科学技術振興事業団（JST）戦略的基礎研究推進事業（CREST）
研究領域「高度メディア社会の生活情報技術」
セマンティック・タイポロジーによる言語の等価変換と生成
平成 14 年度研究状況報告会　(Mar.22 2003)

文型パターン化の方法と文型照合方式

徳久雅人 *1　池原悟 *1　村上仁一 *1

*1 鳥取大学工学部 知能情報工学科
E-mail: {tokuhisa,ikehara,murakami}@ike.tottori-u.ac.jp

アブストラクト　等価的類推思考の原理に基づく機械翻訳方式を実現するために，日英対訳コーパス，および，日英文型パターン対が作成されている．その性能調査の 1 つに文型パターンの被覆率調査があげられている．本稿では，その調査に必要とされる文型パターン照合方式について，現状を報告する．文型パターンには離散記号という任意の要素と適合するパターン要素がある．そこで，ネットワーク文法は離散要素の記述が簡潔であることから，ATN に基づく照合方式を採用する．さらに，文型照合では照合できる全ての解を出力しなければならない．そこで，複数解の管理をわかりやすくするために，ネットワーク上にエージェントを配置した方式を提案する．実装により照合が正しく行なえることを確認した．今後は，処理速度の向上を目指す．

キーワード　ネットワーク文法，ATN，構文解析，文型パターン照合，エージェント

1　はじめに

等価的類推思考の原理に基づく機械翻訳方式 [1] を実現するために，日英対訳コーパス [4] を作成し，日英文型パターン対 [5] を作成した．その次の作業として，文型パターン照合器を用いてパターンの被覆率を調査することがあげられている．

そこで，本稿では，日本語文型パターンの作成方法を概説し，照合方式についての現状を報告する．

なお，文型パターン化の原則は [1]，文型パターンの記述言語仕様（原案）は [2]，そして，平成 14 年度後期に行った文型パターン作成の実作業の結果は [5] に記載されている[1]．

2　文型パターンの作成

文型パターンの作成上の原則についての説明は既に [1] で述べた．本章では，次章での照合方式を理解する上で必要な部分について概説する．

2.1　文型パターンの構成要素

文型パターンは「字面」,「変数」,「関数」および「制御記号」で構成される．文型パターンは，意味的等価性を保ち訳出に有効であること，かつ，汎用性が高いこと，を狙って作成される．

前者のために，文の構造を支える単語は,「字面」,「変数」あるいは「関数」に置き換えられる．逆にそうでないものは，パターンの構成要素にはならない．

後者のために，文型パターンには,「単語，句，節」という 3 レベルがある．単語レベル文型パターンには訳出精度を，節レベル文型パターンには汎用性を，それぞれ期待している．なお，文型パターンのレベルは，パターン内の変数の汎化レベルによる．

2.1.1　変数

変数は，単語，句，節の 3 レベルがある．

単語レベル： 名詞 N, 動詞 V, 形容詞 AJ, 形容動詞 AJV, 副詞 ADV, 時詞 $TIME$, 数詞 NUM

句レベル： 名詞句 NP, 動詞句 VP, 形容詞句 AJP, 形容動詞句 $AJVP$, 副詞句 $ADVP$

節レベル： 節 CL

句および節の定義は難しい．現段階では，用言型の句は「は格」,「が格」以外の格要素（「を」,「に」,「で」,「から」,「まで」）および用言で構成されるものとする．節は「は格」,「が格」のいずれか 1 つ，0 個以上の格要素，および，用言で構成されるものとする．名詞句は，名詞単独，単独の形容詞と名詞の組，あるいは「の」で連結された名詞とする．

[1] 日英対訳コーパス，日英文型パターン，および，説明書については，http://unicorn.ike.tottori-u.ac.jp/crest/を参照

2.1.2 関数

関数には，変数との関係から次の 2 種類がある．

- **内包制約型関数：** 「変数名」「アンダーバー演算子」「関数名」と繋げて表記する．変数値に対して制約条件を指定する．
- **隣接制約型関数：** 「変数名」「ドット演算子」「関数名」，または,「関数名」「ドット演算子」「関数名」と繋げて表記する．変数または関数に隣接し後続する部分に対して制約条件を指定する．

