# ブラウザ上でユーザが編集可能な言語パターンマッチシステムの構築
# Building a user-editable language pattern matching system in the browser

桂　辰弥[1)]　　竹内　孔一[1)]
Tatsuya Katsura　Koichi Takeuchi

## 1　はじめに

テキスト中の特定のフレーズや表現を見つけることは，言語および教育分野において必要となることがある．テキストデータから特定のキーワードやフレーズの出現位置や文脈を抽出するためのプログラムとしてコンコーダンサがある．コンコーダンサは語学学習において特定のフレーズや表現の使用例を実際の文脈で把握することで，語彙や文法の理解、単語の使用法や文脈の把握に役立ち，学習者の語彙や表現力の向上に役立つ．パターンマッチングはテキストの表層で検索を行う正規表現とは異なり，情報を抽出したい文を対象に予め関係する文や文の一部に対応する文構造のパターンを用意し，そのパターンに合致する結果を取得するものである．有名なコンコーダンサの例として，Sketch Engine がある．Sketch Engine[1)], はクエリ言語として CQL（Corpus Query Language）[2)]が使用されており，コーパス内で正規表現や演算子を組み合わせることでパターンマッチを行うことができる．しかし Sketch Engine は，コーパスベースの言語分析や統計的な情報抽出が主な役割であるため，テキスト中から依存関係解析を持つ表現を抽出する際には前処理が必要となる．

ユーザ自らがこれらを考慮してテキスト中の特定フレーズや表現を抽出するようなシステムを構築することは容易ではない．そこで本研究では解析モジュールで解析した結果をユーザ自身が求める表現をあらかじめ用意された検索ブロックで組み合わせてシステムに投入し，事例を検索できるシステムの開発を行っている．先行研究において WEB アプリケーションとして JavaScript と Python を利用した基本システムを構築したが，システムの本格利用にはいくつかの課題が残されている．そこで本報告では検索エンジンの中心部分である Prolog データベースの実装の改良，および，大規模なテキストが扱えるためにデータベースをシステムに導入したので、この改良について報告する．

## 2　提案するパターンマッチシステムの概要

本章では開発したパターンマッチシステムを構築する環境と実際のシステムの処理の流れについて述べる．また先行研究との大きな変更点である Prolog データベースの実装の改良，および，大規模なテキストが扱えるためにデータベースをシステムに導入についても触れている．

### 2.1　提案するパターンマッチシステムの構成

本システムは図 1 のようにユーザが視覚的に操作を行うフロントエンドシステムとユーザが要求したテキストの処理バックエンドシステムに切り分けて構成している．

フロントエンドシステムは Javascript のライブラリである React.js で構成されており主な機能としてはテキストファイルのアップロード，検索する言語パターンの構築，解析結果の表示，検索結果の表示などがある．

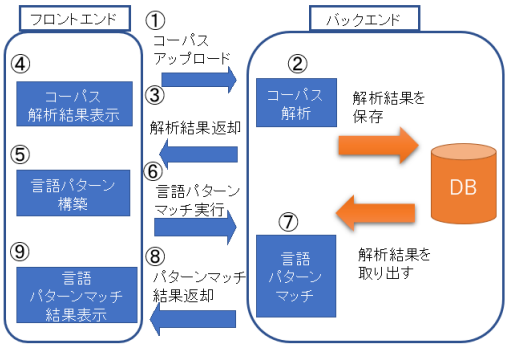


図 1　システムの構成図

バックエンドシステムは Python の Web フレームワークである Django とデータベースシステムである Elasticsearch で構成されており，主な機能としてはテキストファイルの解析，ユーザが構築した検索クエリの言語パターンマッチ実行などがある．

