別添　6

AI 活用技術に関する調査研究の報告

船用品整備における品質管理高度化に向けたAI活用技術に関する調査研究に関する完了報告書

令和7年3月

（株）オフィスＳ．Ｋ．Ｙ

https://www.officesky.co.jp

内容

（株）オフィス S．K．Y

1. 概要

船用品整備の品質管理高度化に向け、適した AI 技術を調査し選定を行い、実際の整備美術指導書（膨張式救命いかだについてのみ）を入力とした試作プログラム開発を通した検証を行った。

2. 調査研究手法および結果

仕様書の(1)(2)については以下の方法により調査および研究を行い得られた結果についてここで述べる。(3)については手法を記し、結果の詳細は次章に述べる。

(1) 新管理システム基本設計支援を行うこと

新管理システムを構築する上での基本設計において下記(2)(3)のヒューマンエラー防止のための AI 技術の利用の観点から助言、提案を全体会議において行うこと

→

新管理システムを構築する上での基本設計において下記(2)(3)のヒューマンエラー防止のための AI 技術の利用の観点から助言提案を 2024 年 8 月 2 日の技術開発委員会にて行った。

また海上技術安全研究所様を訪問および Web 会議を介したヒアリングと提案を行った。具体的な日時については別紙物品等一覧(オフィスＳ．K．Y(4 月～25 年 3 月).xls の各月の作業日誌シートを参照されたい。

(2) 整備記録記載におけるヒューマンエラー調査を行うこと

(ア) 膨張式救命いかだ整備技術指導書（電子データ）を入力とし、自動（最小限の人による入力や出力やファイル操作、実行操作等によりほぼ自動で結果が出力される）処理または半自動処理（人による作業操作も処理内容に含まれる）によって(3)の AI 技術が活用できるか（前段階）調査を行うこと

→

膨張式救命いかだ整備技術指導書（電子データ）を入力とし、自動（最小限の人による入力や出力やファイル操作、実行操作等によりほぼ自動で結果が出力される）処理または半自動処理（人による作業操作も処理内容に含まれる）によって(3)の AI 技術が活用できるか（前段階）調査を行った。その専門的な内容から大自然言語処理用 AI の技術である大規模言語モデル（以下 LLM）だけでは一般的なことしか答えられないため、より論理的で厳密なオントロジー技術を用いることが適しているという結論に達した。

（イ）GMDSS 救命整備指導書については、新管理システムの入力となるエクセルファイルについて、とくに入力時のヒューマンエラー（綴りミスや表現の揺れ）について調査すること

→

GMDSS 救命整備指導書については、新管理システムの入力となるエクセルファイルの一部（検査機の種類）に着目して調査を行った。入力時のヒューマンエラー（綴りミスや表現の揺れ）については自然言語処理 AI のなかでも大規模言語モデル(以下 LLM)を用いた形態素解析（日本語の文章を品詞ごとに借り受けまで自動で分解する手法）が適切であることが分かった。

(3) ヒューマンエラー防止のための AI 技術活用調査研究を行うこと

(2)の前段階調査をへて、適した自然言語処理（NPL）による AI 技術を選択し、膨張式救命いかだについては指導書本文の電子データ、GMDSS 救命整備については DX 化に取り組まれている海上技術安全研究所様より入手した CSV ファイルをそれぞれ入力とし、前者に対してはオントロジーの自動生成の可能性、後者に対してはヒューマンエラー（表現の揺れや綴りミスなど）を吸収して、将来のオントロジー作成のためのデータ（標記）の整理の自動化の可能性の試作プログラム作成を通した検証を行うこと

→GMDSS については Python プログラムを試作し、誤記や表現の揺れを吸収し、LLM による事前学習済みの形態素解析ライブラリを用いて、読みを見出し語とする辞書を作成するコードにより、例えば EPIRB 作用用.csv の仕様測定器の項目にある 5 万件以上のヒューマンエラー（表現の揺れや綴りミスなど）を含む単語群が 90 語程度の類似単語として分類され、最終的には 65 個にまで絞られた。詳細は 3.1 節で述べる。

膨張式救命いかだについては、200 ページ以上ある救命いかだ整備記録技術指導書の本文および参考資料の電子データから単語を LLM で事前学習した形態素分解ライブラリで品詞付きで自動抽出し、オントロジーを構成するクラス、個体、関係を自動生成する Python プログラム作成した。詳細については 3.2 節で述べる。

（株）オフィス S. K. Y

3. 調査研究結果((3)について)

この章では前章の(2)の調査結果を受け(3)の手法の詳細とその結果について述べる。

3.1 GMDSS 救命整備指導書にかかる検査装置の入力データのヒューマンエラー吸収プログラムの試作

海上技術安全研究所様から、GMDSS 救命設備の整備システム DX 化のための入力データとして、EPIRB 作業用.csv と START 作業用.csv および TW.csv をいただいた。今回はヒューマンエラーのターゲットとして最も単純な「仕様測定器１」の列のみを 3 つの csv についてそれぞれ選び、その列における表現の揺れやご記入を選んだ。図 1 に EPIRB 作業用.csv の一部を示す。

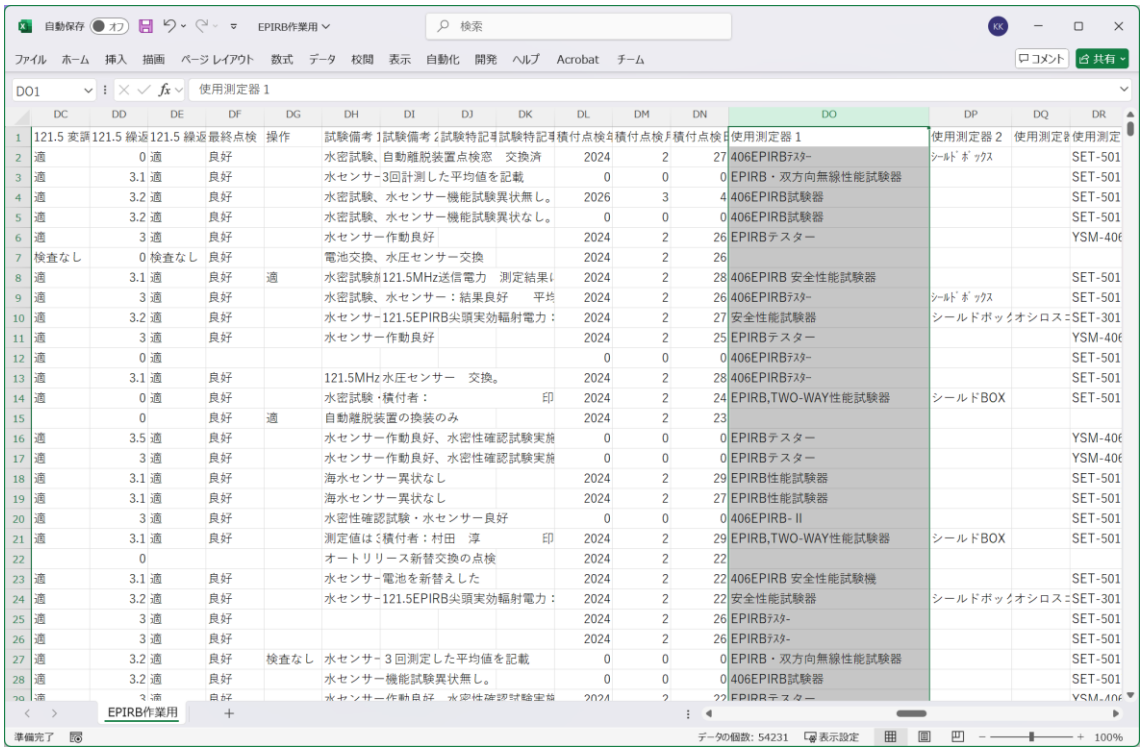
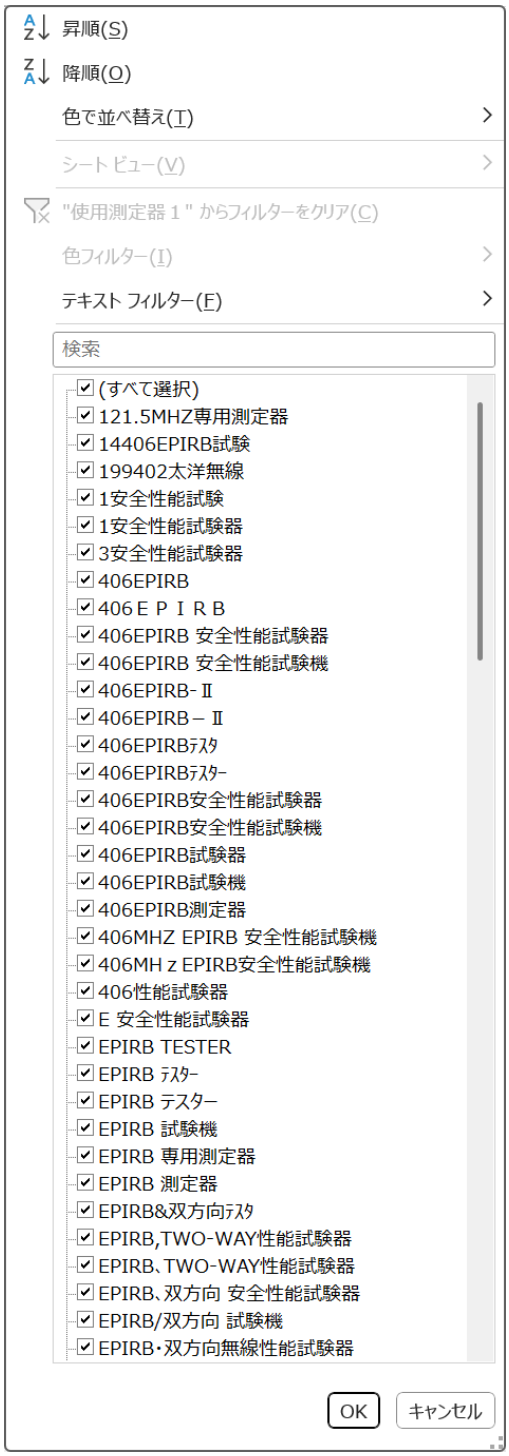
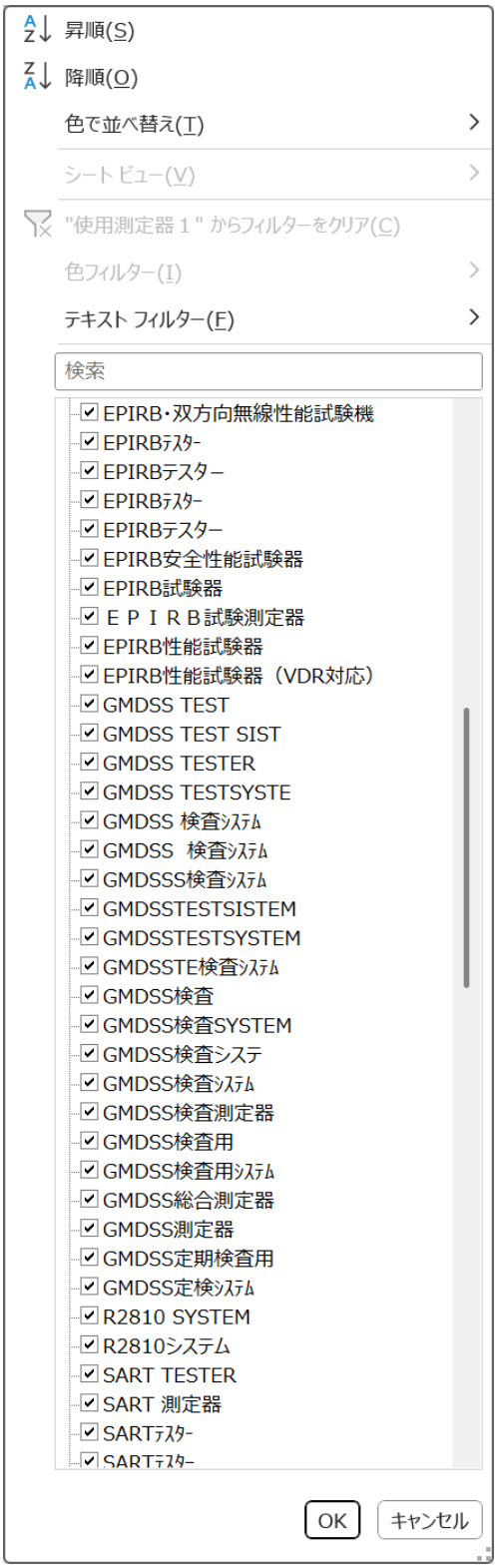


図 1 ヒューマンエラーのサンプル　EPIRB 作業用.csv の「仕様測定器１」列

EPRIB では(有効なデータは)54231 個、SART では 65972 個、TW（双方向）では 75963 個となる。これをエクセルに読込み選択してフィルタをかけると、図 2 に示す様に 128 個になるが、一部は半角や全角、同音異語、伸ばし文字（―、ー、など）の有無が含まれている。結論としてはこれが約半数の 65 語までになるようなプログラムを作成した（SART では 136 から 61 語、TW では 213 から 117 語)。以下にこの方法について具体的に述べる。


昇順(S)
降順(O)
色で並べ替え(T)
シート ビュー(V)
"使用測定器 1 " からフィルターをクリア(C)
色フィルター(I)
テキスト フィルター(F)
検索
(すべて選択)
121.5MHZ専用測定器
14406EPIRB試験
199402太洋無線
1安全性能試験
1安全性能試験器
3安全性能試験器
406EPIRB
406ＥＰＩＲＢ
406EPIRB 安全性能試験器
406EPIRB 安全性能試験機
406EPIRB-Ⅱ
406EPIRB－Ⅱ
406EPIRBﾃｽﾀ
406EPIRBﾃｽﾀｰ
406EPIRB安全性能試験器
406EPIRB安全性能試験機
406EPIRB試験器
406EPIRB試験機
406EPIRB測定器
406MHZ EPIRB 安全性能試験機
406MH z EPIRB安全性能試験機
406性能試験器
E 安全性能試験器
EPIRB TESTER
EPIRB ﾃｽﾀｰ
EPIRB テスター
EPIRB 試験機
EPIRB 専用測定器
EPIRB 測定器
EPIRB&双方向ﾃｽﾀ
EPIRB,TWO-WAY性能試験器
EPIRB､TWO-WAY性能試験器
EPIRB､双方向 安全性能試験器
EPIRB/双方向 試験機
EPIRB･双方向無線性能試験器
OK
キャンセル


昇順(S)
降順(O)
色で並べ替え(T)
シート ビュー(V)
"使用測定器 1 " からフィルターをクリア(C)
色フィルター(I)
テキスト フィルター(F)
検索
EPIRB･双方向無線性能試験機
EPIRBﾃｽﾀｰ
EPIRBテスター
EPIRBﾃｽﾀｰ
EPIRBテスター
EPIRB安全性能試験器
EPIRB試験器
ＥＰＩＲＢ試験測定器
EPIRB性能試験器
EPIRB性能試験器（VDR対応）
GMDSS TEST
GMDSS TEST SIST
GMDSS TESTER
GMDSS TESTSYSTE
GMDSS 検査ｼｽﾃﾑ
GMDSS 検査ｼｽﾃﾑ
GMDSSS検査ｼｽﾃﾑ
GMDSSTESTSISTEM
GMDSSTESTSYSTEM
GMDSSTE検査ｼｽﾃﾑ
GMDSS検査
GMDSS検査SYSTEM
GMDSS検査システ
GMDSS検査ｼｽﾃﾑ
GMDSS検査測定器
GMDSS検査用
GMDSS検査用ｼｽﾃﾑ
GMDSS総合測定器
GMDSS測定器
GMDSS定期検査用
GMDSS定検ｼｽﾃﾑ
R2810 SYSTEM
R2810システム
SART TESTER
SART 測定器
SARTﾃｽﾀｰ
SARTﾃｽﾀｰ
OK
キャンセル

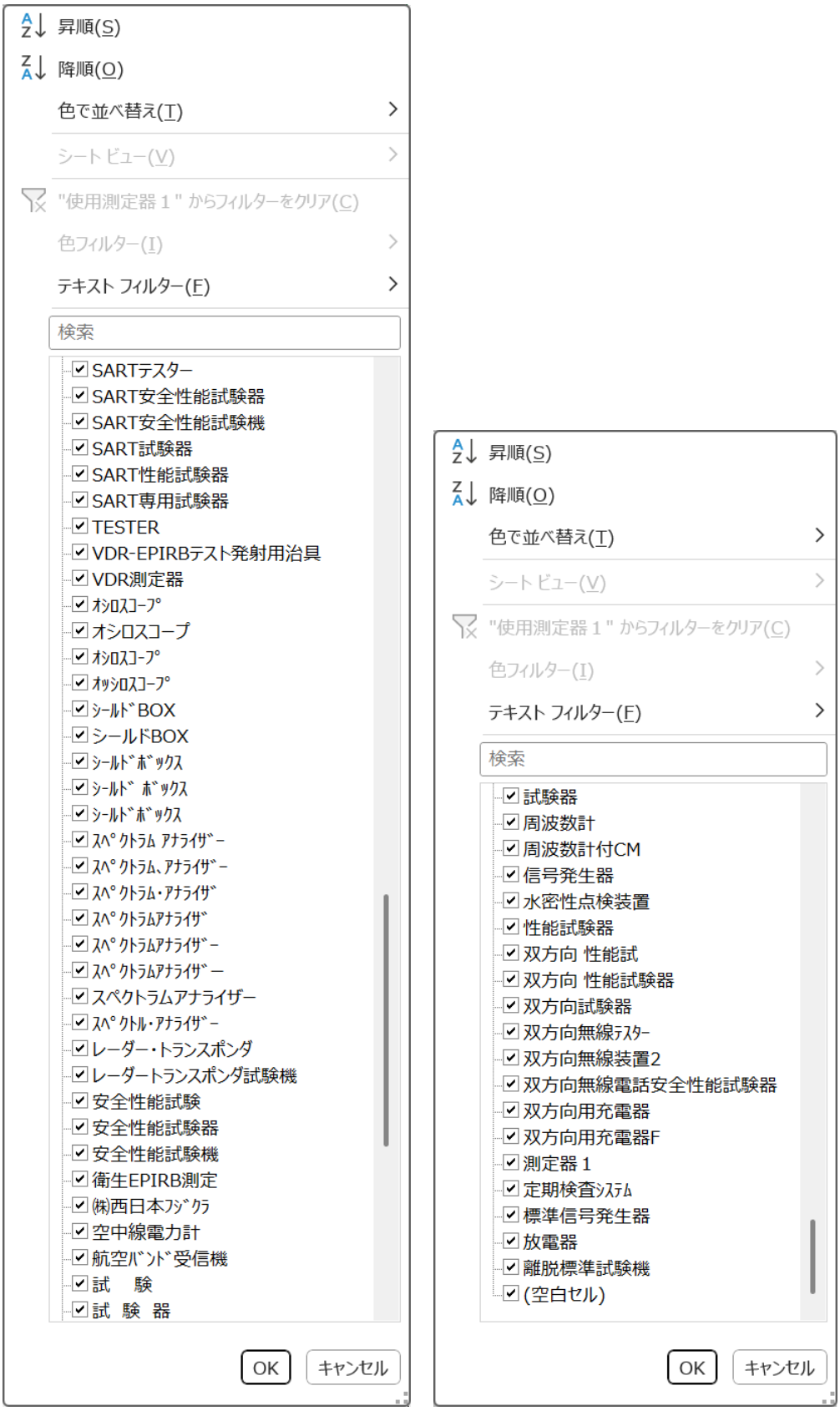


図 2 EPIRB 作業用.xls で「使用測定器 1」列を選択してフィルタをかけた結果

Excel の機能で、ピボットを使った整理すると、「電池種類」の列では「LITHIUM LITHUM RITHIUM ｱﾙｶﾘ アルカリ ﾘﾁｳ ﾘﾁｳﾑ リチウム リチュウム」のように表現の揺れが分かりやすい。ここでの例（EPIRB から）では「使用測定器１」は GMDSS 関連だけ抜粋しても「GMDSS TEST GMDSS TEST SIST GMDSS TESTER GMDSS TESTSYSTE GMDSS 検査ｼｽﾃﾑ GMDSSTESTSISTEM GMDSSTESTSYSTEM GMDSS 検査 GMDSS 検査 SISTEM GMDSS 検査システ GMDSS 検査ｼｽﾃﾑ GMDSS 検査測定器 GMDSS 検査用 GMDSS 検査用ｼｽﾃﾑ GMDSS 総合測定器 GMDSS 測定器 GMDSS 定期検査用 GMDSS 定検ｼｽﾃﾑ GMSDSS 検査ｼｽﾃﾑ GUDSS 検査ｼｽﾃﾑ」となり、誤記の吸収だけでなく、互いの包含関係[1]についても抽出することを考える。

各行のそれぞれから抽出したテキストファイルを入力としたアルゴリズムとして、思いつくままに個別ルール（(全角⇔半角、区切り文字（、スペース、ー、…）や同類語（計測器、測定器、テスターなど...※）でコーディングすることも考慮したが、なるべく汎用的でシステマティックなアプローチをとるために、観察結果からの考察として①機械的(syntactic)に変換できるものと②意味を考えた階層構造的な分類が必要なもの(semantic)なものの二つに分けられる。前者は同値類として分類列挙したあとにルール（※）で変換が可能であるが、後者はオントロジー自動生成の範疇となる[2]。まずは確実に結果が出ると予想される①のアプローチをとった。事前に LLM や Wikipedia（プログラムからは Wikidata ライブラリ）の可能性を調べた結果自動処理で使えるレベルに達してないこと[3]から実際に専門家のヒアリングを必要とすると判断し後回しとした。

①Syntactic ルールとして以下のようなものが考えられる。

①-1 半角/全角→同類

①-2 区切り文字で分割（半角/全角スペース、句読点(半角/全角コンマ、・、)、半角/全角ハイフン、英語の大文字/小文字

①-3 英語訳のブレ（例：リチウム、リチゥム→これは①-2 に分類できない；カタカナの大文字/小文字)、カタカナ/英語（例：リチウム/LITHIUM)

①-4 変換ミス（一定のルールがあるはず)

なお、①に一見近いが①に分類できないものとして、② -1 つづり誤記（例 Rithium これはリチウムの英語としては間違い)、他にも日本語でも散見される。この解決策として、LLM で学習済みの形態素分解（品詞に分解）ライブラリを用いて**『読み』が同じものを同値類（同じものとして分類）とする辞書**（見出しに対して複数の要素からなるリストを対応させるデータ構造）で実現することとした。（結果的には半数に絞り込むことができた一番の要因はこれである。）

---

[1] これは次節のオントロジーに繋がる

[2] ただし今回は入力の分量としてははるかに多いいかだ整備指導書のみを対象とした（次節）ため含まれない

[3] 人が対話的に使う上ではまだよいが、今回はプログラムをとおした自動処理に耐えるレベルではない

②についてはいくつかのライブラリを調査したが結果的には使わなかった[4]。

それぞれの処理結果を図 3～8 に示す。見方として最初に代表としての見出し語が（番号の後の）「同義語：」のあとで、その後ろの「->集合：」の後が同じとみなされた各入力語である(図 3,5,7)。そのあとの図 4,6,8 は同値類として絞り込まれた用語同士の包含関係で、「->」の手前が後者の集合に含まれることを示している。

