E-008

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[2)]
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．テキストコーパス内の文字列や単語を検索し，検索単語を中心として，前後の文脈とともに示される KWIC 形式のように表示するコンコーダンサは既に数多く存在し [1][2]，これらを用い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[2)]が使用されており，正規表現を用いて複雑な条件やパターンに基づいたデータの検索や抽出が可能である．他にも茶器 [3] や StruAP[4] があり，係り受け構造を利用し、部分木パターンによる関係抽出ができるツールである．しかしこれらのツールは商用である場合や用途が異なるため，言語解析者や語学学習者が文構造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といった目的に利用するのは容易ではない．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 [5][6] 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．

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 で構成されており主な機能として

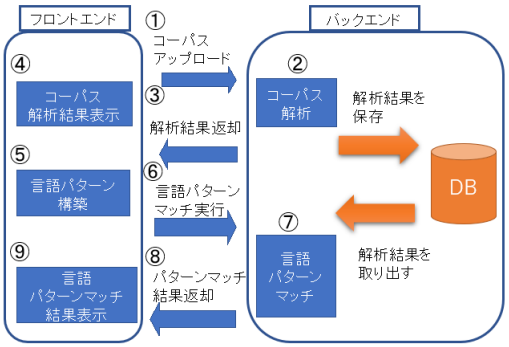

図 1: システムの構成図

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．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[3)]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，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．

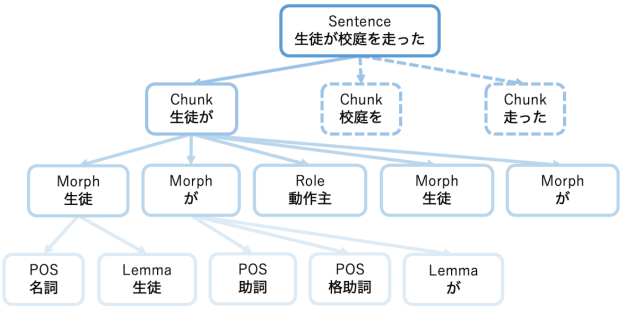

図 2: 文を解析した木構造の例

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

1）　岡山大学大学院環境生命自然科学研究科　Graduate School of Environmental, Life, Natural Science and Technology, Okayama University
2）　岡山大学学術研究院　Academic Research Assembly, Okayama University

1）　https://www.sketchengine.eu/
2）　https://www.sketchengine.eu/documentation/corpus-querying/
3）　https://www.elastic.co/jp/elasticsearch/

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．送られたテキストファイルの文に ASA[4] を用いて形態素解析と項構造を適応させる．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．データベースに保存する際には，1 文ごとに対応した解析データを保存している．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言語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 データを取得し，1 文ずつ Prolog データと検索クエリでパターンマッチを実行し，マッチ解の生成を行う．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[5] は C で記述された高機能の 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．

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 (ASA)

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する．以下の図 3，4 は解析結果の表示例である． 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しているので，解析データの表示の際には文をクリックするたびに

図 4: 解析結果の例 (Prolog)

その文に対応する解析データを取得するように実装している．

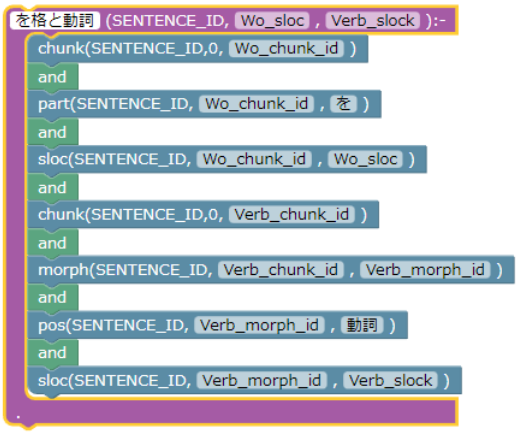

図 5: 「ヲ格と動詞」の言語パターン

言語パターンを構築するには Blockly[6] で定義されたブロックを利用する．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 5 に例を示す．パターンマッチを実行するとバックエンドから検索結果を受け取り，結果を表示する．表示の際にはテーブル，KWIC，強調の 3 つの表示形式を選択することができる．以下の図 6，7，8 は図 5 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 7 は「ヲ格」，図 8 は「動詞」の要素を強調して表示することで視覚的にわかりやすくしている．

