NPCMJ Interface and Users Manual no.1 (2018-11-30)

Stephen Wright Horn and Alastair Butler (trans. Kei Yoshimoto)

Contents

1 General information

The following is a general introduction to the data in the NINJAL Parsed Corpus of Modern Japanese (NPCMJ) and to the online Interfaces available for accessing that data. The starting point is the Interface gallery page:

http://npcmj.ninjal.ac.jp/interfaces/

Please note that this is a work in progress. There are functionalities in the Interfaces that are not discussed in this manual. Furthermore, there are aspects of the NPCMJ that are not accesible through the Interfaces. This guide is meant to be a basic primer for anyone wishing to search the data in the NPCMJ in general. More functions and techniques will be introduced as time permits.

1.1 About the buttons

Once you enter the interface environment from the Interface gallery page, you can navigate to all the functionalities without returning to the Interface gallery page. The row of buttons at the top on the left side of the pages (links to the Interface gallery page, the Tags page, and the specialized search functions) and the row of buttons at the top on the right side of the pages (Kanji-Kana/Roman alphabet toggle, Japanese/English toggle, and Credits) will always be visible. For each specialized search function, a second row of buttons will appear. The orange buttons provide documentation about features in the search function. Clicking a documentation button toggles the documentation text. Note that any search function can be used from the page where its documentation is displayed. The violet buttons (for the String search and Tree search) are specialized search functions. Once a specialized search function is selected, an orange documentation button for that function will appear in the second row of buttons. Clicking that button sends you to a documentation page describing details about the use of the search function. Again, all search functions can be used as presented, or from windows within their respective documentation pages.

1.2 Acknowledgements

The annotated data includes:

- Articles from the Kahoku Shimpo newspaper (newswire_KAHOKU) with the permission of the KAHOKU SHIMPO PUBLISHING CO.
- Data from the example sentences of "Kiso Nihongo Bunpou — Revised edition" (textbook_kisonihongo) with the permission of the authors, Takasi Masuoka and Yukinori Takubo, and the publisher, Kurosio Syuppan.
- Data from the example sentences of "Basic Japanese: A Grammar and Workbook" and "Intermediate Japanese: A Grammar and Workbook" with the permission of the authors, Takae Tsujioka and Shoko Hamano, and the publisher, Routledge.
- Data from the example sentences of "A dictionary of basic Japanese grammar", "A dictionary of intermediate Japanese grammar", and "A dictionary of advanced Japanese grammar" with the permission of the authors, Seiichi Makino and Michio Tsutsui, and the publisher, The Japan Times.
- Data from the example sentences of "A Dictionary of Japanese Particles" with the permission of the author, Sue A. Kawashima, and the publisher, Kodansha International.

1.3 Guidelines for citation

In general, when providing citation for a particular example, the full text of the example (either as leaves in a tree or as text) plus a general citation (e.g., "NPCMJ") linked to a bibliographic entry in the form provided below is normally sufficient to identify the example. A full ID number is always available, but not strictly necessary. For any example in the corpus accessed through the interfaces, choosing the Context button in the tree view for that example sends you to a page which has complete information for the text at issue: title, date, source, genre, subcorpus, and terms of use.

When presenting research results taken from this corpus, please be sure to include a citation in the following form:

National Institute for Japanese Language and Linguistics (2016).
&ldblquote;NINJAL Parsed Corpus of Modern Japanese.&rdblquote;
(Version 1.0) (http://npcmj.ninjal.ac.jp/interfaces/ accessed YYYY/MM/DD).

(Change the version number and access date as necessary.)

2 Interface guide and users manual

2.1 An invitation to the user

What can you do with a parsed corpus that you couldn't do without? Let's say you wanted to see all the things that horses are depicted as doing in Japanese texts. You want to find all the predicates in relation to which 馬 (or noun phrases formed on 馬) is a subject. Now, along with expressions that are fairly reliably subjects such as 馬が, there are other expressions that may or may not be subjects: 馬は, 馬も, 馬さえ, 馬なら, 馬に, 馬から, 馬の, etc. Furthermore you know that there are sentences such as 「馬が追い手を振りきって、走って、山へ逃げた」 where 馬 is the subject of three verbs, but only appears local to one of them. You know there might be pronouns that refer to previously mentioned horses, and you would like to look at the predicates these are associated with as well. On top of this, there are expressions like 鳴く馬 and しっぽが白い馬, which depict horses as things that whinny or that have white tails, but the status of 馬 as subject is not directly inferable by reference to particle marking or word order alone. With a parsed corpus, you can find all the predicates in relation to which 馬 (or a noun phrase formed on 馬, or a pronoun co-referent with 馬) is a subject by referring to syntactic structure, specifications of grammatical role, and annotation for semantic relations. This will be explained in more detail in section 2.4.5.

Another example: Let's say you are interested in expressions like 笠を手に in sentences such as 「笠を手に出かけた」. This expression is interesting because in the sentence there seems to be a missing verb (like 持って) that mediates the relation between 笠を and 手に. This expression is especially interesting because (unlike expressions such as 「料理を中心に」or 『首都を皮切りに」), the relation between 笠を and 手に cannot be paraphrased as 「A を B として」. With a parsed corpus, you can search for all the (i) subordinate clauses that (ii) are missing predicates and (iii) contain only an accusative object and a postpositional phrase headed by に. You will find these are very rare indeed. Here are a few:

- 地図を手に目的地を探した。
- 共産党は独自候補の擁立を視野に準備を進めている。
- 親交のあった本願寺門主蓮如の留守中に居室に上がりこみ、蓮如の持念仏の阿弥陀如来像を枕に昼寝をした。
- 荒涼とした市街地を前に、学生らは津波被害の大きさを初めて体感した。
- 「奈良・国宝室生寺の仏たち」の開幕を前に、同展の見どころを一足早く紹介する催しが１３日、仙台市中心部で始まります。
- オリンピックを前に、選手たちは泳ぎ込んだ.

These are just a few examples of what you can do with a parsed corpus, but couldn't do possibly with a search engine in a browser or a concordancer. This is made possible by human analysis of the meaning of sentences, and organisation of the data based on a limited number of abstract principles. There are many questions about Japanese grammar that haven't been explored systematically, but are amenable to investigation through a parsed corpus. This is because a parsed corpus analyzes everything, rather than just the things that linguists have already identified as interesting. We invite you to explore and see what you may find.

2.2 String searches

In order to use the corpus effectively, you need to know in general terms what is inside and how it is organized. But one simple way to access information without needing any background information beyond a general knowledge of Japanese is the "Basic string search" interface in the corpus, and its related functions. This is a very powerful tool which has features that simplify searches and make extensive use of wild cards unnecessary (unlike some other search engines). For example, you can easily search for a string that does not actually form a word in Contemporary Japanese, but nonetheless appears in texts as combinations of parts of words. Anyone who knows Japanese can say that the contiguous string 「にはい」does not spell a simplex word. A search that looks for a segment of that precise shape (that is, <wb>にはい<wb>, where "<wb>" indicates a word boundary) will yield nothing. (This search can be done by applying the 'Strict' option for string searches to the search expression にはい.)

But the string にはい does appear in combinations of parts of words (as long as those parts are written in ひらがな). For example, the 'Character' string search option looks for <wb>に‥は‥い<wb> (where "‥" indicates that a word boundary is possible in that position.) This yields

- 「家<wb>に<wb>は<wb>い<wb>ない」 (家には居ない).

The 'Liberal' string search option allows word boundaries to be ignored when searching for a string, so it looks for ‥に‥は‥い‥. Added to the results from the 'Character' search, this option yields examples like these:

- 「簡単<wb>に<wb>は<wb>いか<wb>ない」(簡単には行かない)
- 「エジプト<wb>に<wb>はいろ<wb>う<wb>と<wb>して」(エジプトに入ろうとして)
- 「<wb>わけにはいかない<wb>」
- 「屋根<wb>に<wb>はい上がる」(屋根に這い上がる), etc.

