ブロック形式を利用したパターンマッチシステムの構築

竹内 孔一 [1]　小笠原 崇 [2]　岡田 魁人 [2]　今田 将也 [3]
[1] 岡山大学学術研究院 [2] 岡山大学大学院 [3] 岡山大学工学部
takeuc-k@okayama-u.ac.jp

概要

外国語など言語を学習する際に表現を検索する場合や，大規模な意見テキストから不満点や問題点などを分析するテキストマイニングでは，必要に応じて様々なパターンの表現を抽出する必要がある．そこで本研究では，情報抽出や言語の学習で必要なパターンの構築に注目して，ユーザが即時に必要なパターンを作成・編集し，試して共有できる環境を構築することを目標とする．本稿では既存のパターンマッチの事例をとりあげ，それらを包含できるマッチングシステムとしてブロック形式の可能性について議論する．さらに，Prolog 形式のパターンマッチについてブロック形式によるパターンを作成・編集する機構をもつシステムを試作したので報告する．

1　はじめに

テキストもしくは構造化したテキストから必要な情報をパターンを利用して取り出したいことがある．例えば，英語の構文で as〜as one can の表現を取り出して事例を確認したい場合 [1] など，文字列のマッチだけでは不要な事例を取り出しすぎてしまう．近年，自然言語処理では深層学習が様々な手法に取り入れられて大きな成果を挙げているが (例えば [2])，基本的には取り出したいデータに対して正解データを用意して学習する必要があるため，一時的に必要になった言語表現を獲得する分類器を機械学習で作成するのは難しいと考えられる．通常，言語表現の獲得ではコンコーダンサが開発されており，こうしたものを正規表現を中心としたパターンマッチで処理するシステムが提案されている．また NLP では構文解析ツールや構文木ベースのデータ[1)]が構築されているため構文木に対する検索ツールも展開されている [7]. さらに質問応答という応用処理の観点からテキストに対して，係り受け解析などの情報を付与したデータに対してパターンを適用して必要な情報を獲得する情報抽出が Prolog 形式を利用して構築されている [8].

一方で，これらの先行研究ではパターンによる事例を獲得したいユーザがそれぞれシステムを作成することが前提となっているがこうした個別のシステム (プログラム) で作成されているパターンは例え同じ内容のものを検索するパターンでも共有することができず，似たような表現検索パターンを作成していたとしても何度も個人が繰り返して作成して再利用されていないと考えられる．しかしながらテキストから事例が欲しいユーザにとって，プログラムではなく，編集可能 (つまり制御可能) なパターンと自身の持つテキストを投入して結果が得られれば良く，多大な時間を必要とするシステム構築は必須ではないと考えられる．この問題点を整理すると下記のように集約できる

1. あらかじめ取り出したい言語表現は想定できない
2. テキストの表層文字列だけではなく文法やタグなど NLP ツールを利用した検索ツールが必要
3. 基本のパターンを処理する言語処理システムを組むのは容易ではない
4. システムが違うためパターンは共有できない
5. 既存のパターンを変更したり組合わせることが現状では容易ではない

この問題に対して本研究ではユーザがパター

1） 例えば Penn Treebank[3] や PropBank[4]，Hinoki[5], NPCMJ[6] など．

ンを作成してテキストに対して検索結果を得られるだけでなく，他人が作成したパターンを編集して再利用可能な環境の構築を目指す．本論文では様々なパターンを受け入れられる枠組としてブロック形式のパターンが可能性があることを議論し，具体的な事例として Prolog 形式のパターンマッチ編集再利用環境を試作したので報告する．

2 組合わせを実現する抽象的なパターンマッチ言語の検討

現在，様々な検索意図に基づいて様々なツール上で作成されているパターンを包含するようなパターンマッチ言語はどのような形式が可能であるかを議論する．

全ての言語検索の状況を考慮するのは難しいが，少なくとも前節でとりあげた (1) 言語分析などで利用するコンコーダンサ，(2) 構文木構造テキストに対する情報抽出，(3) 質問応答における情報抽出の 3 つの場合について具体的に例をあげて考察する．次にこの議論を踏まえてブロック形式のパターンマッチシステムの可能性について記述する．

2.1 既存のパターンマッチング

例 1) 言語分析などで利用するコンコーダンサ

例えば 1 節で示したように as X as Y can の事例をテキストから抽出したいとする．事例として下記のようなものが考えられる [1].

