別添資料　主要プログラムコード

本調査研究で開発したプログラムのうち、処理において主要なものを抜粋して示す。

表記パターン解析ツール

- app.py

条文生成ツール

- main.js

以下リスト UI モック

- app.js

表記パターン解析ツール app.py

```
import streamlit as st
import spacy
from spacy import displacy
from spacy.matcher import Matcher
import ginza
import pandas as pd

nlp = spacy.load('ja_ginza')
sample_text = "氏名又は名称及び住所並びに法人にあっては、その代表者の氏名"

def divide(doc):
    matcher = Matcher(nlp.vocab)
    mataha = [{"LEMMA":"又", "POS":"CCONJ"}, {"LEMMA":"は", "POS":"ADP"}]
    moshikuha = [{"LEMMA":"若しくは", "POS":"CCONJ"}]
    oyobi = [{"LEMMA":"及び", "POS":"CCONJ"}]
    narabini = [{"LEMMA": {"REGEX": "^並び"}, "POS":"CCONJ"}, {"LEMMA":"に", "POS":"ADP",
"OP":"?"}]

    matcher.add("conj",[mataha, moshikuha, oyobi, narabini])
    matched = matcher(doc)

    ## CCONJ を頼りにして分割位置を取得
    pos_list = [0]
    for _, start, end in matched:
        pos_list.extend([start, end])
    pos_list.append(len(doc))

    ## こんなんでテキストを CCONJ で分割する
    str_list = []
    for i in range(len(pos_list)):
        if i < len(pos_list)-1:
            start = pos_list[i]
            end = pos_list[i+1]
            span = doc[start:end]
            str_list.append(span.text)

    return str_list

def divide_bunsetu(doc):
    bunsetsu_head_list = ginza.bunsetu_head_list(doc)
    words = []
    arcs = []
    for i,chunk in enumerate(ginza.bunsetu_spans(doc)):
        words.append({'text': chunk.text, 'tag': chunk.label_ })
        for token in chunk.lefts:
            arcs.append({
                'start': bunsetsu_head_list.index(token.i),
                'end': i,
                'label': token.dep_,
                'dir': 'left'
            })
```

```
        for token in chunk.rights:
            arcs.append({
                'start': i,
                'end': bunsetsu_head_list.index(token.i),
                'label': token.dep_,
                'dir': 'right'
            })
    return { 'words': words, 'arcs': arcs }

st.set_page_config(page_title="Demo", layout="wide")

with st.form("text_data", clear_on_submit=False):
    text = st.text_area("解析するテキストを入力してください", value=sample_text)
    submit_button = st.form_submit_button(label="解析開始")

if submit_button:
    st.header("解析結果")
    st.write("spacy + ginzaによる解析結果です")

    doc = nlp(text)
    df = pd.DataFrame({
        'text':[t.text for t in doc],
        'lemma_':[t.lemma_ for t in doc],
        'pos_':[t.pos_ for t in doc],
        'tag_':[t.tag_ for t in doc],
        'dep_':[t.dep_ for t in doc],
        'head_text':[t.head.text for t in doc]})
    st.dataframe(df, use_container_width=True)

    st.header("係り受け解析")
    svg = displacy.render(doc, style="dep", jupyter=False)
    # st.image(svg, use_container_width=False)
    st.image(svg, width=None)

    st.header("文節によるトークン分解と係り受け解析")
    st.json(ginza.bunsetu_spans(doc), expanded=False)
    bunsetu_svg = displacy.render(divide_bunsetu(doc), manual=True, style="dep",
jupyter=False)
    st.image(bunsetu_svg, width=None)

    st.divider()

    divided = divide(doc)

    st.header("ここから独自検証")
    st.subheader("法令表記を考慮した文節への分かち書き処理")
    st.text("通常の文節では解釈しづらい点もあり、接続詞を含む法令表記に適した文節への分割処
理の可能性についての検証です")
    st.json(divided)
```