内包制約型の例：$AJV2\_syushi$　次のパターンと日本文では，変数 $AJV2$ に字面「当然だ」が合致し，終止形である制約 $syushi$ は $AJV2$ の値に対してかかる．

（パ）$N1$ に/$AJV1\_renyo$ にする/のは/$AJV2\_syushi$

（文）老人に親切にするのは当然だ

隣接制約型の例：$V3.teyoi$　次の例では，変数 $V3$ に字面「い（居る）」が合致し，後続の字面「てよい」について制約 $teyoi$ がかかる．

（パ）$N1$ は/$ADV2$/いたいだけ/$V3.teyoi$

（文）君はここにいたいだけいてよい

2.1.3 制御記号

制御記号には，次の 3 種類がある．

離散記号「/」： 任意の文字列と適合

要素選択記号「$\#n$(…|…)**」：** 括弧内の「|」で分けられた要素のうちいずれかが適合

任意要素記号「$\#n$[…|…]**」：** 括弧内の「|」で分けられた要素のうちいずれかが適合，または，非適合

なお,「n」は整数である．要素選択記号の「$\#n$」は省略してもよい．

2.2 文型パターンの作成方法

2.2.1 変数化

単語レベルの変数化では，日本文の形態素解析結果より品詞属性に注目し，かつ，英文の対応に注意を払いながら，変数化を行う．句節レベルはさらに文法的関係に注目して変数化する．

（日本文）太郎は山に登った

（単語レベル文型パターン）$N1$ **は** $N2$ **に** $V3$ **た：**
（結果） $N1 =$ 太郎, $N2 =$ 山, $V3 =$ 登る

（句レベル文型パターン）$NP1$ **は** $VP2$ **た：**
（結果） $NP1 =$ 太郎, $VP2 =$ 山に登る

（節レベル文型パターン）$CL1$ **た：**
（結果） $CL1 =$ 太郎は山に登る

なお，入力文と文型パターンを照合し変数が文の一部と合致すると，変数には文の合致部分が標準表記に変換されてバインドされる．このように，対応がとれることを念頭に，対訳文の一部分を変数化する．

なお，節レベルの変数化は，本年度は，狭い範囲で実施された．汎用的かつ訳出に悪影響のない変数化条件を定めることが困難であるためである．[3] では広義の節について変数化の基準を提案している．

2.2.2 関数化

次の 2 つの日本文は，節変数化すると様相表現が字面で残される．ここで「てみる」も「でみる」も様相表現としては同義のものである．そこでこれらは隣接型形式指定関数に置き換えることができる．

（日本文 1）花子は飴を食べてみる
（節変数化）$CL1$ てみる

（日本文 2）太郎は酒を飲んでみる
（節変数化）$CL1$ でみる

（文型パターン）$CL1.temiru$

文型パターンは，わかりやすさも配慮して作られている．そこで関数の命名方法も問題となる．

さて，関数照合の処理では意味解析を行わない．つまり様相表現に対する意味の判定は行わない．たとえば「〜られる」という入力について，その意味が「受身」であるか「可能」であるかを判定することはできない．したがって，この様相表現と照合する関数が「$ukemi$」や「$kanou$」という名称では不適切である．

そこで，表層表現とその意味の多様性の関係から関数名を定めることにする．この関係は 4 通りある（表 1）．

表 1: 関数の命名方法

命名タイプ	関係タイプ	例「表現」→「意味」（使用例）
単一字面指定	一形一義	「**〜すぎる**」→「過剰」 ($V.sugiru$)
	一形多義	「**〜そうだ**」→「伝聞，推量」 ($V.souda$)
	多形多義	「**〜られる，〜える，〜ことができる**」→「自発，受身，可能，尊敬」 ($V.rareru, V.eru, V.kotogadekiru$)
集合字面指定	多形一義	「〜事にする，〜つもり」→「**予定**」 ($V.yotei$)

2.2.3 制御記号の挿入

離散記号 離散記号により任意の要素を無視して照合できる．たとえば以下の日本文に対して文型パターン 1 は「グビグビと」のために合致しないが，文型パターン 2 は離散記号により合致する．

（日本文）太郎は酒をグビグビと飲んでいる

（文型パターン 1）$N1$ は $N2$ を $V3.teiru$

（文型パターン 2）$N1$ は/$N2$ を/$V3.teiru$

このように文節境界に挿入する．

また，パターンの先頭と末尾に挿入することで，文型パターンを入力文の途中の部分に照合させることもできる．以下の例では「太郎は」,「酒を」,「ようだ」に関わることなくパターンが合致する．

