自然言語処理 78－4
（1990．7．19）

意味コード体系の自動生成

杉村領一 柿ヶ原康二 石川雅彦 川越睦 青山昇一
松下電器産業 情報通信関西研究所

平成2年 6月 26日

意味コード体系の生成手法について述べる。本論で述べる方法は、意味コード体系の構築作業において人間の判断が入る部分を局所化し、、構築する意味コード体系の客観性を高めることを狙っている。本手法は 2つの手順から成る。第一の手順は、述語とその補語の組の収集であり、実際の例文からこれを集める。どのような述語と補語の組を収集するかは、人間の判断に委ねられる。第2の手順は、この組の書き換えである。書き換えは本論で紹介する手順により自動的に行なわれ、この結果として、意味コード体系を得ることができる。書き換えアルゴリズムとして3つの方法を示し、これらの比較を行なう。

Automatic Generation of Semantic Code Tree

Ryôichi Sugimura, Kôji Kakigahara, Masahiko Ishikawa, Mutsumi Kawagoe, Shôichi Aoyama
Kansai Information and Communications Research Laboratory
Matsushita Electric Industrial Co.Ltd.

A basic method for generating a SEmantic Code Tree (SECT to be abbreviated) is presented. The method presented here has two procedures. In the first procedure, predicates like verbs or adverbs in natural language are collected from example sentences, and pairs of the predicates and their arguments in subject case are choosen from the examples. In the second procedure, the choosen pairs are transformed into SECT.

Three kind of algorithm for the transformation from the choosen pairs into SECT play an important role in the method. The algorithms always terminate. They enables us to get SECT of desirable features. In SECT, a group of words are represented using a generated non-terminal tree node. SECT is suitable for checking an agreement between predicates and their argument. Some other fundamental features are also described.

1 はじめに

自然言語解析等に用いる意味コード体系の構築手法を紹介する。

自然言語の解析では、述語とその引数となる語の間に成り立つ制約 (Appropriateness) を計算機へ実装し、用いることにより、1文の解析によって得られる膨大な解釈の数を減少させている。この制約を適切さに関する制約と呼ぼう。

適切さに関する制約の数は、述語とその引き数になれる語の全ての組み合わせの数である。このため、個々の組み合わせを全て実装することはメモリー効率上得策ではない。そこで、何らかの形で大量の語を縮約した体系を用いて上記の適切さに関する制約を実装する方法が取られてきている。

語を縮約した体系としては、シソーラス (thesaurus)[3][4] が知られている [5]。シソーラスは木構造を用いて語をまとめあげる。語（または、概念）は木のノードに割り付けられ、語と語の関係は木のリンクとして表現される。語と語の関係としては、上位/下位関係 ("is_a") や部分/全体 (hasa, part-of) を示す関係などがあるが、"is_a" 関係はその意味が比較的明確なため、シソーラスの骨格として用いやすい [7]。"is_a"関係では、「木の語の性質はその語の葉の側の語へ伝搬される」。そこで、この性質を用いて、語をグループ化つまり縮約し、前記の適切さに関する制約や、視点 [9] などを実装し、自然言語処理に用いることができる [8]。

しかし、シソーラス [3][4] は、元来人間が読むことを前提として多くの人手を介して構築されている。更には、「性質」の厳密な定義が難しく、上位語と下位語を関係付ける手順に客観性を持たせるのが難しい [5]。このため、構築された体系は開発者が異なれば相違する。また、既存のシソーラスを適切さに関する制約の実装に利用するには、その構造や内容の修正に多くの労力を要するのが現状である。

本論では、これらの問題に鑑み、構築において人間の主観を含みにくく、かつ、適切さに関する制約の実装においては例外を含まない、語を縮約する体系の構築方法について述べる。

まず、構築にあたっては、「語の性質は、語と語の関係のみによって捉えられる」という考え方 [1][2] に乗っとり、語の性質を示す意味素のような存在は仮定しない。

この考えにそって、本手法ではまず述語とこの関係概念の引き数に立つ単語の組を実際の文章から収集する。そして、収集されたデータを、中間ノードを持たず、リーフを共有する木の集合として表現する。すなわち、各木のルートノードには述語を対応させ、リーフノードには名詞など、意味体系で分類すべき単語を対応させる。