Results can be specified to a certain extent by using the 'Mine' string search option. This allows you to introduce word boundaries into the string by typing a single space (symbolized here with an underscore "_"). For example, entering に_はい under this option looks for ‥に<wb>はい‥, yielding examples with verbs 入る and 這い, but none with verb 居る or 行く.

Results can be specified still further by choosing the 'Strict' option for string searches, which enforces word boundaries at the edges of the string. For example, entering に_はい under this option looks for <wb>に<wb>はい<wb>. This yields no results at present because there are no isolated words 肺, 灰, 杯, or 這い in the corpus

(at least none written in ひらがな) that immediately follow に. Such a result also tells us that 這いあがる is one segment (that is, the whole sequence is analyzed as a single lexical item, rather than as two words). This is useful information: While productive auxiliary verbs following i- and e- combining stems (始める, 終わる, 出す, 止む, 忘れる, etc.) are treated as separate words, there are also sequences of verbs in i- and e-combining stem form in lexical compounds with other verbs: 這い上がる, かき集める, 言いふらす, 撒き散らす, 追いかける, etc. For these latter forms, there is no word boundary between verb 1 and verb 2.

Results can be expanded using the Greedy string search function in combination with the 'Liberal', 'Character', 'Mine' and 'Strict' options. Greedy string search enables you to specify the upper limit of intervening characters that may appear at a word boundary site. For the string search に_はい, using Greedy string search in combination with the 'Character' string search option and setting the intervening character number to a value of 1 will yield examples containing things such as

- 「間には悪い感情」,
- 「基本的に鬼は怖いもの」,
- 「調べるのには向いていない」,
- 「活動には狙いが二つ」,
- 「チリにはない考え方」,
- 「おいしいにはおいしいけれど」, etc.

Setting the intervening character number to a higher value yields even more results. Greedy string search interacts with the string search options in the following ways:

- With the 'Liberal' option for a string A_BC, Greedy string search set at n allows n characters to intervene between A and BC (with no word boundary restrictions for A and for BC).
- With the 'Character' option for a string A_BC, Greedy string search set at n allows n characters to intervene between A and BC (with word boundary restrictions after A and before BC).
- With the 'Mine' option for a string A_BC, Greedy string search set at n allows n characters to intervene between A and BC, (with no word boundary restrictions before A and after BC).
- With the 'Strict' option for a string A_BC, Greedy string search set at n allows n characters to intervene between A and BC, (with word boundary restrictions for A and for BC).

The principles by which Basic string search and Greedy string search operate for Kanji/Kana text do not change when the corpus is accessed in the Roman alphabet.

There are limitations to the string search: (i) the results are limited to a particular orthographic form (e.g., うま, ウマ, and 馬 all give different results); and (ii) searches based on lemmas are not possible (e.g., while expressions あります, あれば, ある, and あった all contain a morpheme with the same lemma, 有る, but to get results for all attestations of the the word in question, the various forms must each be searched for

separately: あり, アリ, 在り, 有り, あれ, アレ, 在れ, 有れ, etc.). These limitations can be overcome to a large extent using Tgrep-lite search (section 2.4).

Accordingly string searches will yield complete results for sentences containing a specific string, but string search is most useful in cases when either i) the user doesn't know how a string is segmented, or ii) the user doesn't know what structural assignments are given to elements in a sequences of words. For example, if you don't happen to know that in the corpus the string ならでは is marked up as a single word, searching for any of なら, らで, では, ならで, らでは, or ならではの using the 'Liberal' option will yield ならでは in the results. If you don't know that the string に向けて is marked up in two different ways, depending on the grammatical functions involved, using the 'Liberal' option with that string will yield the pattern (P-ROLE に向けて) and also the pattern (P-ROLE に) (VB 向け) (P-CONN て). (The significance of the patterns is discussed in section 2.3 below.

In sum, any well-formed string search that has a corresponding pattern in the corpus will yield search results. The default displays show you for each attestated example how the elements defined in the string search are distributed in the structure of that example. Examining the displays is one way to familiarize yourself with the segmentation practices with regard to morphemes in the corpus and with specifics of the structure of the corpus.

2.3 Basic structure

This corpus is primarily a description of texts. The texts are divided into basic units (primarily sentences and sentence fragments). A basic unit is organized on the principle of a tree in which a trunk branches into smaller parts, ending with leaves (the words of the text). Alternatively you can think of a big box with smaller boxes inside, and the smallest boxes contain blocks (the words of the text). Both ways of visualizing are useful. As a mathematical model, both visualizations amount to the same thing, but the simplest way to express a tree structure in electronic text is to use labeled brackets (the boxes within boxes idea), while in writing about tree structures we use the tree metaphor most often, together with some kinship terms (as might be used in a genealogical tree): ancestor::descendant; sibling::sibling; parent::child, etc. The only confusing thing about the tree metaphor is that linguistic trees are written upside down, with the leaves (words) at the bottom.

2.3.1 From leaf to root

First let's consider how trees get built from the bottom up. This discussion is meant to give the user an idea of how a sentence might be analyzed in the NPCMJ. Let's start with a sentence that we know is already in the corpus:

- 鈴木さんの言葉はすがすがしくさえあった

As a piece of data this sentence is first a string of characters. It is possible to search for the whole sentence using the String search interface on-line. As this sentence is unique in the corpus, a properly formed search will yield only one entry in the result. A poorly formed search will yield nothing. Again, at the moment, the sentence above is only a string of characters, while sentences in the corpus have been parsed into segments and organized under a lattice of nodes. In order for String search to find a reflex in the corpus, it has to match strings to segments in some way or other. One way is to conduct a String search with either "Liberal" or "Character" selected.

Either of these will regard each point between characters as a potential segment (word) boundary.

- 鈴‖木‖さ‖ん‖の‖言‖葉‖は‖す‖が‖す‖が‖し‖く‖さ‖え‖あ‖っ‖た

This will return a result of one entry, containing one attestation of each of the segments in the sentence. Another way is to segment the string exactly as it is segmented in the corpus, and then conduct a String search with “Mine” or “Strict” selected as a condition:

- 鈴木さん<wb>の<wb>言葉<wb>は<wb>すがすがしく<wb>さえ<wb>あっ<wb>た

This also will return a result of one entry.

While the data in the corpus is segmented in a specific way, the segmentation should be fairly easy for a user to predict, as it is mostly based on school grammar: The main parts of speech in school grammar (noun, verb, adjective, adverb, pre-noun, particle, auxiliary, etc.) are set apart as separate segments (words) and each is dominated by a pre-terminal node that is labeled with the name of the part-of-speech for that word.

NPR	P-ROLE	N	P-OPTR	ADJI	P-OPTR	VB2	AXD
鈴木さん	の	言葉	は	すがすがしく	さえ	あっ	た

NPR (“noun-proper”) is a subcategory of nouns. The pre-terminal node contains the segment, and this can be expressed in labelled bracket notation in the following way:

(NPR 鈴木さん)

P-ROLE is a sub-category of particles that express grammatical roles:

(P-ROLE の)

N is the label for common nouns:

(N 言葉)

P-OPTR is a subcategory of particles, including what are known as **toritate** particles:

(P-OPTR は)

ADJI is the category of い-adjectives, excluding auxiliary usages such as -たい, -がたい, -やすい, -にくい, and -づらい. While the ADJI すがすがしく in the example above is in the Infinitive inflection, this information is not included in the pre-terminal node label:

(ADJI すがすがしく)

P-OPTR appears after inflected forms as well:

(P-OPTR さえ)

VB2 is a sub-category of verb that follows a core predicate and carries out one or more grammatical functions. VB2s are usually derived from full-fledged verbs, but lose much of their lexical meaning when they appear as VB2s. Here a form of the verb ある appears as a ‘dummy’ to carry inflectional information for the preceding core predicate. Note that the actual form of the verb is a combining stem:

(VB2 あっ)

AXD is the category of auxiliaries for past-tense morphology:

(AXD た)