（日本文）太郎は とうとう 酒を 飲んだ ようだ

（文型パターン）/とうとう/$V1.ta$/

要素選択記号と任意要素記号 要素選択記号は，文型の特徴を表す字面などが複数通り存在する場合に有効である．

任意要素記号は，たとえば，日本文における主格の省略に対処できる．

（文型パターン 1）(非常に|あまりに|とても)$AJ1.ta$ので/$CL2.dekiru.nai.ta$

（文型パターン 2）#1[$N2$は/]$VP3$と/$VP4$

3 文型パターンの照合方式

3.1 ネットワーク文法に基づく照合方式

3.1.1 概要

文型パターンの照合は，ATNG(Augmented Transition Network Grammer) に代表されるネットワーク文法に基づく解析方法が適していると考えた．離散記号がネットワークで単純に表現できること，また，関数で指定する処理がネットワーク上に記述しやすいことが理由である．また，DP マッチングによる照合も検討したが採用しなかった．日本文と文型パターンの DP マッチングでは，日本文にギャップが存在しないこと，文型パターンではギャップの位置があらかじめ決まっていることから，DP マッチングは冗長な照合となるためである．

本稿では，文型パターンにおける字面列（連続する字面のみの列）および変数を，ネットワークのノードに対応づけ，要素選択記号および任意要素記号はアークで表現する．離散記号および関数は，ノードを処理する際の特殊条件とする．そして，句節変数は，ATNG と同様，別のネットワークにより対処する．

文型パターンの照合は，常に全ての解を出力する．全解出力のためには，照合結果を蓄積する点において，工夫をしなければならない．そこで，本稿のアルゴリズムでは，照合結果をエージェント[2]に保持させる．

エージェントの動作周期は，入力文の 1 文字を基準とする．1 単語（形態素）区切りとしないのは，文型パターンの要素に字面があるためである[3]．

エージェントには，管理屋と検査屋の 2 種類がある．エージェントが動作するのは,「ネットワーク上のノードに配置されたとき」および「動作周期によりトリガーのかかったとき」である．

3.1.2 管理屋の動作

管理屋は，パターン本体や名詞句など別ネットワークを呼び出すノードの上に存在し，その役割はネットワーク上のエージェントを管理することである．

配置後動作： ネットワークの開始ノードに検査屋を配置

[2]エージェントとは解析過程におけるレジスタ等をまとめたものであり，複数解のためのデータの引き継ぎを分り易く説明するために用いた用語である．一般的なエージェントのように並列分散処理を行うわけではない．

[3]今後，文型パターンの記述仕様が定まり単語区切りで計算可能であることが明らかになれば，単語区切の周期とする．

し，その検査屋の管理を始める．管理下のエージェントが新しい検査屋や管理屋を作ると，それらも管理対象となる．なお，これは周期的動作の途中でも生じる．

周期的動作： 管理下のエージェントに処理をさせ，その結果を集計する．そして，ネットワークの終了ノードに到達している検査屋が存在すれば，このネットワークの成立となり，自身を管理している上位の管理屋に報告する．

消滅条件： 管理下のエージェントがすべて消滅すると自身も消滅する．

3.1.3 検査屋の動作

検査屋の役割は，入力文の字面および形態素情報とネットワーク上のノードと比較することである．ネットワークの開始ノードから検査屋が存在するノードまでは，ある 1 つの経路において照合が成功している．

配置後動作： まず，配置されたノードがネットワークの最終ノードであるか確認する（{ok} ラベルがある）．最終ノードである場合，このネットワークが解析できたことになる．この結果は管理屋に報告される．次に，次ノードが別のネットワークを有するか調べる．有する場合，すなわち，句節レベルの変数の場合は，次ノードに管理屋を配置する．この管理屋は，別のネットワークを管理するものであり，この検査屋を管理している管理屋により管理される．

周期的動作： 次ノードと入力文字あるいは単語（形態素）を照合する．次ノードが関数の場合，関数の内容も照合する．照合に成功すると，自身の子供を産み，照合成功した次ノードに配置する．子供は自身のこれまでの照合結果を保持し，かつ，次ノードの照合結果も持つ．