この構造に対して、書き換えを行い、非終端のノードを自動的に生成させ語を縮約する。書き換えアルゴリズムとしては、3種類のものを紹介する。第一のアルゴリズムは木のリンク数を縮退させることに主眼を置くもので、体系の生成にあたっては、後述するアルゴリズムのような方向性は無い。第二のアルゴリズムは、体系を構成する木を、そのリーフからルートへ向かってボトムアップに生成する。第三のアルゴリズムは、体系を構成する木を、ルートからリーフへ向かって、トップダウンに生成する。生成によって得られた中間ノードへの名前は便宜上つけ、解析に用いる。この生成は、帰納的に意味コード体系を学習する手順としても捉えることができる [6]。

ＳＥＣＴは、具体的な用例をもとに構築されるため、シソーラス体系に例外的な要素は含まれない。得られた構造は、既存の意味体系と類似点を持つ。但し、各中間ノードの意味は述語によって明確に規定される。

木のリンクを縮退させる第一のアルゴリズムを用いた場合には、書き換え規則を適用する対象の順番が異なれば、"適切さの制約"の意味は変化しないが、得られる木の枝の表現は異なる。

トップダウンおよびボトムアップどちらの構築手法を用いても得られる結果は同一になる。また、書き換え規則を適用する対象の順番が異なっても、"適切さの制約"の意味は変化しない。また、得られる木の枝の表現も変化しない。

2 意味体系の概要

一般に日本語や英語の解析は、ある語がある述語の引き数として適切か（appropriateness）どうかを調べながら進められる。

例として「機械が 生き生きした 鳥に かなうわけがない。」と「羽が生き生きとした鳥にかなうわけがない。」を解析する場合を取り上げる。

まず、第一文の解析では「機械」が「生き生きした」の主体となれるかどうかを調べる。計算機には予め1）「生き生きした」の主体になれる語は「生きている物」であるという知識と2）「鳥」は「生きている物」であるという知識が、"is_a"関係を用いて組み込まれている。「機械」は「生き生きした」の主体にはなれない。そこで、「機械」は、それが修飾（係る）ことのできる唯一の動詞「かなうわけがない」の主体となる。第二の文では、「羽」は「生きている物」であり、「生きている物」は「生き生きした」の主体になれるため、「羽」は「生き生きした」の主体になる。

ここで、"is_a"を以下のように定義する。

$$is\_a(A,B) \leftarrow is\_a(A,C) \wedge is\_a(C,B) \quad (1)$$

$$is\_a(A,A). \quad (2)$$

つまり、is_a を、反射的 (reflexive)、推移的 (transitive) な関係とする。

そこで、例えば述語$_k$の主体として$atom_j$が適切である場合、以下の推論規則3によって得られる論理的な帰結を適切さの制約として用いることができる。

$$\forall A(\text{述語}_k(atom_j) \wedge is\_a(atom_j, A) \rightarrow \text{述語}_k(A)) \quad (3)$$

"is_a"そのものには問題は無いが、実際のシソーラスでは データの分類に客観的な尺度を設けるのが難しいので種々の問題を体系が含む。例えば、「ペンギン」は「鳥」である (is_a(鳥, ペンギン))。しかし、述語「飛ぶ」の引き数として、「鳥」をあげると (飛ぶ (鳥))、推論規則 3によって得られる論理的な帰結として「ペンギン」も飛ぶことになる (飛ぶ (ペンギン))。また、述語「駆ける」の引き数として「人間」をあげるなら、「赤ん坊」も「駆ける」ことにる。