With the parts of speech established for each word, it is possible to refer to the immediate context in which words appear and infer what kind of structures they occupy. We can try this for the sentence above, segment by segment, working from

left to right. The basic principle for building structure is that heads project phrases (constituents) and combine with other constituents under those phrases. Accordingly, NPR (as a sub-category of noun) will project an NP (noun phrase):

```
(NP (NPR 鈴木さん))
```

This NP is immediately followed by a genitive particle P-ROLE, which is a head projecting a PP. To put it another way, this NP modifies the following P-ROLE as its complement:

```
(PP (NP (NPR 鈴木さん))
    (P-ROLE の))
```

A genitive particle immediately preceding a noun typically marks the complement of that noun. Accordingly the PP is inside of the NP projected by the N which it precedes:

```
(NP (PP (NP (NPR 鈴木さん))
        (P-ROLE の))
    (N 言葉))
```

Again, a following particle P takes the NP as complement, projecting a PP containing that NP:

```
(PP (NP (PP (NP (NPR 鈴木さん))
            (P-ROLE の))
        (N 言葉))
    (P-OPTR は))
```

Noun phrases marked with focus particles typically relate to predicates (according to some grammatical role or other). The segment すがすがしく immediately following は has the part-of-speech of い-adjective (ADJI). This is one of the three basic types of expressions that form core parts of predicates: verbs, adjectives, and copular expressions. In general, "predicate" is an inflecting category that can project clauses. Predicates in Japanese can be formed by a single segment, but they are frequently formed of a string of segments of different types. For now let's assume that すがすがしく projects a clause that contains the immediately preceding PP. Clauses are labelled IP (inflectional phrase).

```
(IP (PP (NP (PP (NP (NPR 鈴木さん))
                (P-ROLE の))
            (N 言葉))
        (P-OPTR は))
    (ADJI すがすがしく))
```

The remaining elements さえ、, あっ, and た can be seen as parts of a predicate that has an ADJI as its core. Specifically, a past tense adjectival predicate すがすがしかった has been broken up so that the core part of it can receive focus from the P-OPTR さえ. The other parts of the sequence (the verbal syntagm) appear at the same level as the core part.

```
(IP (PP (NP (PP (NP (NPR 鈴木さん))
                (P-ROLE の))
            (N 言葉))
        (P-OPTR は))
    (ADJI すがすがしく)
    (P-OPTR さえ)
    (VB2 あっ)
    (AXD た))
```

The resulting structure is a basic constituency tree for a sentence, but when necessary the corpus adds more information about grammatical function in the form of

extensions to the node labels. For example, the PP 鈴木さんの言葉は is the subject of the ADJI すがすがしく, but the particle は doesn't specify that grammatical function, so more information needs to be added: The subject constituent is labelled PP-SBJ. Constituents take on grammatical functions when they combine with other elements, so the specification of function is made at the phrase level where combination of constituents takes place. The subject constituent combines with the predicate under the phrase that the predicate projects, so that is the level under which the labelling for function is added. By contrast, particle の inside of an NP has a function with a limited set of interpretations (possessor::possessed (言葉の音節); whole::part (言葉の殆ど); argument::predicate (言葉の解釈), etc.), and as such doesn't need an extension in that context. Finally, IPs always get an extension. The sentence above is a simple declarative utterance ending in a predicate. We label such sentences IP-MAT (inflectional phrase-matrix).

```
(IP-MAT (PP-SBJ (NP (PP (NP (NPR 鈴木さん))
                        (P-ROLE の))
                    (N 言葉))
                (P-OPTR は))
        (ADJI すがすがしく)
        (P-OPTR さえ)
        (VB2 あっ)
        (AXD た))
```

The above has been an example of the generation of structure for a relatively simple sentence, but the same basic principles of modification and argumenthood can be applied recursively to build very complex structures.

2.3.2 From root to leaf

Now let's consider a sentence from the top down, starting with the root. Each sentence (or sentence fragment) has a root node, which is in turn divided into smaller nodes (normally other complex phrases). Sentence fragments (typically truncated utterances that lack a predicate) have a root node labeled FRAG. Full sentences are labeled depending on the type of utterance they represent. A question is labeled CP-QUE, an exclamation is labeled CP-EXL, and a statement followed by a non-question sentence-final particle or an afterthought is labeled CP-FINAL. Each of these nodes typically contain an IP-SUB node plus some following element (a sentence-final particle or an afterthought). The remaining root node labels (for sentences that don't involve additional elements marking utterance types) are for statements (IP-MAT) and imperatives (IP-IMP).

All full utterances have an IP (inflectional phrase) clause node of some type as a main clause, but there can be other clauses embedded inside these. Subordinate clauses have the label IP-ADV-SCON. Such clauses typically modify predications specifying either manner or condition. Non-final coordinate clauses have the label IP-ADV-CONJ. Within a noun phrase, clauses directly modifying nouns are labeled either IP-REL or IP-EMB. With certain exceptions, an IP node directly contains a predicate of some sort as a head. Predicates are roughly defined as inflecting expressions that say something about a subject. A common predicate type is that of verbs (labeled VB). Directly under (that is "directly dominated by") IP you will often find a node indicating a part-of-speech of the core predicate (e.g., VB) and under the part-of-speech label is the word itself (that is, a segment of text). The word is a leaf node. Incidentally, the other core predicates that head IPs are ADJI (い-adjective), ADJN (な-adjective), and [NP-PRD + AX] (nominal predicate plus copula).

There is an inventory of phrase types that appear as components under IP. Of course, IP-ADV can appear recursively under IP, as suggested above. The other principle phrase types are NP (bare noun phrase); PP (particle phrase); ADVP (adverbial phrase); INTJP (interjectional phrase); FRAG (fragment); PU (punctuation). Excepting PU, any of these phrases may be complex, containing other phrases in addition to a head.

When using brackets to describe a given structure, we may say that "node A directly contains node B" or that "node B is directly contained by node A" but we can describe the same relation as "node A directly dominates node B", or "node B is directly dominated by node A". In a tree, such a relationship is represented by a branch extending from a (higher) node A close to the root to a (lower) node B farther from the root. Trees are connected structures. That is, every element in a tree is related to every other element through branches. Branches express two independent relations: Precedence and direct dominance. These two relations can be combined to define other generalized relations in a tree (e.g., indirect dominance, immediately following sibling, dominated by root, etc.). So you can specify relations between nodes to create search expressions. If, for example, you wanted to find all the questions that are headed by a verb in the form of the combining stem 食べ, you could look for a CP-QUE that directly dominates an IP-SUB that directly dominates a VB that directly dominates the node 食べ.

2.4 Tgrep-lite searches

Tgrep-lite is an adaptation of Tgrep (Pito 1994)– a language for describing relationships in bracketed trees– for use in an on-line interface. Tgrep has provided the foundation for more expressive tree search languages, notably Tgrep2 (Rohde 2005) and Tregex (Levy and Andrew 2006). The search interface also supports XPath search syntax, and that will be referred to from time to time in the explanation that follows.

Tgrep-lite is simple, powerful, and easy to learn. A few quick examples of what Tgrep-lite can do: At the part-of-speech level and the phrase level, nodes are specified by character strings inside square brackets (e.g., [IP-MAT], [NP-SBJ], [AX], etc.). We call a simple expression specifying a node according to the content of the node label a 'node description'. The square brackets in these expressions form regular expression environments. Thus [P] matches any string containing P, including phrase level nodes like IP-MAT, PP-SBJ as well as part-of-speech level nodes like P-ROLE, and NPR, etc. Depending on what you are searching for, it may be necessary to specify a node very clearly. For example, [^IP] matches any label beginning with IP, [SBJ$] matches any label ending with SBJ, etc. Note that a well-formed search expression may not yield any results. For example, [^IP$] matches the exact label IP, but the NPCMJ always adds some kind of functional information to every IP, so this search will not return any results. Inside a regular expression environment, disjunction is available between node descriptions by separating them with "|". Thus, a search term [IP-REL|IP-EMB] will find all nodes that exactly match either gapped adnominal clauses (IP-REL), or ungapped adnominal clauses (IP-EMB). A wildcard (two underscores: "__") matches any node. Relations between nodes are specified using symbols such as direct dominance ("<"), dominance ("<<"), immediate precedence ("."), precedence (".."), etc. The complement of a node description is specified by appending "!" to the beginning of either a simple string (e.g., "[P-OPTR] < !は" finds **toritate** particles with forms other than "は"), or to the beginning of a regular expression that describes that node. Likewise, the complement of a relation can be specified by appending "!" to the front of the symbol for that relation (e.g., "[P-OPTR] !< は" finds **toritate** labels that don't dominate particles with the form "は". Because P-OPTR] always dominates a single **toritate**

particle in the data for this corpus, the results for “[P-OPTR] !< は” are equivalent to those for “[P-OPTR] < !は”).