消滅条件： 周期的動作の後，現ノードに離散記号「/」が無ければ消滅する．

3.2 照合過程の具体例

日本語文型パターン「$/NP1$で$/NP2$を$/V3.teiru$」と日本文「太郎の友達の家で酒を飲んでいる（形態素情報付き）」を照合する過程を具体的に説明する．

3.3 初期化フェーズ

文型パターンは図 1 の「パターン本体ネットワーク」の領域に展開される．そして，このネットワークを管理するエージェント m_1 がこの世界に産まれる．

m_1 は，初期動作として，開始ノード s_1 に検査屋 c_1 を配置する．c_1 は，初期動作により，次ノード $NP1$ 上に管理屋 m_2 を配置する．m_2 は，$NP1$ のために「NP ネットワーク領域」を管理することになる．そして，s_2 に検査屋 c_2 を配置する．

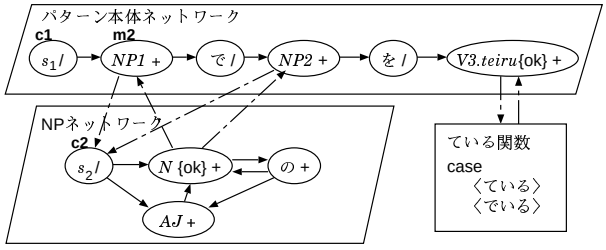

図 1: 初期化直後の様子

3.4 照合フェーズ

まず，最初の 1 文字目に注目して照合を始める．c_2 は次ノードが変数 N であるため，1 文字目に始まる形態素情報を比較する．「太郎=N」と成立するので，子供の検査屋 c_3 を産む．c_3 は c_2 の知識を引き継ぎ，そして，新しい知識を持つ．その結果 c_3 は「n(太郎)」という値を持つ．

c_3 は次ノードに配置される．ここで，c_3 は「NP」の成立を表すノード上にいるため m_2 に報告する．m_2 は，新たな検査屋 c_4 を $NP1$ 上に配置する．c_4 は，m_2 の照合経路に関する知識と c_3 の知識を引き継ぐ．その結果 c_4 は「$np(n$(太郎$))$」という値を持つ．

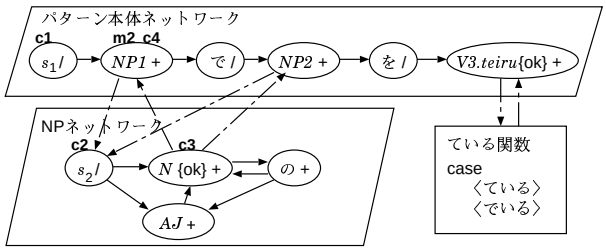

図 2: 「太郎」の照合後の様子

次に 2 文字目「郎」は，いずれのエージェントも反応

しない．なお,「太郎」に合致した検査屋は，文字列長だけ眠っている．

次に 3 文字目「の」は，c_3 が成立し，子供 c_5 を産み，次ノード「の」に配置する．c_4 は次ノードに「の」が無いため何もしない．その後，この周期が終了すると c_3 および c_4 は現ノードが「+」であるため消滅する．なお，c_3 は実質的には「の」に移動したとみなせる．

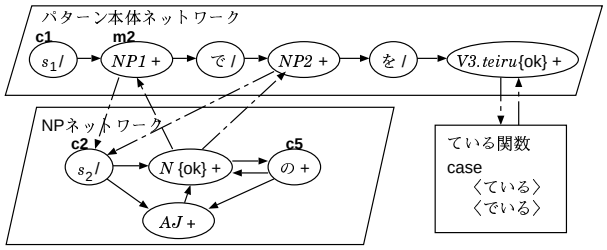

図 3: 「の」の照合後の様子

次に 4 文字目「友達」は，c_2 および c_5 が成立する．そこで，c_2 は子供 c_6 を産み N に配置する．c_5 は c_7 を産み N に配置する．なお，c_6 は「n(友達)」，c_7 は「n(太郎),の,n(友達)」が対応する．もちろん，両者とも最終ノードに存在することから，$NP1$ に c_4 と同様の検査屋 c_8, c_9 が配置される．それぞれは「$np(n$(友達))」,「$np(n$(太郎),の,n(友達))」という値を持つ．なお「の」上に存在していた c_5 は消滅する．