詳しい処理の流れについては以降の節で述べる．

### 2.2　バックエンドの処理の流れについて

バックエンドの処理の流れとしてテキスト解析，言語パターンマッチ実行の処理についてそれぞれ説明する．

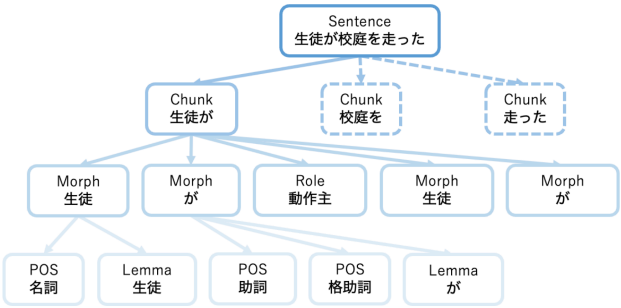


図 2　文を解析した木構造の例

表 1　Prolog の述語一覧

| 述語 | 第 1 引数 | 第 2 引数 | 第 3 引数 |
|---|---|---|---|
| chunk(_, 0,_) | 文番号 | 0 固定 | 文節 ID |
| morph(_, _, _) | 文番号 | 文節 ID | 形態素 ID |
| main(_, _, _) | 文番号 | 文節 ID | 主形態素 |
| part(_, _, _) | 文番号 | 文節 ID | 副形態素 |
| role(_, _, _) | 文番号 | 文節 ID | 意味役割 |
| semantic(_, _, _) | 文番号 | 文節 ID | 概念 |
| surf(_, _, _) | 文番号 | ノード ID | 表層 |
| surfBF(_, _, _) | 文番号 | 形態素 ID | 基本形 |
| sloc(_, _, _) | 文番号 | 文節／形態素 ID | 文中出現位置 |
| pos(_, _, _) | 文番号 | 形態素 ID | 品詞 |
| dep(_, _, _) | 文番号 | 文節 ID | 係り受け文節 ID |

テキスト解析の処理はユーザがテキストファイルのアップロードを行うことで実行される．送られたテキス

1）　岡山大学
1）　https://www.sketchengine.eu/
2）　https://www.sketchengine.eu/documentation/corpus-querying/

トファイルの文に ASA を用いて形態素解析，係り受け解析，項構造解析を適応させる．解析した結果の文の木構造を図 2 に示す．その後 Prolog 述語に変換を行い，データベースに保存する．変換する Prolog 述語は以下の表 1 のように定義している．

ASA で解析したデータが JSON 形式であるため，Elasticsearch をデータベースとして活用することで，JSON 形式の大規模な解析データの柔軟な取り扱いや高速な検索，リアルタイムな更新，スケーラビリティ，高度な分析など，データベースとしての利点を最大限に生かすことができる．

言語パターンマッチの処理はフロントエンドからユーザが構築した検索パターンが送られるとデータベースから各文に対応する Prolog を取得し，Prolog パターンマッチが実行される．パターンマッチを実行する Prolog 処理系として SWI-Prolog の python モジュールである pyswip を使用している．Prolog 処理系として採用した SWI-Prolog は C で書かれた高機能の Prolog 処理系であり，SWI-Prolog の強力な論理プログラミング機能と Python のデータ処理能力を組み合わせることで，高速でより拡張性の高い環境が提供される．

また先行研究のシステムとのバックエンドの大きな改良点としてはバックエンド側でデータベースに保存する際に，以前は全文の解析データを 1 つのデータとして保存していたが，このために保存できる解析データに制限がかかってしまっていたため，1 文ごとに対応した解析データを保存するように改良を行った．またデータベースの取得を 1 文ずつ Prolog データベースの処理を行い，マッチ解の生成を行うように改良した．