Conjunctions of relations with respect to a given $node_1$ beginning a search expression (that is, a focus expression) are formed by stating more than one [relation + node] pair to the right of $node_1$. Disjunctions of relations are not directly expressible in Tgrep-lite. However, DegMorgan’s law can be used as a workaround (discussed in section 2.4.4 below).

Unfortunately, Tgrep-lite is not able to do searches on lemmas. For example, if in a search expression you use the dictionary entry form of a verb (for example, the dictionary entry form for ある –viz., “有る”), Tgrep-lite will not return results for matching structures with leaves of any other inflection or orthographic form for that verb (in this case, あり, 有り, ある, あっ, 有っ, あろ, 有ろ, あれ, 有れ, etc.). It is possible to search for words by lemma using a different search language, XPath. To search for the item mentioned above, in the Tree search text box, enter the following expression:

```
//node[@lemma='有る']
```

2.4.1 Preliminaries

A fairly complete description of the relations available in Tgrep-lite and their uses is provided in the sections that follow. But there are a few things to bear in mind when using Tgrep-lite. The first is that Tgrep-lite queries over a database of XML encoded trees by re-writing Tgrep language into XPath language. Every element in NPCMJ’s XML format corresponding to a part of a tree has the element-name `node`. We will call these elements ‘nodes’. Nodes in the XML are distinguished from each other by the attributes they carry. There are two attributes which have values that comprise node descriptions: `@word`, `@cat`. In the trees in the XML format, terminal nodes have `@word` attributes, and all other nodes (viz, part-of-speech nodes and phrasal nodes) have a `@cat` attribute, but no node has both a `@word` attribute and a `@cat` attribute. The form of the search expression in Tgrep-lite determines which type of node is described. The values of `@word` attributes are the segments that make of the leaves of the trees (the terminal nodes). The re-write program interprets simple strings of text (e.g., “馬”) in the search box as the value of a `@word` attribute in the XML format. The exact XPath expression that is used can be seen by clicking the “reveal” button under the search results box and re-submitting the search:

```
//node[self::node[@word='馬']]
```

To find a leaf that contains a particular substring (e.g., “りな”), a regular expression environment is used. The search expression

```
/りな/
```

will find any terminal node that contains a match for the substring.

```
//node[self::node[matches(@word,'りな')]]
```

Results for this search include かぎりなく, 織りなす, 極まりない, 怠りなく, 取りなす, etc.

Note that for nodes with a `@word` attribute there is always an accompanying `@pt` attribute. For `@word` attributes with overt content as values, the `@pt` attribute has the value “word”. But for `@word` attributes with null elements such as `*T*`, `*pro*`, etc., the `@pt` attribute has the value “zero”, e.g.,:

```
//node[self::node[matches(@pt='zero' @word='*T*')]]
```

In a tree the structure looks like this:

(ZERO *T*)

In the online search interface the node descriptions for these special nodes must be put inside of braces “{}”. In Tgrep-lite the @pt value and the @word value are defined as being equal. Accordingly, while the relationship {WORD} == 宇宙人 holds, {WORD} == 宇宙人 does not. However, equality being transitive, the relationship [N] < 宇宙人 also holds.

The label for a part-of-speech node or phrasal node in a tree is the value for a @cat attribute in a non-terminal node in the XML format. A simple string inside of square brackets (e.g., “[WPRO]”) will trigger a search for the value of a @cat attribute (in this case, matching the part-of-speech label for interrogative pronouns):

```
//node[self::node[matches(@cat,'wpro')]]
```

Note that in addition to triggering a search for the value of a @cat attribute, the square brackets create a regular expression environment. Thus in order to search for nodes with the part-of-speech label N (while at the same excluding longer labels that include N such as NPR and NP and PRN), it is necessary to delimit the string inside the regular expression. The symbol “^” indicates the beginning of a string and the symbol “$” indicates the end of a string. Accordingly,

```
[^N$]
```

searches for

```
//node[self::node[matches(@cat,'^n$')]]
```

and finds only one-character long strings composed of N at the part-of-speech or phrase level.

A wildcard (“__”) in a Tgrep-lite search matches any node. This can be used to define terminal nodes (as nodes that do not dominate any other node):

```
__ !< __
```

In the present release there are 338,709 such nodes in the NPCMJ. This is the number of words in the corpus. The roots of trees (20,425 in the present release) can be found in a similar way. This is left as an exercise for the reader.

The “self” relation is specified with “==”. This is useful in a variety of ways. For example, IPs directly under other IPs are always typed with functional information, but such IP-ADVs are further sub-typed. In order to find all those IP-ADVs except for coordinated IP-ADV-CONJs, it is possible to search for a general description and negate a substring in the selfsame node:

```
[IP-ADV-] > [IP] !== [CONJ]
```

Using the node description recognition conditions skillfully requires knowing the set of node labels. The full list is provided in the Appendix (4) below.

Note that the on-line interface is case-sensitive for terminal nodes. Furthermore, certain symbols such as “*” must be escaped with a “¥” inside of a regular expression environment in order to be read by Tgrep-lite. While the search expression “*pro*” is sufficient to find the underspecified null pronoun *pro* in isolation, when paired with other expressions in a regular expression, “/*pro*/” must be used. For part-of-speech nodes and phrasal nodes, the search is case-insensitive (that is, part-of-speech node “[pro]” yields the same result as “[PRO]” as “[pRo]” or “[PrO]”, etc.). These searches will find the PRO (pronoun) part-of-speech tag directly over non-interrogative pronouns (as well as the WPRO tag over interrogative ones). To search for only “PRO” it it necessary to delimit the search (e.g., “[^PRO$]”). Directly under the label in the search results can be found such items as 私, あなたがた, これ, 自分, etc.

When you compose a search expression in the box provided, if the search expression

uses Tgrep-lite syntax, the interface will recognize that and interpret the expression accordingly. If the expression is well-formed but there are no matches in the corpus, the screen shows

There were no results returned.

Note that the same message appears if the search expression is not well-formed.

Again, if you check the "reveal" box in the search result and re-submit, a translation of the Tgrep-lite expression into XPath syntax is displayed. This allows you to check whether the expression you have composed reflects the search that is intended. One final word of caution: In online searches using Tgrep-lite, single spaces must be inserted between node descriptions and relation symbols. Parentheses used for grouping (described in section 2.4.4 can be placed immediately before or after node descriptions.

2.4.2 Describing nodes and constructing expressions

Tgrep-lite expressions begin with a node description. The form depends on whether it is a terminal or non-terminal node. When specifying a terminal node (i.e., a text segment, or string) as the first form in a search expression, the node description can take the form of

- an exact match (with a simple character string (e.g., 馬; 国立国語研究所; etc.),
- a contentful regular expression (e.g., /馬|牛|羊/; /*pro*/; etc.),
- a wildcard for a node that directly dominates no other node (e.g., __ !< __ or / (.*)/ !< /(.*)/),
- a complemented match or regular expression (e.g., !犬; !/^ど/; etc.)

