A Platform for Searching Texts for Desired Expressions in a User-Editable Pattern Matching Environment for Language Learning

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—In this paper we propose a platform of pattern matching system that can extract required phrases or sentences in texts. Finding certain expressions in texts are often needed in language learning, e.g, examples of case markers between a predicate and an argument, or possible nouns in subject of a verb in a certain meaning. In previous studies, several types of systems, containing concordancers, are proposed. However, it is not easy to apply combined patterns because the pattern mathing templates are previously fixed. Thus, we propose a flexible phrase searching system in which users can create search patterns by combining blocks of basic search templates. One of the characteristics the proposed system is that user can also specify where to be highligted in texts with the blocks. To realize the function of combining patterns by users, the proposed system employs Prolog as the base of the data structure. The platform of the searching system is implemented on an Web server with JavaScript-based interface and database system. In the performance test, we shows that the proposed system can deal with relatively large sclale texts (10,000 sentences), and also demonstrate the combined patterns can be applied to the texts. In this paper we discuss the system architecture and the extendability of the pattern matching.**

***Index Terms*—pattern matching, concordancer, browser-based pattern matcher, Prolog**

I. Introduction

Extraction of some phrases and expressions in texts is considered essential function in langugage education. For example, case markers located between a predicate and an object in Japanese language are various rules and alternations[1], then language learners need to search for examples of predicate-argument examples in Japanese texts. As a tool for searching texts, various kinds of concordancer are proposed, however, most concordancers have the liminations in the use of their functionality. For example, Sketch Engine[2], one of the famous concordancers, proposes Corpus Query Language (CQL)[3] that has rich templates of pattern maching for words and characters, however, most of the templates are mainly effective for English and the templates are realized at the level of regular expressions. Thus, users are unable to employ patterns at the level of Context Free Grammar (CFG) which incorporates dependency parsing or predicate-argument relations. From the viewpoint of Natural Language Processing (NLP) research, dependency parsers (KNP, CaboCha, GiNZA) have already been developed and are available, but it is not easy to build a pattern matching system with NLP tools from scratch.

Thus, we propose an environment that users can combine patterns searching for phrases or expressions in texts without programming. The system allows users to combine the basic search templates connected with Prolog predicates that have all of the information about dependency, POS, and lemmas of sentences analyzed by NLP tools. The combination of search templates are consistently composed and the combined search pattern works to find the intended phrases in texts thanks to the the Prolog inference engine. The system has a browser-based interface based on Blockly[4] that allows users to combine patterns in a visual environment.

The contributions of this study are as below: 1) we propose a new type pattern matching system providing the environment in which users can combine patterns, 2) Prolog-based search templates allow users flexibility of pattern combinations, and 3) browser-based system that can be suitable for non-programmers. In this paper we focus on Japanse pattern matching system because most of the concordancers are build for English, and the target of the system is not only for language learners but also for analysing texts to check the studetnt essays.

In the performance test, we shows that the proposed system can deal with relatively large sclale texts (10,000 sentences), and also demonstrate the combined patterns can be applied to the texts. In this paper we discuss the system architecture and the extendability of the pattern matching.

II. Related Studies

One of the famous concordancers sketch Engine. it can seaerch inflections it works effectvenly on English grammars

[1]The verbs *morau* (get) and *eru* (obtain) are almost the same meaning but *morau* has alternation between the case markers *ni/kara* as *kare ni/kara morau* ((I) get (it) from him.), and *eru* can only takes *kara* in this meaning as *kare kara/*ni eru*.

[2]https://www.sketchengine.eu/

[3]https://www.sketchengine.eu/documentation/corpus-querying/

[4]https://developers.google.com/blockly?hl=ja

words, poses can be searched regular expression grammsars. For English. Japease, word base and keyword list and morphmeme. Sketch Englihs is tagged corpus and tag and surface are extarctd.

For Japese text search, for language learner, tregex based searching interface in NPCMJ. they construct Japaennse parsed texts and provide the searchin for syntactical informations based on annotated tags. They are CFG level concordancer, however, the tools are available only for NPCMJ text corpus, and it cannot be applied to plan texts.

Not for education, but extraction tools in NLP, not for language learner, but Chaki is for corpus annotation tool

The is the part of programming, then user is requited to programming skill to use this.

III. Platfrom of Pattern Matching System

A. *Total View of the Proposed System*

user editable pattern with visual handling, Construction of pattern is not easy, the user shoud apply the assumed patterns, see the results and then tune up the patterns. Thus, editing space and results should be exist.

Patterns are combinations, conbining by basic blocks and extract every where matched.

B. *Prolog-Based Data Structure in Backend*

C. *Example of Patterns*

IV. Performance Tests

V. Discussions and Conclusions

References