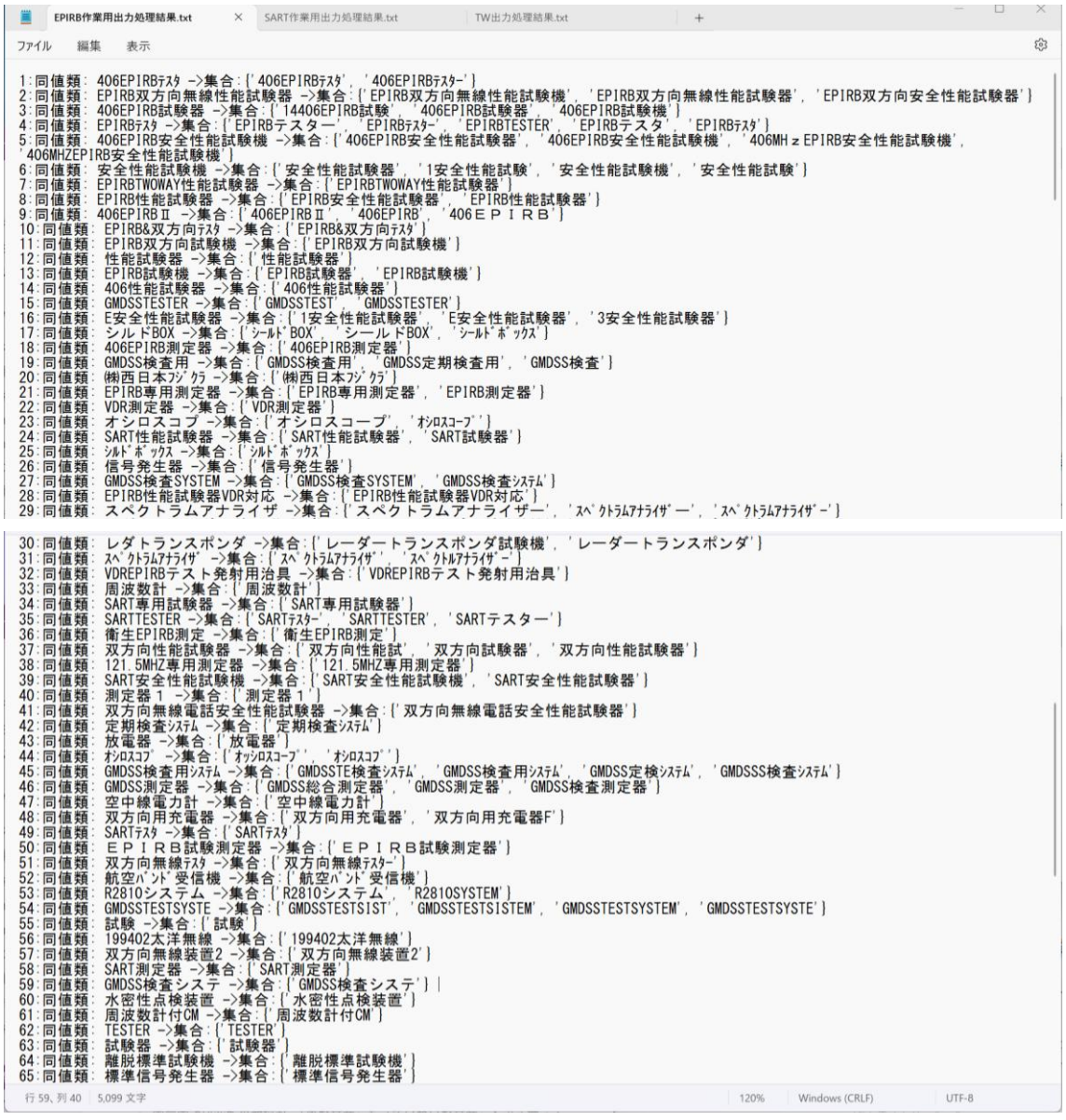
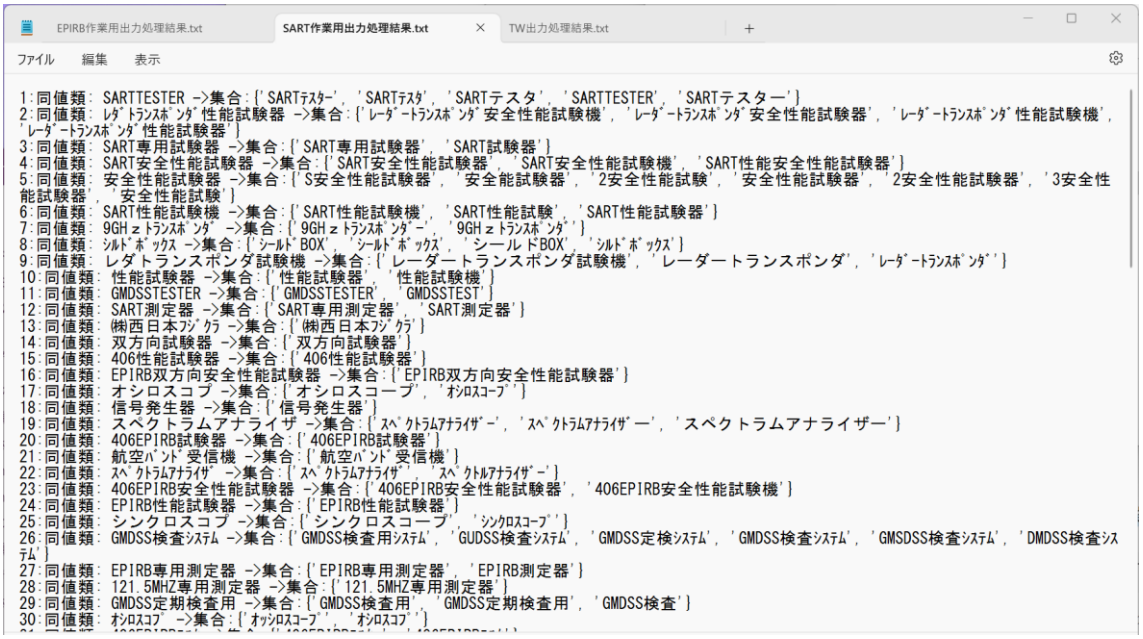


図 3　EPRIB で同じとみなされた 65 語

[4] オートコレクト機能など、変換ミスについては結局形態素分解を用いることが多いため上記の LLM で学習済みの形態素分解ライブラリに吸収された。

```
1:包含関係:406EPIRBﾃｽﾀ ->集合:['EPIRBﾃｽﾀ']
2:包含関係:EPIRB双方向無線性能試験器 ->集合:['試験器', '性能試験器', '試験']
3:包含関係:406EPIRB試験器 ->集合:['試験器', '試験']
4:包含関係:EPIRBﾃｽﾀ ->集合:['406EPIRBﾃｽﾀ']
5:包含関係:406EPIRB安全性能試験機 ->集合:['安全性能試験機', '試験']
6:包含関係:安全性能試験機 ->集合:['SART安全性能試験機', '406EPIRB安全性能試験機', '試験']
7:包含関係:EPIRBTWOWAY性能試験器 ->集合:['試験器', '性能試験器', '試験']
8:包含関係:EPIRB性能試験器 ->集合:['EPIRB性能試験器VDR対応', '試験器', '性能試験器', '試験']
9:包含関係:EPIRB双方向試験機 ->集合:['試験']
10:包含関係:性能試験器 ->集合:['406性能試験器', 'EPIRB双方向無線性能試験器', '双方向無線電話安全性能試験器', '試験器', 'EPIRBTWOWAY
性能試験器', '双方向性能試験器', 'EPIRB性能試験器', 'SART性能試験器', 'EPIRB性能試験器VDR対応', 'E安全性能試験器', '試験']
11:包含関係:EPIRB試験機 ->集合:['試験']
12:包含関係:406性能試験器 ->集合:['試験器', '性能試験器', '試験']
13:包含関係:GMDSSTESTER ->集合:['TESTER']
14:包含関係:E安全性能試験器 ->集合:['試験器', '性能試験器', '試験']
15:包含関係:GMDSS検査用 ->集合:['GMDSS検査用ｼｽﾃﾑ']
16:包含関係:SART性能試験器 ->集合:['試験器', '性能試験器', '試験']
17:包含関係:信号発生器 ->集合:['標準信号発生器']
18:包含関係:EPIRB性能試験器VDR対応 ->集合:['試験', '試験器', '性能試験器', 'EPIRB性能試験器']
19:包含関係:周波数計 ->集合:['周波数計付CM']
20:包含関係:SART専用試験器 ->集合:['試験器', '試験']
21:包含関係:SARTTESTER ->集合:['TESTER']
22:包含関係:双方向性能試験器 ->集合:['試験器', '性能試験器', '試験']
23:包含関係:SART安全性能試験機 ->集合:['安全性能試験機', '試験']
24:包含関係:双方向無線電話安全性能試験器 ->集合:['試験器', '性能試験器', '試験']
25:包含関係:GMDSS検査用ｼｽﾃﾑ ->集合:['GMDSS検査用']
26:包含関係:ＥＰＩＲＢ試験測定器 ->集合:['試験']
27:包含関係:試験 ->集合:['離脱標準試験機', 'EPIRB性能試験器VDR対応', '406EPIRB安全性能試験機', '安全性能試験機', 'EPIRB双方向試験
機', '406性能試験器', '双方向無線電話安全性能試験器', '406EPIRB試験器', 'SART専用試験器', '試験器', '性能試験器', 'ＥＰＩＲＢ試験測
定器', 'EPIRB双方向無線性能試験器', 'SART性能試験器', 'SART安全性能試験機', 'EPIRBTWOWAY性能試験器', '双方向性能試験器', 'EPIRB性能
試験器', 'EPIRB試験機', 'E安全性能試験器']
28:包含関係:周波数計付CM ->集合:['周波数計']
29:包含関係:TESTER ->集合:['SARTTESTER', 'GMDSSTESTER']
30:包含関係:試験器 ->集合:['406性能試験器', 'EPIRB双方向無線性能試験器', '双方向無線電話安全性能試験器', '406EPIRB試験器', 'SART専
用試験器', 'EPIRBTWOWAY性能試験器', '双方向性能試験器', 'EPIRB性能試験器', 'SART性能試験器', 'EPIRB性能試験器VDR対応', 'E安全性能試
験器', '性能試験器', '試験']
31:包含関係:離脱標準試験機 ->集合:['試験']
32:包含関係:標準信号発生器 ->集合:['信号発生器']
```

図 4 図 3 の結果からさらに導かれた互いの包含関係(EPIRB)



EPIRB作業用出力処理結果.txt　SART作業用出力処理結果.txt　TW出力処理結果.txt

ファイル　編集　表示

```
1:同値類: SARTTESTER ->集合:['SARTﾃｽﾀｰ', 'SARTﾃｽﾀ', 'SARTテスタ', 'SARTTESTER', 'SARTテスター']
2:同値類: ﾚﾀﾞﾄﾗﾝｽﾎﾟﾝﾀﾞ性能試験器 ->集合:['ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ安全性能試験機', 'ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ安全性能試験器', 'ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ性能試験機',
'ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ性能試験器']
3:同値類: SART専用試験器 ->集合:['SART専用試験器', 'SART試験器']
4:同値類: SART安全性能試験器 ->集合:['SART安全性能試験器', 'SART安全性能試験機', 'SART性能安全性能試験器']
5:同値類: 安全性能試験器 ->集合:['S安全性能試験器', '安全能試験器', '2安全性能試験', '安全性能試験器', '2安全性能試験器', '3安全性
能試験器', '安全性能試験']
6:同値類: SART性能試験機 ->集合:['SART性能試験機', 'SART性能試験', 'SART性能試験器']
7:同値類: 9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞｰ ->集合:['9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞｰ', '9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞ']
8:同値類: ｼﾙﾄﾞﾎﾞｯｸｽ ->集合:['ｼｰﾙﾄﾞBOX', 'ｼｰﾙﾄﾞﾎﾞｯｸｽ', 'シールドBOX', 'ｼﾙﾄﾞﾎﾞｯｸｽ']
9:同値類: レダトランスポンダ試験機 ->集合:['レーダートランスポンダ試験機', 'レーダートランスポンダ', 'ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ']
10:同値類: 性能試験器 ->集合:['性能試験器', '性能試験機']
11:同値類: GMDSSTESTER ->集合:['GMDSSTESTER', 'GMDSSTEST']
12:同値類: SART測定器 ->集合:['SART専用測定器', 'SART測定器']
13:同値類: (株)西日本ｼﾞｸﾗ ->集合:['(株)西日本ｼﾞｸﾗ']
14:同値類: 双方向試験器 ->集合:['双方向試験器']
15:同値類: 406性能試験器 ->集合:['406性能試験器']
16:同値類: EPIRB双方向安全性能試験器 ->集合:['EPIRB双方向安全性能試験器']
17:同値類: オシロスコプ ->集合:['オシロスコープ', 'ｵｼﾛｽｺｰﾌﾟ']
18:同値類: 信号発生器 ->集合:['信号発生器']
19:同値類: スペクトラムアナライザ ->集合:['ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞｰ', 'ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞｰ', 'スペクトラムアナライザー']
20:同値類: 406EPIRB試験器 ->集合:['406EPIRB試験器']
21:同値類: 航空ﾊﾞﾝﾄﾞ受信機 ->集合:['航空ﾊﾞﾝﾄﾞ受信機']
22:同値類: ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞ ->集合:['ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞｰ', 'ｽﾍﾟｸﾄﾙｱﾅﾗｲｻﾞｰ']
23:同値類: 406EPIRB安全性能試験器 ->集合:['406EPIRB安全性能試験器', '406EPIRB安全性能試験機']
24:同値類: EPIRB性能試験器 ->集合:['EPIRB性能試験器']
25:同値類: シンクロスコプ ->集合:['シンクロスコープ', 'ｼﾝｸﾛｽｺｰﾌﾟ']
26:同値類: GMDSS検査ｼｽﾃﾑ ->集合:['GMDSS検査用ｼｽﾃﾑ', 'GUDSS検査ｼｽﾃﾑ', 'GMDSS定検ｼｽﾃﾑ', 'GMDSS検査ｼｽﾃﾑ', 'GMSDSS検査ｼｽﾃﾑ', 'DMDSS検査ｼｽ
ﾃﾑ']
27:同値類: EPIRB専用測定器 ->集合:['EPIRB専用測定器', 'EPIRB測定器']
28:同値類: 121.5MHZ専用測定器 ->集合:['121.5MHZ専用測定器']
29:同値類: GMDSS定期検査用 ->集合:['GMDSS検査用', 'GMDSS定期検査用', 'GMDSS検査']
30:同値類: ｵｼﾛｽｺﾌﾟ ->集合:['ｵｯｼﾛｽｺｰﾌﾟ', 'ｵｼﾛｽｺﾌﾟ']
```

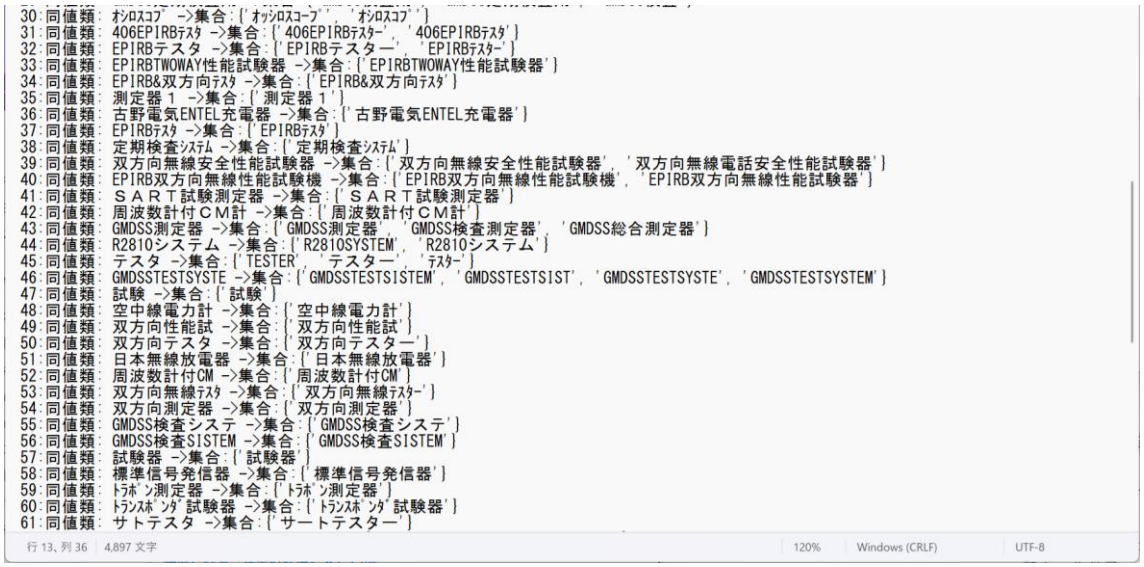


図 5　SART で同じとみなされた 61 語

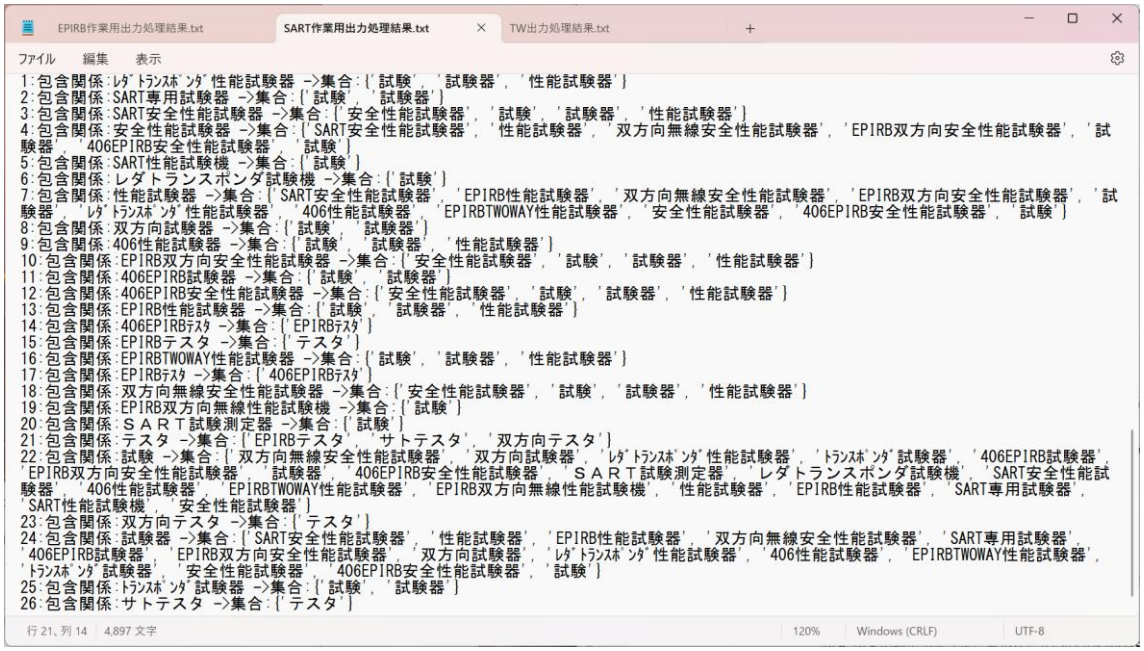


図 6 図 5 の結果からさらに導かれた互いの包含関係(SART)

EPIRB作業用出力処理結果.txt　SART作業用出力処理結果.txt　TW出力処理結果.txt

ファイル　編集　表示

```
1:同値類: 周波数計付ＣＭ計 ->集合:{'周波数計付ＣＭ計'}
2:同値類: 双方向無線ﾃｽﾀ ->集合:{'双方向無線ﾃｽﾀｰ', '双方向無線ﾃｽﾀ'}
3:同値類: EPIRB双方向無線性能試験器 ->集合:{'EPIRB双方向無線性能試験機', 'EPIRB双方向無線性能試験器', 'EPIRB双方向安全性能試験器'}
4:同値類: ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞ ->集合:{'ｽﾍﾟｸﾄﾙｱﾅﾗｲｻﾞｰ', 'ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞｰ', 'スペクトラムアナライザー', 'ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞ', 'ｽﾍﾟｸﾄﾗﾑｱﾅﾗｲｻﾞｰ'}
5:同値類: 安全性能試験器 ->集合:{'安全性能試験', 'E安全性能試験器', '安全性能試験器', '4安全性能試験', '1安全性能試験器', '3安全性能試験器', '4安全性能試験器'}
6:同値類: EPIRBTWOWAY性能試験器 ->集合:{'EPIRBTWOWAY性能試験器'}
7:同値類: 双方向無線装置2 ->集合:{'双方向無線装置', '双方向無線装置2'}
8:同値類: 双方向試験器 ->集合:{'双方向試験測定器', '双方向試験機', '双方向性能試験器', '双方向試験器', '双方向性能試'}
9:同値類: 双方向無線電話安全性能試験器 ->集合:{'双方向無線電話安全性能試験機', '双方向無線電話装置性能試験器', '双方向無線安全性能試験器', '双方向無線電話安全性能試験器'}
10:同値類: EPIRB&双方向ﾃｽﾀ ->集合:{'EPIRB&双方向ﾃｽﾀ'}
11:同値類: 性能試験器 ->集合:{'性能試験器'}
12:同値類: 周波数測定機 ->集合:{'周波数測定機', '周波数測定器'}
13:同値類: デジタル電力計 ->集合:{'デジタル電力計'}
14:同値類: 電力周波数測定器 ->集合:{'電力周波数測定器'}
15:同値類: 406EPIRB安全性能試験器 ->集合:{'406EPIRB安全性能試験器', 'EPIRB安全性能試験器', '406EPIRB安全性能試験機'}
16:同値類: VHF性能試験器 ->集合:{'VHF性能試験器'}
17:同値類: 406性能試験器 ->集合:{'406性能試験器'}
18:同値類: GMDSSTESTER ->集合:{'GMDSSTEST', 'GMDSSTESTER'}
19:同値類: ｼﾙﾄﾞﾎﾞｯｸｽ ->集合:{'ｼﾙﾄﾞﾎﾞｯｸｽ'}
20:同値類: 406EPIRB試験器 ->集合:{'406EPIRB試験機', '406EPIRB試験器'}
21:同値類: 空中線電力計 ->集合:{'空中線電力計'}
22:同値類: 406EPIRB測定器 ->集合:{'406EPIRB測定器'}
23:同値類: EPIRB性能試験器 ->集合:{'EPIRB性能試験器', 'EPIRB試験器'}
24:同値類: シルドBOX ->集合:{'ｼｰﾙﾄﾞﾎﾞｯｸｽ', 'ｼｰﾙﾄﾞBOX', 'シールドBOX'}
25:同値類: 周波数計 ->集合:{'周波計', '周波数計'}
26:同値類: 周波数ｶｳﾝﾀ ->集合:{'周波数ｶｳﾝﾀｰ付', '周波数ｶｳﾝﾀｰCM', '周波数ｶｳﾝﾀｰ'}
27:同値類: (株)西日本ﾌｼﾞｸﾗ ->集合:{'(株)西日本ﾌｼﾞｸﾗ'}
28:同値類: 高周波電力計 ->集合:{'高周波数電力計', '高周波電力計'}
29:同値類: 無線機ﾃｽﾀ ->集合:{'無線機ﾃｽﾀｰ'}
30:同値類: 双方向無線充電器古野電気 ->集合:{'双方向無線充電器古野電気'}
31:同値類: EPIRB双方向試験機 ->集合:{'EPIRB双方向試験機'}
32:同値類: 双方向試験器121.5MHZ ->集合:{'双方向試験器121.5MHZ'}
33:同値類: SARTテスタ ->集合:{'SARTTESTER', 'SARTテスター', 'SARTﾃｽﾀｰ'}
34:同値類: 周波数ｶｳﾝﾀ付電力計 ->集合:{'周波数ｶｳﾝﾀｰ付電力計'}
35:同値類: 充電器ENTEL ->集合:{'充電器ENTEL'}
36:同値類: 信号発生器 ->集合:{'信号発生器'}
37:同値類: JHS7充電器 ->集合:{'JHS7充電器'}
38:同値類: SART試験器 ->集合:{'SART性能試験器', 'SART試験器'}
39:同値類: FM8放電器 ->集合:{'FM8放電器'}
40:同値類: 充電器 ->集合:{'充電器(ﾌﾙﾉ)', '充電器J', '充電器', '充電器(ｱﾝﾘﾂ)'}
41:同値類: 双方向測定器 ->集合:{'双方向測定器'}
42:同値類: 周波数電力計 ->集合:{'周波数計電力計', '周波数電力計'}
43:同値類: 周波数計付CM ->集合:{'周波数計付CM'}
44:同値類: SARTﾃｽﾀ ->集合:{'SARTﾃｽﾀ'}
45:同値類: SART安全性能試験器 ->集合:{'SART安全性能試験器'}
46:同値類: FM8充電器 ->集合:{'FM8充電器'}
47:同値類: ﾊﾞｯﾃﾘﾃｽﾀｱﾝﾘﾂ ->集合:{'ﾊﾞｯﾃﾘﾃｽﾀｱﾝﾘﾂ'}
48:同値類: SART専用測定器 ->集合:{'SART専用測定器', 'SART測定器'}
49:同値類: GMDSS定期検査用 ->集合:{'GMDSS検査用', 'GMDSS定期検査用', 'GMDSS検査'}
50:同値類: 双方向無線充電器日本無線 ->集合:{'双方向無線充電器日本無線'}
51:同値類: 充電器FM8 ->集合:{'充電器FM8'}
52:同値類: レダトランスポンダ試験機 ->集合:{'レーダートランスポンダ試験機'}
53:同値類: GMDSS検査ｼｽﾃﾑ ->集合:{'GMDSSD検査ｼｽﾃﾑ', 'GMDSS検査ｼｽﾃﾑ', 'GMDSS定検ｼｽﾃﾑ', 'GMDSS検査用ｼｽﾃﾑ', 'GMDSS検査ｽｽﾃﾑ', 'GMDSS検査ｼｽﾃﾑ'}
54:同値類: ﾚﾀﾞﾄﾗﾝｽﾎﾟﾝﾀﾞ安全性能試験器 ->集合:{'ﾚｰﾀﾞｰﾄﾗﾝｽﾎﾟﾝﾀﾞ安全性能試験器'}
55:同値類: RU230AB充電器 ->集合:{'RU230AB充電器'}
56:同値類: 406EPIRBﾃｽﾀ ->集合:{'406EPIRBﾃｽﾀｰ', '406EPIRBﾃｽﾀ'}
57:同値類: 周波数計付電力 ->集合:{'周波数計付電力'}
58:同値類: 周波数計及び ->集合:{'周波数計及び'}
59:同値類: 充電器古野 ->集合:{'充電器古野'}
60:同値類: RU207A充電器 ->集合:{'RU207A充電器'}
61:同値類: 双無線試験 ->集合:{'双無線試験'}
62:同値類: 双方向無線電話装置用テスタ ->集合:{'双方向無線電話装置用テスター'}
63:同値類: ﾊﾞｯﾃﾘﾃｽﾀﾌﾙﾉ ->集合:{'ﾊﾞｯﾃﾘﾃｽﾀﾌﾙﾉ'}
64:同値類: 放電器JRC ->集合:{'放電器JRC'}
65:同値類: GMDSS測定器 ->集合:{'GMDSS測定器', 'GMDSS検査測定器', 'GMDSS総合測定器'}
66:同値類: 定期検査ｼｽﾃﾑ ->集合:{'定期検査ｼｽﾃﾑ'}
67:同値類: EPIRBﾃｽﾀ ->集合:{'EPIRBﾃｽﾀ'}
68:同値類: 古野電気充電器 ->集合:{'古野電気充電器'}
69:同値類: 充電装置(ｱﾝﾘﾂ) ->集合:{'充電装置', '充電装置(ｱﾝﾘﾂ)'}
70:同値類: 充電器JRC ->集合:{'充電器JRC'}
71:同値類: ＣＭ計 ->集合:{'ＣＭ計', 'CM計'}
72:同値類: 双方向テスタ ->集合:{'双方向テスター'}
73:同値類: 双方向充電器 ->集合:{'双方向充電器', '双方向充電器N', '双方向用放電器A', '双方向充電器A', '双方向用充電器A', '双方向充電器F'}
74:同値類: 充電器古野電気 ->集合:{'充電器古野電気'}
75:同値類: R2810システム ->集合:{'R2810システム'}
76:同値類: GMDSSTESTSYSTE ->集合:{'GMDSSTESTSYSTE', 'GMDSSTESTSISTEM', 'GMDSSTESTSISTE', 'GMDSSTESTSYSTEM', 'GMDSSTESTSIST'}
77:同値類: 放電器 ->集合:{'放電器'}
78:同値類: 通過型電力計 ->集合:{'通過型電力計'}
79:同値類: 双方向無線測定 ->集合:{'双方向無線測定'}
80:同値類: 406EPIRBⅡ ->集合:{'406EPIRB', '406EPIRBⅡ'}
81:同値類: 試験 ->集合:{'試験'}
82:同値類: EPIRB測定器 ->集合:{'EPIRB測定器'}
83:同値類: 9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞ ->集合:{'9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞ', '9GHｚﾄﾗﾝｽﾎﾟﾝﾀﾞｰ'}
84:同値類: 周波計出力計 ->集合:{'周波計出力計', '周波計出力'}
85:同値類: 日本無線放電器 ->集合:{'日本無線放電器', '日本無線充電器'}
86:同値類: TOKIMEC充電器 ->集合:{'TOKIMEC充電器'}
87:同値類: 放電器古野 ->集合:{'放電器古野'}
88:同値類: 充電器ｱﾝﾘﾂ ->集合:{'充電器ｱﾝﾘﾂ'}
89:同値類: GMDSS検査SISTEM ->集合:{'GMDSS検査SISTEM'}
90:同値類: ｼﾝｸﾛｽｺｰﾌﾟ ->集合:{'ｼﾝｸﾛｽｺｰﾌﾟ'}
```