そこで、1) 適切さの制約の実装に例外無く用いることが可能で、2) 分類の明確な基準を持つ体系が必要である。しかし、このような体系は現在まで提案されていない。

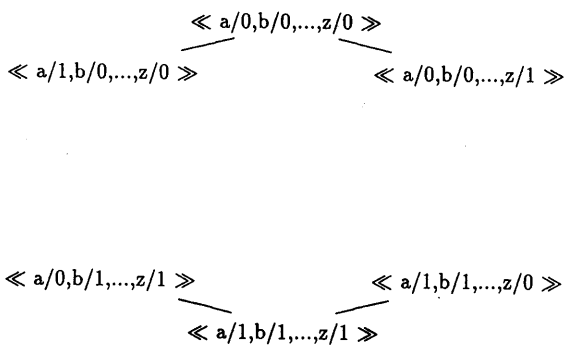
3 意味コード体系の生成

3.1 体系のモデル

意味コード体系のモデルは、体系を構築するにあたって用いられた述語名を並びの名前とする下記のようなベクトルで構成される。

<< 笑う/1, 動く/1, 甘える/0 >>

直観的には、下図のような空間の部分空間に意味コード体系が生成される。



以降、本空間の底から上方向へ意味コード体系を生成するアルゴリズムをボトムアップ、上から下方向へ生成するアルゴリズムをトップダウンと呼ぶことにする。

3.2 望ましい性質

本論が紹介する体系 (SECT) は、下記の性質を満足する。すなわち、述語 P の引き数として A が適切なら ば、A の "is_a"によって得られる推移的閉包 (Transitive Closure) も必ずその述語の引き数になり得る。

$$\forall(P(X) \wedge is\_a(A, X) \leftarrow P(A)) \quad (4)$$

$$is\_a(A, B) \leftarrow is\_a(A, C) \wedge is\_a(C, B) \quad (5)$$

3.3 述語と引き数の収集

まず、述語とその述語の引き数に成り得る語を実際の文章から、例えば以下のように収集する。ここで、v() は述語を、t() はその引き数を示す。

```
is_a(v(甘える),t(子供)).   is_a(v(甘える),t(子犬)).
is_a(v(甘える),t(赤子)).   is_a(v(食べる),t(子供)).
is_a(v(食べる),t(子犬)).   is_a(v(食べる),t(赤子)).
is_a(v(食べる),t(大人)).   is_a(v(食べる),t(虎)).
is_a(v(食べる),t(微生物)). is_a(v(動く),t(子供)).
is_a(v(動く),t(子犬)).     is_a(v(動く),t(赤子)).
is_a(v(動く),t(大人)).     is_a(v(動く),t(虎)).
is_a(v(動く),t(微生物)).   is_a(v(動く),t(機械)).
is_a(v(動く),t(車)).       is_a(v(動く),t(電車)).
is_a(v(存在する),t(子供)). is_a(v(存在する),t(子犬)).
is_a(v(存在する),t(赤子)). is_a(v(存在する),t(大人)).
is_a(v(存在する),t(虎)).
is_a(v(存在する),t(微生物)).
is_a(v(存在する),t(機械)). is_a(v(存在する),t(車)).
is_a(v(存在する),t(電車)). is_a(v(故障する),t(機械)).
is_a(v(故障する),t(車)).   is_a(v(故障する),t(電車)).
is_a(v(笑う),t(子供)).      is_a(v(笑う),t(子犬)).
is_a(v(笑う),t(赤子)).      is_a(v(笑う),t(大人)).
is_a(v(笑う),t(虎)).        is_a(v(生まれる),t(子供)).
is_a(v(生まれる),t(子犬)). is_a(v(生まれる),t(赤子)).
is_a(v(生まれる),t(虎)).
```

3.4 書き換え結果の一例

前記の関係から書き換えによって、本方式は以下のような関係を生成する。n() は新しく生成されたノードである。

```
is_a(v(存在する),n(16)). is_a(v(動く),n(16)).
以上より、n(16) は、存在し、且つ、動く
      下位ノード : n(7),n(15)
is_a(v(食べる),n(15)).
以上より、n(15) は、X を、食べる
      下位ノード : n(13),t(微生物)
is_a(v(笑う),n(13)).
以上より、n(13) は、笑う
      下位ノード : n(12),t(大人)
is_a(v(生まれる),n(12)).
以上より、n(12) は、生まれる
      下位ノード : n(5),t(虎)
is_a(v(甘える),n(5)).
以上より、n(5) は、甘える
      下位ノード : t(子供),t(子犬),t(赤子)
is_a(v(故障する),n(5)).
```

以上より、n(7) は、故障する

下位ノード：t(機械),t(車),t(電車)