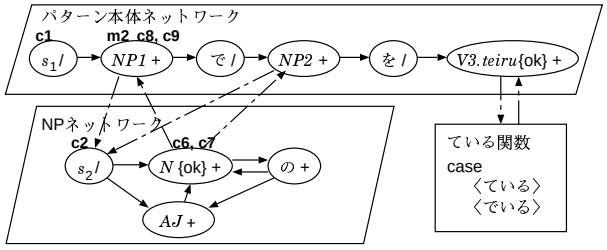

図 4: 「友達」の照合後の様子

次に 5,6 文字目は「達」と「の」であるため同様の処理が行われる．その結果，c_6, c_7 の子供となる c_{10}，c_{11} がノード「の」上に存在し，それぞれ「n(友達),の」,「n(太郎),の,n(友達),の」という値を持つ．

7 文字目「家」では，s_2 上の c_2，ノード「の」上の c_{10}，c_{11} が成立し，N 上にそれぞれ子供を配置する．そして，$NP1$ 上に名詞句の成立結果を持つ検査屋が配置される．それは c_{15}, c_{16}, c_{17} である．

8 文字目「で」により，NP ネットワーク上の c_2 以外の検査屋は消滅する．c_{15}, c_{16}, c_{17} の子供となる c_{18}, c_{19}, c_{20} は，ノード「で」に辿り付く．それぞれ「$np(n$(家)),で」,「$np(n$(友達),の,n(家)),で」,「$np(n$(太郎),の,n(友達),の,n(家)),で」という値を持つ．そして，それら 3 つの検査屋は，次ノード $NP2$ に対して，c_2 と同様に，3 つの管理屋 m_3, m_4, m_5 を配置する．

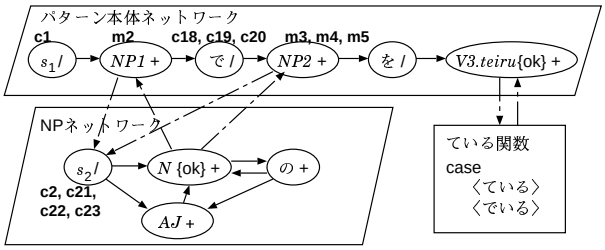

図 5: 「で」の照合後の様子

以上の処理を進めて行き，最後の文字が与えられると，3 つのエージェントが $V3$ および付属する関数 $teiru$ を照合する．こうして，最終ノードについて照合が成功すると，この文型パターンの照合が成功となり，3 つの解が出力される．

4 インプリメンテーション

4.1 実装規模

Linux 上の C 言語により約 6,000 行で作成した．

4.2 入力条件と出力形式

入力は，ALT-J/E の形態素解析結果を流用する．すなわち，単語区切りで改行を入れ，通し番号を付け直したものである（図 6 参照）．

文型パターンは，前述の日本語文型パターンにパターン番号を付与したものである．

照合結果は，入力文と文型パターンの対応位置，合致変数，および，変数バインド値である．バインド値には，対応する字面，標準表記，構文構造，および，入力文と変数の対応位置である．

4.3 設定ファイル

変数，および，関数の構造は，設定ファイルに定義を記述する．次の設定ファイルが必要である．

- 単語レベル変数定義ファイル

- 句節レベル変数定義ファイル
- 内包制約型形式指定関数定義ファイル
- 隣接制約型形式指定関数定義ファイル

定義ファイルの一部を付録（図 8〜11）に示す.

単語レベル変数，および，両関数の定義ファイルでは，行と行は選言条件，1 行内では連続条件とみなされる．< >で囲まれた文字列は単語の標準形と照合し，囲まれていない文字列は字面と照合する．包含制約型の場合，字面の前に「-」をつけて変数内字面の末尾側から文字列を照合することもできる.

句節レベル変数定義ファイルでは，ネットワークを記述している．開始ノードのノード番号は「0, 1」と決まっている．さらに「0」には離散記号「/」が,「1」には連続関係を明示する「+」が対応するように決まっている[4]. 終了ノードには「{ok}」をラベル付ける.

4.4 動作例

以下の文とパターンの照合について説明する.

（日本文）彼が彼女のいえを出るとすぐ不運にも雨が降りはじめた

（日本語文型パターン）P0123[5]

$N1$が/$NP1$を/$V1$とすぐ/$CL1.kaishi.ta$

日本文は図 6 の形式で入力される．実行の様子および出力結果を図 7 に示す.