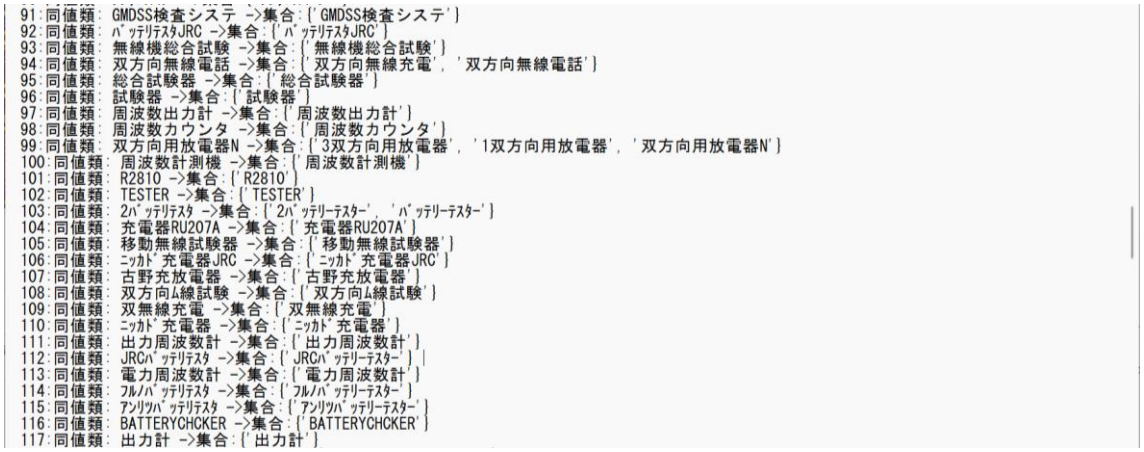


図 7　TW(双方向) で同じとみなされた 117 語

図 8 図 7 の結果からさらに導かれた互いの包含関係(TW)

以下に作成した Python プログラムのソースコードを載せる。

```
import pandas as pd
from tkinter import filedialog
import os
import re

# CSV ファイルの読込
file_name1 = filedialog.askopenfilename(
    title = "入力 CSV ファイルを開きます",
    filetypes = [('CSV file',".csv")],
    initialdir = './'
)
df = pd.read_csv(file_name1, encoding="cp932", index_col = 0).reset_index()

# 今回は'使用測定器１'列を読む
column_name ='使用測定器１'
fname_without_ext = os.path.splitext(os.path.basename(file_name1))[0]
```

```
textfilename = fname_without_ext + '出力.txt'

out_file = open(textfilename, 'a', encoding = 'utf-8')

nlp = spacy.load('ja_ginza_electra') # Ginza Transformers モデル

print('総件数:' + str(len(df)) + '件')

douon_dict = {} # 読みをキー、token リストのリストを値にした辞書

for i in range(len(df)):
#for i in range(100):
    reading = str()
    #t_list = []
    t_list =str() # 2024/8/17
    text = df.loc[i,column_name]
    if type(text) is str :
        doc = nlp(text)
        for sent in doc.sents:

            for token in sent:
                if token.pos_ not in ['PUNCT', 'SYM','X']:
                    t_read_list = token.morph.get("Reading")
                    if len(t_read_list) > 0:
                        reading += t_read_list[0]
                        #t_list.append(token)
                        t_list += str(token) # 2024/8/17
                        #print(t_list)
            #print(reading)
    # 行ごとの処理
    if len(reading) > 0:
        if reading not in douon_dict:
            #douon_dict[reading] = [t_list]
            douon_dict[reading] = set([t_list]) # 2024/8/17
        else:
            #douon_dict[reading].append(t_list)
```

```
                douon_dict[reading].add(t_list) # 2024/8/17

out_file.write(f'同音リスト:₩n')
d_count = 0
for reading in douon_dict:
    d_count += 1
    out_file.write(f'{d_count}:{reading} -> {douon_dict[reading]}₩n')

print('計：' + str(d_count) + '語')
out_file.close()
```

① ルールに相当した Python コード

```
from tkinter import filedialog
import os
import re
import difflib


# text ファイルの読込 ~ 出力.txt ファイルが対象
file_name1 = filedialog.askopenfilename(
    title = "入力 TEXT ファイルを開きます",
    filetypes = [('Text file',".txt")],
    initialdir = './'
)
#df = pd.read_csv(file_name1, encoding="cp932", index_col = 0).reset_index()
in_file = open(file_name1, 'r', encoding = 'utf-8')

tfnwoext = os.path.splitext(os.path.basename(file_name1))[0]

out_filename =  tfnwoext + '処理結果' + '.txt'
out_file = open(out_filename, 'a', encoding = 'utf-8')


equivalence_dict = {}
family_dict = {}
```

```
for line in in_file:
    if ('->') in line:
        tail = line.split('->')[1]
        #print(tail)
        tail = tail.lstrip(' {')
        tail = tail.rstrip('}₩n')
        w_list = tail.split(', ')
        w_list = [w.strip("'") for w in w_list] # 両側の引用符をとる

        if len(w_list)>0:
            # 辞書のキーにはマイナス抜きで,同じ読みのものの見出しにない後も－なしでキーに一致すれば同値類に入る
            wominus = w_list[0].replace('−', '')
            wominus = wominus.replace('ー','')
            wominus = wominus.replace('-','')
            if  wominus in equivalence_dict:
                for w in w_list:
                    equivalence_dict[wominus].add(w)
            else:
                equivalence_dict[wominus] = set(w_list)

# 日本語⇔英語　変換（翻訳）して同じであれば同値類に入れる
#from translate import Translator
from deep_translator import GoogleTranslator # 月 50 万文字まで無料
#translator = Translator(from_lang='en', to_lang='ja')
translator = GoogleTranslator(source='en', target='ja')

merge_keys_pair_list = [] # [(key1,key2),...,(keyn,keym)] -> key1 が key2 へ（あるいはその逆）値ごと併合
i = 0
j = 0
for tek1 in equivalence_dict.keys():
    i = i+1
    j = 0
    for tek2 in equivalence_dict.keys():
```

```
            j = j+1
            #if tek1 != tek2:
            if i < j:
                # 翻訳してキーが同じならば併合ペアに
                e_word1 = re.sub(r"[^a-zA-Z]", "", tek1) #  英語だけ取り出す
                e_word2 = re.sub(r"[^a-zA-Z]", "", tek2)
                #予約語は除く
                e_word1 = e_word1.replace('EPIRB', '')
                e_word1= e_word1.replace('SART','')
                e_word2 = e_word2.replace('EPIRB', '')
                e_word2 = e_word2.replace('SART','')
                if len(e_word1)>0:
                    j_word1 = translator.translate(e_word1.lower())
                    if j_word1 != e_word1:
                        j_word1 = j_word1.replace('−', '')
                        j_word1 = j_word1.replace('ー','')
                        j_word1 = j_word1.replace('-','')
                        replaced_tek1 = tek1.replace(e_word1,j_word1)
                        #print(replaced_tek1)
                        if replaced_tek1 == tek2:
                            merge_keys_pair_list.append((tek1,tek2))
                if len(e_word2)>0:
                    j_word2 = translator.translate(e_word2.lower())
                    if j_word2 != e_word2:
                        j_word2 = j_word2.replace('−', '')
                        j_word2 = j_word2.replace('ー','')
                        j_word2 = j_word2.replace('-','')
                        replaced_tek2 = tek2.replace(e_word2,j_word2)
                        #print(replaced_tek2)
                        if replaced_tek2 == tek1:
                            merge_keys_pair_list.append((tek1,tek2))
                #   互いに文字列（シーケンス）としての類似性が spelling_miss%以上であれ
ば併合ペアに  2024/9/1
                spelling_miss = 0.85
                similarity = difflib.SequenceMatcher(None,tek1,tek2).ratio()
                if similarity > spelling_miss:
```

```
                    merge_keys_pair_list.append((tek1,tek2))
print(merge_keys_pair_list)
for mk_pair in merge_keys_pair_list:
    # どちらかのこっているほうをにマージ
    if mk_pair[0] in equivalence_dict:
        if mk_pair[1] in equivalence_dict:
            equivalence_dict[mk_pair[0]] = equivalence_dict[mk_pair[0]].union(equivalence_dict[mk_pair[1]])
        else:
            for ek_new in equivalence_dict:
                if mk_pair[1] in equivalence_dict[ek_new]:
                    if mk_pair[1] in equivalence_dict:
                        equivalence_dict[mk_pair[0]]=equivalence_dict[mk_pair[0]].union(equivalence_dict[ek_new]) # すでにキー（であり値の元でもある mk_pair[0]の値集合（=同値類）は ek_new の値集合にマージされているので、カウンターパートである mk_pair[1]の値集合をマージ
                        break
    else: # 今回の mk_pair[0]が過去にペアの２番目として消されている場合、値が一致するキーに連結する
        for ek_new in equivalence_dict:
            if mk_pair[0] in equivalence_dict[ek_new]:
                if mk_pair[1] in equivalence_dict:
                    equivalence_dict[ek_new]=equivalence_dict[ek_new].union(equivalence_dict[mk_pair[1]]) # すでにキー（であり値の元でもある mk_pair[0]の値集合（=同値類）は ek_new の値集合にマージされているので、カウンターパートである mk_pair[1]の値集合をマージ
                    break
    if mk_pair[1] in equivalence_dict:
        del equivalence_dict[mk_pair[1]]
# 包含関係
for ek1 in equivalence_dict.keys():
    for ek2 in equivalence_dict.keys():
        if ek1 != ek2:
            if ek1 in ek2 or ek2 in ek1:
                if ek1 in family_dict:
                    family_dict[ek1].add(ek2)
```

```
                    else:
                        family_dict[ek1] = set([str(ek2)])


# 最終結果のファイル出力
count = 0
for ek,ev in equivalence_dict.items():
    count += 1
    out_file.write(f'{count}:同値類: {ek} ->集合:{ev} ₩n')
fcount = 0
for fk,fv in family_dict.items():
    fcount += 1
    out_file.write(f'{fcount}:包含関係:{fk} ->集合:{fv} ₩n')

# 終了処理
in_file.close()
out_file.close()
```

上記ルール適用後に同値類および包含関係を生成する Python コード

3.2 膨張式救命いかだ整備指導書を入力としたヒューマンエラー防止のためのオントロジー自動生成試作

オントロジー（事例である「個体」とその集合である「クラス」及びそれらの「関係」の三つから成る）は、一旦作成できるとその論理的な推論機能によりベテラン設計者の判断や決定の見える化が可能で、さらに複数の推論過程を利用することにより人が見落とし易い様々な要因や波及効果を網羅的に抽出できることから、ヒューマンエラー防止の観点から技術指導書の知的な検索・推論エンジンとして最適である。しかし、オントロジーデータ作成は人に依存するところが大きく自動生成には問題があった。

そこで、最近進歩が目覚ましい生成系 AI の技術を活用し、この問題を解決するための第一歩に着手した。ChatGPT 等に代表される生成系 AI は広く公開されている情報（データ）を基に事前学習を行い、従来からある自然言語処理技術により自然発話文を検索可能な単語の組合せへ変換して、効率的な検索やそれらの組合せ文を作成（生成）するが、公開されていない専門知識に対する処理は不十分で、自動化の可能性については完全にオープンな（やってみないと分からない）状況である。

ここではヒューマンエラー防止に向けて実際の膨張式救命いかだ整備指導書を入力対象とし、その中で使われている単語を共通する述語（動詞、形容詞、形容動詞）に着目して類

型化し、自然言語処理向けのデータ駆動型 AI 技術を活用することでオントロジーの構成要素である「個体」「クラス」「関係」を自動生成して、作成したオントロジーと比較しながら有効性を確認することを目指した。

この段階では、人手によるオントロジーには及ばない点も予想できるが、人が 200 ページをこえる文書を読みモデルを作成するのは現実的ではなく、近年発展が目覚しい生成 AI 技術のなかでも自然言語処理において使われている技術を援用することにより、オントロジーを構成するビルディングブロックである「個体」（とその集合である「クラス」）間の「関係」を入力テキストから自動的に抽出するには主語（S）述語（V）目的語（O）または補語（C）に分解（SVO/SVC 分解）することを考えた。この部分は前処理として大規模言語モデル（LLM）で事前学習済みの形態素分解ライブラリを使用することにより解決した。

形態素解析（けいたいそかいせき）とは、自然言語処理の分野で用いられる手法の一つで、文章を単語や形態素（最小意味単位）に分割し、それぞれの形態素の品詞（動詞、名詞、助詞など）を識別することを指す。例えば、「私は学生です」という文を形態素解析すると、「私/名詞」「は/助詞」「学生/名詞」「です/助動詞」といった具合に分割される。形態素解析は、日本語のような複雑な文法を持つ言語において特に重要であり、テキストマイニングや機械翻訳、検索エンジンのインデクシングなど、さまざまな応用分野で利用されている。従来から MeCab など人による発見的な（経験的な）アルゴリズムによって、入力テキストから自動的に品詞分解を行っていたが、近年は圧倒的なビックデータである入力例文と形態素解析結果を機械学習させることによりその精度は十分実用化されるレベルに至った。これは LLM による事前学習によるところが大きい。

また大規模言語モデル（Large Language Model, LLM）とは、膨大な量のテキストデータを使って訓練された機械学習モデルの一種で、これらのモデルは、自然言語処理（NLP）のタスクを非常に高い精度で実行できるよう設計されている。例えば、文章の生成、翻訳、要約、質問応答、対話などが含まれる。

大規模言語モデルの主な特徴には以下のようなものがある：

大量の訓練データ：インターネット上のニュース記事、書籍、ウェブページなど、非常に多くのテキストデータを使って訓練されている。

高度なパターン認識：文章の意味や文脈を理解し、適切な応答や生成を行うことができる。

多様な応用：チャットボット、検索エンジン、翻訳ツール、コンテンツ生成など、さまざまな分野で応用されている。ChatGPT や Copilot も LLM を基にして動作している。

日本語を対象にした際のオントロジーの人による作成の手順は以下のようになる。これらを自動化することが目的となる。

STEP 1（文書の前処理）

予めモデル仕様書（要求書、仕様書、解析文書）からテキスト抽出して、一つのテキストファイルを入力とした大規模言語モデルによる辞書を用いた形態素解析（分かち書き及び品

詞分解、係り受け解析が全自動化）を行う。→（出力）S、V、O/C に分解された各単語（品詞）とその他の修飾語等の係り受け関係。

STEP2（交互関係個体辞書作成）

個体候補リストにある一つ一つの個体をキー/関連する関係リストを値とし、次に関係候補リストの一つ一つ（関係）をキー/関連する個体リストを値として、交互に共通関係辞書、共通個体辞書を作成する。

STEP3（個体と関係の構造化）

個体の構造化はクラス生成上の候補となり、関係の構造化は関係同士の階層構造を作ることに繋がる。

STEP3-1

関係候補リストの一つ一つの関係キーに対応する値を共通個体辞書から取り出す。複数ある場合はクラス候補リストに入れる。

STEP3-2

個体候補リストの一つ一つの個体キーに対応する値を共通関係辞書から取り出す。複数ある場合は関係階層候補リストに入れる。

STEP4（クラス候補リストと関係階層候補リストの精査）

STEP4-1

「同じ関係で繋がる個体が同じクラスになるのか？」「どのような場合に違うクラスにすべきか？」を精査する。

STEP4-2

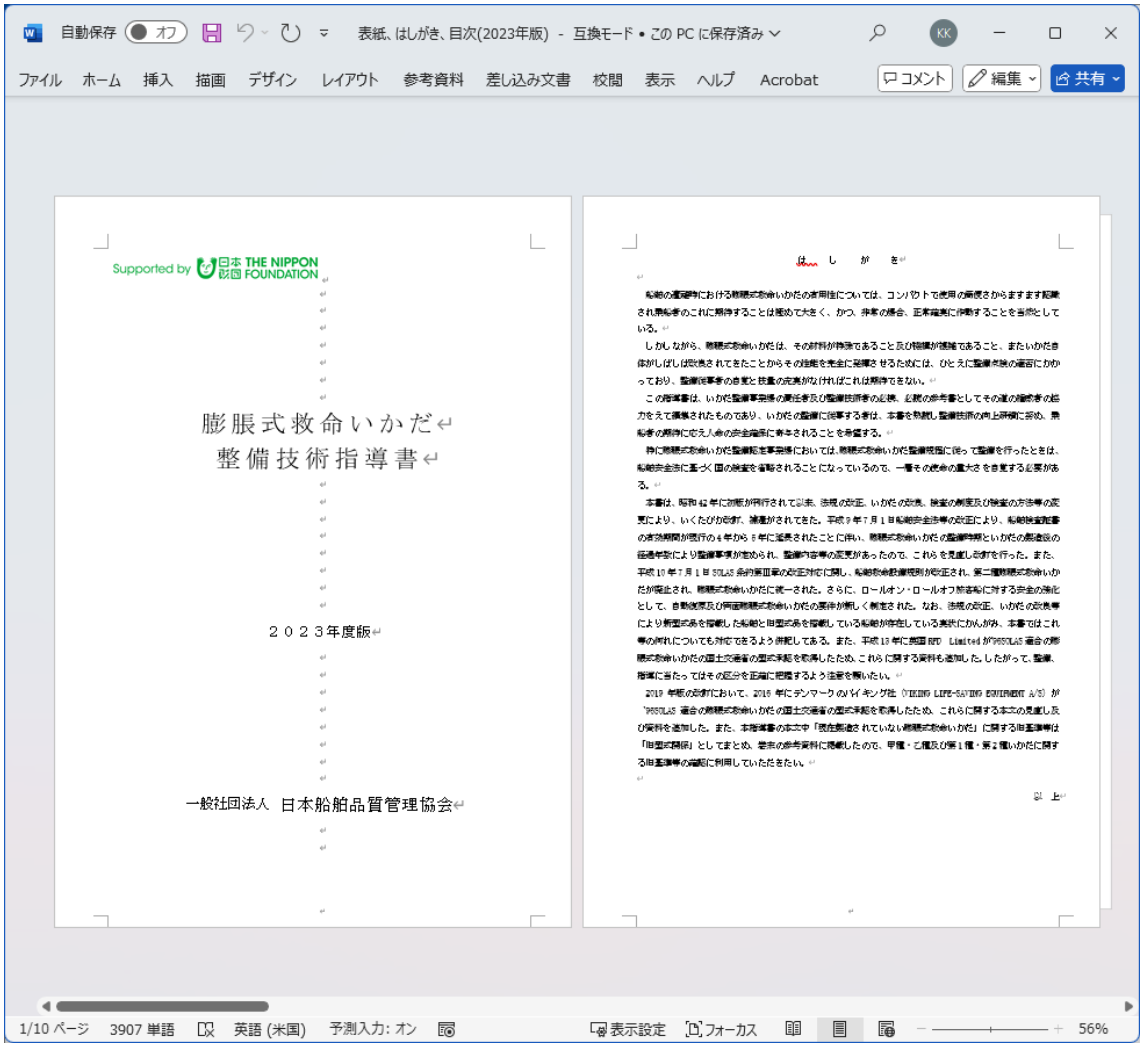
「同じ個体で繋がる関係のどちらが親でどちらが子か？」「それ以外の関連（並列）は何か？」を精査する。

本調査研究では上記の STEP1 から STEP４-1 までを実装し、関係と個体をそのままクラスにした自明なクラスまではオントロジーデータ（OWL ファイル）として自動生成したが、STEP4-2 まで達成するには至らなかった。実装したアルゴリズムをトライしたところ、クラス候補となる個体の集合と同じ名前を持つ自明なクラスまでは同じ関係から自動作成できたが、クラス間の包含関係や抽象化には決定的なロジック（アルゴリズム）が不足しており、現状の LLM や Wikidata[5]を援用しても実用にあるオントロジー作成ができないことが分かったからである。それについては終章の課題として述べる。

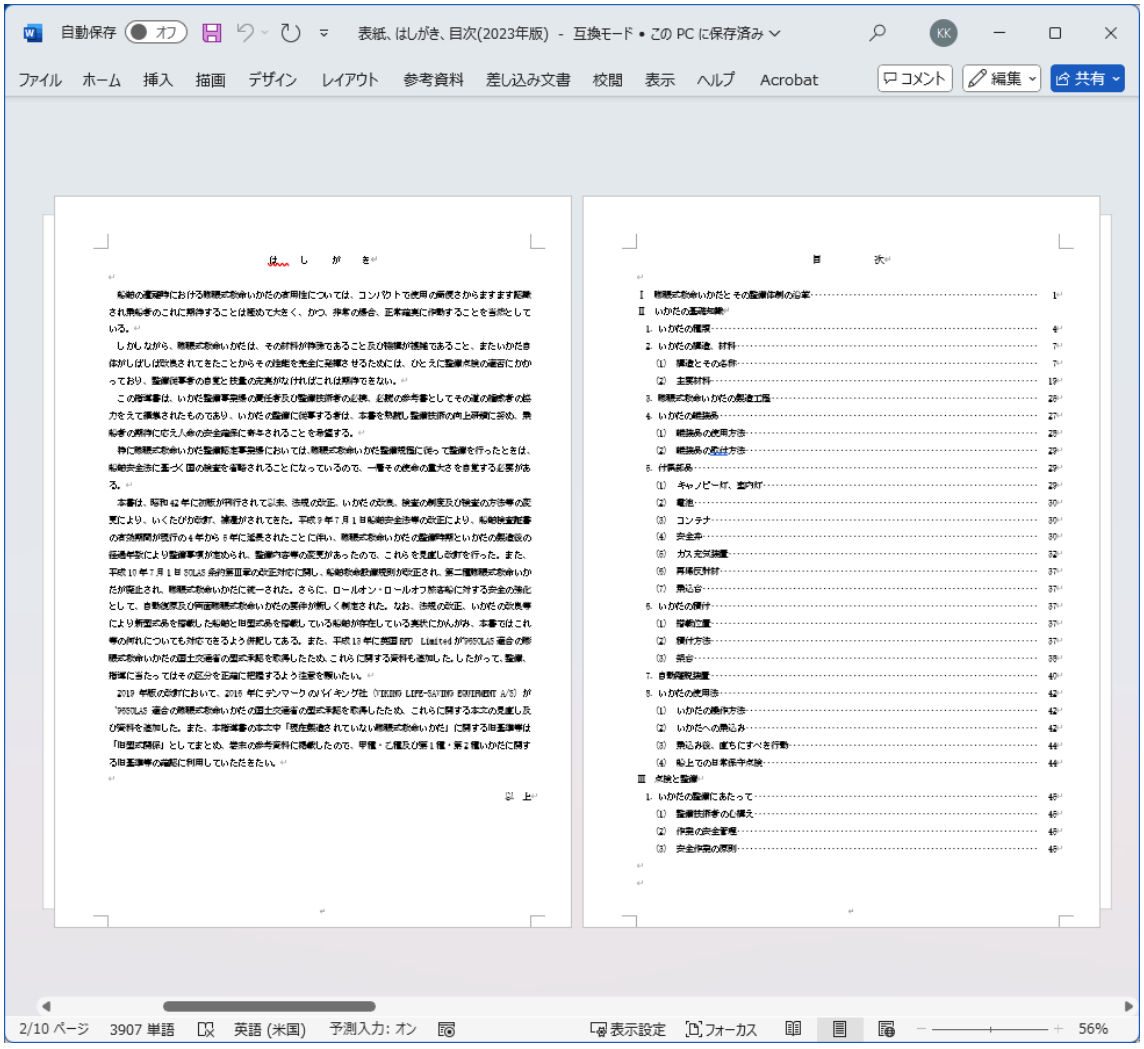
図 9 に日本船舶品質管理協会様よりお借りしている膨張式救命いかだ整備技術指導書の冒頭から目次の最後のページを載せる。