A simple string that has an exact match in the corpus will yield a search result for terminal nodes. The results may be marked up in different ways, depending on whether the form is ambiguous in function or meaning. For example, a search of the form

と

will yield conjunctional particles P-CONN (e.g., 草と木と), grammatical role ("case") particles P-ROLE (e.g., 人と競い合う), complementizers P-COMP (e.g., 「信じます」と言って), and copulas AX (e.g., 観光の拠点となる).

Partial matches can be specified by using regular expressions surrounded by slashes "//". For example, a search of the form

/と/

will yield all segments that contain the character と. A search of the form

/^と/

will yield all segments that begin with the character と. A search of the form

/と$/

will yield all segments that end with the character と. A search of the form

/^(と|ど)/

will yield all segments that begin with either the character と or the character ど.

Note that, because in Tgrep-lite the square brackets "[ ]" have been designated to express a regular expression ranging over values for the attribute @cat, a search expression like /^[とど]/ is not equivalent to /^(と|ど)/ above, but rather is not recognized as well-formed, and so yields no results at all. Also, because the online search

expressions of the interface ultimately form part of a url, symbols with dedicated uses within urls (e.g., “?” and “&”) cannot be used as they would in Java regular expressions. For example, /か(.*?)ら$/ yields no results at all.

When specifying a non-terminal (part-of-speech or phrasal) node as the form that begins a search expression, the node description can take the form of

- an exact match (e.g., [^N$]; [^NP-SBJ$]; [^IP-ADV-CONJ$]; etc.),
- a partial match (e.g., [N]; [VB]; [IP]; etc.),
- a wildcard for a node that directly dominates some other node (e.g., __ < __” or /(.*)/ < /(.*)/).

Note that in contrast to terminal nodes, for non-terminal nodes, a complemented match cannot be written simply with a pre-posed “!” (such as ![^N$]). Instead, a wild card and a negated “self” relation must be used:

```
__ !== [^N$]
```

For phrasal nodes in particular, you may want to specify a word boundary as part of a node description. You can use a hyphen (e.g., [NP-]) to find NPs with extensions of various kinds (e.g., NP-SBJ, NP-OB1, etc.), but if you want to find NPs both with and without extensions in the same search, a search of the form

```
[NP¥b]
```

will yield all forms that contain the string NP with a word boundary at the end of the string, thus excluding a form such as NPR.

You can use the “immediately precedes” relation (“.”) to specify word boundaries between strings. For example, a search expression of the form

```
/に$/ . /^はい/
```

yields the same results as the “Mine” option of the Basic string search for the string に_はい (i.e., 〃に<wb>はい〃).

Note that the search expression for the “immediately follows” relation (viz. “.”) between terminal strings in the opposite order yields the same results.

To search for a particular part-of-speech for と, a relation with another node description must be specified. For example, a search of the form

```
と > [-ROLE]
```

looks for a node with a @word attribute value of と, immediately dominated by a node which has a @cat value containing the string “-ROLE”. The expression finds the comitative case particle (P-ROLE と). Given that the interface is case-insensitive for non-terminal nodes, and that the only node that contains the string -ROLE is the part-of-speech node label P-ROLE, either the search expression と > [p-role] or と > [P-ROLE] or と >> [-ROLE] yields the same results.

The first node in an expression is the “master” node for the expression as a whole, and any following sibling [relation + node] pair (that is, any following [relation + node] pair within the same grouping) indicates a relation with respect to that “master” node. The “master” node gets focus highlighting in the tree view and search results as well. Thus, for “[^NP$] < [^N$]”, the phrasal node NP gets focus.

The first node description in a search expression starts a grouping by default. Furthermore, as [relation + node] pairs refer to the leftmost node in a grouping, a search of the form

```
[^N$] < 馬 > [^NP$]
```

finds Ns that (i) dominate a node equal to the terminal node 馬 and (ii) are also dominated by NP. That is, in generalized form, "A R_1 B R_2 C" is logically equivalent to "(A R_1 B) & (A R_2 C)". (Note: In contrast to Tgrep2 and Tregex, which provide extensions to the Tgrep language, in Tgrep-lite, the conjunction of two relations cannot be expressed using an explicit conjunction symbol "&" between [relation + node] pairs.)

Relations can be grouped with parentheses "()". The first node following a left parenthesis "(" is a master node for the expression contained between that parenthesis and the right parenthesis ")" with which it is paired. Accordingly, a search of the form

```
[^CP-QUE$] < ([^IP-SUB$] < ([^VB$] < 食べ))
```

finds a CP-QUE that directly dominates an IP-SUB that directly dominates a VB that directly dominates 食べ. It bears repeating here that in order to search all forms of the verb 食べる (including たべる and タベル), and not just its combining stem form 食べ, an XPath search "//node[@lemma='食べる']" is required.

2.4.3 Relations between nodes in Tgrep-lite

Various relations between nodes can be defined by specifying aspects of the two basic relations of dominance and precedence. Tgrep-lite has a set of pre-defined relations. All relations are symbolized with a string of one or more characters. Each relation is also paired with its converse relation, which makes it possible to freely choose what to put in leftmost (focus) position. For example, the relation "directly dominates" ("<") is paired with its converse "is directly dominated by" (">"), so [^PP$] < [^NP$] describes exactly the same structure as [^NP$] > [^PP$], but with the focus switched.

Available relationships are as follows:

```
A << B     A dominates (is an ancestor of) B
A >> B     A is dominated by (is a descendant of) B
A < B      A immediately dominates (is the parent of) B
A > B      A is immediately dominated by (is the child of) B
A .. B     A precedes B
A ,, B     A follows B
A . B      A immediately precedes B
A , B      A immediately follows B
A $ B      A is a sister of and not equal to B
A $.. B    A is a sister of and precedes B
A $,, B    A is a sister of and follows B
A $. B     A is a sister of and immediately precedes B
A $, B     A is a sister of and immediately follows B
A $, B     A is a sister of and immediately follows B
A == B     A and B are the same node
A <<, B    B is a leftmost descendant of A
A <<- B    B is a rightmost descendant of A
A >>, B    A is a leftmost descendant of B
A >>- B    A is a rightmost descendant of B
A <1 B     B is the 1st child of A
A >1 B     A is the 1st child of B
A <-1 B    B is the last child of A
A >-1 B    A is the last child of B
A <, B     B is the first child of A (synonymous with A <1 B)
A >, B     A is the first child of B (synonymous with A >1 B)
```

```
A <- B      B is the last child of A (also synonymous with A <-1 B)
A >- B      A is the last child of B (also synonymous with A >-1 B)
A <: B      B is the only child of A
A >: B      A is the only child of B
A <<: B     A dominates B via an unbroken chain (length > 0) of unary branches
A >>: B     A is dominated by B via an unbroken chain (length > 0) of unary branches
```

2.4.4 Notes on more complex relations

A terminal node description can be negated by placing an exclamation mark ("!") immediately before the node description (either a simple string or a regular expression). (Note that non-terminal node descriptions cannot be negated in this way.)
Relations between node descriptions can also be negated by placing an exclamation mark ("!") immediately before the symbol for that relation.

The first node in an expression is the "master" node for the expression as a whole, and any [relation + node] pair appearing to the right refers to the "master" node. Accordingly,

```
A < B < C
```

yields node A which directly dominates both B and C.

Relations can be grouped with parentheses "()". The first node following a left parenthsis "(" is a master node for the expression contained between that parenthesis and the right parenthesis ")" with which it is paired. Accordingly,

```
A < (B < C)
```

returns node A which directly dominates node B, where B directly dominates node C. There is basically no limit to the depth of nesting allowed. For example,

```
A < (B < (C < D))
```

returns node A which directly dominates node B, where B directly dominates node C, where C directly dominates node D.

The first node description in a search expression is a master node by default. Any [relation + node description] pair that is not explicitly grouped with some non-initial master node is related to the first node description in a search expression. Accordingly an open parenthesis before the first node description in a search expression is vacuous, regardless of which node description to the left of which a matching close parenthesis might be placed. That is,

```
A < B < C
```

is equivalent to

```
((A < B) < C)
```

and so on.