上記得られた結果から、"is_a"の引き数に n() と t() だけを含む関係を以下のように択び出す。この結果が所望の意味コード体系である。

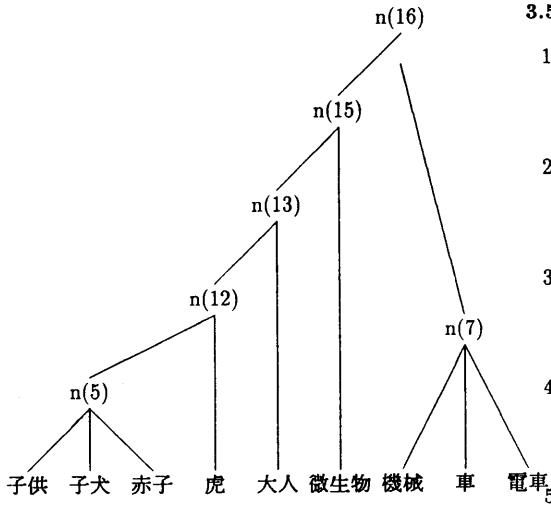


図 1: 生成された意味コード体系

ＳＥＣＴの各ノードＸの性質は、関係 "is_a(v(Ｖ), Ｘ)"のＶによって与えられる。例えば、n(12) 以下のノードは、「生まれる」の主体になれるものと言える。そしてこれらのノードは更に、「笑う」もの n(13) の下位分類であり、更に、「笑う」ものは「食べたり、動いたり」するもの n(15) の下位概念である。以上のように、これらの分類は全て述語とその引き数の直接的な関係から自動的に得られ、かつ、各ノードは分類上のいわゆる意味的な不明瞭さを全く持たない。

続いて、このような結果を得るための書き換え手順を示す。

3.4.1 書き換え対象

書き換えの対象は以下の形の述語の集合である。

$$\text{書き換え対象} \stackrel{def}{=} \{X | X = is\_a(A,B)\} \quad (6)$$

書き換えの前後で保たれるべき性質は $is\_a(v(X),t(Y))$. である。

v() は述語、t(Y) はその引き数となる語である。つまり、書き換え前後で v(X) から (7) (8) を用いて辿れる t(Y) が変化してはならない。

$$is\_a(A,A) \quad (7)$$

$$is\_a(A,B) \leftarrow is\_a(A,C) \wedge is\_a(C,B). \quad (8)$$

3.5 ランダム法

順次、トップダウン、ボトムアップの方向に関係無く、書き換えを行なう手順をまず紹介する。ランダム法は、順次木のリンク数を縮退する手順から成る。

3.5.1 アルゴリズム

1. 2つ以上のノードの共通の子ノード (C_nodes1) として持つ2つの親ノード (P_nodes1) を見つける。無ければ終了する。
2. C_nodes1 を子ノードとするノードを一つ作り、これを P_nodes1 の子ノードとする。P_nodes1 から C_nodes1 への枝を消す。
3. 子ノードを一つ持ち、親ノードを複数持つノードがあれば、このノードと子ノードを同一のノードに書き換える。
4. 親ノードを一つ持ち、子ノードを複数持つノードがあれば、このノードと親ノードを同一のノードに書き換える。
5. 1へ

1－2の手順では例えば次のような書き換えを行う。

3－4の手順では例えば次のような書き換えを行う。

このランダムアルゴリズムは、外延を与えられた幾つかの集合を、各集合の共通部分集合による表現へ書き換える手順として捉えることができる。

つまり、手順1は2つの集合で共有部分を持つものが無いかを調べるものであり、2は共有部分を新たに集合として表現しなおすものである。3と4は、書き換えによって同一の集合を異なるものとして扱う表現を除外するものである。

一旦共通部分集合が作られると、たとえいくつかの共通部分集合の集合がより大きな共通部分集合に成り得る場合でも、これを生成することはしない。実は、このことが以下でのべる完備性を損なう原因になっている。

3.5.2 書き換え1の完備性

上記の書き換えが停止し、且つ、合流性を満たすかどうかを考察する。

書き換えを、複数のリーフを共用する木構造を前記制約(7)と(8)を満たすように書き換える問題として捉えても良いが、ここでは説明の都合上、上記の書き換え対象および書き換え手順を、前記の集合の操作として捉える。

まず、一つの木のノードとそのリーフを、外延の与えられた集合とみなす。リーフは集合の外延であり、ノードはその集合そのものを表すとする。

書き換えの最初の対象Wは以下のようなものである。$V_i(1 \geq i \geq n)$は、述語を示し、t_{ij}は述語の引き数を示す。

$$W = \{V_1, V_2, V_3, \ldots, V_n\} \quad (9)$$
$$V_1 = \{t_{1a}, \ldots, t_{1m}\} \quad (10)$$
$$\vdots \quad (11)$$
$$V_n = \{t_{na}, \ldots, t_{np}\} \quad (12)$$