[5] このように事前学習した LLM だけに頼らず、既に人が概念や用語を整理した外部情報をもちいることを RAG(Retrieval Augment Genration)と呼ぶ

（株）オフィスS．K．Y


自動保存
表紙、はしがき、目次(2023年版) - 互換モード • この PC に保存済み
ファイル ホーム 挿入 描画 デザイン レイアウト 参考資料 差し込み文書 校閲 表示 ヘルプ Acrobat
コメント
編集
共有
Supported by 日本財団 THE NIPPON FOUNDATION
膨脹式救命いかだ
整備技術指導書
２０２３年度版
一般社団法人 日本船舶品質管理協会
はしがき
1/10 ページ 3907 単語 英語 (米国) 予測入力: オン
表示設定 フォーカス
56%

（株）オフィスS．K．Y


自動保存
表紙、はしがき、目次(2023年版) - 互換モード • この PC に保存済み
ファイル ホーム 挿入 描画 デザイン レイアウト 参考資料 差し込み文書 校閲 表示 ヘルプ Acrobat
コメント
編集
共有
はしがき
目次
2/10 ページ
3907 単語
英語 (米国)
予測入力: オン
表示設定
フォーカス
56%

（株）オフィス S．K．Y

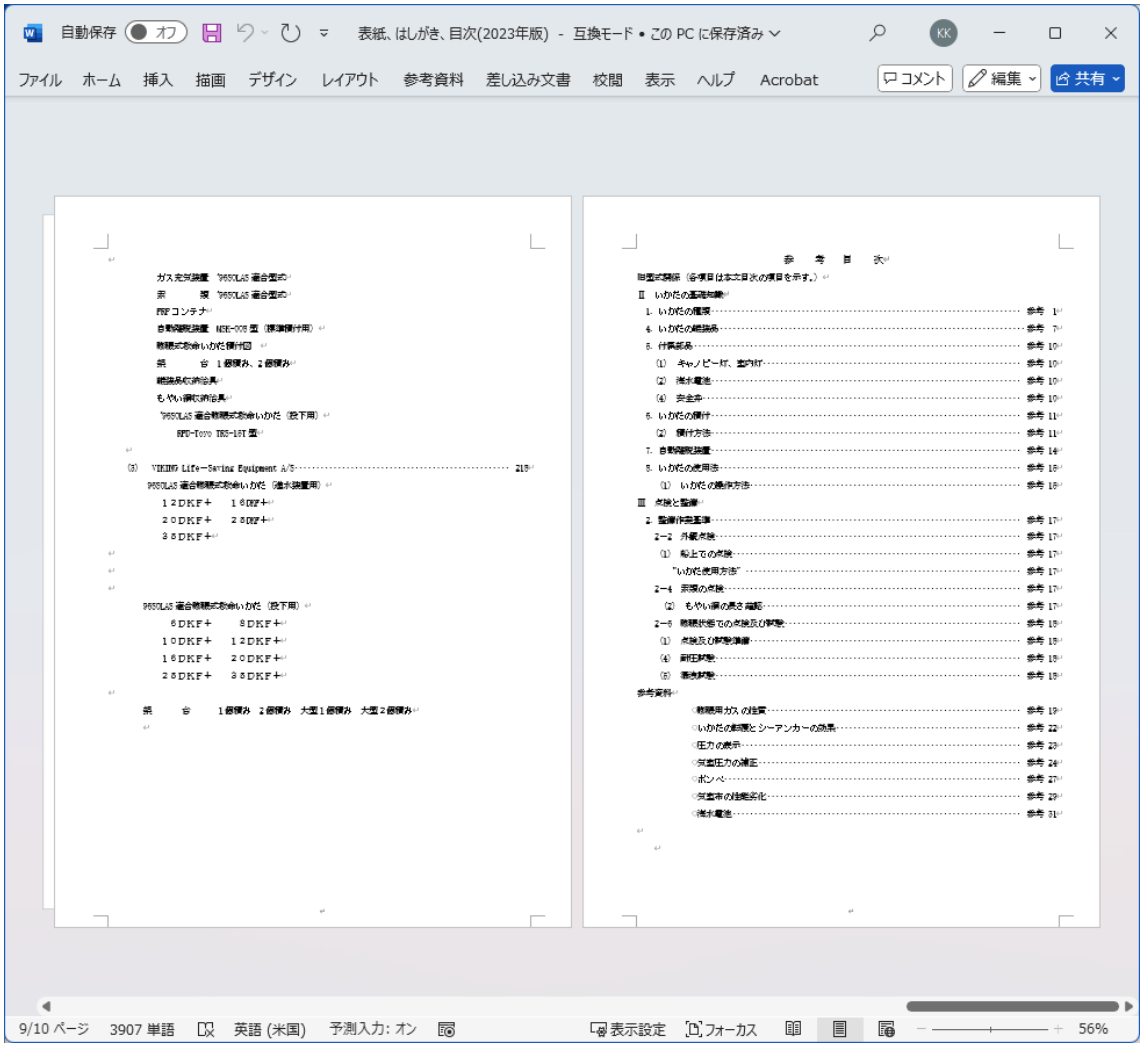


図 9 膨張式救命いかだ整備指導書（Word）冒頭から目次の最後まで

入力対象対象としては Word 文書と PDF および Excel のファイルである（図 10）。

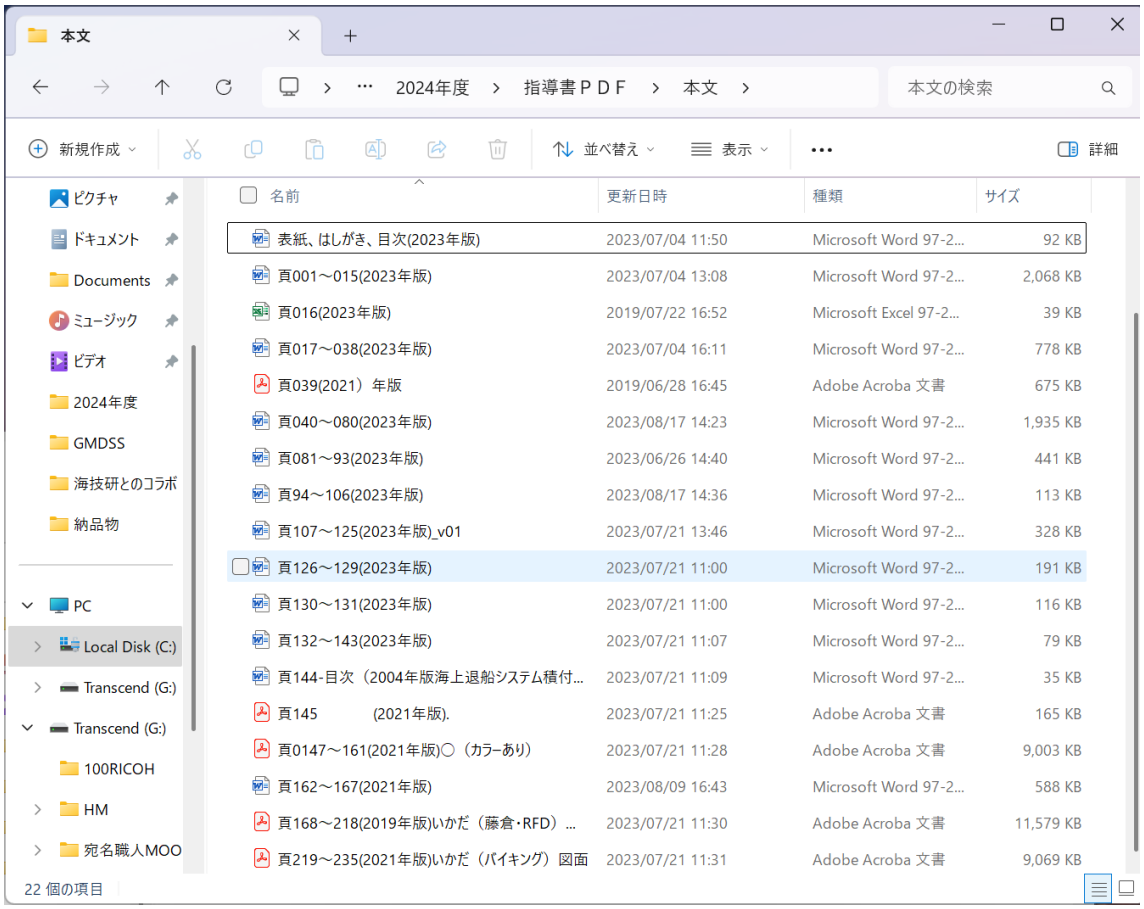


図 10 入力ファイル一覧

これらのファイルに対し、Windows に標準装備されている編集可能なスクリプト（自動処理）である Power Automate を用いて Word/PDF/Excel からテキストのみを抽出する。図 11～13 は Power Automate のスクリプトの内容である[6]。

6 結果をファイルとして出力できない仕様となっているのでスクリーンショットをとった

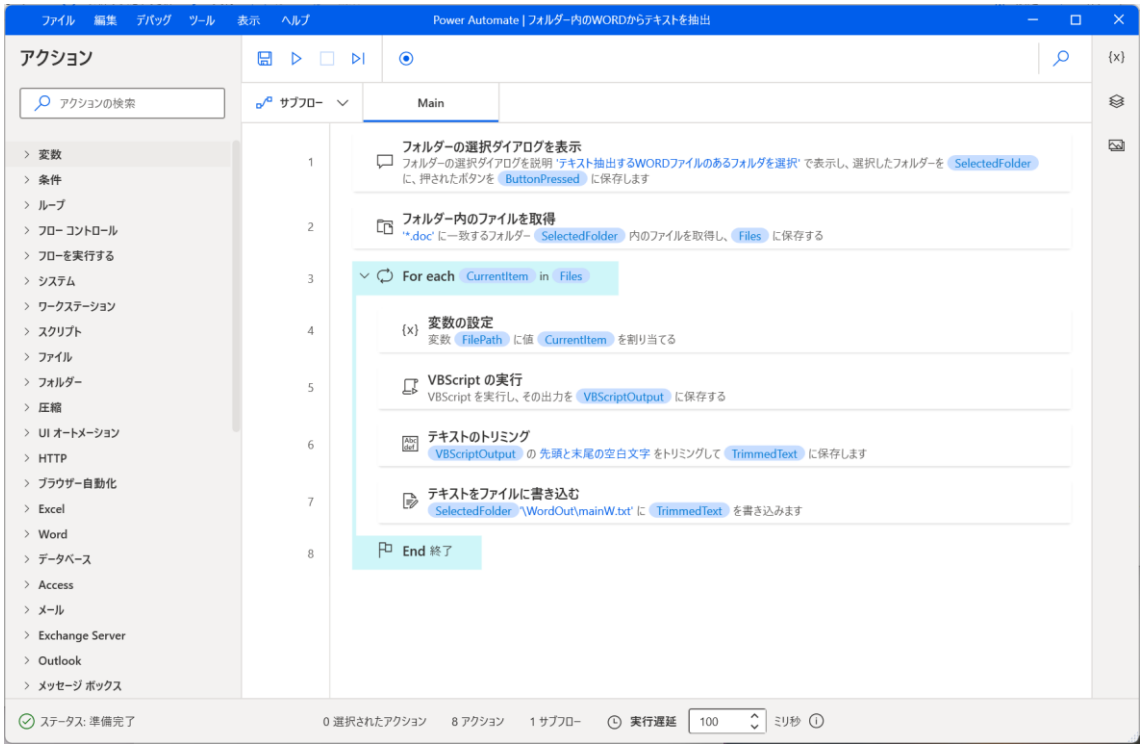


図 11　Power Automate フォルダー内の Word からテキストを抽出すフロー

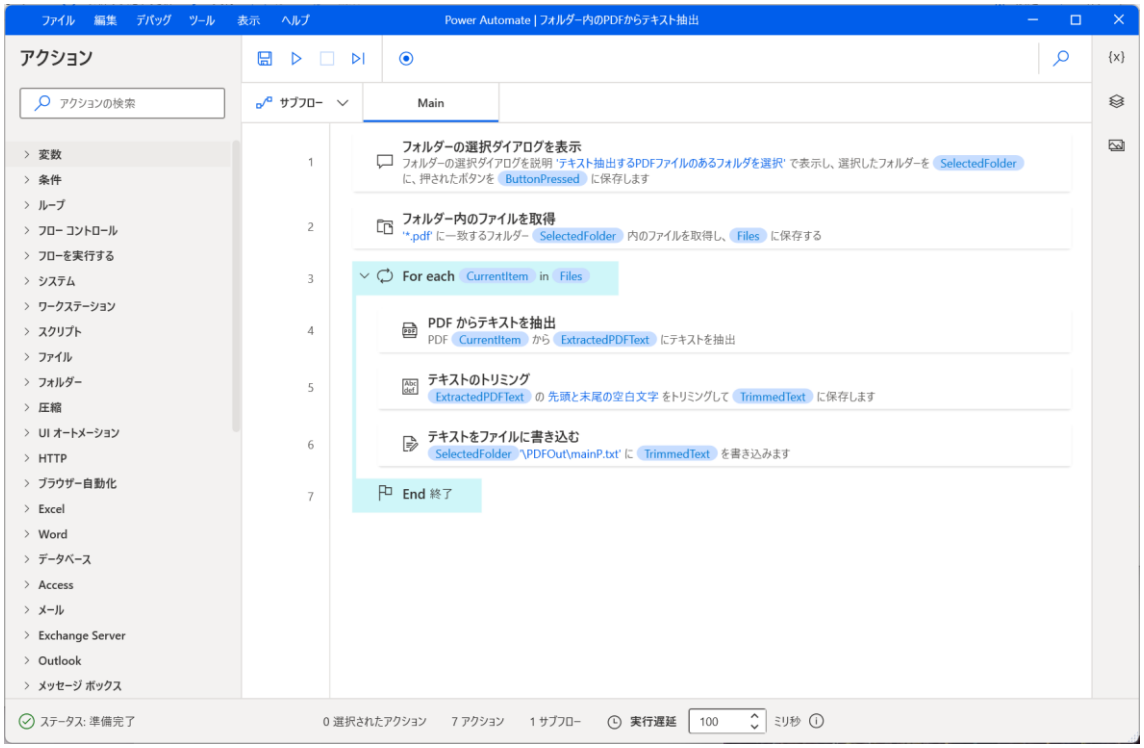


図 12　Power Automate フォルダー内の PDF からテキストを抽出するフロー

（株）オフィス S．K．Y

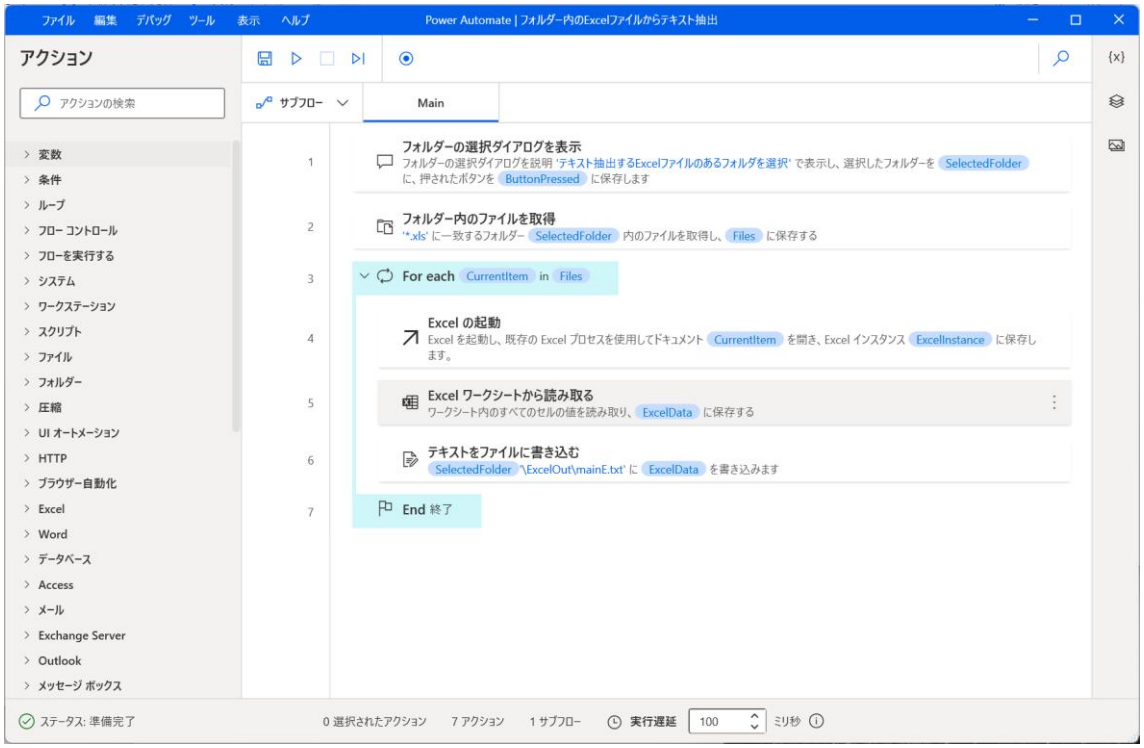


図 13　Power Automate フォルダー内の Excel からテキストを抽出するフロー

図 14 は Word から抽出したテキストファイルの冒頭部分である。

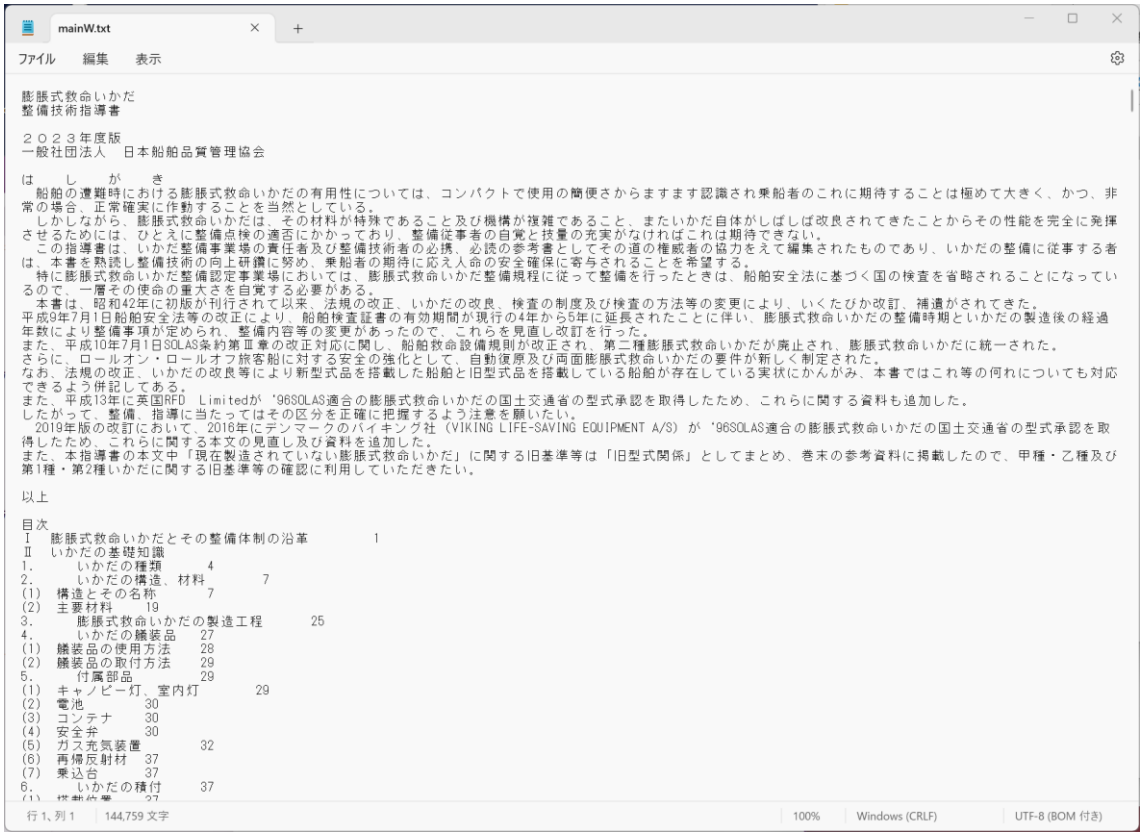


図 14 Word から抽出されたテキストファイル

このようにして Word/PDF/Excel ファイルから抽出されたテキストファイルを 1 つのテキストファイルにつなげて[7]できた入力テキストファイルは 372KB である。この入力に対して Python で LLM で事前学習済みの形態素解析ライブラリを使ったコードで前処理を行うのだが、そのライブラリが一度に処理できるトークン（単語）の数に制限(5000 文字以下)があるため以下の Python コードで入力テキストファイルを制限単語数以下になるように自動分割を行った。

```
import math
from tkinter import filedialog


# text ファイルの読込
file_name1 = filedialog.askopenfilename(
    title = "入力 TEXT ファイルを開きます",
    filetypes = [('Text file','.txt')],
```

7 つなげる作業も Power Automate でフローを作成して行った

```
    initialdir = './'
)
#df = pd.read_csv(file_name1, encoding="cp932", index_col = 0).reset_index()
in_file = open(file_name1, 'r', encoding = 'utf-8')

filename_count = 0
textfilename = './入力' + str(filename_count) + '.txt'
out_file = open(textfilename, 'a', encoding = 'utf-8')

char_count = 0
for line in in_file:
    # 2024/4/20 改行（行の最後）の半角スペースがあれば除去する
    line = line.strip()
    #
    char_count += len(line)
        # 5000 文字を超えたら別ファイルに分割
    if char_count > 5000:
        char_count = 0
        out_file.close()
        filename_count += 1
        textfilename = './入力' + str(filename_count) + '.txt'
        out_file = open(textfilename, 'a', encoding = 'utf-8')
    #print(char_count)
    out_file.write(line + '₩n')

in_file.close()
out_file.close()
```