> as fast as he could, as much as he could, as well as he could, as loudly as she could, as many stones as he could, as many reapers as I can hire

言語分析では事例が漏れない (recall を高くする) ことが重要であるため正規表現をフラットに適用したパターンマッチシステムが利用される．as と as の間の単語[2] を具体的に指定するよりも単語数に応じて skip してマッチするような構造が必要である．

この必要性に対して例えば Sketchengine における Corpus Query Language (CQL)[3] では正規表現だけでなく単語の skip 数などが指定できる検索言語を用意している．また日本語では茶器 [9] が形態素解析や係り受け解析のタグを利用してフラットなパターンマッチ[4] を適用することで何語か離れた語に対して指定したパターンを適用した検索が可能である．

よってまとめるとコンコーダンサでは正規表現ベースで十分であるが単語の skip などを受け付けるパターンマッチ言語が必要であると考えられる．

例 2) 構文木構造テキストに対する情報抽出

1 節で述べたように Penn Treebank をはじめ構文木構造のテキストコーパスがあることから構文木に対して必要な部分を取り出すパターン検索言語が提案されている．例えば tregex[7] では括弧形式の構文木[5] に対して必要な情報を取り出すことが出来る．ユーザは構文木の非終端記号や入れ子構造の各要素間に対する制約を指定することで必要とする情報を取り出す．例えば，単語だけを取り出す検索式は __ !< __ であり，これに on の前置詞を持つ VP の支配下での主辞の動詞を取り出すパターンは (__ !< __)»# (VP < (PP «# on)) のように追加した形式となる．近年では tregex を考慮してさらに発展した構文木処理ツール StruAP[10] も提案されている．

入れ子構造に対するデータの処理は言語だけで無く情報処理で利用されることから検索言語が提案されている．XML 形式であれば XPath[6] があり，利用する場合は PythonBeautiful Soup ライブラリで XPath を利用できる．

XPath の事例は省略するがパターン編集の追加性を考えると入れ子構造を扱うことから単純な構造の追加で無く取り出す目標に対してパターンを大きく書き換える必要があることがわかる (上記の__ !< __に対して，制約を加える際，全体を () で括る必要がある点).

例 3) 質問応答における情報抽出の例

質問応答システム Watson における質問文解析では文献 [8] が示すようにテキストは内部的に構文情報など与えられた後，Prolog による関係データベース化を行い様々な必要となる要素を複数のタグを利用して取り出して

2) 場合によっては単語ではなくトークンとなる．
3) https://www.sketchengine.eu/documentation/cql-basics/ 2022/1/14 アクセス
4) ここでフラットと述べているの検索途中の内容を変数などに保存してさらに検索する再帰的な検索をイメージしている．形式言語におけるスタック付きの文法をイメージしている．
5) 例えば (PP-SBJ (NP (PRO それ))(P-OPTR は)) のような形式.
6) https://ja.wikipedia.org/wiki/XML_Path_Language 2022/1/12 アクセス

いる．例えば He published Songs of a Sourdough. という文は lemma や POS など述語化 (例えば lemma(1, “he”)) したあと，authorOf(A,C):-createV(Vb),subj(Vb,A),obj(Vb,C). などの Prolog 述語で createV(生成する動詞) の主語である著者 (Author)，と目的語の作品名 (Composition) を取り出す[7]．Prolog 形式の特徴は検索結果が大文字の A や C といった変数に入ることである．この A や C の変数に対して，さらに制約を与えたい場合は，論理積で追加することが可能である．一方，例 1 の場合と異なり，テキストの単語列に対して 2 単語後など位置を指定してパターンを適用するのは簡単ではないと考えられる．

よって情報抽出における例では Prolog 形式のパターン言語の場合は追加編集は容易である一方で，コンコーダンサなどで利用されている正規表現や単語の skip する構造は明示的に書くのは難しいと考えられる．

2.2 ブロックベースのパターン構想

前節にあげた複数の検索タスクを可能な限り包含するパターンマッチシステムを考察する．まず例 3 に注目すると検索した内容を変数として取り出して再利用可能な点が特徴的である．Prolog の場合ならば述語を 1 つのブロックとして検索されたものを変数として 1 つのパターンを作成するとする．前節の例では authorOf(A,C) であるが，さらに著者について He などの代名詞ではなく固有名詞が必要な場合はさらに authorOf(A,C) の A に対してさらなる制約を論理積として実現することができる．