条文生成ツール main.js

```
/let OUTDIV,
    WORDS = [],
    CONNECTORS = {},
    ROOT_LEVEL_AND = 99,
    ROOT_LEVEL_OR = 99,
    LAST_LEVEL_AND = 0;

const textbox = function(trash = false) {
    return `
        <div class="text-item">
            <div class="dots">
            </div>
            <input type="text" class="text-input" placeholder="テキスト" data-kind="item"/>
            <button onclick="add_container(this, ${trash})" type="button">
                <img src="./src/add-folder-svgrepo-com.svg" alt="plus-circle" style="width:
20px;">
            </button>
            ${trash ? `
                <button type="button" onclick="delete_textbox(this)">
                    <img src="./src/trashbox.svg" alt="trashbox">
                </button>
            ` : ''}
        </div>
    `;
};

const container = function(remove = false, trash = false) {
    return `
        <div class="group-box">
            <div class="alignC">
                <div class="text-item">
                    <input type="text" class="text-input" placeholder="先頭テキスト" data-
kind="first"/>
                </div>
                ${textbox(false)}
                ${textbox(false)}
                <div>
                    <button type="button" onclick="add_textbox(this)">
                        <img src="./src/plus-circle.svg" alt="plus-circle">
                    </button>
                </div>
                <div class="text-item">
                    <input type="text" class="text-input" placeholder="末尾テキスト" data-
kind="last"/>
                </div>
            </div>
            <div class="line-r">
                ${remove ? `
                    <div class="justify-end">
                        <button type="button" onclick="revert_to_textbox(this, ${trash})">
                            <img src="./src/xmark-circle-.svg" alt="plus-circle">
```

```
                        </button>
                    </div>
                ` : ''
                }
                <select class="connector_selector">
                    <option value="and">AND</option>
                    <option value="or">OR</option>
                    <option value="concat">継続</option>
                </select>
            </div>
        </div>
    `;
};

const add_textbox = function(e) {
    const cont = document.createElement('div');
    cont.innerHTML = textbox(true);
    e.parentElement.before(cont.children[0]);
};

const add_container = function(e, trash) {
    e.closest('.text-item').innerHTML = container(true, trash);
};

const delete_textbox = function(t) {
    t.closest('div.text-item').remove();
    walk();
};

const revert_to_textbox = function(t, trash) {
    const cont = document.createElement('div');
    cont.innerHTML = textbox(trash);
    t.closest('div.text-item').replaceWith(cont.children[0]);
    walk();
};

const make_text = function() {
    const is_first_item = function(obj) {
        let first = WORDS.filter((o) => (o.group == obj.group) && (o.kind == 'first'))[0];
        let iAmFirstItem = false;
        for (var i = 0, f = false; i < WORDS.length; i ++) {
            if (WORDS[i] == first) {
                f = true;
                continue;
            }
            if (f) {
                if (WORDS[i].group != obj.group) {
                    break;
                } else if (WORDS[i] == obj) {
                    iAmFirstItem = true;
                    break;
                } else {
                    break;
                }
```

```
            }
        }
        return iAmFirstItem;
    };

    const check_first_is_not_empty = function(group) {
        let firstObj = WORDS.filter((o) => (o.group == group) && (o.kind == 'first'))[0];
        let sameLevelFirstIsEmpty = firstObj.value.trim() == '';

        if (! sameLevelFirstIsEmpty) return true;

        let p = 0;
        for (var i = 0; i < WORDS.length; i ++) {
            if (WORDS[i] == firstObj) {
                p = i;
                break;
            }
        }
        for (var i = p - 1; i >= 0; i --) {
            if (WORDS[i].kind == 'first') {
                if (WORDS[i].value.trim()) {
                    return true;
                } else {
                    continue;
                }
            } else {
                return false;
            }
        }
    };

    const get_parent_connector = function(elm) {
        const check_first_group = function(elm) {
            let p = 0;
            for (var i = 0; i < WORDS.length; i ++) {
                if (WORDS[i] == elm) {
                    p = i;
                    break;
                }
            }
            let o = 0;
            let current_group = elm.group;
            for (var j = p - 1; j >= 0; j --) {
                if (WORDS[j].group > current_group) {
                    continue;
                }
                current_group = WORDS[j].group;
                if (WORDS[j].group < elm.group && WORDS[j].level == elm.level) {
                    return false;