また、書き換えで満たされるべき性質を、以下のように捉え直す。

$$is\_a(A,B) \leftarrow A = B. \quad (13)$$
$$is\_a(A,B) \leftarrow A \ni C \wedge is\_a(C,B). \quad (14)$$

制約(7)は(13)に、(8)は(14)に各々対応する。また、"is_a"は下記の型式で与える。

$$is\_a(A,B) \stackrel{def}{=} A \ni B.$$

書き換えを⇒で示すとする。書き換えは、上記の集合表現をから、下記の集合表現を得る操作である。V_i, V_jは、V_i', V_j'へ書き換えられ、新たにV_kが作られる。

$$V_i \cap V_j = \{E_1, \ldots E_n\} \wedge n > 1 \quad (15)$$
$$\Rightarrow \quad (16)$$
$$V_i' = \{V_i \cap (V_j)^c, V_k\} \quad (17)$$
$$V_j' = \{V_j \cap (V_j)^c, V_k\} \quad (18)$$
$$V_k = V_i \cap V_j \quad (19)$$

また、この書き換えにおいて、ある集合Aが一つだけの集合Bの要素となっている場合は、集合Aの要素をBの要素とし、集合Aを消去する。この条件は、制約(14)を満足する表現として無限のものを生成することを抑える。

外延を与えられた二つの集合の共通部分集合を求める操作は有限回で停止する。上記手順は、この操作を繰り返すことにより、有限要素の有限個の集合の共通部分集合を、帰納的な手続きによって求めるものである。よって、停止性は保証される。

しかし、本手順によって得られる木構造の合流性は保証されない。これには反例を一つ示せば十分であろう。例えば、以下の構造の書き換えを考える。

$$V_2 = \{2,4,6,12,10,20,30,60\} \quad (20)$$
$$V_3 = \{3,6,9,12,15,30,45,60\} \quad (21)$$
$$V_5 = \{5,10,15,20,25,30,45,60\} \quad (22)$$

上記の書き換え結果としては、3つあるが、以下にその内の2つを示す。

第1の結果 V_2とV_3の共有集合(26)をまず求める手順から始めると、以下の結果となる。

$$V_2' = \{2,4,V_{23},V_{25}\} \quad (23)$$
$$V_3' = \{3,9,V_{22},V_{35}\} \quad (24)$$
$$V_5' = \{5,25,V_{25},V_{35},V_{235}\} \quad (25)$$
$$V_{23} = \{6,12,V_{235}\} \quad (26)$$
$$V_{25} = \{10,20\} \quad (27)$$
$$V_{35} = \{15,45\} \quad (28)$$
$$V_{235} = \{30,60\} \quad (29)$$

第2の結果 V_2とV_5の共有集合(34)をまず求める手順から始めると、以下の結果となる。

$$V_2' = \{2,4,V_{23},V_{25}\} \quad (30)$$
$$V_3' = \{3,9,V_{23},V_{35},V_{235}\} \quad (31)$$
$$V_5' = \{5,25,V_{25},V_{35}\} \quad (32)$$
$$V_{23} = \{6,12\} \quad (33)$$
$$V_{25} = \{10,20,V_{235}\} \quad (34)$$
$$V_{35} = \{15,45\} \quad (35)$$
$$V_{235} = \{30,60\} \quad (36)$$

以上のように、木構造に対する合流性は保証されない。これは、本来一つの性質を持つノードが存在しても、この部分を定義するノードがすでに存在する場合、本アルゴリズムは、新たにノードを生成しないことに起因する。

ただし、得られた結果の"is_a"によって得られるv()とt()の関係は変化しない。

また、例えば、前記(27)(28)のように、二つ以上のノードに共通の要素であるにも関わらず、それが一纏まりのノードとして定義されない場合があるが、二つのノードが一纏まりのノードとしての意味を持つかどうかは、以下の推論規則を用いて調べることができる。

$$parent(A,Parent) \leftarrow is\_a(Parent,A)$$
$$parent(A,Parent) \leftarrow is\_a(X,A) \wedge parent(X,Parent)$$

