解析表現文法によるCNL日本語文法の試作

若杉 祐依 *　秋信 有花　渡邉 遥輔　倉光 君郎
日本女子大学大学院　理学研究科 *
kuramitsuk@fc.jwu.ac.jp

1　はじめに

構文解析は，コンパイラ構成法だけではなく，自然言語処理でも重要な要素技術である．自然言語処理分野では，曖昧さを扱うため，非決定的な形式文法が用いられてきた．しかし，非決定的文脈自由文法の場合は，最悪計算量が $O(n^3)$ となる．これは確率で解決される場合もあるが，根本的な解決にはならない．また，曖昧さ除去 (disambiguration) も必要となり，手軽な方法とはいえない．

一方で，プログラミング言語の世界では，文法に制約をかけることで，$O(n)$ 処理が可能な決定的文法が開発されてきた．例えば以下のようなものが挙げられる．

- LR 文法
- LL 文法
- 解析表現文法
  Parsing Expression Grammar(PEG)[1][2]

これらの形式文法は，自然言語の曖昧さを扱えないため，自然言語処理には不適切であるとされてきた．そこで，我々は「制限・制約を加えることで曖昧さをなくした自然言語ならば，決定的形式文法で扱えるのではないか」と考えるに至った．

このような制限された自然言語は，Controlled Natural Language(CNL)[3] として多く開発され，有用性も知られている．しかし，CNL の設計は非形式的に行われており，既存のものでは機械処理は難しい．

本研究の目的は，決定性形式文法の制約によって，機械処理しやすい CNL 文法を形式的に定義することである．形式化する対象自然言語は日本語とし，形式化する際には PEG を用いるものとする．また，PEG の利点を活かすため，ターゲットをコードなどの記号が多用されている「ソフトウェア工学文書」に定めた．

本稿の残りは，以下の通りである．２節では，解析表現文法 (PEG) について述べる．３節では，設計方針について述べる．４節では CNL 日本語文法 (CJ) について述べる．５節では，実験と評価した結果について述べる．６節では，本稿をまとめる．

2　解析表現文法 (PEG) とは

解析表現文法 (Parsing Expression Grammar, PEG) は，2004 年に Bryan Ford によって示された形式文法の一種である．軽量かつ決定的な構文解析アルゴリズムとの親和性が高いため，プログラミング言語処理系で広く導入されてきた．

PEG は，A を非終端記号，e を解析表現としたとき，$A \leftarrow e$ で表される規則の集合である．PEG の演算子を表 1 に示す．

表 1: PEG の演算子

e	名前	説明
'abc'	文字 (列) リテラル	同じ文字列とマッチする
[abc]	文字クラス	これらの文字のひとつとマッチする
.	ワイルドカード	任意の一文字にマッチする
A	非終端記号	非終端記号が参照する構文規則にマッチする
(e)	グループ化	
e?	オプション	e はマッチしなくてもよい
e*	0 回以上繰り返し	e の 0 回以上の貪欲な繰返し
e+	1 回以上繰り返し	e の 1 回以上の貪欲な繰返し
!e	否定先読み	もし e のマッチが失敗したら成功，そうでなければマッチは失敗とする
e_1 e_2	連接	e_1 と e_2 を順番にマッチする
e_1 / e_2	優先度付き選択	最初に e_1 をマッチし，失敗したときのみ e_2 をマッチする

PEG のオプションや繰り返しは正規表現と共通の動作をするが，これらの表現は貪欲であり，最長の位置でマッチする．例えば，$a* a$ （ただし a は終端記号）という解析表現は，$a*$ が全ての a を消費するため，後ろの a は必ず失敗する．

また，PEG の優先度付き選択 / は，バッカス・ナウア記法の選択 | のようにどちらか一方にマッチするという定義ではなく，先に e_1 のマッチを試し，失敗

した場合のみバックトラッキングして e_2 のマッチを試す.