                }
                if (!WORDS[j].value) {
                    continue;
                }
                if (WORDS[j].group < elm.group && WORDS[j].level < elm.level) {
```

```
                o = j;
            }
        }
        return WORDS[o];
    };

    let o = check_first_group(elm);
    if (o) {
        return {
            conn: CONNECTORS[o.group],
            level: o.level
        };
    }
    let arr = WORDS.filter((o) => (o.level < elm.level) && (o.group < elm.group));
    let g = 0;
    let e;
    for (var i = 0; i < arr.length; i ++) {
        if (arr[i].group > g) {
            g = arr[i].group;
            e = arr[i];
        }
    }
    return {
        conn: CONNECTORS[g],
        level: e.level
    }
};

const connect_str = function(conn, level) {
    if (conn == 'and') {
        if (ROOT_LEVEL_AND == 99 || level == LAST_LEVEL_AND) {
            return '及び';
        } else {
            return '並びに';
        }
    }
    if (conn == 'or') {
        if (ROOT_LEVEL_OR == 99 || level == ROOT_LEVEL_OR) {
            return '又は';
        } else {
            return '若しくは';
        }
    }
    return '';
};

let out = '';
let e = null;

for (
    var i = 0,
        l = WORDS.length,
        str = '',
        p = null;
```

```
        i < l;
        i ++
    ) {

        p = null;
        e = WORDS[i];
        str = e.value.trim();

        if (!str) {
            before = e;
            continue;
        }

        if (e.kind == 'first') {
            if (out) {
                p = get_parent_connector(e);
                out += connect_str(p.conn, p.level);
                out += str;
                before = e;
                continue;
            } else {
                out += str;
                before = e;
                continue;
            }
        }

        if ( e.kind == 'item' ) {
            if (is_first_item(e)) {
                if (check_first_is_not_empty(e.group)) {
                    out += str;
                    before = e;
                    continue;
                } else {
                    p = get_parent_connector(e);
                    if (out) {
                        out += connect_str(p.conn, p.level);
                    }
                    out += str;
                    before = e;
                    continue;
                }
            } else {
                out += connect_str(CONNECTORS[e.group], e.level);
                out += str;
                before = e;
                continue;
            }
        }

        if (e.kind == 'last') {
            out += str;
            before = e;
            continue;
```

```
        }
    };

    OUTDIV.textContent = out;
};

const walk = function() {
    const count_parent_box = function(e, level = 0) {
        const parent = e.parentElement.closest('div.group-box');
        if (parent) {
            level ++;
            return count_parent_box(parent, level);
        } else {
            return level;
        }
    };

    let walker = document.createTreeWalker(document.querySelector('div.group-box'),
NodeFilter.SHOW_ELEMENT);
    let c, kind, level, value, group;
    WORDS = [];
    CONNECTORS = {};
    ROOT_LEVEL_AND = 99;
    ROOT_LEVEL_OR = 99;

    while (walker.nextNode()) {
        const n = walker.currentNode;
        if (
            (n.nodeName == 'INPUT' && n.classList.contains('text-input')) ||
            (n.nodeName == 'SELECT')
        ) {
            c = n.closest('div.group-box');
            for (var i = 0, elms = document.querySelectorAll('div.group-box'), l =
elms.length; i < l; i ++) {
                if (elms[i] == c) {
                    group = i + 1;
                    break;
                }
            }
            kind = n.dataset.kind;
            level = count_parent_box(n);
            value = n.value;
            if (n.nodeName == 'INPUT' && n.classList.contains('text-input')) {
                WORDS.push({
                    group: group,
                    kind: kind,
                    level: level,
                    value: value.trim()
                });
            }
            if (n.nodeName == 'SELECT') {
                CONNECTORS[group] = value;
                if (value == 'and' && (ROOT_LEVEL_AND > level)) {
                    ROOT_LEVEL_AND = level;
```

```
                }
                if (value == 'or' && (ROOT_LEVEL_OR > level)) {
                    ROOT_LEVEL_OR = level;
                }
                if (value == 'and' && (LAST_LEVEL_AND < level)) {
                    LAST_LEVEL_AND = level;
                }
            }
        }
    }

    make_text();
};

const init = function() {
    document.querySelector('#content-wrapper').innerHTML = container();
    OUTDIV = document.querySelector('#text-out');
    const form = document.querySelector('form');
    form.addEventListener('change', walk);
    form.addEventListener('input', walk);
};

window.addEventListener('DOMContentLoaded', init);
```