すなわち、前記述語 "parent" を用い、下記を成立させるノード "Parent" が $node_1, node_2$ に対してあれば、その組 $node_1, node_2$ は、一纏まりの概念として扱って良い。

$$parent(node_1, Parent) \wedge parent(node_2, Parent)$$

3.5.3 書き換え 1 の計算量

ある時点 i までに新たに生成された s(i) 個の部分集合全て対して、新たに処理対象となる集合が s(i) 個の集合と共有集合を持つかどうか時点 i においてチェックする必要がある。時点 i の次に生成される部分集合の数は S(i) なので、結果として得られる部分集合の数は、最大 $S(i+1) = S(i) \times 2 + 1$ となる。$S(1) = 1$ であるので、本漸化式を解くと、最終的時点 n における部分集合は $S(n) = 2^n - 1$ となる。そこで、各チェックの計算量 a を与え、全ての計算量を求めると、$\sum_{i=1}^{n} a(2^n - 1) = a(2^{n+1} - 2 - n)$ となる。しかし、現実にあらゆる部分集合が重なり合う場合は非常に稀である。実際にはこの性質を用いて処理の実行時間を短縮するヒューリスティクスを用いる必要がある。最小の時間は、部分集合が全く生成されない場合で n^2 である。

3.6 ボトムアップ法による書き換え

意味コード体系を、前記モデルの底から上方向へ生成するアルゴリズムを示す。

3.6.1 アルゴリズム

手順は、二つから成る。まず、is_a(A,B) の組を、B を中心とした例えば以下のようなベクトルへまとめ上げる。

```
子供,子犬,赤子 :: 甘える/1,生まれる/1,食べる/1,
              :: 笑う/1,動く/1,故障する/0,存在する/1
      虎      :: 甘える/0,生まれる/1,食べる/1,
              :: 笑う/1,動く/1,故障する/0,存在する/1
      大人    :: 甘える/0,生まれる/0,食べる/1,
              :: 笑う/1,動く/1,故障する/0,存在する/1
    微生物    :: 甘える/0,生まれる/0,食べる/1,
              :: 笑う/0,動く/1,故障する/0,存在する/1
機械,車,電車  :: 甘える/0,生まれる/0,食べる/0,
              :: 笑う/0,動く/1,故障する/1,存在する/1
```

次に、これらのベクトル間の関係を以下の手順で計算する。

1. 任意のベクトル V_I, V_J を取りだし、その包含関係を調べる。一方が他方の真部分集合であれば、真部分集合となっているベクトルから他方へポインターを張る。互いに疎であれば、なにもしない。共通部分があるならば、まず、これと同一のベクトルがないか調べ、ある場合には、これから各々のベクトルからポインターを張る。無ければ、これを新たにベクトルとして登録し、各々のベクトルへポインターを張る。
2. 新たに生成されたベクトルが一つでもあれば、これを書き換え対象として加え、1 へ戻る。無ければ、停止する。

例えば、前記の例に対して、書き換え手順 1 を一通り適応した結果は以下のようになる。

```
子供,子犬,赤子 :: 甘える/1,生まれる/1,食べる/1,    (37)
              :: 笑う/1,動く/1,故障する/0,存在する/1
            =>
      虎      :: 甘える/0,生まれる/1,食べる/1,    (38)
              :: 笑う/1,動く/1,故障する/0,存在する/1
            => (37)
      大人    :: 甘える/0,生まれる/0,食べる/1,    (39)
              :: 笑う/1,動く/1,故障する/0,存在する/1
            => (38)
    微生物    :: 甘える/0,生まれる/0,食べる/1,    (40)
              :: 笑う/0,動く/1,故障する/0,存在する/1
            => (39)
機械,車,電車  :: 甘える/0,生まれる/0,食べる/0,    (41)
              :: 笑う/0,動く/1,故障する/1,存在する/1
            =>
  新ベクトル  :: 甘える/0,生まれる/0,食べる/0,    (42)
              :: 笑う/0,動く/1,故障する/1,存在する/1
            => (41),(40)
```

なお、本アルゴリズムへ書き換え 1 の特徴であるリンク数を縮退させる手順を埋め込むには、2 つのベクトルの共通の要素（つまり、動詞）が二つ以上である場合にのみ、新たなベクトル生成を行なうようにすれば良い。