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．

4） https://github.com/Takeuchi-Lab-LM/python_asa/
5） https://www.swi-prolog.org/

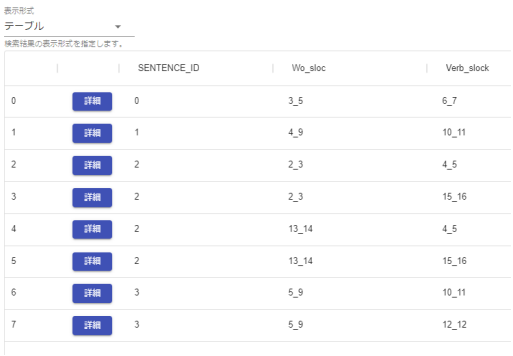

図 6: テーブル表示

図 7: KWIC 表示

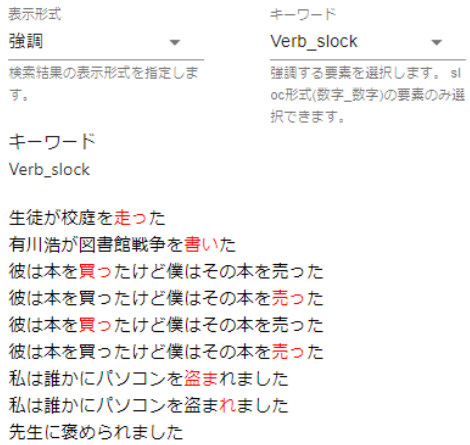

図 8: 強調表示

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

3.1 実験内容

表 2 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．検索クエリは表 5 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	0.083	0.144
10	0.414	0.453
100	3.50	3.56
1000	35.3	36.4
5000	174	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	0.440	0.495
100	8.15	1.65
1000	960	30.9
5000	計測不能	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については先行研究のシステムは 10000 文のテキストファイルを処理する際には動作が止まってしまったが，提案するシステムでは 10000 文でも動作が確認できた．提案するパターンマッチシステムはは先行研究のシステムに比べ，処理時間が少し大きくなっているが微差である．

パターンマッチシステムの処理時間については先行研究のシステムは 5000 文のテキストファイルを処理する際には動作が止まってしまったが，提案するシステムでは 10000 文でも動作が確認できた．提案するシステムは 10000 文のテキスト解析，パターンマッチ実行がともに動作を確認できた．また 100 文以上のパターンマッチの処理時間が向上しており，全体的にシステムの処理性能の向上が確認できた．

ただし 10000 文の処理を行った際にはブラウザの挙動が重くなるため，ユーザの操作に対する影響という新たな課題を確認した．

4 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

6） https://developers.google.com/blockly?hl=ja

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，バックエンドにおける 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．今回の実装では 1 文ずつ Prolog データベースに追加し，パターンマッチを生成した．実際には全文で解探索するほうが，処理時間が小さくなるが，全文の Prolog データの記述されたファイルの生成に多くの時間がかかるため，1 文ずつ Prolog データベースに追加する実装に変更した．しかしこの実装の問題点として，例えば 10000 文のパターンマッチを行う際には 10000 回処理を行う Prolog ファイルの生成を行う必要があり，バックエンドシステムへの負荷が大きい，そのためファイル生成を行わずにパターンマッチを実行する実装方法を検討する必要がある．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 10000 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手順では対応できない．また提案するパターンマッチシステムは 10000 文まで解析やパターンマッチが可能となったが，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がある．

現在の実装では，柔軟なパターンマッチを行うためには，ユーザが Prolog に精通している必要がある．より簡単な操作を実現する方法としては表層に”*” のような曖昧性を含んだ正規表現を持つクエリと Prolog のパターンマッチの実装が考えられる．今回 Prolog 処理系として導入した SWI-Prolog は正規表現に対応可能な regex パッケージが存在するため，今後の課題として正規表現マッチのシステムの導入が考えられる．

5 まとめ

本研究では，先行研究のシステムをベースに開発を進め, 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として，Prolog 処理系として SWI-Prolog を導入し，1 文ずつ Prolog データと検索クエリでパターンマッチを実行し，マッチした解を生成するように実装した．また，大規模なテキストを扱うためにデータベースとして Elasticsearch をシステムに導入した．実装した機能の処理性能を確認するための動作評価実験を行い，10000 文のテキストの動作することを確認した．今後の課題としてさらなる処理性能の向上と正規表現マッチの追加が考えられる．

謝辞

本研究の一部は、日本学術振興会科学研究費補助金 (助成番号 22K00530) の助成を受けた．

参考文献

[1] 中條清美, 西垣知佳子, アントニ・ローレンス．フリーウェア WebParaNews オンライン・コンコーダンサーの英語授業における活用．日本大学生産工学部研究報告 B (文系), Vol. 47, pp. 49–63, 2014.

[2] 小木曽智信, 中村壮範．通時コーパス用『中納言』: Web ベースの古典語コンコーダンサー．第 2 回コーパス日本語学ワークショップ, 2012.

[3] 松本裕治, 浅原正幸, 岩立将和, 森田敏生．形態素・係り受け解析済みコーパス管理・検索ツール「茶器」．情報処理学会研究報告. 自然言語処理研究会報告, Vol. 2010, No. 18, pp. 1–6, Nov 2010.

[4] Kohsuke Yanai, Misa Sato, Toshihiko Yanase, Kenzo Kurotsuchi, and Yoshiki Niwa Yuta Koreeda. StruAp: A tool for Bundling Linguistic Trees through Structure-based Abstract Pattern. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 31–36, 2017.

[5] 岡田魁人, 竹内孔一．Blockly を利用したタグ付きコーパス検索パタン構築ツール．言語処理学会第 *27* 回年次大会発表論文集 *D7-2*, pp. 1291–1294, 2021.

[6] 小笠原崇, 竹内孔一．意味役割付与テキストに対する prolog ベースの探索木による言語パタンマッチシステム構築．言語処理学会第 *27* 回年次大会発表論文集 *C5-1*, pp. 875–876, 2021.