以下リスト UI app.js

```
const defines = document.querySelectorAll('.define');
const defineds = document.querySelectorAll('.defined');
const dialog = document.querySelector('#link-dialog');
const dialog02 = document.querySelector('#defineTextDialog');
const closeButton = dialog.querySelector('.close-dialog');
const closeButton02 = dialog02.querySelector('.close-dialog');
const dialogTitle = dialog.querySelector('p');
const dialogTitle02 = dialog02.querySelector('p');
const urlParams = new URLSearchParams(window.location.search);
const jsonFile = urlParams.get('json');

// 以下定義箇所・定義リンク用データ 仮の形式
// ここでいう id は法令標準 XML から導かれる e-gov 法令検索での HTML でつかわれている id のこと

// keyword: 定義語
// definedTextId: 定義文を含む element の id
// displayTextIds: 定義文を表示するときにつかう element の id
// displayExtraElements: 定義文を表示するときに id で引っぱってこれないものを html string で保持
// definedTexts: 定義語がつかわれている element の id

// 定義する部分→以降のデータで出現している箇所へのリンクをダイアログ表示
defines.forEach(span => {
    const icon = document.createElement('img');
    icon.src = './src/Button-popover.svg'; // アイコンの内容
    icon.alt = 'アイコン';
    icon.className = 'icon';
    icon.role = 'button';
    icon.tabIndex = 0;

    span.appendChild(icon);

    icon.addEventListener('click', (event) => {
        const keyword = span.textContent;
        const closestElementWithId = event.target.closest('[id]');
          if (closestElementWithId) {
            const id = closestElementWithId.id;
            console.log("最も近い祖先要素の ID:", id);
          } else {
            console.log("ID を持つ祖先要素が見つかりませんでした");
          }

        const keywordId = closestElementWithId.id;
        //const defineData = DATA.find((obj) => { return obj.keyword == keyword});
        const params = new URLSearchParams({
          keyword: keyword,
          definedTextId: keywordId
        });

                            const data = WORDS_LIST;
                            console.log('読み込まれた JSON データ:', data);
```

```
                            // ここでデータを使用して必要な処理を行います
                            const defineData = data.find((obj) => { return obj.keyword ==
keyword && obj.defineTextId == keywordId});

                            const rect = icon.getBoundingClientRect(); // アイコンの位置を
取得
                            dialog.style.top = `${window.scrollY + rect.top +
rect.height}px`; // アイコンの下に表示
                            dialog.style.left = `${window.scrollX + rect.left}px`; // アイ
コンの左位置に揃える
                            dialogTitle.textContent = span.textContent.trim();

                            // 動的にリンクをつくる
                            const defineLinks = document.querySelector('ul#defineLinks');
                            defineLinks.innerHTML = "";
                            if(defineData.definedTexts.length==0){
                                    const li = document.createElement("li");
                                    li.textContent = "データ範囲での使用箇所がありません";
                                    defineLinks.appendChild(li);
                            }
                            else{
                                    defineData.definedTexts.forEach(link => {
                                    const li = document.createElement("li");
                                    const a = document.createElement("a");
                                    a.href = "#" + link.id;
                                    a.textContent = link.text;
                                    a.onclick = () => {dialog.close()};
                                    li.appendChild(a);
                                    defineLinks.appendChild(li);
                                    });
                            }

                            dialog.showModal();
    });

});

// 定義語→定義されている箇所の表示とリンクをダイアログ表示する
defineds.forEach(span => {
    const icon = document.createElement('img');
    icon.src = './src/Button-popover02.svg'; // アイコンの内容
    icon.alt = 'アイコン';
    icon.className = 'icon';
    icon.role = 'button';
    icon.tabIndex = 0;

    span.appendChild(icon);

    icon.addEventListener('click', (event) => {
        const keyword = span.textContent;
        let closestElementWithId = 0;

        const targetElement = event.target.closest('div');
```

```
        if (targetElement && (targetElement.classList.contains('_div_Subitem1Sentence') ||
targetElement.classList.contains('_div_Subitem2Sentence'))) {
           let currentElement = targetElement
           while (currentElement) {
             if (currentElement.id) {
                closestElementWithId = currentElement;
                break;
             }
             currentElement = currentElement.previousElementSibling;
           }
        } else {
           closestElementWithId = event.target.closest('[id]');
        }

        if (closestElementWithId) {
          const id = closestElementWithId.id;
          console.log("最も近い祖先要素の ID:", id);
        } else {
          console.log("ID を持つ祖先要素が見つかりませんでした");
        }

        const keywordId = closestElementWithId.id;
        const params = new URLSearchParams({
          keyword: keyword,
          defineTextId: keywordId
        });

                                        const data = WORDS_LIST;
                                        console.log('読み込まれた JSON データ:', data);
                                        // ここでデータを使用して必要な処理を行います
                                        const defineData = data.find((obj) => { return