3.6.2 ボトムアップ法の完備性

書き換え方法の 1 と異なり、本手法による書き換えは完備である。これは、全ての部分集合の関係を計算することによる。

3.6.3 ボトムアップ法の計算量

計算量は、書き換え 1 と同等であるが、計算対象をベクトル化できるため、実装方法によっては、定数項が小さくできるメリットを持っている。

3.7 トップダウン法による書き換え

意味コード体系を、前記モデルの上から下方向へ生成するアルゴリズムを示す。

3.7.1 アルゴリズム

手順は、二つから成る。まず、is_a(A,B) の組を、A を中心とした例えば以下のようなベクトルへまとめ上げる。

```
甘える  :: 子供/1,子犬/1,赤子/1,
        :: 虎/0,大人/0,微生物/0,
        :: 機械/0,車/0,電車/0
生まれる :: 子供/1,子犬/1,赤子/1,
        :: 虎/1,大人/0,微生物/0,
        :: 機械/0,車/0,電車/0
笑う    :: 子供/1,子犬/1,赤子/1,
        :: 虎/1,大人/1,微生物/0,
        :: 機械/0,車/0,電車/0
食べる  :: 子供/1,子犬/1,赤子/1,
        :: 虎/1,大人/1,微生物/1,
        :: 機械/0,車/0,電車/0
動く,存在する :: 子供/1,子犬/1,赤子/1,
        :: 虎/1,大人/1,微生物/1,
        :: 機械/1,車/1,電車/1
故障する :: 子供/0,子犬/0,赤子/0,
        :: 虎/0,大人/0,微生物/0,
        :: 機械/1,車/1,電車/1
```

次に、これらのベクトル間の関係を以下の手順で計算する。

1. 任意のベクトル V_I, V_J を取りだし、その包含関係を調べる。一方が他方の真部分集合であれば、大きい集合となっているベクトルから真部分集合へポインターを張る。互いに疎であれば、なにもしない。共通部分があるならば、まず、これと同一のベクトルがないか調べ、ある場合には、これへ各々のベクトルからポインターを張る。無ければ、これを新たにベクトルとして登録し、各々のベクトルからポインターを張る。

2. 新たに生成されたベクトルが一つでもあれば、これを書き換え対象として加え、1へ戻る。無ければ、停止する。

例えば、前記の例に対して、書き換え手順1を一通り適応した結果は以下のようになる。

```
甘える  ::  子供/1,子犬/1,赤子/1,    (43)
        ::  虎/0,大人/0,微生物/0,
        ::  機械/0,車/0,電車/0
      =>
生まれる ::  子供/1,子犬/1,赤子/1,    (44)
        ::  虎/1,大人/0,微生物/0,
        ::  機械/0,車/0,電車/0
      =>  (43)
笑う    ::  子供/1,子犬/1,赤子/1,    (45)
        ::  虎/1,大人/1,微生物/0,
        ::  機械/0,車/0,電車/0
      =>  (44)
食べる  ::  子供/1,子犬/1,赤子/1,    (46)
        ::  虎/1,大人/1,微生物/1,
        ::  機械/0,車/0,電車/0
      =>  (45)
動く,存在する ::  子供/1,子犬/1,赤子/1,    (47)
        ::  虎/1,大人/1,微生物/1,
        ::  機械/1,車/1,電車/1
      =>  (46)(48)
故障する ::  子供/0,子犬/0,赤子/0,    (48)
        ::  虎/0,大人/0,微生物/0,
        ::  機械/1,車/1,電車/1
      =>
```

なお、本アルゴリズムへ書き換え1の特徴であるリンク数を縮退させる手順を埋め込むには、2つのベクトルの共通の要素（つまり、名詞）が二つ以上である場合にのみ、新たなベクトル生成を行なうようにすれば良い。

3.7.2 トップダウン法の完備性

書き換え方法の1と異なり、本手法による書き換えは完備である。これは、全ての部分集合の関係を計算することによる。

3.7.3 トップダウン法の計算量

計算量は、書き換え1と同等であるが、計算対象をベクトル化できるため、定数項が小さくなるメリットを持っている。

4 実験

本方式により、中規模のシソーラス構築実験を行なった。述語、およびその引数となる語として、予め、使用頻度の高い組を3000組選別し、これらの組を書き換え対象とした。

表1: 書き換え時間

アルゴリズム	時間	実装言語
ランダム	80時間/3000組	Prolog
トップダウン	30時間 / 3000組	Prolog,C
ボトムアップ	30時間 / 3000組	Prolog,C