「% 照合結果」から「----------」までが 1 パターンに対する全ての照合結果が出力されている.「- - - - -」が複数解の区切りである.

照合結果を詳しく見てみよう.

最初の行は，照合したパターン番号，および，パターン要素と日本文の対応位置である．対応位置は，入力文の最初の文字を最下位ビットとし，対応する場合にビットを立てたものである．4 番目が最も多くビットが立っており，離散記号による照合漏れが「不運にも」だけであることがわかる.

次の行からは各変数のバインド結果である．内容は，標準表記，入力文の対応字面，構文構造，および，対応位置である.

[4]図 1〜5 では s に添字を書いているが，定義ファイルには書かない．また，これらの図では連続記号の場合を省略している．上位ネットワークにおいて直前のノードが「/」か「+」により下位ネットワークにおける「s/」と「s+」が決まる.

[5]これは動作確認用のパターンである

```
1.  彼 (1710)
2.  が (7410)
3.  彼女 (1710)
4.  の (7410)
5.  いえ (1100, 家)
6.  を (7430)
7.  出る (2216)
8.  と (7610)
9.  すぐ (4140, 直ぐ)
10.  不運に (3215)
11.  も (7530)
12.  雨 (1100)
13.  が (7410)
14.  降り (2183, 降る)(2213, 下りる)
15.  はじめ (2813, 始める)
16.  た (7216)
```

図 6: 入力形式の例

表 2: 登録パターン数毎の照合時間（1 文照合時）

文型パターン照合件数	処理時間 (sec)		使用メモリ (MB)
	1 文照合時	4 文照合時	
50	0.08	0.14	–
10,000	3.60	8.53	20
50,000	60.52	110.15	96
100,000	254.66	413.00	191
150,000	624.66	992.23	286

5 実験

今後の大規模な照合実験に備え，照合速度を計測する.

実験に使用する計算機は，CPU は PentiumIII 866MHz，メモリは 384MB である．文型パターンは 50 種類を用意し，パターン数の多い場合の実験は再利用することにする.

5.1 文型パターン登録件数の依存性

文型パターン辞書への登録数に，照合時間がどれだけ依存するか調べた（表 2）．入力する日本文を 1 文として照合すると，登録件数 n に対して，処理時間が指数オーダで増えている．次に 4 文の入力を照合すると，1 文照合時と比べて，処理時間が 4 倍にはなっていない．現在のところ，この原因の 1 つにメモリ管理が指摘されている.

本実験より，照合速度を向上させるために，照合対象となるパターン数を，前処理の段階で削減すると高速化につながると期待できる.

```
[tokuhisa]$ jptc2 -env env -in in0123 -j P0123
[tokuhisa]$ cat P0123.output
PATTERN=P0123=0x1f81fe3
N1=彼=彼=n(彼)=0x1
NP1=家=いえ=np(n(いえ [家]))=0x60
V1=出る=出る=v(出る)=0x300
CL1=降る|下りる=降り=cl(vp(v(降り [降る, 下りる])))
   =0x180000
- - - - -
PATTERN=P0123=0x1fe1fe3
N1=彼=彼=n(彼)=0x1
NP1=家=いえ=np(n(いえ [家]))=0x60
V1=出る=出る=v(出る)=0x300
CL1=雨が降る|雨が下りる=雨が降り
   =cl(np(n(雨)),が,vp(v(降り [降る, 下りる])))
   =0x1e0000
- - - - -
PATTERN=P0123=0x1f81fff
N1=彼=彼=n(彼)=0x1
NP1=彼女の家=彼女のいえ=np(n(彼女),の,n(いえ [家]))=0x7c
V1=出る=出る=v(出る)=0x300
CL1=降る|下りる=降り=cl(vp(v(降り [降る, 下りる])))
   =0x180000
- - - - -
PATTERN=P0123=0x1fe1fff
N1=彼=彼=n(彼)=0x1(1)
NP1=彼女の家=彼女のいえ
   =np(n(彼女),の,n(いえ [家]))=0x7c
V1=出る=出る=v(出る)=0x300
CL1=雨が降る|雨が下りる=雨が降り
   =cl(np(n(雨)),が,vp(v(降り [降る, 下りる])))
   =0x1e0000
---------
[tokuhisa]$
```