Note that, in contrast with Tgrep2 and Tregex, functions associated with square brackets ("[ ]") are not available in Tgrep-lite, so while disjunctions between full node descriptions within a regular expression can be done with "|",

```
[PRO] > [^NP-SBJ$|^NP-OB1$|^NP-OB2$]
```

and disjunctions between partial or full node descriptions can be done within regular expressions with "|",

```
[PRO] > [SBJ|OB1|OB2]
```

disjunctionsi between [relation + node] pairs, etc. are not directly expressible. For example, if we wanted to search for subject NPs headed by the noun "人" (as discussed in an example in section 2.4.5 below) we would want to look for the head of either a

bare subject NP with extension -SBJ or a particle-marked subject, that is, inside a PP with extension -SBJ. Although both phrase types are labeled with -SBJ, they hold different structural relations to the head. Instead of incorporating these two descriptions into a disjunction in one search, with Tgrep-lite it is possible to conduct two separate searches and combine the results.

An indirect way to express disjunction between relations in Tgrep-lite is to use De Morgan's law. Accordingly, to find the head of either a bare subject NP with extension -SBJ or a particle-marked subject inside a PP with extension -SBJ, the following is also possible:

```
人 > ([N] !== (__ !> [NP-SBJ] !> ([NP] > [PP-SBJ])))
```

2.4.5 Instantiations of Tgrep-lite

Given the set of relations and operations available in Tgrep-lite, there are many relationships that can be specified for searching. To begin, consider the trivial case of co-occurrence of node A and node B within a sentence. It is possible to compose a search expression that looks for any root node (that is, a node that is not immediately dominated by any other node) that dominates both A and B. Let A be 船 and B be 海:

```
__ !> __ << 船 << 海
```

the same results can be gotten with 船 as the focus by using the following search:

```
船 >> (__ !> __ << 海)
```

By putting 船 in a regular expression, the search can be broadened to include any node that includes 船 as a character anywhere in the string:

```
/船/ >> (__ !> __ << 海)
```

By using the symbol for disjunction ("|") within a regular expression, any number of node descriptions can be added for searching under the same set of conditions at the same time:

```
/船|魚/ >> (__ !> __ << 海)
```

The relation of co-occurrence within a given context is more interesting when structures and grammatical roles are specified. For example. it is possible to look for any clause that contains (immediately dominates) both a subject and a direct object. Recall that both PPs introducing particle-marked arguments and NPs introducing bare arguments receive extensions that specify grammatical role. The search can be generalized by including only the extension in the node description:

```
[IP] < [SBJ] < [OB1]
```

An equivalent search specifies an IP immediately dominating a subject, where the subject is sibling to a direct object:

```
[IP] < ([SBJ] $ [OB1])
```

The following search, then, focuses many (but not all) of the transitive verbs in the corpus.

```
[VB] > ([IP] < ([SBJ] $ [OB1]))
```

The linear order of the subject and object can be specified using Tgrep-lite relations. The following search looks for a subject that is sibling to and follows an object (a scrambling structure):

```
[SBJ] $,, [OBJ]
```

Equivalently, an object that precedes a sibling subject:

```
[OBJ] $.. [SBJ]
```

Now consider a variation of the first problem set out at the beginning of this Users Guide, substituting 人 for 馬: How to find all the instances where common noun 人 is either the head of a subject NP or instances where an NP with 人 as head co-refers to an empty subject position. Part of the solution can be found with two simple searches: one for bare subjects and the other for particle-marked subjects:

```
[NP-SBJ] < ([^N$] < 人)
```

```
[PP-SBJ] < ([NP] < ([^N$] < 人))
```

We can also look for subject traces (NP-SBJ *T*) inside of relative clauses modifying heads with content 人.

```
[IP-REL] < ([NP-SBJ] < /¥*T¥*/) $. ([N] < 人)
```

The rest of the problem can be partially solved by looking for arguments headed by ([^N$] < 人) in IPs that directly dominate at least one IP that is missing a subject position (that is, looking for cases where NPs headed by 人 are potential antecedents in a Control relationship). The extensions for arguments are -SBJ2, -OB2, -LGS, OB1, and SBJ. Here again we need one search for bare arguments and another for particle-marked ones:

```
[SB|OB|LGS] < ([^N$] < 人) > ([IP] < [IP] !< [SBJ]))
```

```
[SB|OB|LGS] < ([NP] < ([^N$] < 人)) > ([IP] < ([IP] !< [SBJ]))
```

The results of theses searches will contain the desired examples, but those will have to be filtered out by hand. (There is actually an off-line tool called TIGERSearch which can identify every instance where 人-headed NP has a subject role using one simple expression. This is introduced in slightly more detail in section 3.2)

It is possible to specify the ordering of a node with respect to its siblings as counted from either the leftmost or rightmost branch under a dominating phrase. For example, to find sentences beginning with adverbs, you can do the following:

```
[IP] <, [ADVP]
```

Alternatively, to put the focus on the ADVP, you can search for ADVP instances (under IP) that do not follow any sister nodes:

```
[ADVP] !$,, __ > [IP]
```

This has been a cursory introduction to some of the relations available in Tgrep-lite. Further documentation is available in the online documentation:

http://npcmj.ninjal.ac.jp/interfaces/cgi-bin/tree_search.sh?db=npcmj

3 Using the corpus for research

The NPCMJ allows access, through a set of interfaces, to a subset of the data in the **Keyaki Treebank** (Butler, Yoshimoto, Hiyama, Horn, Nagasaki, and Kubota 2017). The subset employs publicly available texts annotated to a high standard. Furthermore, there are many functionalities available through the interface that can be used for research. However there are systematic differences in the way that the information is presented and stored. The data in the NPCMJ is stored in an XML format that includes the results of morphological analysis with mecab-UniDic, and information in the linear ordering of segments. Accordingly, searches using lemma information, and searches with conditions specifying distance between one segment and another (as measured by the number of intervening segments) are executable in the NPCMJ with relative ease. This information is not provided in the data format for the Keyaki Treebank.

The Keyaki Treebank format is designed for ease of annotation. The format is that of bracketed trees. Extra information about lemma and linear order is not included. Furthermore, some of the functional information for constituents is presented offset from those constituents as "empty" phrases (that is, as phrases without textual content). For example, a が-marked subject noun phrase in the Keyaki Treebank will take the following generalized form:

```
(PP (NP (N xxx))
    (P が))
(NP-SBJ *が*)
```

This sort of information is added to the node labels of the NPCMJ in the form of label extensions

```
(PP-SBJ (NP (N xxx))
        (P-CORE が))
```

3.1 Research with off-line tools

One way to count of the number of attestations of a particular search is by downloading in csv format. Each result appears on a single line consisting of the containing sentence, preceded by a number to count the attestation, and the ID of the sentence. The attestation itself is identified between XML-like open and close tags (e.g., "<NP-OB2> ... </NP-OB2>" when the focus of the search is NP-OB2), so that an example ID and the associated sentence may appear multiple times, only with likely altering placements for the tags that isolate each search result. Note that the individual words in the sentence are separated by spaces.

For a simple example of how one might want to manipulate search results data, consider sorting the results according to the length of each sentence (as measured by the number of words in the sentence). One way to do this is to apply the following script:

```
awk -F "¥",¥"" '{ print length($3), $0 }' | sort -n
```

to the file contents. Specifically:

```
$ cat /path/filename | awk -F "¥",¥"" '{ print length($3), $0 }' | sort -n
```

gives character length for each example sentence (identified with $3) and sorts the lines according to the character count.

Another feature of the "comma-separated values" format is that the focus of the search is marked off by a preceding and following XML-like tag, and these can be replaced with separators to provide the focus in its own column, if desired. This can be accomplished thus:

```
$ cat /path/filename | sed 's@<[^/ ]¥+>@",@; s@<¥/[^/ ]¥+>@",@'
```

The downloaded files lend themselves to search and manipulation with various tools. For example, the "comma-separated values" format can be manipulated and imported into a spreadsheet program, or it can be used to generate a ".Tok" file for a Goldvarb analysis

http://albuquerque.bioinformatics.uottawa.ca/GoldVarb/GoldManual.dir/GVManual.html

3.1.1 Case study with offline tools

The following is a brief case study outlining how to download data from the online Interface and analyze it using offline tools. Given direct access to the source files of

trees, a set of data relevant to a particular question can be collected using a general search. Then the data can be analyzed with tools that are more powerful than what can be offered online.