トップダウンおよび、ボトムアップ法では、ベクトルの計算にPrologを用い、ベクトルの依存関係の計算にCを用いた。書き換えによって得られた意味コード体系の深さは、18であった。以下に結果の一部を示す。

```
n(1)0:v(ある);v(いる);
   t(類 たぐい)1:
   n(11)1:v(取り替える);
      n(10)2:v(降る);
         t(そっち そっち)3:n(320);
   n(21)1:v(くたびれる);
      n(20)2:v(かみ付く);
         n(19)3:v(うなだれる);
            n(18)4:v(かき集める);
               t(自身 じしん)5:n(334);n(369);n(390);v(取りこむ);
               n(24)5:v(けしかける);
                  t(参数 たすう)6:v(治す);
                  n(26)6:v(こぼす);
                     t(後任 こうにん)7:v(取り替える);
                     n(35)7:v(辞する);
                        t(小数 しょうすう)8:
                        n(48)8:v(治す);v(借りる);v(弱める);
                           t(過半数 かはんすう)9:v(取り替える);
                           t(尼 あま)9:
                     n(38)7:n(337);
                        n(37)8:v(腰掛ける);
                           n(59)9:n(358);n(371);
                              t(海軍 かいぐん)10:n(361);
                              n(64)10:v(弱める);
                                 t(敵 かたき)11:
                                 t(両親 りょうしん)11:n(370);n(387);
                           n(133)9:n(356);n(392);
                              t(売り手 うりて)10:n(368);
                              n(136)10:v(降る);v(降ろす);v(取り外す);
                                 n(132)11:v(差し出す);
                                    n(118)12:n(365);
                                       t(親父 おやじ)13:
                                       t(父母 ちちはは)13:v(置す);
                                    n(131)12:v(催す);
                                       n(130)13:v(行き過ぎる);
                                          n(126)14:n(395);
                                             n(125)15:v(置す);
                                                t(あなた あなた)16:
                                                t(院長 いんちょう)16:
                                             t(愛児 あいじ)15:
```

5 ＳＥＣＴの性質について

以下では、ＳＥＣＴの性質を特に学習と関係させて考察する。

5.1 木構造の容量

ランダム法の書き換えにより、木を表現するオーダードペアの数は書き換え以前以下になる。書き換え以前と数が同一になるのは、２つの親ノードが２つの子ノードを共有している場合だけである。

言語を学習の際、言葉の用いられ方から新しい概念を人間は作っているとすることができるならば、本方式は、これを具現化したものとなっている。つまり、言葉の用いられ方を一つ一つそのまま学び、ある程度様々な用いられ方を記憶した段階で、これらをある一群の単語を纏め上げることを人間は行っているとするならば、種々の述語とその引き数の関係だけから複数の単語を縮約するランダム法は、これをシミュレートしているという意味を持つかもしれない。

5.2 木構造の詳細化

ＳＥＣＴでは、述語とその引き数となる単語を多く与えるほど、その単語を含む木構造は複雑になる。この性質も学習と関連があるように思われる。

6 終わりに

意味コード体系の自動生成方式について述べた。本方式を基礎に現在我々は意味コード体系の構築を行っている。我々は本方式で得られた体系を自然言語解析に用いる予定であるが、自然言語解析用の意味コード体系として用いるためには、本方式に更に「類似」したノードの縮約方法を取り入れる必要がある。また、大量の述語とその引き数の組の内、どのような組を択ぶべきかを詳細に検討する必要がある。

認知的な意味についても、上記の作業を行いながら順次検討して行きたい。

参考文献

[1] Barwise,J. and Perry,J., Situations and attitudes, MIT Press, Cambridge 1983

[2] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial In telligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[3] 分類語彙表, 国立国語研究所

[4] 大野晋, 浜西正人, 類語新辞典, 1981

[5] 長尾真言語工学昭晃堂pp.147–173,1983

[6] 中川聖一, 山本幹雄, 若原一彰, 自然言語の文法と意味解析規則の帰納的学習システム, 情報処理学会論文誌,vol.30,No.1, pp.72–80,1989

[7] 日本電子化辞書研究所, EDR電子化辞書, TR-016,EDR, 1989

[8] 徳永健伸, 岩山真, 田中穂積, 上脇正 自然言語解析システムＬａｎｇＬＡＢ, 情報処理学会論文誌,vol.29,No.7, pp.703-711,1988

[9] 徳永健伸, 奥村学, 田中穂積, 概念階層への視点の導入, 情報処理学会論文誌,vol.30,No.8, pp.970–975, 1989