しかし Prolog に基づいた規則を守ると例 1 にあるような n 単語後の語など指定した条件で単語を読み飛ばす機構は容易に書くことはできない．しかし Prolog に限定しなければプログラムの正規表現は利用可能である．例えば Prolog で検索した結果に対して，常にその変数の結果を含む文も取り出すとすると，文に対して n 単語後という制約を加えて対応する語を取り出すことは可能である．よって Prolog だけでは無く，ある検索クエリを持つブロックを仮定して，入力と取り出したい目的出力とマッチした文の出力を仮定すれば例 1 における検索要求はブロックの組合わせで満たすのではないだろうか．

検索ブロック (Q,I,Oa,Os) (1)

= Q:検索クエリ,I:検索対象の入力,
Oa:出力引数,Os:マッチ要素を含む文

このブロックの論理積や論理和などを記述することで Prolog でも正規表現も両方検索する．ここで検索クエリは当然，Prolog と正規表現では処理方法が違うため検索ブロックに種類を仮定する必要がある．また正規表現でも n 単語読み飛ばすものと，文字列に対する正規パターンとは扱いが異なる．そうしたものも別ブロックとして用意しておく．

また，Prolog 形式では authorOf(A,C) の検索でヒットした A や C はマッチした文の要素番号の集合が入るが，番号では正規表現パターンマッチは適用できない．つねに文の要素の **span 情報の集合**を検索結果としておくことで，正規表現適用の際には内部的に文字列に戻すことでテキスト (タグ付きテキストも含めて) の検索が可能であると考えられる．この span によるデータの構造化は既に IBM の Watson で利用された UIMA と同じ方向である [11].

こうした抽象化した検索ブロックを仮定すると例 2 も同様である．tregex などで検索にマッチした結果は必ずテキストの集合として獲得できる．そこでその出力集合に対してより制約をかける場合には他の検索ブロックとの論理積で実現できる．ただし，構文木ベースのパターンマッチ間の組み替えなどは難しい．先の例にも挙げたように，tregex の編集は全体の構造を変える必要があることが多く，単に文字列結合などでは実現できないことが多い (2.1 節参照)．よって構文木そのもののパターンを書き換えることは仮定せず，様々なパターンをあらかじめ用意しておき，独立に論理積で利用して制約する．

このように検索ブロックを仮定して抽象化すると，品詞や単語といった従来の文法ベースに特化しない検索の組合せが可能となる．例えば入力文の参照情報を取り出す分類器を深層学習で作成できたとする．この分類器を検索ブロッ

7) テキストは解析されて subj や obj タグならびに createV の動詞集合などは与えられているとする．

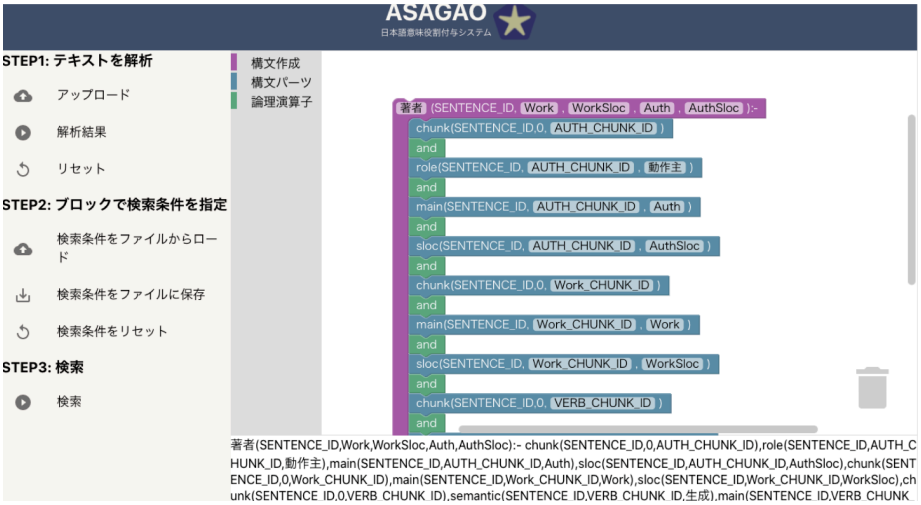

図 1　ブロックベースのパターンマッチシステム

クとして設定しておくと利用することが可能になる．具体的には上述の Prolog の例における authorOf(A,C) の X に対して著者 A が代名詞で合った場合に固有表現を探すブロックに適用して A に対する固有表現 P を取り出すことが可能になる．