The implementation for doing the case study (and many other operations on data of this type) uses tsurgeon, bash, sed, and gawk code. We recommend using a Linux operating system, or alternatively, on a Windows system you can install the Windows Subsystem for Linux (https://docs.microsoft.com/en-us/windows/wsl/install-win10). With a few more steps, you can take advantage of a variety of tools for manipulating bracketed files for trees. First, clone the directory from the GitHub website by executing the command below:

```
$ git clone https://github.com/ajb129/normalize.git
```

The instructions in the README file direct you to download munge-trees and stanford-tregex.jar. You will need to add the /bin folder from the cloned directory into your PATH, and change the content of " tregex_location" (found in the same directory as the README) to point to your local copy of stanford-tregex.jar.

Now clone the following directory and add the /bin folder to your PATH as well:

```
$ git clone https://github.com/ajb129/keyaki-aid.git
```

These steps enable you to carry out the procedures in the case study below.

Consider sentences with more than one "subject" (i.e., "double nominative sentences" and sentences derivable from these). One example might be 「すなわち駒は1手ごとに性能が変わる」. A survey of these shows that the が-marked subject phrases closest to the predicate are frequently relational nouns. Suppose you wanted to extract a list of triples with lexical content: <head noun of SBJ, head noun of SBJ2, predicate>. Our example above would yield the triple <駒,性能,変わる>. These sentences are usually marked up in the corpus as containing a phrase (PP or NP) marked with an extension -SBJ, followed by a second phrase (PP or NP) marked with an extension -SBJ2. In the interface, you can isolate all the sentences containing SBJ2 by a tree search for [SBJ2]. But if you want to take advantage of syntactic annotation to manipulate results or extract information, you will need access to files in bracket format.

A one-step download of all the data released in the NPCMJ in the form of bracketed trees available through the "Overview" page of the interface or the top page of NPCMJ project :

http://npcmj.ninjal.ac.jp/interfaces/cgi-bin/index.sh?db=npcmj

http://npcmj.ninjal.ac.jp/

.JP http://npcmj.ninjal.ac.jp/interfaces/cgi-bin/index.sh?db=npcmj&lang=jp

There are at present 136 such files presently released in this way. The following assumes that these steps have been taken and that a folder "my_NPCMJ" with all the data (136 files) in bracket form exists.

First, create a dedicated directory (e.g., sandbox) and put the folder my_NPCMJ inside it.

At this point you will need to use some offline tools.

The munge-trees tool at:

http://web.science.mq.edu.au/~mjohnson/Software.htm

The Tgrep2 tool at:

https://tedlab.mit.edu/~dr/Tgrep2/

And the tsurgeon and tregex tools at:

https://nlp.stanford.edu/software/tregex.html

The immediate goal is to create one file containing all and only the trees from each file. When you download the bracket format files they have meta information, which should be left out of the data you want to use. There is a command csearch_collect that will do all this automatically, but first, the extensions of the files in my_NPCMJ must be changed from .txt to .psd. From a terminal inside my_NPCMJ execute the following:

```
$ for f in *.txt; do  mv "$f" "${f%.txt}.psd"; done
```

Now the command can be used to collect the trees and deposit them in a file. From inside the my_NPCMJ folder, execute the following:

```
$ csearch_collect > ../gold.psd
```

Now from the sandbox, execute the following:

```
$ cat gold.psd
```

You will see that now each tree is collapsed onto a single line.

At this point you will need to use Tsurgeon to excise the WORD nodes and ZERO nodes between the terminal nodes and the part-of-speech nodes. These nodes are not employed to carry information in the NPCMJ, but in other corpora that use the same set of tools they carry information about word formation or orthography. In order for the tree_to_tnt command (in the pipeline below) to return terminal string/part-of-speech label pairs, these nodes must be removed. In the same sandbox, create a file change.tsurgeon with the following content:

```
8<8<8<8<8<8<8<8<8<8<8<

/WORD|ZERO/=x

excise x x

8<8<8<8<8<8<8<8<8<8<8<
```

With this file in place, run the following:

```
$ cat gold.psd | tsurgeon_script change.tsurgeon | sponge gold.psd
```

Now collect the trees with node SBJ2 by executing the following:

```
$ cat gold.psd | grep "SBJ2" > SBJ2.psd
```

At this point you will be using Tgrep or Tgrep2 instead of Tgrep-lite, so the syntax of the search language is slightly different. Tsurgeon scripts operate on files in this format. Now it is possible to count the number of attestations of phrases marked with -SBJ2 using the following:

```
$ tregex "/SBJ2/" SBJ2.psd
```

There will be a report after the execution of the command in a form like

```
There were 699 matches in total.
```

Such a file will also include attestations of subject noun phrases that have no lexical content: traces in relative clauses, null pronouns of various types, empty positions that inherit their reference from antecedents higher in the structure, etc. Subjects in the form of traces and null pronouns always appear in the clause as NP under IP, with the null element directly under NP in the general form *form*. Those under SBJ2 can be searched for with the following:

```
$ tregex "/SBJ2/ < /¥*/" SBJ2.psd
```

This search gives 24 matches in total. We can do the same sort of search for null NP-SBJ co-occurring with phrases marked SBJ2:

```
$ tregex "/SBJ/ < /¥*/ $ /SBJ2/ " SBJ2.psd
```

This search gives 294 matches in total. Another search for cases where the subject position is empty:

```
$ tregex "/SBJ2/ !$ /SBJ/ " SBJ2.psd
```

This search gives 151 matches in total. That is, 445 out of 699 instances (about 64%) of clauses with SBJ2 phrases have "major subjects" that get their reference from a non-local source. This in itself is an interesting result.

Incidentally, there are 13 instances in which an SBJ2 phrase doesn't occur with an overt predicate. Doing a search combining all these factors will give you an idea of how many attestations of SBJ2 will be missing from a list of complete triplets with lexical items as members.

```
$ tregex "/SBJ2/ [ < /¥*/ | !$ /ADJ|VB|PRD/ | !$ /SBJ/ | $ (/SBJ/ < /¥*/) ]" SBJ2.psd
```

The above yields 482 matches in total. Accordingly, a list of triplets with overt lexical items will have approximately 699 - 482 = 217 entries.

Now, in order to produce a complex list, it is necessary to put focus on each element. This cannot be done with a simple search. Instead, we have to manipulate the original data, "highlighting" the items we want to extract. Lexical content heading NPs will most often be found under a common noun (N), a proper noun (NPR), or a pronoun (PRO). Predicates will be found under adjectives (ADJI, ADJN), under verbs (VB), or under the heads of nominal predicates (NP-PRD). Attaching a label to each of these will allow us to collect them for display in a list. In the same sandbox, modify your change.tsurgeon file to have the following content:

```
8<8<8<8<8<8<8<8<8<8<8<

/SBJ¥b/ [ < N|NPR|PRO=a | < (NP < N|NPR|PRO=a) ] $ (/SBJ2¥b/ [ < N|NPR|PRO=b | < (NP
< N|NPR|PRO=b) ]) [ $ /ADJ|VB/=c | $ (/PRD/ < __=c) ]

relabel a N_sbj_of_interest
relabel b N_sbj2_of_interest
relabel c predicate_of_interest

8<8<8<8<8<8<8<8<8<8<8<
```

With this file in place, run the following:

```
$ cat SBJ2.psd | tsurgeon_script change.tsurgeon | grep interest
```

Now execute the following:

```
$ cat SBJ2.psd | tsurgeon_script change.tsurgeon | grep interest > interesting_trees.psd
```