## 2.3 フロントエンドの表示機能について

次にフロントエンドでの解析結果，検索クエリとなる言語パターンの構築，検索結果の表示について説明する．

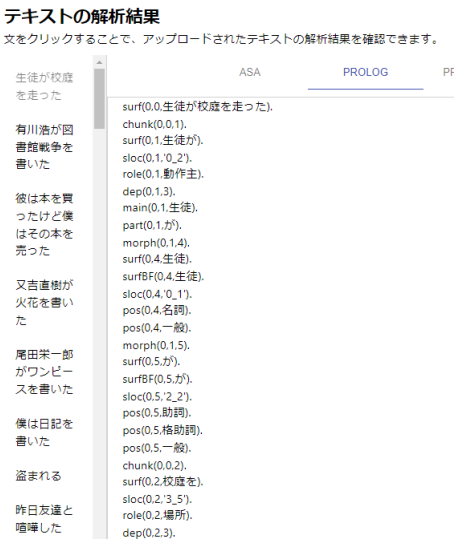


図 3　解析結果の例

フロントエンドはテキスト解析の処理終了後，解析結果をデータベースから取得して表示できる．以下の図 3 は解析結果の表示例である．

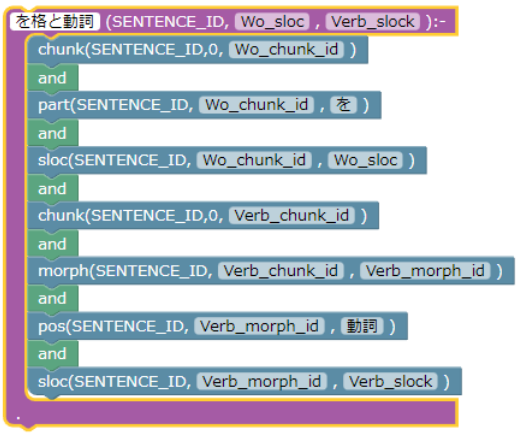


図 4　「ヲ格と動詞」の検索パターン

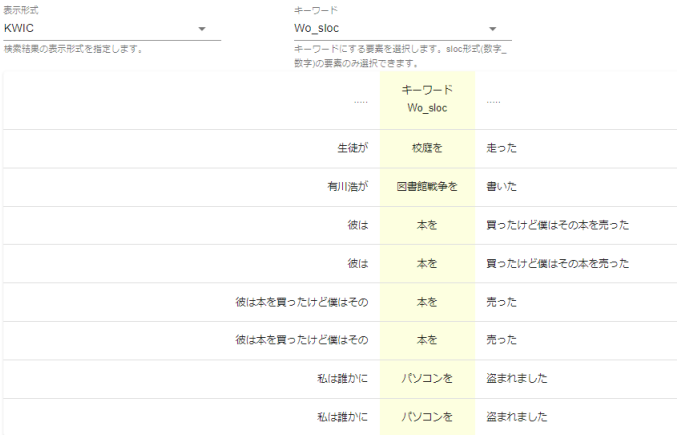


図 5　KWIC 表示

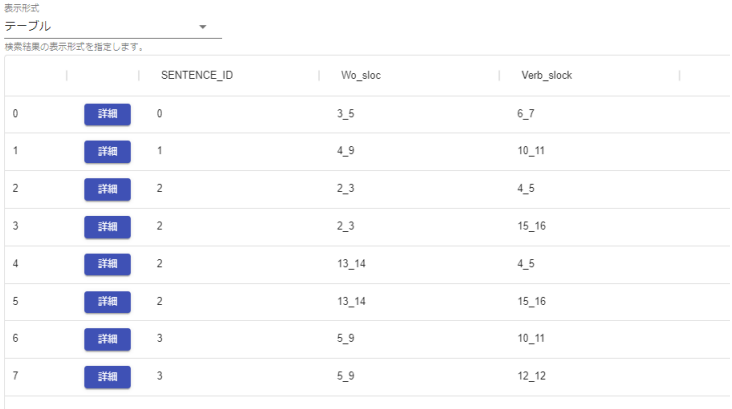


図 6　テーブル表示

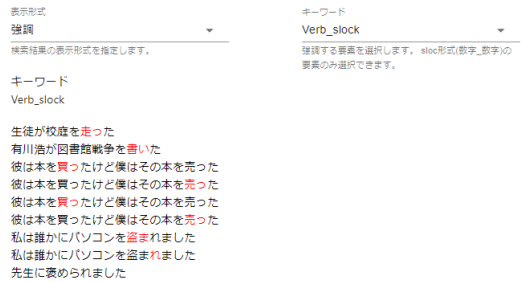


図 7　強調表示

先行研究のシステムとの変更点として，１文ごとに対応した解析データを保存するようにデータベースに保存する変更を行ったため，解析データの表示の際にはクリックした文の解析データがそれぞれ取得するように改良を行った．