図 7: 照合の様子

6 考察

6.1 高速化に向けて

6.1.1 照合本体系の効率化

複数のエージェントがネットワーク上で同一ノードに存在する場合，それらのエージェントは，以降の照合処理で全く同一の処理をするにもかかわらず，独立して処理をしている．

そこで，同一のノードに検査屋が存在する場合には「まとめ屋」エージェントにより照合・比較処理を 1 つにまとめる．そして，まとめ屋が，過去の照合結果を別々に管理し，今後の照合結果を共通化する．

こうして，照合回数の削減，および，記憶領域の省力化が期待できる．

6.1.2 照合前処理の可能性

前処理では,「文型パターン内の字面の全てが，入力文に含まれていること」という条件でその文型パターンを照合するかどうかを判定するものである．

表 3: 自立語変数化ファイルにおける字面の出現状況

	タイプ 1	タイプ 2
文型パターン数	134,136	112,292
異なり字面数	11,715	9,786
頻度 1 の字面数	6,463	5,398
高頻度字面 1 位	を (73,524)	を (60,160)
2 位	の (63,172)	は (57,557)
3 位	は (62,539)	が (42,584)
4 位	が (47,994)	に (41,770)
5 位	に (47,872)	た (29,376)

[4] における自立語変数化ファイルを対象に，この前処理の効果について予備調査をする．

まず，文型パターンの内訳について表 3 に示す．任意化の深さから，2 タイプのパターンがある．

テストは，パターンと照合する入力日本文から，格助詞や様相表現などが抽出できたとして，10 万件台の文型パターンから候補を何件にまで減らすことができるかを調べる．

以下の 6 つの列が入力日本文のキーワード列である．

列 1 が，の，事を，だと，のは，だ

列 2 は，ので，は，だ

列 3 が，ているんだから，は，だろう

列 4 が，ては，だ

列 5 と，が，られる

列 6 ので，だ

検索の結果を表 4 に示す．粗い検討であるが，絞り込みの前処理により照合対象を減らすことができる見込みがある．

ただし，この予備調査では，入力文から格要素や様相表現などを抽出できたことを前提にしているため，絞り込みが鋭く効いている．

しかしながら，本来は，日本文全体の文字列とパターン中の字面とを比較するものである．これについては今後調査を行う．

表 4: 絞り込み件数

列名	絞り込み後の件数	
	タイプ 1	タイプ 2
列 1	160 件	168 件
列 2	171 件	163 件
列 3	18 件	155 件
列 4	68 件	107 件
列 5	1 件	97 件
列 6	42 件	41 件

6.2 文型パターン記述能力の問題点

ALT-J/E の形態素解析結果の都合により文型パターンに加工が必要な場合がある．現在わかっているものは以下のとおりである．

- $V1.dekiru$：ALT-J/E では「サ変動詞」は「できる形」があるため別の単語にならない．$V1$ にすべてがバインドされてしまう．その場合は，包含制約型の関数で記述しなければならない．
  （対策）$V1.dekiru \Rightarrow (V1.dekiru|V1\_dekiru)$
- $AJV1.desu$：上記と同様に「形容動詞」は「です形」がある．
  （対策）$AJV1.desu \Rightarrow AJV1\_desu$
- $V1.souda$：不確実な断定を表す関数 $souda$ は，動詞の連用形に接続することが決められている．本パターン照合器では，包含制約と隣接制約を 1 つの関数で処理できない．
  （対策）$V1.souda \Rightarrow V1\_renyo.souda$
- $V1.meirei$：外見は隣接制約であるが，実質的には内包制約であるため変換が必要である．
  （対策）$V1.meirei \Rightarrow V1\_meirei$
- 1 つのパターン内で同一の変数を使う場合

関数が連続する場合や，付属する変数が単語レベルか句節レベルかなど，対処方法を慎重に考えておく必要がある．

最後のパターン内で同一の変数が使われる場合について，具体的な照合方法を検討する余地がある．まず，名詞変数が使われる場合について考える．「僕」の形態素解析結果は「私」という標準形になる．もし，1 文中で「僕」と「私」が使い分けられているならば，標準形により複数の名詞変数の一致をとることは不都合になる．次に，パターン内で同一の用言変数が使われている場合があると，標準形で照合しなくては，活用形のゆれにより一致しないことが多い．