3 設計方針

PEG は「美しい水車小屋の娘」などの係り受けによって複数の解釈が存在する曖昧性に関しては，係り受け先を判定することができない．この点は，自然言語に制限をかけ文法定義する際に組み込む予定である．

その一方で，PEG は意味の区切りを見つけることを得意としている．そのため，係り受け解析の正確さは求めずにチャンクごとに区切って後処理できるようにする．

ここでいうチャンクとは，意味のひとかたまりであるとしていて，主に名詞・動詞・形容詞・形容動詞などの直前で切り分けている．

また，解析に使用する辞書は，大きく分けて以下の四種類から構成される．

- 名詞辞書
- 動詞辞書
- 形容辞書
- 修飾辞書

4 CJ: CNL 日本語文法

4.1 概要

PEG による機械処理しやすい日本語文法として CJ を作成している．用途としてはソフトウェア工学文書処理を考えている．ソフトウェア工学文書を選出した理由としては，会話文・命令形などの口語的表現が含まれないことや，PEG が数式や記号の処理を得意としていることが挙げられる．

CJ は日本語の文章を意味の塊ごとに小分けすることで，機械処理の準備とする．このとき分けた塊に，四種類のチャンクの中から最適なものを振り分ける．四種類のチャンクは以下の通りである．

- 名詞チャンク
- 動詞チャンク
- 形容詞チャンク
- 修飾チャンク

実際の処理結果はリスト 1 のようになる．このとき，チャンクは 3, 4, 8, 12 行目に示されている．

リスト 1: 処理結果例

```
1  >>> とても赤いリンゴを食べる
2  [#S
3          [#Adverb 'とても']
4          [#AChunk
5                  [#Adjective '赤']
6                  [#Base 'い']
7          ]
8          [#NChunk
9                  [#Noun 'リンゴ']
10                 [#Object 'を']
11         ]
12         [#VChunk
13                 [#Verb1 '食べ']
14                 [#Base '']
15         ]
16 ]
```

4.2 名詞チャンク

CJ は，次のルールから名詞チャンクを解析する．

- 名詞辞書に含まれる用語
- 1 つ以上の漢字が続く (例. 特許)
- 1 つ以上のカタカナが続く (例. メソッド)
- 5 段活用連用形の辞書（例. 組み)
- 「おにぎり」のように「」で囲まれた文字列
- 上記の組み合わせ

しかしこれでは,「段組み」や「組み込み」などの漢字とひらがなが入り混じる名詞は上記のパターンに含まれず，不十分である．そこで，CJ では「組み」など 5 段活用動詞の連用形を名詞パターンの構成辞書として導入することでこの問題を解決している．

しかし，日本語では先頭の単語が名詞だとしても，後に続くものによって，動詞的な使われ方をされたりする．したがって，先頭が名詞であったとしても，名詞チャンクであると判定することができない．そこで，名詞に続き NounSuffix で定義されたルールによって，接辞詞からチャンクの種類を再判定する．NounSuffix を以下に定義する．

- 「のみ」や「など」など，名詞接辞が続くとき，引き続き名詞チャンクとして続ける
- 「する」などサ行変格活用が続くとき，動詞チャンクとして続ける
- 「な」など形容動詞の活用が続くとき，形容詞として続ける
- 「てにをは」などの助詞が続くとき，名詞チャンクの終端とする

4.3 動詞チャンク

動詞チャンクは，動詞の語幹で始まるチャンクである．動詞の語幹は，カ行・サ行変格活用動詞以外，あらかじめ準備された辞書 (表 2) を用いる．

表 2: CJ 辞書一覧

	辞書名	内容	例
名詞	NOUN	名詞	重み, しきい値
	XVERB	動詞連用形	組み, 受け
動詞	VERB5KA	カ行 5 段	書
	VERB5SA	カ行 5 段	探, 出
	VERB5TA	タ行 5 段	勝
	VERB5NA	タ行 5 段	死
	VERB5MA	タ行 5 段	読
	VERB5RA	タ行 5 段	切
	VERB5WA	ワ行 5 段	笑
	VERB5GA	ガ行 5 段	防
	VERB5BA	バ行 5 段	遊
	VERB1	上一段・下一段	着, 述べ
形容	ADJ	修飾詞	美し
修飾	CONJ	接続詞	しかし, つまり
	ADVERB	副詞	順序よく, とても
	UNIT	助数詞	度, 分, 回
	NONUNIT	非助数詞	分野, 日本

語幹に続く活用形は，動詞活用ごとに定義する．語幹には活用形ごとの活用語尾が続き、さらに後ろに続く助詞や助動詞を、表 3 の ASuffix, ISuffix 等の接辞パターンとして定義する。接辞パターンは,「書かな〈い〉」の〈い〉のように更に活用するものがある．このような活用は，再帰的に定義されている．

動詞は，活用において「ひらがな」が続く．CJ では，動詞語幹辞書のあと，先読みで活用形をチェックしている．これにより,「視る (動詞)」と「視野 (名詞)」のように品詞を区別できる．

動詞チャンクでは，動詞の連用形，連体形に続く文字列も解析している．たとえば,「書くとき」のような文字列は,「書く」で終端とせず,「書く」(#Base) +「とき」(#If) として，接続情報も動詞チャンクに含めている．