文字数を 5000 文字以下にテキストファイルを分割する Python コード

入力のテキストは 27 個に分割された（入力 1.txt、入力 2.txt、…入力 27.txt）。これらに対して以下の Python コードで示す処理を行った。内容としては入力のテキストに対して、形態素分解をおこない、品詞ごとに分割され、互いに係り受けの関係を保持ままの各単語に、自動的に降られた ID に加え、入力ファイル名（例、入力 1）を追加して、主語に対する動詞の関連、動詞に対する対する主語や目的語、補語の関連、および修飾語が最終的に係り受ける主要な単語の関連全てを漏らさず記録した「辞書」として保存し、最終的には SVO/SVC リストとして出力する内容である。これによって LLM を使った前処理が実行される。

```
import os
from tkinter import filedialog
import re


#Batch 版（複数ファイル名対応→PowerAutomate で使うことを想定
print(sys.argv[1])
textfilename = sys.argv[1]
in_file = open(textfilename, 'r', encoding = 'utf-8')
text = in_file.read()

in_file.close
# print(text)

textfilename2 = './未処理.txt'
out_file = open(textfilename2, 'a', encoding = 'utf-8')

#textfilename3 = './正規処理出力.txt'

# 2024/4/20
tfnwoext = os.path.splitext(os.path.basename(textfilename))[0]
#print(tfnwoext)
#textfilename3 = textfilename + '処理結果.txt'
textfilename3 = tfnwoext + '処理結果.txt'
#
out_file2 = open(textfilename3, 'a', encoding = 'utf-8')

# 文章から直接単語単位の解析の使い方のサンプル
# for sent in doc.sents:
#       for token in sent:
#           print(                                  # CoNLL-U フォーマット
#               token.i,                            # ID 単語インデクス
#               token.orth_,                        # 表層形、形式　単語形式または句読点番号
#               token.lemma_,                       # 辞書見出し形、補題 補題または語形の語幹（変形前）
```

```
#                 token.norm_,                    #
#                 token.morph.get("Reading"),    # 読み
#                 token.pos_,                     # part of speech tag ユニバーサル品詞タグ
#                 token.morph.get("Inflection"),
#                 token.tag_,                     # 品詞
#                  token.dep_,                     # 依存関係 例 obj のあとに次の行の ID で ID の目的語
#                 token.head.i,                   # 文節を跨げるが…
#             )
#       print('EOS')
# print('EOD')

# 目的は複文を(単文にばらす必要はない) そのまま SVO に分解すること

dep = dp.DependencyAnalysis()

span_deps = dep.analyze(text) # mode はデフォルトの 0 のまま (係り受け元も先も文節、主辞にするとおちる)

s_individuals = set() # 主語 (S) の個体候補集合
o_individuals = set() # 目的語(O)の個体候補集合
c_individuals = set() # 補語(C)の個体候補集合
o2_individuals = set() # 目的語 (O2,ヒト) の個体候補集合
v_properties = set()  # 動詞 (V)の関係候補集合
sv_dict = {}    # 主語個体をキーにして、関係集合をバリューにした辞書
vs_dict = {}    # 関係をキーにして、主語個体集合をバリューにした辞書
vo_dict = {}    # 関係をキーにして、目的語個体集合をバリューにした辞書　vs_dict の関係とは逆関係になる
ov_dict = {}    # 目的語個体をキーにして、関係集合をバリューにした辞書
vo2_dict = {}    # 関係をキーにして、目的語個体集合をバリューにした辞書　vs_dict の関係とは逆関係になる
o2v_dict = {}    # 目的語個体をキーにして、関係集合をバリューにした辞書
b_properties = set() # 補語(be)動詞の関係候補集合
sb_dict = {}    # 主語個体をキーにして、be 動詞関係集合をバリューにした辞書
bs_dict = {}    # be 動詞関係をキーにして、主語個体集合をバリューにした辞書
```

```
bc_dict = {}  # be 動詞関係をキーにして、補語個体集合をバリューにした辞書
cb_dict = {}  # 補語個体をキーにして、be 動詞関係集合をバリューにした辞書
rvspan_dict = {} # ROOT にある VERB をキーにして、の係り受け元の文節集合をバリューにした 辞書
words_bundles = [] # 同じ役割(nmod)の複数の単語(token)のリスト（連語）のリスト。
# 2024/05/07 主語に対する複数目的語/補語のケースが漏れていたので新たに以下を追加
soc_dict = {} # 主語個体をキーにして、（複数の）目的語/補語をバリューにした辞書
ocs_dict = {} # O/C 個体をキーにして、複数の主語をバリューにした辞書

out_file2.write('...Main in process...¥n')
for span_dep in span_deps: # 文節(span)の係り受け(dependency)毎に
# ほとんどの場合、係り受け間の V-O、V-C 関係を係り受け先の末尾に注目して取り出せる
# それ以外のケースとしては ROOT から辿る宣言・定義文で別扱いが必要→係り受け元で回している for ループの中には出てこない
# → 二つ目の for ループ↓で

    # print(f'{str(span_dep[0])} → {str(span_dep[1])}')
    # 係り受け元と場合によっては係り受け先の単語単位での依存関係を解析に使う
    # token.dep_ による switch 文(match 文)
    # print('係り受け元から先へ')
    for token in span_dep[0]:
        to_token = token.doc[token.head.i]
        j_tokens = [] # 毎回空リストに初期化↓ j は (名詞への係り受け元中で)「述語」の意 述語トークン
        o_tokens = [] # 毎回空リストに初期化↓ o は（限定子がいる文節中で)「目的語」の意 目的語トークン
        n_tokens = [] # 毎回空リストに初期化↓ n は（限定子がいる文節中で)「名詞」の意 名詞トークン
        # print(token.dep_)
        match token.dep_: # 係り受けのタグで分類
            # 2024/4/21 改行や空白のみの主語を除く
            #case 'nsubj': # 主語（S)
            case 'nsubj' if str(token) != '¥n' and str(token) != ' ' and str(token) != '　': # 主語（S)
                # out_file2.write('S...¥n')
```

```
                    # out_file2.write(f'{token} → {to_token}¥n')
                    s_individuals.add(token)
                    # 主語の時のみ to_token は複数の可能性あり、文章全体を候補に変更
                    for to_token in token.sent:
                        if to_token == token:
                            continue
                        else:
                            if to_token.pos_ in ['VERB','ADJ']: # 述語のみ
                                v_properties.add(to_token)
                                # キーになければキー値を追加、あれば値に係り受け先を追加(集合の要素として)
                                                                                #
sv_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))  # 旧リスト版

                                # 2024/4/20
                                #if (str(token.lemma_)+str(token.i)) not in sv_dict: # 無ければキーと値毎新規に
                                    #       sv_dict[str(token.lemma_)+str(token.i)] = {to_token} # キーはユニークな文字列、値は集合
                                    if (str(token.lemma_)+str(token.i)+tfnwoext) not in sv_dict: # 無ければキーと値毎新規に
                                        sv_dict[str(token.lemma_)+str(token.i)+tfnwoext] = {to_token} # キーはユニークな文字列、値は集合
                                else: # あればキーの値に要素を追加（集合なので重複なし）
                                    #sv_dict[str(token.lemma_)+str(token.i)].add(to_token)
                                    sv_dict[str(token.lemma_)+str(token.i)+tfnwoext].add(to_token)
                                                                                #
vs_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_)) # あれば値に係り受け元を追加
                                # ID の uniqueness のため分割ファイル名も ID に付加

                                # 2024/4/21 改行を含まなければ条件を追加して次の if 以下の 4 行にタブ付加
```

```
                                    if '₩n' not in str(token.orth_) and '₩r' not in
str(token.orth_):
                                if str(to_token.lemma_) not in vs_dict: # キーになけ
れば追加
                                    vs_dict[str(to_token.lemma_)] = {token}
                                else:
                                    vs_dict[str(to_token.lemma_)].add(token)
                            #
                        # 2024/05/07-08 単独主語の複数 O/C のケースが漏れていた
 (S = C)
                        else:
                            if to_token.pos_ in ['NOUN']: # 主語が係り受け先の名詞
は目的語か補語
                                #print(to_token)
                                #o_individuals.add(to_token) # この段階では O か C
かわからないので、下の ocs_dict に入れとく
                                 # 以下は vo/ov_dict に o が漏れているので必要、
sv_dict と組み合わせて SVO が完成する
                                 if (str(token.lemma_)+str(token.i)+tfnwoext) not in
soc_dict:
                                    soc_dict[str(token.lemma_)+str(token.i)+tfnwoe
xt] = {to_token}
                                else:
                                    soc_dict[str(token.lemma_)+str(token.i)+tfnwoe
xt].add(to_token)
                                 if (str(to_token.lemma_)+str(to_token.i)+tfnwoext)
not in ocs_dict:
                                    ocs_dict[str(to_token.lemma_)+str(to_token.i)+t
fnwoext] = {token}
                                else:
                                    ocs_dict[str(to_token.lemma_)+str(to_token.i)+t
fnwoext].add(token)

            case 'obj' | 'obl' : # 目的語 (O1:モノ) obl は obj より弱い
                # out_file2.write('O...₩n')
                # out_file2.write(f'{token} → {to_token}₩n')
```

```
                    o_individuals.add(token)
                    if to_token.pos_ in ['VERB','ADJ']:
                        v_properties.add(to_token)
                                                                                #
vo_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_))
                        if str(to_token.lemma_) not in vo_dict:
                            vo_dict[str(to_token.lemma_)] = {token}
                        else:
                            vo_dict[str(to_token.lemma_)].add(token)
                                                                                #
ov_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))

                        # 2024/4/20
                        #if (str(token.lemma_)+str(token.i)) not in ov_dict:
                        #       ov_dict[str(token.lemma_)+str(token.i)] = {to_token}
                        if (str(token.lemma_)+str(token.i)+tfnwoext) not in ov_dict:
                                    ov_dict[str(token.lemma_)+str(token.i)+tfnwoext]   =
{to_token}
                        else:
                            #ov_dict[str(token.lemma_)+str(token.i)].add(to_token)
                             ov_dict[str(token.lemma_)+str(token.i)+tfnwoext].add(to_tok
en)
                        # ID の uniqueness のため分割ファイル名も ID に付加

                    elif to_token.pos_ in ['NOUN']: #  目的語の係り受け先が名詞の場合は
その名詞は動詞として扱うが、
                        #  文節を取り出し、動詞か形容詞が含まれていれば名詞の to_token
の代わりにそちらを優先する↓
                        j_tokens = [t for t in span_dep[1] if t.pos_ in ['VERB','ADJ']] #
リスト内包表記（述語トークン）
                        if len(j_tokens) > 0: # ↑for 文で毎回初期化している？
                            # out_file2.write(f' {to_token} を {j_tokens} に変更!¥n')
                                                                    to_token      =
j_tokens[0]                                                  # ↑
                            #  ここまでが係り受け先の文節検索
                            v_properties.add(to_token)
```

```
                                                                                #
vo_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_))
                            if str(to_token.lemma_) not in vo_dict:
                                vo_dict[str(to_token.lemma_)] = {token}
                            else:
                                vo_dict[str(to_token.lemma_)].add(token)
                                                                                #
ov_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))

                            # 2024/4/20
                            #if (str(token.lemma_)+str(token.i)) not in ov_dict:
                            #       ov_dict[str(token.lemma_)+str(token.i)] = {to_token}
                            #else:
                            #       ov_dict[str(token.lemma_)+str(token.i)].add(to_token)
                            if (str(token.lemma_)+str(token.i)+tfnwoext) not in ov_dict:
                                    ov_dict[str(token.lemma_)+str(token.i)+tfnwoext]  =
{to_token}
                            else:
                                ov_dict[str(token.lemma_)+str(token.i)+tfnwoext].add(to
_token)
                            # ID の uniqueness のため分割ファイル名も ID に付加

                case 'iobj': #  間接的目的語　(O2:ヒト)
                    out_file2.write('O2...₩n')
                    out_file2.write(f'{token} → {to_token}₩n')
                    o2_individuals.add(token)
                    if to_token.pos_ in ['VERB','ADJ']:
                        v_properties.add(to_token)
                                                                                #
vo2_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_))
                        if str(to_token.lemma_) not in vo2_dict:
                            vo2_dict[str(to_token.lemma_)] = {token}
                        else:
                            vo2_dict[str(to_token.lemma_)].add(token)
                                                                                #
o2v_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))
```

```
                    # 2024/4/20
                    #if (str(token.lemma_)+str(token.i)) not in o2v_dict:
                    #       o2v_dict[str(token.lemma_)+str(token.i)] = {to_token}
                    #else:
                    #       o2v_dict[str(token.lemma_)+str(token.i)].add(to_token)
                    if (str(token.lemma_)+str(token.i)+tfnwoext) not in o2v_dict:
                                o2v_dict[str(token.lemma_)+str(token.i)+tfnwoext]  =
{to_token}
                    else:
                        o2v_dict[str(token.lemma_)+str(token.i)+tfnwoext].add(to_to
ken)
                    # ID の uniqueness のため分割ファイル名も ID に付加

                elif to_token.pos_ in ['NOUN']: # 目的語の係り受け先が名詞の場合は
その名詞は動詞として扱うが、
                    # 文節を取り出し、動詞か形容詞が含まれていればそちらを
to_token の代わりに優先する↓
                    j_tokens = [t for t in span_dep[1] if t.pos_ in ['VERB','ADJ']] #
リスト内包表記
                    if len(j_tokens) > 0: # ↑for 文で毎回初期化している？
                        print(f'{to_token} → {j_tokens} !')
                        # ここまでが係り受け先の文節検索
                        v_properties.add(j_tokens[0])
                                                                                #
vo2_dict.setdefault(str(j_tokens[0].lemma_),[]).append(str(token.lemma_))
                        if str(j_tokens[0].lemma_) not in vo2_dict:
                            vo2_dict[str(j_tokens[0].lemma_)] = {token}
                        else:
                            vo2_dict[str(j_tokens[0].lemma_)].add(token)
                                                                                #
o2v_dict.setdefault(str(token.lemma_),[]).append(str(j_tokens[0].lemma_))

                        # 2024/4/20
                        #if (str(token.lemma_)+str(token.i)) not in o2v_dict:
                                #       o2v_dict[str(token.lemma_)+str(token.i)]  =
```

```
{j_tokens[0]}
                            #else:
                             #       o2v_dict[str(token.lemma_)+str(token.i)].add(j_tokens
[0])
                            if (str(token.lemma_)+str(token.i)+tfnwoext) not in o2v_dict:
                                  o2v_dict[str(token.lemma_)+str(token.i)+tfnwoext]  =
{j_tokens[0]}
                            else:
                                o2v_dict[str(token.lemma_)+str(token.i)+tfnwoext].add(j
_tokens[0])
                            # ID の uniqueness のため分割ファイル名も ID に付加

                case 'fixed'：# 限定子 で VERB/AUX を含み、obj/obl を含まない S+V+C の
時の V をキャッチ
                    # out_file2.write('Fixed…₩n')
                    # out_file2.write(f'{token} → {to_token}₩n')
                    # 自分（単語）が述語であるケースのみ
                    if token.pos_ in ['VERB','ADJ']:
                        # 自分が含まれる文節に目的語が含まれていないときのみに(既に
obj,obl で処理済みなので)
                        o_tokens = [t for t in span_dep[0] if t.dep_ in ['obj','obl']]
                        if len(o_tokens) == 0: # ↑for 文で毎回初期化している？
                            n_tokens = [t for t in span_dep[0] if t.pos_ in ['NOUN']] #
↑名詞トークンリスト
                            if len(n_tokens) > 0:
                                b_properties.add(token) # be 動詞に登録
                                #out_file2.write(f'Be 動詞:{token}₩n')
                                for nt in n_tokens:  # 名詞トークンリストの全ての見出
し型を追加
                                    c_individuals.add(nt)
                                    #out_file2.write(f'C 個体候補:{nt}₩n')
                                                                                #
bc_dict.setdefault(str(token.lemma_),[]).append(str(nt.lemma_))
                                    if str(token.lemma_) not in bc_dict:
                                        bc_dict[str(token.lemma_)] = {nt}
                                    else:
```

```
                                        bc_dict[str(token.lemma_)].add(nt)
                                # C+be 動詞の辞書登録
                                                                        #
cb_dict.setdefault(str(nt.lemma_),[]).append(str(token.lemma_))

                                # 2024/4/22 個体 ID にファイル名の追加漏れ　以下 4
行の代わり
                                #if (str(nt.lemma_)+str(nt.i)) not in cb_dict:
                                #       cb_dict[str(nt.lemma_)+str(nt.i)] = {token}
                                #else:
                                #       cb_dict[str(nt.lemma_)+str(nt.i)].add(token)
                                                                        #
bc_dict.setdefault(str(token.lemma_),[]).append(str(nt.lemma_))
                                if (str(nt.lemma_)+str(nt.i)+tfnwoext) not in cb_dict:
                                        cb_dict[str(nt.lemma_)+str(nt.i)+tfnwoext] =
{token}
                                else:
                                        cb_dict[str(nt.lemma_)+str(nt.i)+tfnwoext].add(
token)
                                # ID の uniqueness のため分割ファイル名も ID に付加

                                if str(token.lemma_) not in bc_dict:
                                        bc_dict[str(token.lemma_)] = {nt}
                                else:
                                        bc_dict[str(token.lemma_)].add(nt)
                        # C(補語)+be 動詞に対応する S(主語)の登録 to_token で
はなく係り受け先の文節の主辞が S に対応
                        # 2024/5/8 ↑は次善の策で正しくは soc_dict/ocs_dict 双
方でマッチした(1 対 1 対応)o/c に対応した s にすべき、以下 if else:を追加
                        s_set = {}
                        if (str(nt.lemma_)+str(nt.i)+tfnwoext) in soc_dict:
                                                s_set = {s for s in
ocs_dict[str(nt.lemma_)+str(nt.i)+tfnwoext] if nt in
soc_dict[str(s.lemma_)+str(s.i)+tfnwoext]}
                        stoken = span_dep[1].root
                        if len(s_set) != 0:
```

```
                                    stoken = s_set.pop()
                                s_individuals.add(stoken)
                                #s_individuals.add(span_dep[1].root)
                                #out_file2.write(f'→S:{span_dep[1].root}₩n')
                                out_file2.write(f'→S:{stoken}₩n')
                                # 2024/5/8
                                                                                #
sb_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_))

                                # 2024/4/22 個体 ID にファイル名の追加漏れ　以下 4 行
の代わり
                                                                                #if
(str(span_dep[1].root.lemma_)+str(span_dep[1].root.i)) not in sb_dict:
                                #       sb_dict[str(span_dep[1].root.lemma_)+str(span_d
ep[1].root.i)] = {token}
                                #else:
                                #       sb_dict[str(span_dep[1].root.lemma_)+str(span_d
ep[1].root.i)].add(token)
                                                                                #if
(str(span_dep[1].root.lemma_)+str(span_dep[1].root.i)+tfnwoext) not in sb_dict:
                                #       sb_dict[str(span_dep[1].root.lemma_)+str(span_d
ep[1].root.i)+tfnwoext] = {token}
                                #else:
                                #       sb_dict[str(span_dep[1].root.lemma_)+str(span_d
ep[1].root.i)+tfnwoext].add(token)
                                # ID の uniqueness のため分割ファイル名も ID に付加
                                                                                #
bs_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))
                                #if str(token.lemma_) not in bs_dict:
                                #       bs_dict[str(token.lemma_)] = {span_dep[1].root}
                                #else:
                                #       bs_dict[str(token.lemma_)].add(span_dep[1].root
)
                                    if (str(stoken.lemma_)+str(stoken.i))+tfnwoext not in
sb_dict: # →stoken
```

```
                                    sb_dict[str(stoken.lemma_)+str(stoken.i)+tfnwoext]
= {token} # →stoken
                                else:
                                    sb_dict[str(stoken.lemma_)+str(stoken.i)+tfnwoext].
add(token) # stoken
                                                                                    #
bs_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))
                                if str(token.lemma_) not in bs_dict:
                                    bs_dict[str(token.lemma_)] = {stoken} # root-
>stoken
                                else:
                                    bs_dict[str(token.lemma_)].add(stoken) #root-
>stoken
                                # 2024/5/8
                        case 'nmod' | 'compound' | 'nummod': # 名詞修飾語か合成語および数値
修飾語も　ここでは並列扱いのケースのみ
                            if str(token.lemma_) != '¥n' and str(to_token.lemma_) != '¥n' :
                                # out_file2.write('Nmod or Compound...¥n')
                                # out_file2.write(f'{token} → {to_token}¥n')
                                if len(words_bundles) == 0：# 初期状態
                                    words_bundles.append([token,to_token]) # 最初のペアをリ
ストとしてリストに追加
                                else:
                                    wb_found = False
                                    for wb_list in words_bundles:
                                        if len(wb_list) > 0：
                                            if wb_list[-1] == token：# 当該リストの末尾がペア
の先頭と同じ単語であれば（線形の係り受け）後ろに追加
                                                wb_list.append(to_token)
                                                wb_found = True
                                            elif wb_list[-1] == to_token：# 当該リストの末尾が
ペアの後ろと同じ単語であれば（係り受けの分岐）末尾の手前に先を挿入
                                                if wb_list[-2] != token：
                                                    wb_list.insert(-1,token)
                                                wb_found = True
                                    if wb_found is not True：# リストに新しいペアをリストして
```