（例文）君はここにいたいだけいてよい

（パターン）$N1$ は/$ADV2$/$V3.tai$ だけ/$V3.teyoi$

7 おわりに

本稿では，日本語文型パターンの記述および照合処理に関して，以下のとおり現状を報告した．

- 文型パターン記述言語の仕様を概説した．
- 文型パターンを照合するアルゴリズムを示した．
- 実走行のための諸設定を紹介した．
- 動作確認の結果を示した．

今後の課題は，次のとおりである．

- 被覆率調査に向け，高速化を行う（照合本体系の改良，前処理系の実装）．
- [5] の文型パターン記述仕様に合わせて定義ファイルの調整を行う．

参考文献

[1] 池原, 佐良木, 宮崎, 池田, 新田, 白井, 柴田：“等価的類推思考の原理による機械翻訳方式”, 信学技報, **TL2002-34**, pp. 7–12 (2002).

[2] 池原, 宮崎, 佐良木, 池田, 白井, 村上, 徳久：“機械翻訳のための日英文型パターン記述言語”, 信学技報, **TL2002-48,NLC2002-90**, pp. 1–6 (2003).

[3] 徳久, 村本, 池原, 村上：“日英文型パターンにおける節の変数化条件について”,「セマンティック・タイポロジーによる言語の等価変換と生成」平成 14 年度研究状況報告会 (2003).

[4] “日英対訳コーパス編成ファイル” (2002).

[5] “日英対訳文型パターン編成ファイル” (2003).

付録 1 設定ファイルのサンプル

```
N ; 名詞
    (110.)  ; 一般名詞
    (111.)  ; 副詞型名詞
    (112.)  ; 連体詞型名詞
    (17..)  ; 代名詞
    (19..)  ; 固有名詞
end
NUM ; 数詞
    (16..)(622.)
    (16..)(623.)
    (16..)(626.)
    (16..)(627.)
    (16..)  限目
    (16..)
end
V ; 動詞
    (2...)
end
```

図 8: 単語レベル変数定義ファイル（一部）

```
NP
    node(0,s,/)
    node(1,s,+)
    node(2,N,+,ok)
    node(3,の,+)
    node(4,AJ,+)
    arc(0,2) arc(0,4)
    arc(1,2) arc(1,4)
    arc(2,2) arc(2,3)
    arc(3,2) arc(4,2)
end
VP
    node(0,s,/)
    node(1,s,+)
    node(2,NP,+)
    node(3,V,+,ok)
    node(4,を,/)
    node(5,に,/)
    node(6,で,/)
    node(7,から,/)
    node(8,まで,/)
    arc(0,2) arc(0,3)
    arc(1,2) arc(1,3)
    arc(2,4) arc(2,5) arc(2,6) arc(2,7) arc(2,8)
    arc(4,2) arc(4,3)
    arc(5,2) arc(5,3)
    arc(6,2) arc(6,3)
    arc(7,2) arc(7,3)
    arc(8,2) arc(8,3)
end
```

図 9: 句節レベル変数定義ファイル（一部）

```
; 用言活用
; 未然形
mizen
    (...1) ; 未然形1 推量形（う、よう）
    (...2) ; 未然形2 受身、使役形（ない、ぬ、せる）
end

; 命令形
meirei
    (...9) ; 命令形
end

; ～できる
dekiru
    (2.3.)  -でき
    (2.3.)  -できる
    (2.3.)  -できれ
    (2.3.)  -できろ
    (2.3.)  -できよ
    (2.4.)  -でき
    (2.4.)  -できる
    (2.4.)  -できれ
    (2.4.)  -できろ
    (2.4.)  -できよ
end

; ～です
desu
    (321.)  -です
    (321.)  -でし
    (321.)  -でしょ
    (322.)  -です
    (322.)  -でし
    (322.)  -でしょ
end
```

図 10: 包含制約型形式指定関数定義ファイル（一部）

```
ta
  <た>
  <だ>
end
dearu
  で<ある>
end
rareru
  <られる>
  <れる>
end
teyoi
  て<良い>
  <てよい>
end
youda
  <ようだ>
end
rashii
  <らしい>
end
darou
  <だろう>
end
desu
  <です>
end
```

図 11: 隣接制約型形式指定関数定義ファイル（一部）