よって抽象的な組合わせ可能な検索ブロックを仮定することで Prolog や正規表現，および分類器まで含めた検索が構築できるのではないと考えられる．ユーザは抽象ブロックの操作のみに集中して意図する情報がとれるかどうかを編集しながら検討することに集中できる．

3　Prolog ベースのパターンマッチシステム

上記の背景議論を基にパターンマッチシステムを試作した．現段階では Prolog の述語の単位を 1 つの検索ブロックとして仮定している．下記の構築を通して実際にパターンマッチを実行する上での問題点について考察する．

テキストの解析として係り受け解析は CaboCha を利用して意味役割と概念フレーム付与は ASA を利用している[8]．テキストは文献 [12] のように prolog の述語として関係付けされる．ブロックの組合わせを実現するために blockly を利用した Javascript によるブラウザベースの編集実行環境を構築した [13]．図 1 にシステムの操作画面を示す．Blokly におけるブロックの引数部分が Prolog の引数として機能している．ユーザは基本ブロックを組合わせることでパターンを作ることができる．基本ブロックとして chunk や role が設定されている．画面下部にはブロックから Prolog の検索クエリに変換した結果が表示されている．

テキストを取り込んだ後にこのパターンを適用することで検索結果が表示される．現在は 1000 文を取り込むために 1 分程度時間がかかっている．Prolog ベースのパターンマッチは Python で実装しているが 1000 文程度では検索そのものには時間はほとんどかかっていない．しかしながら大規模なテキストになった場合，テキストの upload 時の解析と検索に大きく時間がかかることが想定される．

4　おわりに

本研究ではパターンマッチシステムに焦点をあてて，既存のコンコーダンサや情報抽出システムで使われているパターンを取り扱う抽象的なパターンの構築と利用について議論した．考察の結果ブロック形式でまとめることで多くのパターンを包含できることが予測される．この考えを具現化する試作モデルとして Prolog ベースの検索システムを構築した．

試作システムは簡易であるがパターンを保存して読み込み，編集して保存することが可能である．こうした機能を拡充することにより，作成したパターンを共有してユーザが多くのパターンを手軽に利用できる環境を構築することが目標である．

8）　https://github.com/Takeuchi-Lab-LM/python_asa

謝辞

本研究は JSPS 科研費 (課題番号 19K00552) と (課題番号 15H03210) および国立国語研究所機関拠点型基幹研究プロジェクト「統語・意味解析コーパスの開発と言語研究」の助成を受けたものである.

参考文献

[1] 大名力. 言語研究のための正規表現によるコーパス検索. ひつじ書房, 2012.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[3] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, Vol. 19, No. 2, pp. 313–330, 1993.

[4] Paul Kingsbury, Martha Palmer, and Mitch Marcus. Adding semantic annotation to the penn treebank. In *Proceedings of the Human Language Technology Conference*, 2002.

[5] Sanae Fujita, Takaaki Tanaka, Fransis Bond, and Hiromi Nakaiwa. An implemented description of japanese: The lexeed dictionary and the hinoki treebank. In *Proceedings of the COLING/ACL06 Interactive Presentation Sessions*, pp. 65–68, 2006.

[6] Stephen Wright Horn, Iku Nagasaki, Alastair Butler, and Kei Yoshimoto. *Annotation Manual for the NPCMJ*. National Institute of Japanese Language and Linguistics, 2019.

[7] Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the th International Conference on Language Resources and Evaluation (LREC 2006)*, pp. 2231–2234, 2006.

[8] A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll. Question analysis: How Watson reads a clue. *IBM Journal Research and Development*, Vol. 56, No. 3/4, pp. 2:1–2:14, 2012.

[9] 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.

[10] Kohsuke Yanai, Misa Sato, Toshihiko Yanase, Kenzo Kurotsuchi, Yuta Koreeda, and Yoshiki Niwa. Struap: A tool for bundling linguistic trees through structure-based abstract pattern. In *Proceedings of the 2017 EMNLP System Demonstrations*, pp. 31–36, 2017.

[11] David Ferrucci and Adam Lally. Uima:an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering 10*, Vol. 10, No. 3/4, pp. 327–348, 2004.

[12] 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, 2021.

[13] 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, 2021.