追加

```
                                words_bundles.append([token,to_token])
                                wb_found = False #  初期化
            case 'case': # O1 と O2 の区別　'を' 'に'
                out_file.write('CASE...¥n')
                out_file.write(f'{token} → {to_token}¥n')
            case 'nummod': # data property の対象
                out_file.write('NUMMOD...¥n')
                out_file.write(f'{token} → {to_token}¥n')
            case _: #上記以外
                out_file.write(str(token.dep_) + '...¥n')
                out_file.write(f'{token} → {to_token}¥n')

    #（二つ目のループ↑）係り受け先からの逆引き　ROOT ケースのみ→ROOT からしかたどれない述語があるので残す（使ってないが...）→2024/5/8 文末の「である。」のケースがそれに相当
    # print('係り受け先から逆引き')
    for token in span_dep[1]:
        # 2024/5/8 以下 4 行も S-B-C ケースのため追加
        to_token = token.doc[token.head.i] #  文節を跨いでも主要部（とくに主語）にたどり着けるが主語ケースのみ同一となる
        j_tokens = [] #  毎回空リストに初期化↓　j は (名詞への係り受け元中で)「述語」の意　述語トークン
        o_tokens = [] #  毎回空リストに初期化↓　o は（限定子がいる文節中で)「目的語」の意　目的語トークン
        n_tokens = [] #  毎回空リストに初期化↓　n は（限定子がいる文節中で)「名詞」の意　名詞トークン
        #
        i_span_deps = list(token.lefts) + list(token.rights) #  係り受け元（文頭、文末方向)
         i_span_deps = [t for t in i_span_deps if str(t.pos_) != 'PUNCT' and (not re.match(r'¥s*¥n',str(t.lemma_)))] #  句読点と任意の空白が 0 以上＋改行を除く
        if token.dep_ == 'ROOT': # ROOT のみ自分を指す、宣言、定義文処理にかかわらず逆依存の文節 span_dep[1]→span_dep[0]をすべて集める
            # out_file2.write('ROOT...¥n')
            if token.pos_ in ['VERB','ADJ']:
```

```
                    # out_file2.write(f'{token} → {i_span_deps}¥n')

                    # 2024/4/20
                    #if (str(token.lemma_) + str(token.i)) in rvspan_dict: # Root Verb span 辞書(キー:ID 付き術語、値：重複を許さないリスト)
                                                #                rvspan_dict[str(token.lemma_)     + str(token.i)].update(i_span_deps) # update は集合の追加更新
                    #else:
                                    #          rvspan_dict[str(token.lemma_)   +   str(token.i)]   = set(i_span_deps) #  キーと値を新たに追加
                    if (str(token.lemma_) + str(token.i) + tfnwoext) in rvspan_dict: # Root Verb span 辞書(キー:ID 付き術語、値：重複を許さないリスト)
                                                rvspan_dict[str(token.lemma_)   +   str(token.i)   + tfnwoext].update(i_span_deps) # update は集合の追加更新
                    else:
                                    rvspan_dict[str(token.lemma_)  +  str(token.i)  +  tfnwoext]  = set(i_span_deps) #  キーと値を新たに追加
                    # ID の uniqueness のため分割ファイル名も ID に付加
          # 2024/5/8 新たに「である」ケース対応を追加
          else:
                if token.dep_ == 'fixed':
                    # 自分（単語）が述語であるケースのみ
                    if token.pos_ in ['VERB','ADJ']:
                                # 自分が含まれる文節に目的語が含まれていないときのみに(既に obj,obl で処理済みなので)
                        o_tokens = [t for t in span_dep[1] if t.dep_ in ['obj','obl']] # ↑ 0→1
                        if len(o_tokens) == 0: # ↑for 文で毎回初期化している？
                            n_tokens = [t for t in span_dep[1] if t.pos_ in ['NOUN']] # ↑名詞トークンリスト ↑  0→1
                            if len(n_tokens) > 0:
                                b_properties.add(token) # be 動詞に登録
                                #out_file2.write(f'Be 動詞:{token}¥n')
                                for nt in n_tokens:  # 名詞トークンリストの全ての見出し型を追加
                                    c_individuals.add(nt)
```

```
                                    #out_file2.write(f'C 個体候補:{nt}¥n')
                                                                                #
bc_dict.setdefault(str(token.lemma_),[]).append(str(nt.lemma_))
                                    if str(token.lemma_) not in bc_dict:
                                        bc_dict[str(token.lemma_)] = {nt}
                                    else:
                                        bc_dict[str(token.lemma_)].add(nt)
                                    # C+be 動詞の辞書登録
                                                                                #
cb_dict.setdefault(str(nt.lemma_),[]).append(str(token.lemma_))
                                    if (str(nt.lemma_)+str(nt.i) + tfnwoext) not in cb_dict:
                                        cb_dict[str(nt.lemma_)+str(nt.i) + tfnwoext] =
{token}
                                    else:
                                                                cb_dict[str(nt.lemma_)+str(nt.i) +
tfnwoext].add(token)
                                                                                #
bc_dict.setdefault(str(token.lemma_),[]).append(str(nt.lemma_))
                                    if str(token.lemma_) not in bc_dict:
                                        bc_dict[str(token.lemma_)] = {nt}
                                    else:
                                        bc_dict[str(token.lemma_)].add(nt)
                                # C(補語)+be 動詞に対応する S(主語)の登録 to_token で
はなく係り受け先の文節の主辞が S に対応も次善の策として以下
                                s_set = {}
                                if (str(nt.lemma_)+str(nt.i) + tfnwoext) in soc_dict:
                                    s_set = {s for s in ocs_dict[str(nt.lemma_)+str(nt.i)
+ tfnwoext] if nt in soc_dict[str(s.lemma_)+str(s.i) + tfnwoext]}
                                #print(s_set)
                                stoken = span_dep[0].root
                                if len(s_set) != 0:
                                    stoken = s_set.pop()
                                s_individuals.add(stoken)
                                # 2024/5/8
                                out_file2.write(f'→S:{stoken}¥n') # .root→stoken
                                                                                #
```

```
sb_dict.setdefault(str(to_token.lemma_),[]).append(str(token.lemma_))
                                if (str(stoken.lemma_)+str(stoken.i) + tfnwoext) not in
sb_dict:
                                    sb_dict[str(stoken.lemma_)+str(stoken.i) +
tfnwoext] = {token}
                                else:
                                    sb_dict[str(stoken.lemma_)+str(stoken.i) +
tfnwoext].add(token)
                                                                                #
bs_dict.setdefault(str(token.lemma_),[]).append(str(to_token.lemma_))
                                if str(token.lemma_) not in bs_dict:
                                    bs_dict[str(token.lemma_)] = {stoken} #
                                else:
                                    bs_dict[str(token.lemma_)].add(stoken) #
        # ↑ここまで 2024/5/8 S-B-C の追加
            # 2024/5/9 逆引きの時の連語も追加
            else:
                if token.dep_ in ['nmod','compound','nummod'] : # 名詞修飾語か合
成語　ここでは並列扱いのケースのみ->nummod を追加して case は別...
                    if str(token.lemma_) != '₩n' and str(to_token.lemma_) != '₩n' :
                        # out_file2.write('Nmod or Compound...₩n')
                        # out_file2.write(f'{token} → {to_token}₩n')
                        if len(words_bundles) == 0 : # 初期状態
                            words_bundles.append([token,to_token]) # 最初のペア
をリストとしてリストに追加
                        else:
                            wb_found = False
                            for wb_list in words_bundles:
                                if len(wb_list) > 0 :
                                    if wb_list[-1] == token : # 当該リストの末尾が
ペアの先頭と同じ単語であれば（線形の係り受け）後ろに追加
                                        wb_list.append(to_token)
                                        wb_found = True
                                    elif wb_list[-1] == to_token : # 当該リストの末
尾がペアの後ろと同じ単語であれば（係り受けの分岐）末尾の手前に先を挿入
                                        if wb_list[-2] != token :
```

```
                                                wb_list.insert(-1,token)
                                            wb_found = True
                                    if wb_found is not True：# リストに新しいペアをリストして追加
                                        words_bundles.append([token,to_token])
                                        wb_found = False # 初期化
                    # ↑ここまで 2024/5/9 逆引き連語の追加

# 共通「関係」(述語)「個体」辞書の作成 4 つの辞書の連結（値は集合和）
common_property_individuals_dict = {}
for k,v in vs_dict.items():
    if k in common_property_individuals_dict:
        common_property_individuals_dict[k] |= v
    else:
        common_property_individuals_dict[k] = v
for k,v in vo_dict.items():
    if k in common_property_individuals_dict:
        common_property_individuals_dict[k] |= v
    else:
        common_property_individuals_dict[k] = v
for k,v in vo2_dict.items():
    if k in common_property_individuals_dict:
        common_property_individuals_dict[k] |= v
    else:
        common_property_individuals_dict[k] = v
for k,v in bc_dict.items():
    if k in common_property_individuals_dict:
        common_property_individuals_dict[k] |= v
    else:
        common_property_individuals_dict[k] = v
# STEP3-1 共通「関係」辞書の値を共通に持つキー（「関係」）を、キーに共通「個体」、同類「関係」集合（2 次分類）に持つ共通「個体」辞書を作成する。
second_common_individual_properties_dict = {}
for k1 in common_property_individuals_dict:
    for k2 in common_property_individuals_dict:
        if k1 != k2 :
```

```
                #tlist = [(i,j) for i,j in zip(v1,v2) if i.lemma_ == j.lemma_] #　トークンの語幹が同じもの（ID は無視）
                # ↑だと網羅してない、片手落ち↓の４重ループが正しい
                for v1 in common_property_individuals_dict[k1]:
                    for v2 in common_property_individuals_dict[k2]:
                        if v1.lemma_ == v2.lemma_:
                                                                if str(v1.lemma_) in second_common_individual_properties_dict:
                                    second_common_individual_properties_dict[str(v1.lemma_)].add(k1)
                                    second_common_individual_properties_dict[str(v1.lemma_)].add(k2)
                                else:
                                    second_common_individual_properties_dict[str(v1.lemma_)] = set({k1,k2})

# STEP3-2 STEP3-2　(3-1 で値であった）同類「関係」集合に共通要素をもつ（3-1 でキーであった)「個体」集合を同類個体（= クラスの元）集合を値に持ち、
# 共通の「関係」をキーに持つ同類個体（= クラス候補）辞書(second_common_property_individuals_dict)を作成する。
second_common_property_individuals_dict = {}
for k1 in second_common_individual_properties_dict:
    for k2 in second_common_individual_properties_dict:
        if k1 != k2：
            for v1 in second_common_individual_properties_dict[k1]:
                for v2 in second_common_individual_properties_dict[k2]:
                    if v1 == v2:
                        if v1 in second_common_property_individuals_dict:
                            second_common_property_individuals_dict[v1].add(k1)
# 共通の「関係」をキーにして、値はそのときの「個体」の語幹の集合
                        second_common_property_individuals_dict[v1].add(k2)
                    else:
                                second_common_property_individuals_dict[v1] = set({k1,k2})

# 最終結果のファイル出力
```

```
out_file2.write(f'sv_dict:{sv_dict}¥n')
# 2024/4/21 改行の抑制のため以下 2 行を挿入
for vk,sv in vs_dict.items():
    vs_dict[vk]=  {v for v in sv if not re.match(r'¥n+', str(v))}
#
out_file2.write(f'vs_dict:{vs_dict}¥n')

# 2024/4/21 改行の抑制のため以下 2 行を挿入
for vk,ov in vo_dict.items():
    vo_dict[vk]=  {v for v in ov if not re.match(r'¥n+', str(v))}
#
out_file2.write(f'vo_dict:{vo_dict}¥n')

out_file2.write(f'ov_dict:{ov_dict}¥n')
out_file2.write(f'vo2_dict:{vo2_dict}¥n')
out_file2.write(f'o2v_dict:{o2v_dict}¥n')
out_file2.write(f'cb_dict:{cb_dict}¥n')
out_file2.write(f'bc_dict:{bc_dict}¥n')
out_file2.write(f'bs_dict:{bs_dict}¥n')
out_file2.write(f'sb_dict:{sb_dict}¥n')
out_file2.write(f'rvspan_dict:{rvspan_dict}¥n')
# 2024/5/8 以下 2 行追加
out_file2.write(f'ocs_dict:{ocs_dict}¥n')
out_file2.write(f'soc_dict:{soc_dict}¥n')
#

out_file2.write(f's_individual:¥n')
# 2024/4/21 改行の抑制のため以下の 3 行追加
words_bundles2 = []
for wlist in words_bundles:
    words_bundles2.append([w for w in wlist if not re.match(r'¥n+', str(w))])

for st in s_individuals:
    # S トークンが含まれる連語リストのうち先頭から直前までを返す↓
    #out_file2.write(f'{st} -> {[wlist[:wlist.index(st)] for wlist in words_bundles if st in
wlist]}¥n')
```

```
    # ↑のかわりに以下 1 行挿入してタブ付加　2 に変更
    if not re.match(r'¥n+', str(st)):
        # 2024/4/30 後処理のため個体は全て ID 付きに変更　リストのリスト (係り受けが複数ある場合もある：キーがトークンなので同じ個体から)
        #out_file2.write(f'{st} -> {[wlist[:wlist.index(st)] for wlist in words_bundles2 if st in wlist]}¥n')
                                out_file2.write(f'{str(st.lemma_)+str(st.i)+tfnwoext}        -> {[ [str(w.lemma_)+str(w.i)+tfnwoext for w in wl] for wl in [wlist[:wlist.index(st)] for wlist in words_bundles2 if st in wlist]]}¥n')

out_file2.write(f'c_individuals:¥n')
for ct in c_individuals:
    if not re.match(r'¥n+', str(ct)): # 2024/4/30 s_individuals に倣って追加
        # C トークンが含まれる連語リストのうち先頭から直前までを返す↓
        #out_file2.write(f'{ct} -> {[wlist[:wlist.index(ct)] for wlist in words_bundles if ct in wlist]}¥n')
                                out_file2.write(f'{str(ct.lemma_)+str(ct.i)+tfnwoext}        -> {[ [str(w.lemma_)+str(w.i)+tfnwoext for w in wl] for wl in [wlist[:wlist.index(ct)] for wlist in words_bundles2 if ct in wlist] ]}¥n')

#out_file2.write(f'v_properties:{v_properties}¥n')
# ↑トークンの集合なので同じ語幹の現が複数含まれるので出力は一つに抑制↓
out_file2.write(f'v_properties:{ {v.lemma_ for v in v_properties} }¥n')
#out_file2.write(f'b_properties:{b_properties}¥n')
out_file2.write(f'b_properties:{ {b.lemma_ for b in b_properties} }¥n')
out_file2.write(f'o_individuals:¥n')
# 2024/5/8 以下 c→o (間違えていた)
for ot in o_individuals:
    if not re.match(r'¥n+', str(ot)): # 2024/4/30 s_individuals に倣って追加
        # O トークンが含まれる連語リストのうち先頭から直前までを返す↓
        #out_file2.write(f'{ot} -> {[wlist[:wlist.index(ot)] for wlist in words_bundles if ot in wlist]}¥n')
                                out_file2.write(f'{str(ot.lemma_)+str(ot.i)+tfnwoext}        -> {[ [str(w.lemma_)+str(w.i)+tfnwoext for w in wl] for wl in [wlist[:wlist.index(ot)] for wlist in words_bundles2 if ot in wlist] ]}¥n')
```

```
# 2024/4/21 改行の抑制　以下を次の行(2)に変更
#out_file2.write(f'連語：{words_bundles}¥n')
out_file2.write(f'連語：{words_bundles2}¥n')
#

for ck,cv in common_property_individuals_dict.items():
    # 2024/4/22 個体 ID にファイル名の追加漏れ　以下 1 行の代わり
    #out_file2.write(f'共通「関係」(述語): {ck} ->「個体」集合:{ {str(cvi.lemma_)+str(cvi.i) if "接尾辞" not in str(cvi.tag_) else str(list(cvi.lefts)[-1].lemma_)+str(list(cvi.lefts)[-1].i)+str(cvi.lemma_)+str(cvi.i) for cvi in cv} }¥n')
        out_file2.write(f'共通「関係」(述語): {ck} ->「個体」集合:{ {str(cvi.lemma_)+str(cvi.i)+tfnwoext if "接尾辞" not in str(cvi.tag_) or len(list(cvi.lefts)) == 0 else str(list(cvi.lefts)[-1].lemma_)+str(list(cvi.lefts)[-1].i)+str(cvi.lemma_)+str(cvi.i)+tfnwoext for cvi in cv} }¥n')
for sck,scv in second_common_individual_properties_dict.items():
    #if sck != '¥n':
    if not re.match(r'¥n+', sck):
        out_file2.write(f'同類「個体」(名詞) :{sck} ->同類「関係」集合:{ { scvi for scvi in scv} }¥n')
for sck2,scv2 in second_common_property_individuals_dict.items():
    out_file2.write(f'同類「関係」(述語) : {sck2} -> 「クラス」候補集合 : {scv2}¥n')
    # 2024/4/29 SVO/SBC リストの出力追加(for IndividualMaker.py)
SVO_list = [] # [S 個体(ID 入力ファイル名), 関係(語幹), O 個体(ID 入力ファイル名)]からなるリスト
SBC_list = [] # [S 個体(ID 入力ファイル名), 関係(語幹), C 個体(ID 入力ファイル名)]からなるリスト
for s_k, v_v in sv_dict.items(): # ここは k,v いっぺんにとってきてよい
    # s_k は個体文字列になってしまっているので連語を引っ張るには s_k_token を復活させる必要がある 2024/9/20
    s_k_token = [s for s in s_individuals if s_k == str(s.lemma_)+str(s.i)+tfnwoext][0]
    for v_vi in v_v:
        if str(v_vi.lemma_) in vo_dict: # キーがない場合はエラーになるので
            for o_v in vo_dict[str(v_vi.lemma_)]: # 値は token だが、キーは語幹の文字列
                # 2024/5/9 連語リストに変更
                #SVO_list.append( [s_k, v_vi, o_v] )# 語幹や ID は出力時
```

```
                o_v_wlist = [wlist for wlist in words_bundles if o_v in wlist]
                s_k_wlist = [wlist for wlist in words_bundles if s_k_token in wlist] # 主語も連語にする必要あり 2024/9/20 以下の o_v_wlist の if else を残しつつ、入れ子で s_k_wlist の if else を入れ込む
                if len(o_v_wlist)==0：
                    # sv_dict でとってきた v と vo_dict でとってきた o_v がらみの連語の筆頭（目指す O 候補）をキーとして ov_dict の値集合に含まれれば、同一の V を仲立ちにしているので 2 行追加 7/8
                    if str(o_v.lemma_)+str(o_v.i)+tfnwoext in ov_dict:
                        if v_vi in ov_dict[str(o_v.lemma_)+str(o_v.i)+tfnwoext]:# 以下がオリジナルは 1 行なので、そこに s_k_wlist の if else を追加 2024/9/20
                            #SVO_list.append([s_k, v_vi, [o_v]])#9/20
                            if len(s_k_wlist)==0：
                                SVO_list.append([[s_k_token], v_vi, [o_v]])
                            else:
                                SVO_list.append( [ s_k_wlist[0], v_vi, [o_v]]) # S に連語を丸ごと登録 2024/9/20
                else:
                    # sv_dict でとってきた v と vo_dict でとってきた o_v がらみの連語の筆頭（目指す O 候補）をキーとして ov_dict の値集合に含まれれば、同一の V を仲立ちにしているので 2 行追加 7/8
                    if str(o_v.lemma_)+str(o_v.i)+tfnwoext in ov_dict: # キーにないとエラーになる
                        if v_vi in ov_dict[str(o_v.lemma_)+str(o_v.i)+tfnwoext]:# 以下がオリジナルは 1 行なので、そこに s_k_wlist の if else を追加 2024/9/20
                            #SVO_list.append( [ s_k, v_vi, o_v_wlist[0] ] ) # 登録は連語のまま
                            if len(s_k_wlist)==0：
                                SVO_list.append( [ [s_k_token], v_vi, o_v_wlist[0] ] )
                            else:
                                SVO_list.append( [ s_k_wlist[0], v_vi, o_v_wlist[0] ])
for s_k, b_v in sb_dict.items(): # ここは k,v いっぺんにとってきてよい
    # s_k は個体文字列になってしまっているので連語を引っ張るには s_k_token を復活させる必要がある 2024/9/20
    s_k_token = [s for s in s_individuals if s_k == str(s.lemma_)+str(s.i)+tfnwoext][0]
```

```
    for b_vi in b_v:
        if str(b_vi.lemma_) in bc_dict: # キーがない場合はエラーになるので
            for c_v in bc_dict[str(b_vi.lemma_)]: # 値は token だが、キーは語幹の文字にしないといけない
                # 2024/5/9 連語リストに変更
                #SBC_list.append( [s_k, b_vi, c_v] ) # 語幹や ID は出力時
                c_v_wlist = [wlist for wlist in words_bundles if c_v in wlist]
                s_k_wlist = [wlist for wlist in words_bundles if s_k_token in wlist] # 主語も連語にする必要あり 2024/9/20 以下の o_v_wlist の if else を残しつつ、入れ子で s_k_wlist の if else を入れ込む
                if len(c_v_wlist)==0 :
                    # sb_dict でとってきた b と bc_dict でとってきた c_が b らみの連語の筆頭（目指す C 候補）をキーとして cb_dict の値集合に含まれれば、同一の b を仲立ちにしているので 2 行追加 7/8
                    if str(c_v.lemma_)+str(c_v.i)+tfnwoext in cb_dict:
                        if b_vi in cb_dict[str(c_v.lemma_)+str(c_v.i)+tfnwoext]: # 以下がオリジナルは 1 行なので、そこに s_k_wlist の if else を追加 2024/9/20
                            #SBC_list.append( [s_k, b_vi, [c_v]] )
                            if len(s_k_wlist)==0 :
                                SBC_list.append( [ [s_k_token], b_vi, [c_v] ] )
                            else:
                                SBC_list.append( [s_k_wlist[0], b_vi, [c_v] ] )
                else:
                    # sb_dict でとってきた b と bc_dict でとってきた c_v をキーとして cb_dict の値集合に含まれれば、同一の b を仲立ちにしているので 2 行追加 7/8
                    if str(c_v.lemma_)+str(c_v.i)+tfnwoext in cb_dict:
                        if b_vi in cb_dict[str(c_v.lemma_)+str(c_v.i)+tfnwoext]:# 以下がオリジナルは 1 行なので、そこに s_k_wlist の if else を追加 2024/9/20
                            #SBC_list.append( [s_k, b_vi, c_v_wlist[0] ] ) # 登録は連語のまま
                            if len(s_k_wlist)==0 :
                                SBC_list.append( [ [s_k_token], b_vi, c_v_wlist[0] ] )
                            else:
                                SBC_list.append( [ s_k_wlist[0], b_vi, c_v_wlist[0] ])
```

```
# ファイル出力
# 2024/5/9 O を連語にしたので、連結へ(join を使う)
#out_file2.write(f'SVO_list: { ["["+li[0]+", "+str(li[1].lemma_)+", "+str(li[2].lemma_)+str(li[2].i)+tfnwoext+"]" for li in SVO_list] }₩n')
#out_file2.write(f'SVO_list: { ["["+li[0]+", "+str(li[1].lemma_)+", "+"".join([str(o.lemma_)+str(o.i)+tfnwoext for o in li[2]])+"]" for li in SVO_list] }₩n')#2024/9/20 以下 S もリスト join
out_file2.write(f'SVO_list: { ["["+"".join([str(s.lemma_)+str(s.i)+tfnwoext for s in li[0]])+", "+str(li[1].lemma_)+", "+ "".join([str(o.lemma_)+str(o.i)+tfnwoext for o in li[2]])+"]" for li in SVO_list] }₩n')
#out_file2.write(f'SBC_list: { ["["+li[0]+", "+str(li[1].lemma_)+", "+str(li[2].lemma_)+str(li[2].i)+tfnwoext+"]" for li in SBC_list] }₩n')
#out_file2.write(f'SBC_list: { ["["+li[0]+", "+str(li[1].lemma_)+", "+"".join([str(c.lemma_)+str(c.i)+tfnwoext for c in li[2]])+"]" for li in SBC_list] }₩n')#2024/9/20 以下 S もリスト join
out_file2.write(f'SBC_list: { ["["+"".join([str(s.lemma_)+str(s.i)+tfnwoext for s in li[0]])+", "+str(li[1].lemma_)+", "+ "".join([str(c.lemma_)+str(c.i)+tfnwoext for c in li[2]])+"]" for li in SBC_list] }₩n')

# 出力ファイルクローズ
out_file.close()
out_file2.close()
```

LLM を用いた形態素分解ライブラリによる SVO/SVC リスト生成 Python コード

図 15 に分割された入力テキストの一部を、図 16 に結果として出力された SVO/SVC リストを示す。

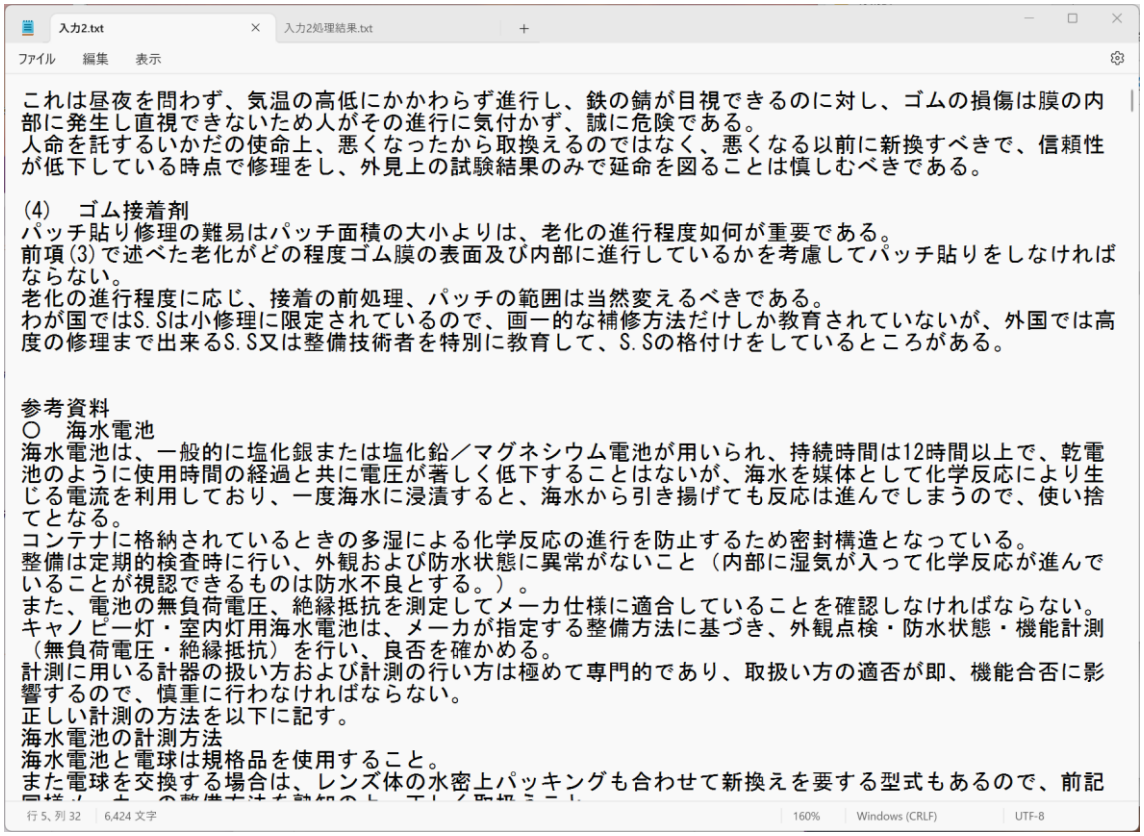


図 15　入力 2.txt の一部

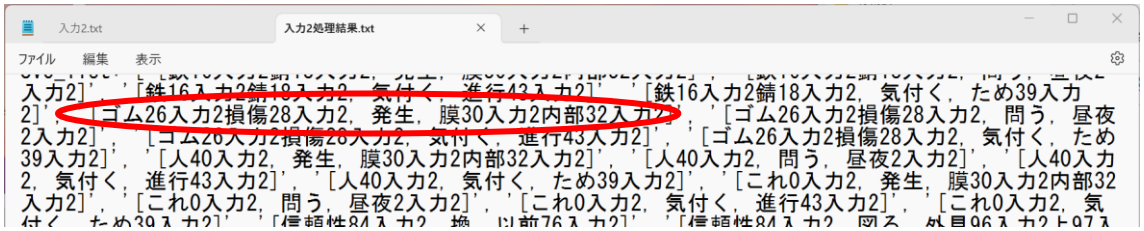


図 16 入力 2.txt に対する出力（SVO/SVC リスト）の一部

図 16 内の赤丸内の[ゴム 26 入力 2 損傷 28 入力 2, 発生, 膜 30 入力 2 内部 32 入力 2] は入力 2.txt 内の「ゴム」という単語にユニークな（その位置だけの）ID(26)が振られ、「ゴム」の「損傷」（修飾語であるゴムがついた損傷）が主語(S)、「発生」が動詞（V)、「膜」の「内部」が目的語（O）であることを示している。このように英語と異なり、日本語は一文の中に複数（入子状に）SVO/SVC の関係を含むことが特徴となっており、現状の LLM を使ってオントロジーを自動抽出するのが困難である要因となっている[8]。

[8] 幾つかの LLM モデルを使ってオントロジーを自動抽出を難しくしている。実際に OpenAI の API である OntoGPT を試したが、一度英語に翻訳してから抽出しているので要約はできても細かい日本語における SVO/SVC 抽出はできなかった

この前処理により、主語（S）および目的語（O)、補語（C）にあたる名詞は「個体」[9]として、一連の修飾語を含めて自動抽出でき、動詞（V）はそれら「個体」どうしを結びつける「関係」として自動抽出できる。一方「個体」の集合である「クラス」においては、日本語の特徴である一番最後の単語（それ以前はすべて修飾語）を「クラス」の名前として生成するにとどめた。上記の例であれば、「ゴム 26 入力 2 損傷 20 入力 2」が「個体」として登録され、その末尾である「損傷」という「クラス」が生成されることとなる。これは「損傷」という概念が確かにあり、他でも使われる（他に「損傷」の個体がいくつかある可能性があるので、その集合体としての「損傷」クラスを新たに登録するので、意味のあることだがもっとも簡単な（抽象化がされてない)「クラス」なので「自明なクラス」と呼んでいる。このようなロジックを組んだオントロジー自動生成(テキストファイルを入力とし、OWL ファイルを出力する)Python コードを以下に載せる。

なお OWL ファイル（Web Ontology Language ファイル）とは、オントロジーを記述するために使われるファイル形式である。オントロジーは、ある領域における概念とその間の関係を明確に定義したものを指し、OWL は、これらのオントロジーを表現し、情報を共有・再利用するための標準化された方法を提供する。