obj.keyword == keyword && obj.definedTexts.find(item => item.id === keywordId);});



                                        const rect = icon.getBoundingClientRect(); // アイコン
の位置を取得
                                        dialog02.style.top = `${window.scrollY + rect.top +
rect.height}px`; // アイコンの下に表示
                                        dialog02.style.left = `${window.scrollX +
rect.left}px`; // アイコンの左位置に揃える
                                        dialogTitle02.textContent = span.textContent.trim();

                                        // 動的に定義箇所のテキストをつくる
                                        const defineTexts =
document.querySelector('div#defineTexts');
                                        defineTexts.innerHTML = "";
                                        defineData.displayTextIds.forEach((id, index) => {
                                        let elm ="";
                                        if(index == 0){
                                                  elm = document.querySelector(`#${id}
div.ChapterTitle`); //章
                                        } else if(index == 1) {
                                                  if(id.includes("Se")) {
```

```
                                                elm =
document.querySelector(`#${id} div.SectionTitle`); //節
                                        } else {
                                                elm =
document.querySelector(`#${id} div._div_ArticleCaption`); //条
                                        }
                                } else if(index == 2) {
                                        if(id.includes("Ss")) {
                                                elm =
document.querySelector(`#${id} div.SubsectionTitle`); //款
                                        } else if(id.includes("Se")) {
                                                elm =
document.querySelector(`#${id} div._div_ArticleCaption`); //条
                                        } else {
                                                elm =
document.querySelector("#"+id);
                                        }
                                } else if(index == 3) {
                                        if(id.includes("Di")) {
                                                elm =
document.querySelector(`#${id} div.DivisionTitle`); //目
                                        } else if(id.includes("Se")) {
                                          if (id.includes("Ss")) {
                                                        elm =
document.querySelector(`#${id} div._div_ArticleCaption`); //条
                                          } else {
                                                        elm =
document.querySelector("#"+id);
                                          }
                                        } else {
                                         elm = document.querySelector("#"+id); //号
                                        }
                                } else if(index == 4) {
                                        if(id.includes("Di")) {
                                                elm =
document.querySelector(`#${id} div._div_ArticleCaption`); //条
                                        } else {
                                         elm = document.querySelector("#"+id); //号
                                        }
                                } else if(index == 5) {
                                        if(id.includes("Di")) {
                                                elm =
document.querySelector("#"+id);
                                        } else {
                                         elm = document.querySelector("#"+id);
                                        }
                                } else {
                                        elm = document.querySelector("#"+id);
                                }

                                const cloned = elm.cloneNode(true);
                                cloned.removeAttribute("id");
```

```
                                        // span を複数含んだ条文センテンスのはじめの部分にリン
クを設定する
                                        const sp = cloned.querySelector("span");
                                        if(sp!=null){
                                        const txt = sp.textContent;
                                        const a = document.createElement("a");
                                        a.href = "#" + id;
                                        a.textContent = txt;
                                        a.onclick = () => {dialog02.close()};
                                        sp.replaceWith(a)
                                        }

                                        defineTexts.appendChild(cloned);
                                });

                                // id で取得できない付加的な html エレメントの処理
                                if(defineData.displayExtraElements) {
                                        defineData.displayExtraElements.forEach(elm => {
                                        defineTexts.insertAdjacentHTML("beforeend", elm);
                                        });
                                }

                                dialog02.showModal();
        });
});

closeButton.addEventListener('click', () => {
        dialog.close();
});

closeButton02.addEventListener('click', () => {
        dialog02.close();
});
```