 'kouen
                 :arg-st '((arg x locus*product)
                           (defarg y concrete))
                 :qualia-st '((formal (kouen x) (at y x))))
          'lexical-entry)

(putprop 'hon (make-lexical-entry
               :phone 'hon
               :arg-st '((arg x information)
                         (arg y concrete)
                         (defarg v human)
                         (defarg w human))
               :event-st '((event e1 process)
                           (event e2 transition))
               :qualia-st '((formal (hon x*y) (hold y x))
                            (telic (read e1 v x*y))
                            (agentive (write e2 w x*y))))
          'lexical-entry)

(putprop 'tanoshisa (make-lexical-entry
```

```
                    :phone 'tanoshisa
                    :arg-st '((arg e2))
                    :event-st '((event e1 state)
                                (event e2 process*state*transition))
                    :qualia-st '((formal (enjoy_act e1 e2))))
          'lexical-entry)

(putprop 'mae (make-lexical-entry
     :phone 'mae
               :arg-st '((defarg x pro*sta*tra*concrete)
                         (defarg y pro*sta*tra*concrete))
               :qualia-st '((formal (mae x y))))
          'lexical-entry)
```