主な用途としては、以下のようなものがある：

知識ベースの構築：様々な分野での知識を体系的に整理し、コンピュータが理解できる形で表現します。

セマンティックウェブ：ウェブ上のデータを意味論的に関連付けることで、検索や情報の統合をより効率的に行えるようにします。

データの相互運用性：異なるシステムやデータベース間での情報交換を容易にします。

OWL は、RDF（Resource Description Framework）や XML などの他のウェブ標準と組み合わせて使用されることが多く、特にセマンティックウェブ技術の一環として重要視されている

```
from tkinter import filedialog
#import textfile
import re
import pandas as pd
import os
import gc

# 先頭の数字ではなく、後ろから 2 回数値（最初はファイル番号、次は ID)を読み飛ばした単語(先頭に来るか、次の数値になったら終わり）（個体→クラス）
def extract_non_numeric_lastword(input_string):
```

[9] 文章に現れた個々の事例として同じ単語でも ID と出現ファイル（入力 2 など）を区別している

```
    non_numeric_prefix = ""
    next_kouho = ""
    return_word = ""
    slen = len(input_string)
    if slen == 0:
        return non_numeric_prefix
    numeric_count = 0
    digit_flag = False
    while(slen > 0):
        slen -= 1
        if  input_string[slen].isdigit():
            digit_flag = True
        else:
            if digit_flag:
                digit_flag = False
                numeric_count += 1
            if numeric_count == 2:
                non_numeric_prefix += input_string[slen]
            if numeric_count == 4:
                next_kouho += input_string[slen]
    return_word = non_numeric_prefix[::-1]
    if next_kouho != "":
        next_kouho = next_kouho[::-1]
        if return_word == "等" or return_word == "など" or return_word == "ら" or
return_word == "他" or return_word =="ほか":
            return_word = next_kouho

    return return_word

# インポート先の　SKY/ものづくりサンプルオントロジー.owl を想定
i_owl_filename = filedialog.askopenfilename(
    title = "入力 owl ファイル(ものづくりサンプルオントロジー)を開きます",
    filetypes = [('OWL file',".owl")],
    initialdir = './'
)
in_file = open(i_owl_filename, 'r', encoding = 'utf-8')
```

```
in_lines = in_file.readlines()
in_file.close

# 出力先の　./自動生成サンプルオントロジーV2.owl を想定
# ([ひな形オントロジー].owl をベースに都度別名で保存
# 予め　ヘッダ（含む[]内のオントロジー名、OP,DP,CL,ID の各セクションがあることが前提
o_owl_filename = filedialog.askopenfilename(
    title = "出力先の（既存の）owl ファイルを開きます",
    filetypes = [('OWL file',".owl")],
    initialdir = './'
)
out_file = open(o_owl_filename, 'r', encoding = 'utf-8') # 各セクションの行をしりたいので'r'
out_lines = out_file.readlines()
out_file.close

# print(out_lines)
# import した OWL ファイルの各セクションをサーチ
op_section_start = [i for i, line in enumerate(out_lines) if '// Object Properties' in line][0]
dp_section_start = [i for i, line in enumerate(out_lines) if '// Data properties' in line][0]
class_start = [i for i, line in enumerate(out_lines) if '// Classes' in line][0]
i_section_start = [i for i, line in enumerate(out_lines) if '// Individuals' in line][0]

buffer_lines = 4
# リストの 0 個目から換算
sec_list = [op_section_start, dp_section_start, class_start, i_section_start]
sorted_sec_list = sorted(sec_list)
op_i = [i for i, val in enumerate(sorted_sec_list) if op_section_start == val][0]
op_next = len(out_lines)-2 # デフォルトで最初は最終行
if op_i < len(sorted_sec_list)-1:
    op_next = sorted_sec_list[op_i+1]-buffer_lines # ここで次のセクションの手前に直している
op_range = range(op_section_start + buffer_lines, op_next) # 次に書き始める行、から終わりまで
```

```
## Property Sction
# SVO 解析の処理結果から
textfilename = filedialog.askopenfilename(
    title = "「関係」のために Merged 処理結果 txt ファイルを開きます",
    filetypes = [('Text file',".txt")],
    initialdir = './'
)
res_file = open(textfilename, 'r', encoding = 'utf-8')

res_lines = res_file.readlines()
res_file.close

#述語の行(v/b_properties:から始まる)を取り出す
p_lines = [l for l in res_lines if 'properties' in l]
#p_list = [] #重複を除くために集合に変える
p_set = set()
for pl in p_lines:
    #print(pl)
    res = re.search(r'properties:{.*}',pl)
    res = res.group()
    new_res= res.replace('properties:{','').replace("'",'').replace('}','')
    new_res = re.sub(r"₩s","",new_res)
    l = new_res.split(',')
    for li in l:
        p_set.add(li)
#print(p_set)

# 「関係」編入辞書の読み込み
csv_filename = filedialog.askopenfilename(
    title = "「関係」編入辞書ファイルを開きます",
    filetypes = [('CSV file', ".csv")],
    initialdir = './'
)
#csv_file = open(csv_filename, "r", encoding="utf-8-sig") #BOM あり UTF8 ので₩ufeff
を除去
```

```
#csv_r = csv.DictReader(csv_file, delimiter = "," )
#print("カラム名",csv_r.fieldnames)
#for r in csv_r:
    #print(r['元'],r['先'],r['型'])
df = pd.read_csv(csv_filename)
#print(df['元'])
# 述語の登録
version = "2024/1/" # for 品管協会様向け自動生成オントロジー
imported_version = "2021/9/"
title = "自動生成オントロジーV2"
imported_title = "ものづくりオントロジー"

# ToDo 逆関係(+される)も自動生成しよう→人によるメンテナンス（の量にしぼれるし、自動生成されたものからもできる、その方がよいオントロジーになることから）で生成(Protege 側)

p_pointer = op_range.start+1
for p in p_set:
    # 毎回　コメントに始まり
    out_lines.insert(p_pointer,'<!--  http://www.semanticweb.org/info/ontologies/'  + version + title + '#' + p + ' -->')
    p_pointer += 1
    out_lines.insert(p_pointer,'₩n')
    p_pointer += 1
    p_range = range(p_pointer,op_range.stop+2) # 2 行たしたので、stop も更新
    #
    bool_index = df['元'] == p
    result = df[bool_index]
    #print(result)
    if result.empty == False: # 編入辞書にある場合 以下に続くので第 1 行目は>で終わる
        out_lines.insert(p_pointer,                    '<owl:ObjectProperty rdf:about="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+p+'">')
        p_pointer += 1
        out_lines.insert(p_pointer,'₩n')
        p_pointer += 1
```

```
            op_range = range(p_pointer,op_range.stop+2)
            for i,r in result.iterrows():
                match r['型']:
                    case '子':
                        out_lines.insert(p_pointer,'₩t<rdfs:subPropertyOf
rdf:resource="http://www.semanticweb.org/info/ontologies/'+imported_version+impor
ted_title+'#'+r['先']+'"/>')
                        p_pointer += 1
                        out_lines.insert(p_pointer,'₩n')
                        p_pointer += 1
                        op_range = range(p_pointer,op_range.stop+2)
                    case '同':
                        out_lines.insert(p_pointer,'₩t<owl:equivalentProperty
rdf:resource="http://www.semanticweb.org/info/ontologies/'+imported_version+impor
ted_title+'#'+r['先']+'"/>')
                        p_pointer += 1
                        out_lines.insert(p_pointer,'₩n')
                        p_pointer += 1
                        op_range = range(p_pointer,op_range.stop+2)
                    case '同 2':
                        out_lines.insert(p_pointer,'₩t<owl:equivalentProperty
rdf:resource="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+r['　先
']+'"/>')
                        p_pointer += 1
                        out_lines.insert(p_pointer,'₩n')
                        p_pointer += 1
                        op_range = range(p_pointer,op_range.stop+2)
            # 編入辞書にあれば必ず以下で閉じる
            out_lines.insert(p_pointer, '</owl:ObjectProperty>')
            p_pointer += 1
            out_lines.insert(p_pointer,'₩n')
            p_pointer += 1
            op_range = range(p_pointer,op_range.stop+2)
        else: # 無い場合　1 行で終わるので末尾が/>で終わる
            out_lines.insert(p_pointer,                                    '<owl:ObjectProperty
rdf:about="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+p+'"/>')
```

```
            p_pointer += 1
            out_lines.insert(p_pointer,'₩n')
            p_pointer += 1
            op_range = range(p_pointer,op_range.stop+2)

del p_set

#csv_file.close()

# 次のセクションのため再検索の必要あり
op_section_start = [i for i, line in enumerate(out_lines) if '// Object Properties' in line][0]
dp_section_start = [i for i, line in enumerate(out_lines) if '// Data properties' in line][0]
class_start = [i for i, line in enumerate(out_lines) if '// Classes' in line][0]
i_section_start = [i for i, line in enumerate(out_lines) if '// Individuals' in line][0]
sec_list = [op_section_start, dp_section_start, class_start, i_section_start]
sorted_sec_list = sorted(sec_list)
dp_i = [i for i, val in enumerate(sorted_sec_list) if dp_section_start == val][0]
dp_next = len(out_lines)-1 # 最終行
if dp_i < len(sorted_sec_list)-2:
    dp_next = sorted_sec_list[dp_i+1]-buffer_lines
dp_range = range(dp_section_start + buffer_lines, dp_next)
cl_i = [i for i, val in enumerate(sorted_sec_list) if class_start == val][0]
cl_next = len(out_lines)-1 # 最終行
if cl_i < len(sorted_sec_list)-2:
    cl_next = sorted_sec_list[cl_i+1]-buffer_lines
cl_range = range(class_start + buffer_lines, cl_next)
id_i = [i for i, val in enumerate(sorted_sec_list) if i_section_start == val][0]
id_next = len(out_lines)-1 # 最終行
if id_i < len(sorted_sec_list)-2:
    id_next = sorted_sec_list[id_i+1]-buffer_lines
id_range = range(i_section_start + buffer_lines, id_next)

## Inidividual Section

# MergedSVOlist.txt から S と O を個体登録、その際に V の関係も使って関係先も挿入
```

```
SVOlistfilename = filedialog.askopenfilename(
    title = "「個体」のために MergedSVOlist.txt ファイルを開きます",
    filetypes = [('Text file',".txt")],
    initialdir = './'
)
SVO_file = open(SVOlistfilename, 'r', encoding = 'utf-8')
SVO_lines = SVO_file.readlines()
SVO_file.close

#SVOlist から(SVO/SBC)_list:から始まる行を取り出す
SVO_lines = [l for l in SVO_lines if '_list' in l]
SVO_list = []
for SVOl in SVO_lines:
    #print(SVOl)
    res= SVOl.replace("SVO_list: ['",'').replace("SBC_list: ['",'') # 先頭分からいらないものをとる
    res = res.replace("']", '') # 行末処理
    tuple_list = res.split("', '") # list の list なので、外側のリストをばらばらに
    tuple_list = [tup.replace('[','').replace(']','').split(', ') for tup in tuple_list ]
    #res = re.sub(r"₩s","",res)
    SVO_list = SVO_list + tuple_list
#print(SVO_list[-1])

# 「個体」の書き込み
i_pointer = id_range.start+1
# for Class
class_set = set()

for SVO in SVO_list:
    S_str = SVO[0]
    V_str = SVO[1]
    O_str = SVO[2]
    S_str = S_str.replace('₩n','').replace('₩₩n','').replace('₩t','').replace('₩₩t','').replace('₩u3000','').replace('#','').replace('₩₩u3000','').replace('₩₩','').replace('"','').replace('>','').replace('<','') # 改行、タブ、全角空白の削除
```

```
    V_str = V_str.replace('₩n','').replace('₩₩n','').replace('₩t','').replace('₩₩t','').replace('₩u3000','').replace('#','').replace('₩₩u3000','').replace('₩₩','').replace('"','').replace('>','').replace('<','')
    O_str = O_str.replace('₩n','').replace('₩₩n','').replace('₩t','').replace('₩₩t','').replace('₩u3000','').replace('#','').replace('₩₩u3000','').replace('₩₩','').replace('"','').replace('>','').replace('<','')
    # Class 生成のため先頭の数値以外を取り出す（数値自体はクラス化しない）,ここの個体には Type としてクラス登録、あとで一括してクラス生成
    S_class_word = extract_non_numeric_lastword(S_str)
    O_class_word = extract_non_numeric_lastword(O_str)

    # 空白や₩から始まる単語ははずす 数字もデータとしていれない。
    if len(S_class_word)>0 and S_class_word[0]!='₩₩':
        # もしSがすでにある「個体」であればそこに属性として追加する。なければ「個体」毎追加
        class_set.add(S_class_word) # クラス候補
        if S_str not in out_lines: # 新規登録
            #print(S_class_word)
            out_lines.insert(i_pointer,'<!-- http://www.semanticweb.org/info/ontologies/' + version + title + '#' + S_str + ' -->')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
            id_range = range(i_pointer,id_range.stop+2)
            #  本体
            out_lines.insert(i_pointer, '<owl:NamedIndividual rdf:about="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+ S_str +'">')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
            id_range = range(i_pointer,id_range.stop+2)
            # 所属クラス
            out_lines.insert(i_pointer, '₩t<rdf:type rdf:resource="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+ S_class_word +'"/>')
```

```
                i_pointer += 1
                out_lines.insert(i_pointer,'₩n')
                i_pointer += 1
                id_range = range(i_pointer,id_range.stop+2)
                # 関係先の挿入、ひな形の出力 OWL のヘッダに省略(this_ont)のフルパスがある、フルパスで書かないと名前空間が解決できずに Object Property として認識されずに
                if len(O_class_word)>0 and O_class_word[0]!='₩₩': # 数字か₩で始まらない場合のみ
                    class_set.add(O_class_word) # クラス候補
                    out_lines.insert(i_pointer,'₩t<this_ont:'+V_str+' rdf:resource="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+O_str+'"/>')
                    i_pointer += 1
                    out_lines.insert(i_pointer,'₩n')
                    i_pointer += 1
                    id_range = range(i_pointer,id_range.stop+2)
                # 以下で閉じる
                out_lines.insert(i_pointer, '</owl:NamedIndividual>')
                i_pointer += 1
                out_lines.insert(i_pointer,'₩n')
                i_pointer += 1 # 次に書く行
                id_range = range(i_pointer,id_range.stop+2)
            else: # S 個体が既にある場合 元に戻るので range を使う
                if len(O_class_word)>0 and O_class_word[0]!='₩₩': # ₩で始まらない場合のみ
                    # ある場所に行って関係先だけ挿入
                    S_found = [i for i, line in enumerate(out_lines) if S_str in line][0]
                    i_pointer = S_found +2 #コメント行とスタート行の後に
                    out_lines.insert(i_pointer,'₩t<this_ont:'+V_str+' rdf:resource="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+O_str+'"/>')
                    i_pointer += 1
                    out_lines.insert(i_pointer,'₩n')
                    #i_pointer += 1 # ここが違う！
                    i_pointer = id_range.start + 2 # ここで未記入の行に戻す（2 行追加しているので）
```

```
                    id_range = range(i_pointer,id_range.stop+2)

    # 空白や₩から始まる単語ははずす 数字はスルーだが type はなし（後）
    if len(O_class_word)>0 and O_class_word[0]!='₩₩':
        class_set.add(O_class_word) # クラス候補
         # O 個体の登録 関係先の追加は逆関係を定義してから以下では新規登録のみ、すでにあれば何もしない
        if O_str not in out_lines：# 新規登録
            #print(O_class_word)
            out_lines.insert(i_pointer,'<!-- http://www.semanticweb.org/info/ontologies/' + version + title + '#' + O_str + ' -->')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
            id_range = range(i_pointer,id_range.stop+2)
            #本体
            out_lines.insert(i_pointer, '<owl:NamedIndividual rdf:about="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+O_str+'">')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
            id_range = range(i_pointer,id_range.stop+2)
             # 所属クラス
            out_lines.insert(i_pointer, '₩t<rdf:type rdf:resource="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+ O_class_word +'"/>')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
            id_range = range(i_pointer,id_range.stop+2)
            # 以下で閉じる
            out_lines.insert(i_pointer, '</owl:NamedIndividual>')
            i_pointer += 1
            out_lines.insert(i_pointer,'₩n')
            i_pointer += 1
```

```
                id_range = range(i_pointer,id_range.stop+2)
del SVO_list

# 次のセクションのため再検索の必要あり
op_section_start = [i for i, line in enumerate(out_lines) if '// Object Properties' in line][0]
dp_section_start = [i for i, line in enumerate(out_lines) if '// Data properties' in line][0]
class_start = [i for i, line in enumerate(out_lines) if '// Classes' in line][0]
i_section_start = [i for i, line in enumerate(out_lines) if '// Individuals' in line][0]
sec_list = [op_section_start, dp_section_start, class_start, i_section_start]
sorted_sec_list = sorted(sec_list)
dp_i = [i for i, val in enumerate(sorted_sec_list) if dp_section_start == val][0]
dp_next = len(out_lines)-1 # 最終行
if dp_i < len(sorted_sec_list)-2:
    dp_next = sorted_sec_list[dp_i+1]-buffer_lines
dp_range = range(dp_section_start + buffer_lines, dp_next)
cl_i = [i for i, val in enumerate(sorted_sec_list) if class_start == val][0]
cl_next = len(out_lines)-1 # 最終行
if cl_i < len(sorted_sec_list)-2:
    cl_next = sorted_sec_list[cl_i+1]-buffer_lines
cl_range = range(class_start + buffer_lines, cl_next)
id_i = [i for i, val in enumerate(sorted_sec_list) if i_section_start == val][0]
id_next = len(out_lines)-1 # 最終行
if id_i < len(sorted_sec_list)-2:
    id_next = sorted_sec_list[id_i+1]-buffer_lines
id_range = range(i_section_start + buffer_lines, id_next)

## Class section

c_pointer = cl_range.start + 1
for c in class_set:
    out_lines.insert(c_pointer,'<!--  http://www.semanticweb.org/info/ontologies/'  +
version + title + '#' + c + ' -->')
    c_pointer += 1
    out_lines.insert(c_pointer,'₩n')
    c_pointer += 1
```

```
    cl_range = range(c_pointer,cl_range.stop+2)
    #本体
    out_lines.insert(c_pointer,                                    '<owl:Class
rdf:about="http://www.semanticweb.org/info/ontologies/'+version+title+'#'+ c +'">')
    c_pointer += 1
    out_lines.insert(c_pointer,'₩n')
    c_pointer += 1
    cl_range = range(c_pointer,cl_range.stop+2)
    # 以下で閉じる
    out_lines.insert(c_pointer, '</owl:Class>')
    c_pointer += 1
    out_lines.insert(c_pointer,'₩n')
    c_pointer += 1
    cl_range = range(c_pointer,cl_range.stop+2)

del class_set

# 最後がなくなっていたら以下で閉じる
#if '</rdf:RDF>' not in out_lines:
#     last_pointer = len(out_lines) -1
#     out_lines.insert(last_pointer, "</rdf:RDF>")
#     out_lines.insert(last_pointer+1,"<!-- Generated by Office S.K.Y python code --
>")
#どうも memory ぶそくで書き出しに失敗しているようなので
gc.collect()

# OWL 出力
base_file_name = os.path.basename(o_owl_filename)
file_name_wo_extention = os.path.splitext(base_file_name)[0]
out_file = open(file_name_wo_extention + "新" + '.owl', 'w+', encoding = 'utf-8')
out_file.writelines(out_lines)
out_file.close
```

「個体」「関係」および自明な「クラス」を OWL ファイルとして出力するオントロジー自動生成 Python コード

このコードにより自動生成されたオントロジー(OWL ファイル)をフリーソフトウェア

Protégé によって開いた画面を図 17、19～21 に示す。図 17 は初期画面で、「クラス」（図中の class count）が 1693 個、「個体」が 6069 個（図中の Individual count）、「関係」が 1308 個（図中の Object property count）あることを示している。なお上位オントロジー（一般的によく使うひな形のオントロジー）としてすでに手動により作成したものづくりオントロジーを読み込んでおり、「クラス」は 159 個 、「個体」は 92 個、「関係」は 29 個なので（図 18）新たに自動生成されたのは「クラス」で 1534 個、「個体」は 5977 個、「関係」は 1279 個となる。