Now you can generate the list by running the following:

```
$ cat interesting_trees.psd | tree_to_tnt | awk '/interest/ { print ; next } ; /EOS/
{printf("¥n") } ' > triples.txt
```

The list will have about 113 entries and look something like this:

```
童舞 N_sbj_of_interest
希少価値 N_sbj2_of_interest
高い predicate_of_interest

北部 N_sbj_of_interest
南東側 N_sbj2_of_interest
分かれる predicate_of_interest

皇位継承 N_sbj_of_interest
不安 N_sbj2_of_interest
つきまとう predicate_of_interest

etc.
```

The following extracts the head words of SBJ2s from triples.txt file, providing a fairly representative list of lexical heads of SBJ2 phrases in the corpus:

```
$ cat triples.txt | grep 'sbj2_of' | sort | uniq -c | sort -n | awk '{ print $2 }'
```

Below is the list, including heads from idiomatic items such as 「しよう・仕方がない」, 「ほうがいい」, 「仲が良い」, and a fair proportion of relational nouns:

ステンドグラス, ファッション, あさって, スピード, ハンドル, しよう, テンポ, ほう, 下地, 不安, 予測, 事, 事故, 二極化, 仕方, 仲, 値うち, 備蓄, 入場整理券, 写本, 分析, 割合, 南東側, 参加者, 取り扱い, 問題, 国力, 売り上げ, 夜景, 大手, 大進, 実現性, 寿命, 工事, 市域, 広島, 建造物, 彩色, 復旧, 思い, 性能, 成績, 文化, 文字, 方, 旧石巻医療圏, 格差, 楽譜, 歩み, 民営化方針, 気仙, 無理, 独眼竜政宗, 現存作, 発色, 発達, 着工, 知名度, 経済規模, 者, 背, 能力, 腹, 自宅, 色, 苦闘, 英語, 行き先, 被害, 補修, 計画, 記憶, 話, 説得力, 車, 運用数, 道路, 部分, 関係, 閲覧, 面, 面積, 類似点, 風景, 高齢化, こと, 場合, 希少価値, 歯, 気, 点, 目, 評価, 違い, 可能性, の

This list does not represent a complete extraction of all the relevant items. (The full inventory involves some structures that are more complicated than what is specified here.)

The series of manipulations in this case study is simply meant to illustrate some of the techniques that can be applied.

3.2 Exploiting the corpus further

The language in the corpus has many complex patterns that recur and vary both in structure and lexical content. The tree structures and node descriptions assigned to these patterns (the annotations) are determined according to some simple basic principles about how grammar works in general, and according to a variety of rules of thumb about how Japanese works in particular. The principles behind the annotation practice are described in detail in an Annotation Manual that is available online at the following address:

http://www.compling.jp/keyaki/

The Annotation Manual is designed with annotators in mind, but has a wealth of infomation about the way that patterns in Japanese grammar are expressed in structural terms in the corpus. Note that while the node descriptions employed in the Annotation Manual differ slightly from those used in the Online Interface, with very few exceptions the structures are exactly parallel. Until a Users' Edition tailored specifically for the online Interfaces is produced, any attempt at exhaustive research using the corpus in any form should be informed by the descriptions in the Annotation Manual.

The Annotation Manual explains, for example, how the principle of Control is defined and used to capture non-local dependencies like that between 太郎 and 苦笑し in the sentence 「太郎は今度苦笑しながら頷いた」, where 太郎 is an upstairs argument local to 頷いた but nevertheless functions as the source for the reference of an empty downstairs subject position in the clause headed by 苦笑し. Non-local dependencies are established by other structural configuations as well, for example by leftmost upstairs constituents in Across The Board extraction, by the presence of a trace in a relative clause, by the use of Binding Information linking antecedents to pronouns, and by a general indexing mechanism that specifies where a dislocated constituent is to be interpreted.

These non-local dependencies are rendered searchable by re-writing the corpus data in the TIGERSearch format. This form of the data, together with an explanation of the TIGERSearch tool, will be made public in the near future.

4 Appendix

This section lists all tag labels used.

Part-of-speech tags

QUOT	quote
-LRB-	left bracket
-RRB-	right bracket
PU	punctuation
ADJI	い-adjective
ADJN	な-adjective
ADV	adverb
AX	auxiliary verb (including copula)
AXD	auxiliary verb, past tense
CL	classifier
CONJ	coordinating conjunction
D	determiner
FN	formal noun
FW	foreign word
INTJ	interjection
MD	modal element
N	noun
NEG	negation
NPR	proper noun
NUM	numeral
P	particle
P-COMP	complementizer
P-CONN	conjunctional particle
P-FINAL	final particle
P-OPTR	operator
P-ROLE	role particle
PASS	passive
PNL	prenominal
PRO	pronoun
Q	quantifier
QN	noun with quantifier
SYM	symbol
VB	verb (or verb stem)
VB0	light verb
VB2	secondary verb

WADV	indeterminate adverb
WD	indeterminate determiner
WNUM	indeterminate numeral
WPRO	indeterminate pronoun

Syntactic tags

ADVP	adverb phrase
CONJP	conjunction phrase
CP-EXL	exclamative
CP-FINAL	projection for sentence final particle
CP-QUE	question (direct or indirect)
CP-QUE-ADV	question used adverbially
CP-QUE-OB1	question used as object
CP-QUE-PRD	question used as a nominal predicate
CP-THT	complementizer clause
CP-THT-ADV	quote used adverbially
CP-THT-SBJ	quote used as subject
FRAG	fragment
FRAG-IMP	imperative with fragment
FS	false start
INTJP	interjection phrase
IP-ADV	adverbial clause
IP-ADV-CND	conditional clause
IP-ADV-CONJ	coordinated clause
IP-ADV-SCON	subordinate clause
IP-EMB	gapless noun-modifying clause
IP-IMP	imperative clause
IP-MAT	matrix clause
IP-REL	relative clause
IP-SMC	small clause
IP-SUB	clause under CP* layer
IP-NML	nominalized clause
multi-sentence	multiple sentence
NML	intermediate nominal layer
NP	noun phrase
NP-ADV	adverbial noun phrase
NP-LGS	logical subject noun phrase
NP-LOC	locational noun phrase
NP-MSR	measure noun phrase
NP-OB1	noun phrase first object
NP-OB2	noun phrase second object
NP-POS	possessive noun phrase
NP-PRD	predicate noun phrase
NP-SBJ	noun phrase subject
NP-SBJ2	noun phrase second subject
NP-TMP	temporal noun phrase
NP-TPC	topic noun phrase
NP-VOC	vocative noun phrase
NUMCLP	numeral-classifier phrase
PNLP	prenominal phrase
PP	particle phrase
PP-CND	conditional
PP-CONJ	coordination
PP-LGS	logical subject
PP-LOC	location
PP-MSR	measurement

PP-OB1	first object
PP-OB2	second object
PP-PRD	predicate
PP-SBJ	subject
PP-SBJ2	second subject
PP-SCON	subordination
PP-TMP	temporal
PP-TPC	topic
PP-VOC	vocative
PRN	parenthetical

Tag extensions to specify clause linkage

-CND	conditional
-SCON	subordinate conjunction
-CONJ	coordinate conjunction

Other tags

LS	list item
LST	list
META	meta information

References

Butler, Alastair, Kei Yoshimoto, Shota Hiyama, Stephen Wright Horn, Iku Nagasaki, and Ai Kubota. 2017. The Keyaki Treebank Parsed Corpus, Version 1.0. (http://www.compling.jp/keyaki/ accessed 2017/10/28).

Levy, Roger and Galen Andrew. 2006. Tregex and Tsurgeon: tools for querying and manipulating tree data structure. In **5th International conference on Language Resources and Evaluation**.

Pito, Richard. 1994. tgrepdoc – documentation for tgrep. University of Pennsylvania.

Rohde, Douglas. 2005. TGrep2 User Manual version 1.15. (http://tedlab.mit.edu/~dr/Tgrep2).