表 3: 接辞パターンとタグ付け

非終端記号	タグ	活用例
ASuffix	#Not	書か〈ない〉, 書かず
	#Make	書かされ〈る〉
	#Been	書かれ〈る〉
ISuffix	#Polite	書き〈ます〉
	#WantTo	書きた〈い〉
	#EasyTo	書きやす〈い〉
	#HardTo	書きにく〈い〉
	#Likely	書きがち
	#And	書き
IXSuffix	#Past	書いた
	#After	書いてから
	#EvenIf	書いても
	#And	書いて
USuffix	#Base	書く
ESuffix	#If	書けば
OSuffix	#Will	書こう
〈ない〉	#If	なければ
	#EvenIf	なくても
	#And	なく て?
	#Not	なく〈ない〉
	#Base	ない
	#Past	なかった

4.4 形容詞チャンク

形容詞チャンクは，形容詞の語幹で始まるチャンクである．全て形容詞辞書 (ADJ) により判定している．

形容動詞は，日本語特有の品詞分類で言語学的にも異説が多い．CJ では，形容動詞は「名詞」の形容詞的な活用として扱う．つまり,「正確な」のように「名詞＋〈な〉」のパターンは，名詞が形容詞的に活用し，名詞節に係るチャンクと解釈する．単純にパターンとして扱うため,「抽象な」のような形容動詞として相応しくない組み合わせも受理するが，こちらは構文解析後に名前解析で確認することで解決できる．

4.5 修飾チャンク

修飾チャンクは，用言もしくは体言を修飾するチャンクであり，大別して次の種類がある．

- 接続詞
- 連体詞 (代名詞も含む)
- 副詞
- 助数詞

これらの品詞は，数もある程度制限されるので，基本的に辞書ベースで定義される．

副詞は,「ゆっくり進む」の「ゆっくり」のように用言を修飾する単語であるが，日本語の場合は，単体として名詞として解釈できる場合も存在する．CJ では，副詞に NounSuffix のパターンが続くかどうかで副詞が名詞節を判定する．

助数詞も,「10 メートル進む」の「10 メートル」のように副詞的に修飾する場合と,「10 メートルは長い」のように名詞として解釈できる場合も存在する．

5 予備実験と評価

我々は予備実験として，ソフトウェア工学文書である javadoc 文書[1]の 93484 文を対象に構文解析実験を行った．解析結果は表 4 の通りであった．

表 4: 解析の結果

	割合
解析成功率	100%
エラーチャンク含有率	4.72%

解析の成功率は，構文解析器が構文木を返せば成功，途中で解析が止まれば失敗としている．解析率が 100% であることから，対象中の全ての文が，CJ に基づいて構文木に変換可能であった．

次に，CJ では，解析中の文字がどのチャンクにもマッチしない場合に，ひらがなの連続をエラーチャンクとして構文木に変換している．エラーチャンク含有率は，そのマッチが 1 文の解析結果中に 1 回でも含まれる場合はエラー，1 回も含まれない場合はエラーでないと判断している．エラーチャンクチ含有率が 4.72% であったことから，9 割以上の文が，CJ 中の，タグに意味を持つ構文規則によって解析できていることが分かった．

ただし，本実験で得られた構文木について，その中身を 1 つ 1 つチェックすることは今回はしていない．もし意図と異なる解析がされた結果が存在した場合，それらはバグとして今後修正を行ってく予定である．

6 むすびに

我々は自然言語を機械処理するために，自然言語に形式的に制約を加えることに辿り着いた．制約を加えることで，自然言語を機械処理する際の最大の問題点である曖昧性を解決することができると考えたためである．本研究では．日本語を形式的に定義した CJ を開発した．予備実験では，javadoc 文書を使用した．結果として，全ての文章を構文木に変換することができた．今後は，他文書の処理への挑戦，文法のバグの発見・改善，係り受けなどの PEG でフォローしきれない部分も含め，構文木構築に取り組んで行く．

参考文献

[1] Daisuke Yamaguchi, Kimio Kuramitsu, CPEG: a typed tree construction from parsing expression grammars with regex-like captures, SAC '19 Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing

[2] Bryan Ford, Parsing expression grammars: a recognition-based syntactic foundation, POPL '04 Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages

[3] Tobias Kuhn, Volume 40, Issue 1 - March 2014, A Survey and Classification of Controlled Natural Languages Computational Linguistics

[1]https://docs.oracle.com/javase/jp/8/docs/technotes/tools/windows/javadoc.html