言語パターンを構築する際には Blockly を用いて生成できる．前節で示した Prolog 述語のブロックをユーザが自ら組み合わせることで，複雑な検索クエリを構築することができる．図 4 はその例である．検索実行後，バックエンドから検索結果を受信し，結果の表示の際には KWIC, テーブル，強調の 3 つの表示形式を用いることができる．以下の図 5，6，7 は図 4 の検索結果の表示例である．引数 *Wo_Slock,Verb_Slock* は文中での出現位置を示しており，検索パターンの引数に*_slock* を含む場合と表示形式で強調する要素を選択できるようになる．図 5 は「ヲ格」，図 7 は「動詞」の要素を強調して表示し，視覚的にわかりやすくなっている．

## 3　動作評価実験

システムの動作評価実験を行い，パターンマッチシステムの処理性能の向上の確認を行う．

### 3.1　実験内容

表 2　ファイルサイズ (バイト)

| 文の数 | ファイルサイズ |
|---|---|
| 1 | 46 |
| 10 | 440 |
| 100 | 5,342 |
| 1000 | 53,248 |
| 5000 | 262,199 |
| 10000 | 524,399 |

図 3 に示すテキストファイルを用意し，テキスト解析とパターンマッチを行い，これらの処理時間を先行研究のシステムと提案するパターンマッチシステムでそれぞれ計測した．具体的にはフロントエンドからバックエンドに送信し，バックエンドからデータが返ってくるまでを処理時間として計測する．これらの処理時間は Chrome のデベロッパーツールを用いて計測を行う．検索クエリは図 4 の「ヲ格と動詞」の言語パターンを用いる．

### 3.2　実験結果

表 3　テキスト解析の処理時間 (秒)

| 文の数 | 先行研究のシステム | 提案するシステム |
|---|---|---|
| 1 | 0.140 | 0.144 |
| 10 | 0.442 | 0.403 |
| 100 | 1.47 | 3.56 |
| 1000 | 28.9 | 36.4 |
| 5000 | 144 | 192 |
| 10000 | 計測不能 | 768 |

表 4　パターンマッチの処理時間 (秒)

| 文の数 | 先行研究のシステム | 提案するシステム |
|---|---|---|
| 1 | 0.110 | 0.242 |
| 10 | 0.531 | 0.495 |
| 100 | 2.41 | 1.65 |
| 1000 | 37.1 | 30.9 |
| 5000 | 98.2 | 78.4 |
| 10000 | 計測不能 | 160 |

パターンマッチの動作評価実験の結果をそれぞれ表 3，4 に示す．テキスト解析，パターンマッチ実行はともに文の数が増えるにつれ，処理時間も増加していることが読み取れる．テキスト解析の処理時間は先行研究のシステムに比べ，少し遅くなったが，先行研究のシステムは 10000 文を解析できなかったが，提案するパターンマッチシステムは解析可能となり，パターンマッチ実行処理も確認できた．これらの結果からシステムの処理性能の向上が確認できた．

### 3.3　考察

テキスト解析の処理時間は先行研究のシステムに比べ，少し遅くなったが，これはデータベースを 1 文ごとに対応した解析データを保存するように改良を行ったためであり，10000 文を解析を行った際には 10000 個の解析データを保存する必要があるため，バックエンドでの Django とデータベースシステムとのやりとりの時間が増加してしたためである．

Elasticsearch は、1 度に取得することができるドキュメントの最大件数は、デフォルトでは 10,000 件であるためであり，10000 文以上のテキストファイルの解析は現状のシステムでは不可能である．提案するパターンマッチシステムは 10000 文を解析可能となったが，さらに解析後のブラウザの挙動がかなり重くなっており，ユーザが利用可能とは言い難い．今後さらに処理性能の向上させるには，パフォーマンスの問題やネットワークの制約などに留意して実装する必要となる．

また今回の実装では 1 文

## 4　まとめ

謝辞

謝辞の文章を`\acknowledgment` で指定します。使わなければ謝辞は出力されません。