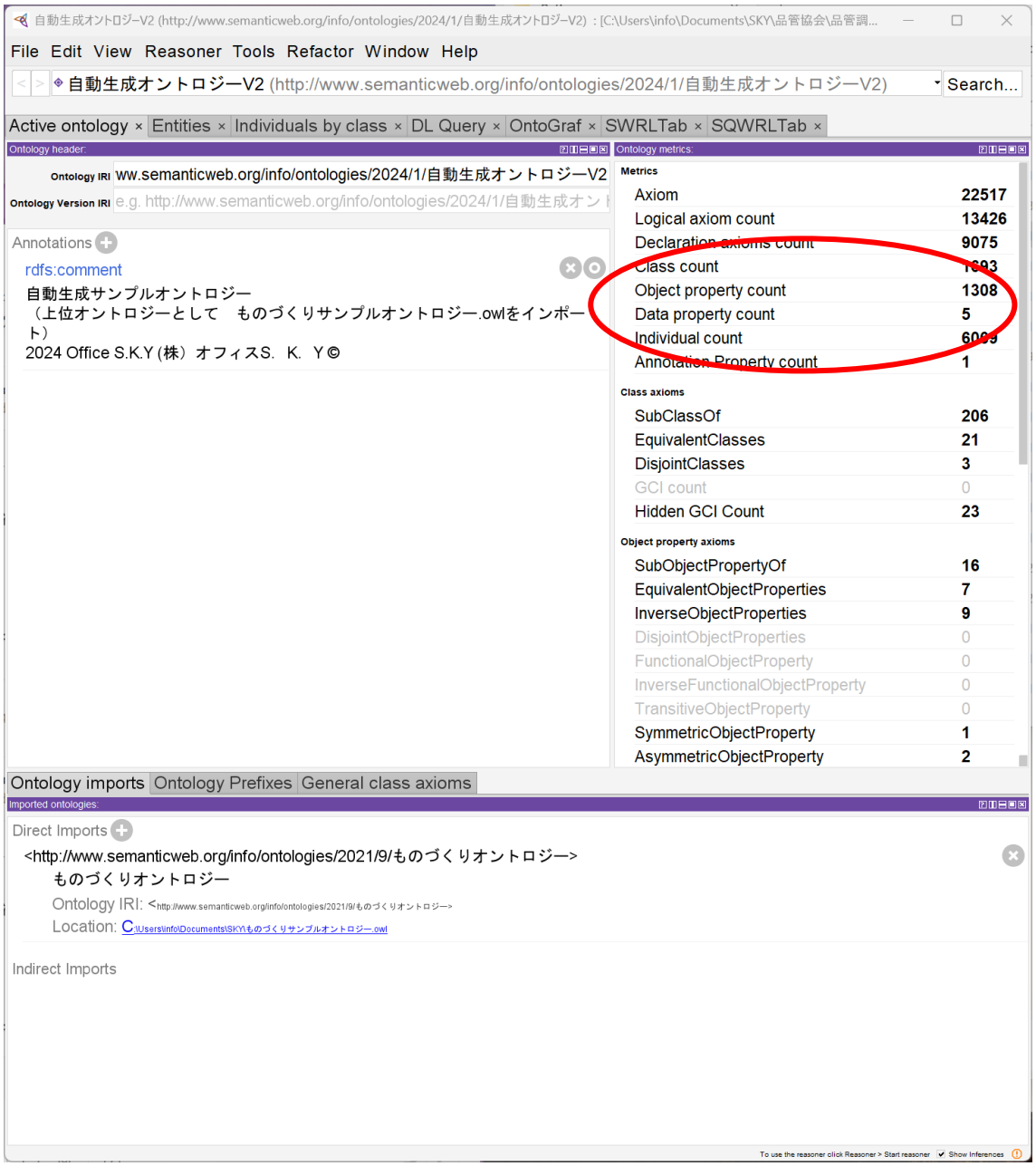


図 17 自動生成オントロジー（フリーのオントロジーエディターProtégé の初期画面）

（株）オフィス S．K．Y

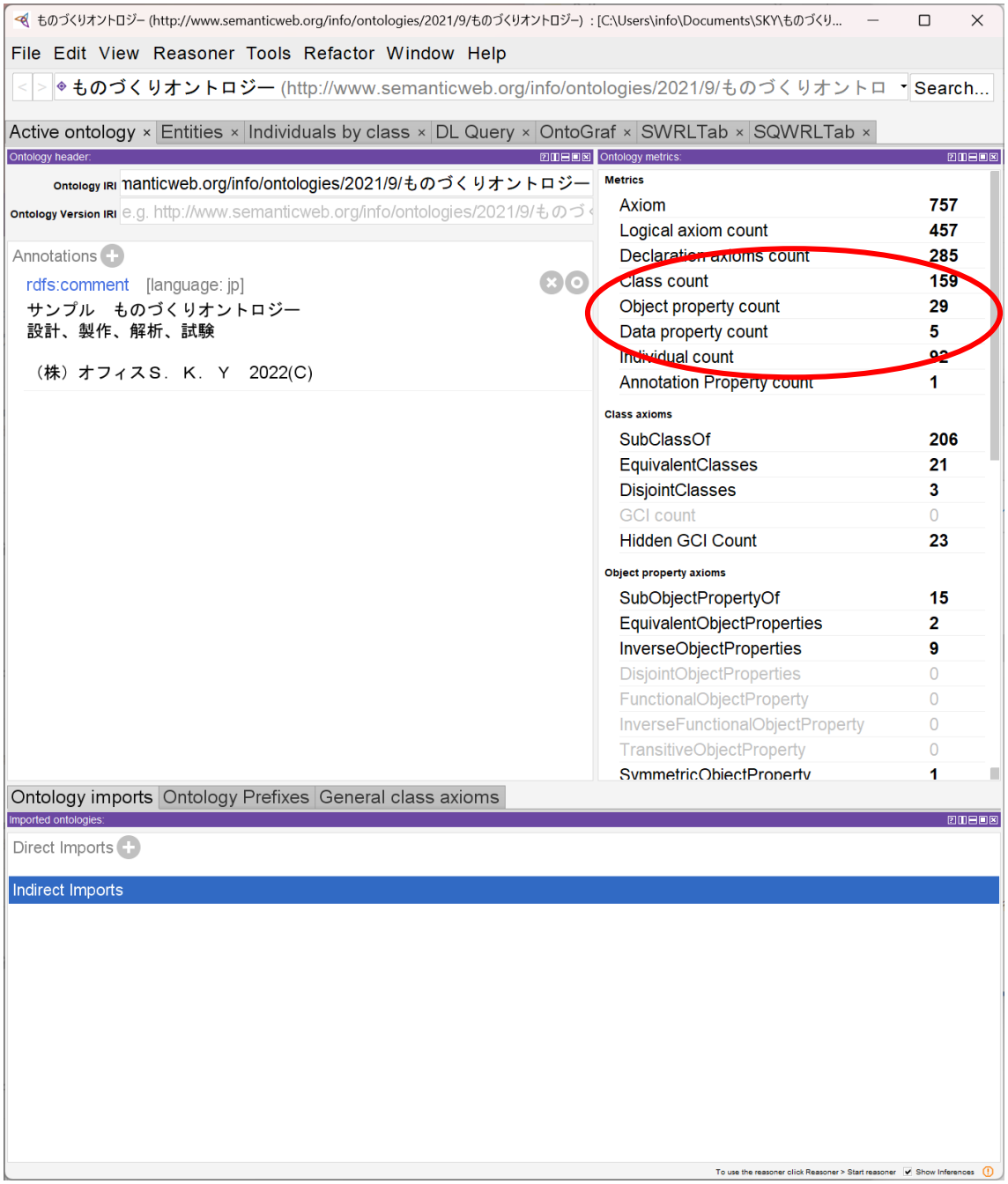


図 18　上位オントロジー（予め作成してあるひな形オントロジー）

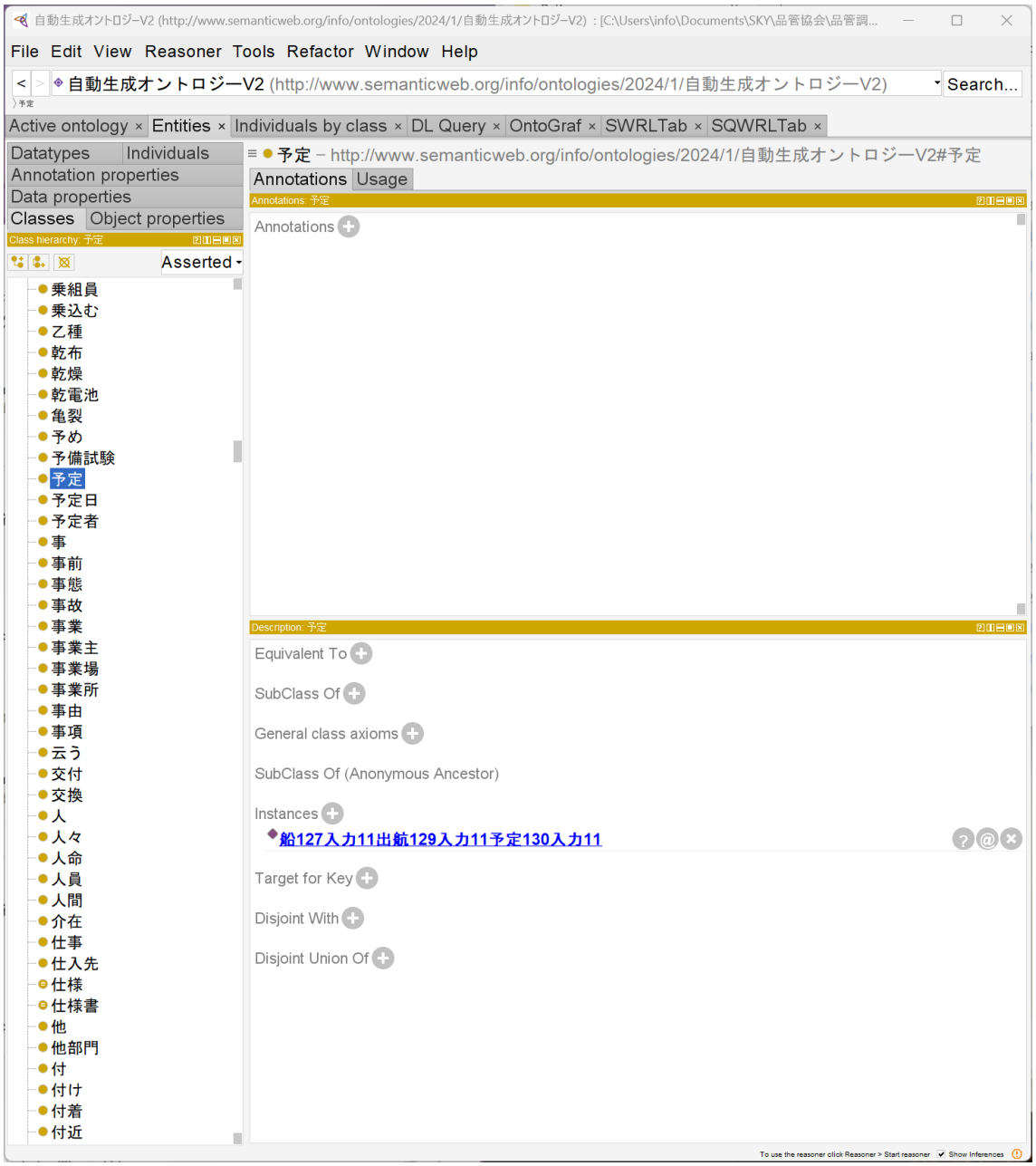


図 19 自動生成オントロジー 「クラス」(一部)

図 19 で左が「クラス」のトリーだが、まだ階層構造はない。右は選択されている「予定」クラスが「船 127 入力 11 出航 129 入力 11 予定 130 入力 11」という「個体」を持っていることを表している。図 20 がその個体である。

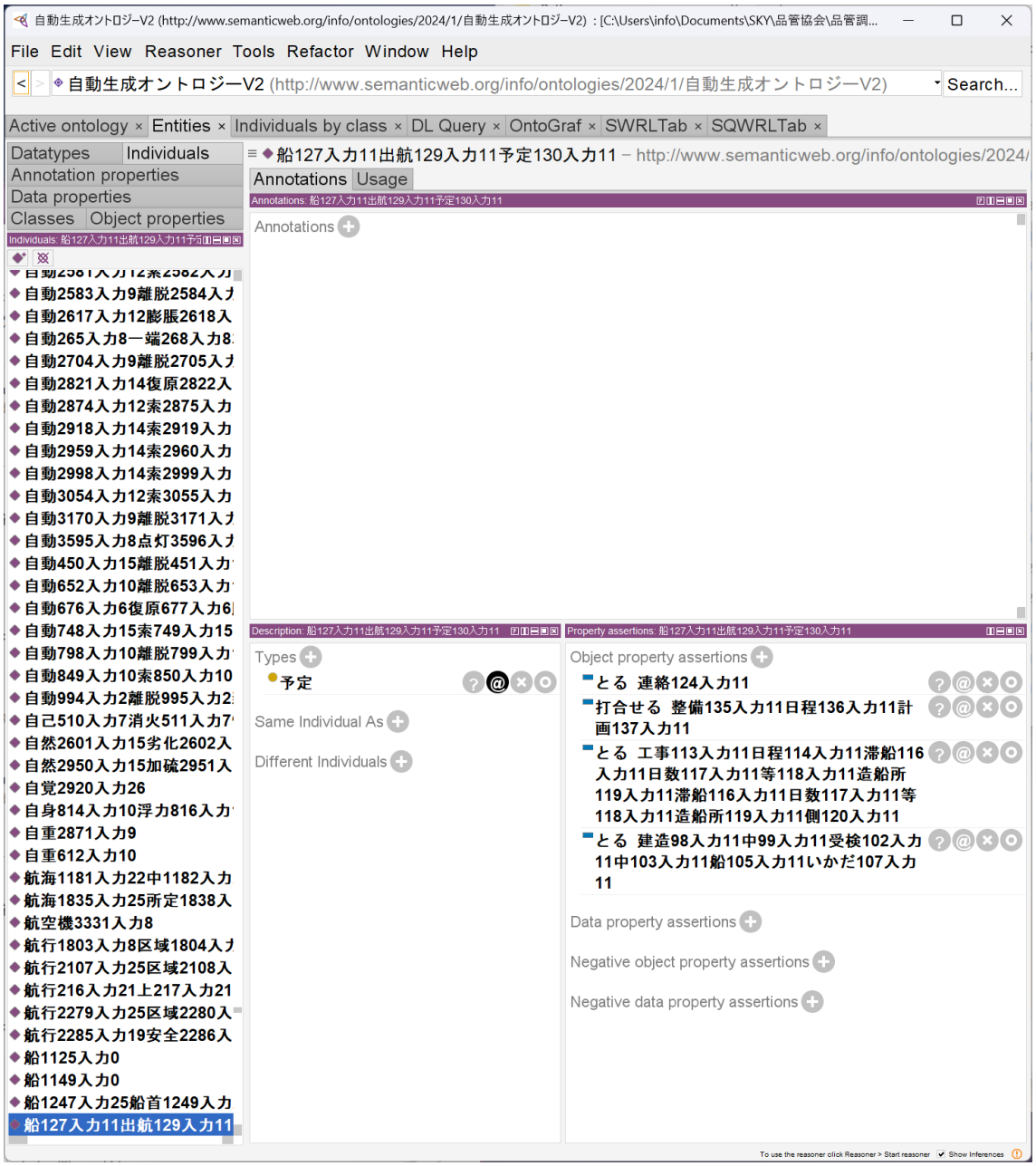


図 20　自動生成オントロジー　「個体」（一部）

図 20 では「船 127 入力 11 出航 129 入力 11 予定 130 入力 11」個体が　別の個体である「連絡 124 入力 11」と　「とる」関係　を、「整備 135 入力 11 日程 136 入力 11 計画 137 入力 11」個体と　「打ち合わせる」関係　…（以下省略）を持っていることを入力テキストから自動的に抽出している。いままでこれを人が行っていたが膨大な「個体」間の「関係」[10]が自動的に生成されていることは大きな省力化となっている。

10　これがそのままデータベースのもととなり、意味のある推論（「関係」のリンクをたどることにより新しい関係が分かる）を行うためのもととなる

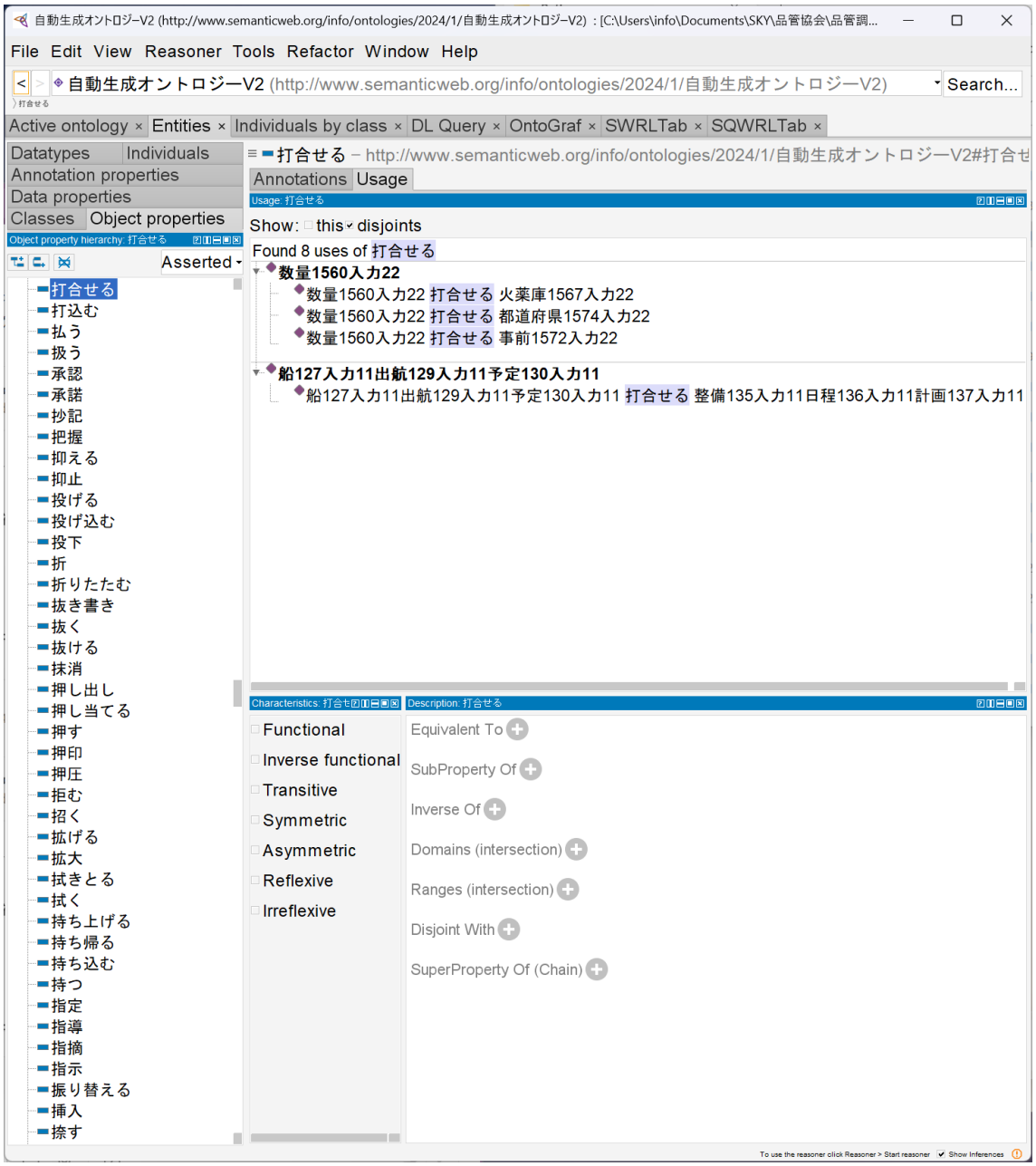


図 21　自動生成オントロジー　「関係」（一部）

図 21 では図 20 で出てきた「打ち合わせる」「関係」が他の「個体」間でも使われていることを示している（図の右）。

4. 結論と課題

ヒューマンエラー防止の AI 技術として、大規模言語モデル（LLM）による事前学習したライブラリを使った形態素解析を利用した Python プログラムを試作し、GMDSS の DX のための入力データの一部を借りて表現の揺れの吸収を試みた。併せて論理知識型 AI 技術であるオントロジーを自動生成するプログラムテスト実装して膨張式救命いかだ技術指導書の全文（テキストのみ、図表は対象外）を入力とし、自動生成を試みた。

ChatGPT 等の生成 AI に使われている大規模言語モデルを用いた形態素解析ライブラリは前処理としては有効であり、今まで人が一つ一つ丁寧に入力のテキストデータ（仕様文書等）を読み込んで SVO 分解をしていたものは、ほぼ自動化可能であることが分かった。

但しここで作成されたオントロジーは初期版であり、このまま高度な推論（最終的には人に代わり、技術指導書の内容をトレースできるようになる）できるレベルにはなっていない。

一方で 200 ページにわたる技術指導書を人が読んで、個々の単語間の関係を記述していることはほぼ不可能な作業（もしくは大変時間のかかる作業）であるため、古典的な知識論理型 AI のアプローチ（ルールベース）を取り、データ駆動型の成果である学習済み大規模言語モデルに基づく辞書を用いた形態素解析でテキストデータを前処理（STEP1）したことの有効性は確認でき、その成果は良好で全文が入力されても SVO/SVC 分解を自動化できることが分かった。

自動生成されたオントロジーの「個体」の集合である「クラス」は自明なものに留まり、ここから抽象的な新たな概念の生成や互いの親子関係などの自動生成には、かなりのインパクトなる技術的な飛躍が必要あり、現時点の LLM や RAG（人が整理した辞書等のデータの援用）をもってしても、特に日本語に対して十分でないことも分かっている。

喫緊の課題解決のために日本語の自然言語処理 AI の発展を待つ暇はないため、今回の初期オントロジーをベースとして、前処理の成果は大幅な省力化、DX 化となっていることはは事実であることから、最終的にはよりユーザーフレンドリーなナレッジグラフのグラフサーチが使えるグラフの元データとしての OWL ファイル作成に向けて、人によるオントロジー整備を進めつつ推論機能の有効性を確認してゆくのが現実的であると考える。

別添　7

2024 年度

船用品整備品質管理高度化技術開発委員会 議事録

2024 年度 第 1 回「船用品整備品質管理高度化技術開発委員会」 議事要旨

1. 日　時　　2024 年 8 月 2 日（金）14:00～16:00
2. 場　所　　朝日生命須長ビル　9 階会議室（Web 会議 併用）
（東京都中央区日本橋馬喰町 2-2-6）
3. 出席者　　別紙参照
4. 議 題
1）委嘱委員の確認
2）委員長の選出
3）事業の推進計画について
4）技術開発の進捗状況について（報告）
① 膨脹式救命いかだの整備の DX 化
② GMDSS の整備の DX 化
③ GMDSS 救命設備の整備記録作成に必要なデータの測定器の開発
④ AI 活用技術に関する調査研究
5）今後のスケジュール
6）その他
5. 配布資料
資料 24DX1-1　2024 年度 船用品整備品質管理高度化技術開発委員会 名簿
資料 24DX1-2　船用品整備における品質管理の高度化に向けたデジタル技術の開発事業計画書
資料 24DX1-3　膨脹式救命いかだの整備の DX 化の進捗状況
資料 24DX1-4　GMDSS の整備の DX 化の進捗状況
資料 24DX1-5　GMDSS 救命設備の整備記録作成に必要なデータの測定器の開発
資料 24DX1-6　AI 活用技術に関する調査研究
資料 24DX1-7　今後のスケジュール（案）
参考資料 1　2024 年 6 月 SS ニュース（Vol.0051）「船用品整備における品質管理の高度化に向けたデジタル技術の開発」プロジェクトの概要
参考資料 2　タブレット説明会日程（L/R 事業場関係）

6. 議事要旨

第 1 回船用品整備品質管理高度化技術開発委員会開催に際し、当会濱田専務理事より挨拶があった。

次いで、事務局より、本委員会の委員及び関係者の紹介があった。

次いで、事務局より、委員会の委員長及び副委員長を選出するにあたり、事務局案として、委員長に清水委員を、副委員長に島田委員を推薦する旨提案したところ、異議なく了承された。以降、清水委員長が議長を務め、配布資料の確認を行ったあとに議事が進行された。

議題１及び２はすでに終了していることから、議題３からの審議となった。

議題３　事業の推進計画について

事務局より、資料 24DX1-2 及び参考資料１に基づき、事業目的、目標、内容、期待される効果、推進体制、開発体制及び事業予算等の事業計画書の概要説明があった。

特に意見等はなく、事業計画案は確認された。

議題４　技術開発の進捗状況についての報告

1. 膨脹式救命いかだの整備の DX 化の進捗状況について

資料 24DX1-3 に基づき、海上技術安全研究所の小沢氏より、開発する新システムの概要、従来システムとの比較等を説明した後に入力等の操作実演を実施した。新システムの特徴は次のとおり。

① 他社の整備記録も WEB から確認が可能。ただし、秘匿すべき情報はアクセス制限あり。
② 端末アプリへの半自動な入力によりいかだの点検及びチェックシートの作成に関する作業性が向上する。また未記入・異常値をハイライトで表現し、ミスの発生を防止。
③ 入力された点検結果から整備記録、総括表を自動作成し、出力可能とする。
④ WEB システムに整備記録の内容を登録し、端末アプリから登録用データを出力可能とする。

以上のことにより、作業性向上及びミスの防止を図ることを目指している。

各委員からの主な質問（○）及び説明者からの回答（●）、委員長コメント（◎）

○　タブレットで入力したデータはエクセルで開くことは可能か？
● Web 登録用のデータについてはエクセルで開くことは可能。一方、印刷用のデータについてはブラウザから開き、出力（印刷）することを想定している。
○　船舶番号で整備情報が管理されるが、船舶所有者及び船名等に変更があった場合はどうするのか？
● 船舶番号は変更されることはないので、船舶番号をベースに管理することとした、船舶所有者や船名は変更されることがあり、また、同じ船名が多数ある場合があり管理しにくい。船名等が変更になった場合は、変更が分かった時点で整備事業者がその都度、今までと同様に変更されたデータを入力していただくことになる。
○　他社が入力した整備データを WEB でどこまで確認できるのか？
● 整備記録に記載されている情報は確認できるが、他社の総括表を見ることはできない。
○　整備記録に前回の整備した事業場名はあるのか？
● 整備記録に前回の事業場名は記載されているので確認できる。
○ どの程度のアクセス制限が加えられるのか？
● 整備記録に書かれている内容は、基本的に他社にオープンできるが、総括表に書かれている船舶所有者とか、免許人といった情報は他社に見せてほしくないという意見があり、それを踏まえて、総括表は基本的には非開示として扱っている。
また、外国船の場合には、整備記録は他社に全く見られないシステムになっており、事

業者の方からの意見に基づいている。その他にこの情報を開示したいとか、見せたくないとか意見があれば、品管に相談のうえ、判断することになる。

○ 入力したデータの値が異常時にハイライトされるが、そのデータの種類は先ほどの例では気圧、圧力及び温度と説明したが、日付も入ってくるのか？

● 日付については一部入れており、例えば、艤装品は有効期間というものが設けられていて、有効期間を過ぎているものは規定に基づき異常値として判定している。

○ 出力される整備記録のフォーマットは、現在のフォーマットとは変わってくるのか？

● 若干変わるが、基本的には現状のものを踏襲している。

○ 輸入の艤装品で有効期限５年のものがあるが、そのような場合の入力はどうなるのか？

● 入力も出力もできる。ただ赤く表示され、要件をそのままで見ると異常な値だけれど大丈夫か？とアラームを鳴らすような検証するためのシステムで、赤く表示された状態で出力することもできる。

○ 開示されるのは、前回の整備内容のデータで、整備記録に限られており、総括表の内容はダウンロードできないと説明だが、基本的な考え方として、品管へリクエストして、前回の整備情報を欲しいと言って出てくる内容と同じものがダウンロードできると考えていればよろしいのか？

● 基本的に整備記録に書かれている内容は、すべてダウンロードできる。

○ SS が品管に出したデータから作成しているが、公的機関が作成している船舶リストに関しては、船名等が更新されていると思うが、これを打ち込み船舶リストを基に完備していくことも考えられるが、あくまでも SS が入力したデータを基に管理することになるのか？

● 基本的には整備事業者から新造船の追加を申請することもできる。もちろん管理者の品管からも船の情報を編集あるいは追加することができる。
それは品管が実施しているので、外部からのデータを取り込むことは考えてない。
今後、一般公開されたデータについて、品管が入力するか、あるいは事業所の方で、必要になったらその都度追加するというところは、管理者である品管が判断すると考えている。

○ 端末アプリについて、いかだの整備を複数台行う時、例えば、２０台ぐらい並べて整備すると言った場合、複数人の作業者が担当部署を持ち回りで点検する場合にいかだ１台ずつに端末アプリを用意すると２０台の端末アプリが必要となる。データの入力を合理的に行う良い方法がないかと考えるところである。

○ 横浜で説明会した時に聞いたが、いかだ１台にタブレット一個ずつ置いて整備するしかない。艤装品の点検する人は、艤装品の点検ばかりするのは個々のいかだにタブレット置いて、その都度入力するしかないとのことであった。

● やり方はいくつかあって、例えば、整備するいかだにタブレットを用意して、自分が整理したところだけ記入して整備が終わったら、一旦、ファイルを保存し、終った後に、デスクに持ち帰って、検査したファイルを最後に開いて、デスクの方でその入力結果を統合するという方法。

アプリを全部開いておいて、流れ作業でやる場合に、アプリをその試験項目を別々の試験を実施している間が共有できるので、例えばその事業所にネットワーク環境を用意しておいて、ネットからアクセスできるメディア、ハードディスクを用意し、今回の検査データを保存しておいて、検査するタイミングでファイルを開く方法。
同じいかだを整備している場合、同時に開いて複数人が同時に一つのファイルを開くという想定はしていない。
要望内容は難しいが、まず技術的に可能か、やったことによるメリットデメリットを検討し、さらに要望している事業者がどれだけ希望するかを調べ、それを踏まえ最終的にやるかやらないかを検討する必要がある。

◎ いろいろ意見があるが、システムの設計コンセプトがうまく共有されてないと思われる。例えば、この端末が常にネットにつながり、サーバーから常に出たデータが形のシステム設計になっていれば、端末アプリが繋がることを想定してないので、その辺の設計思想が違うと思う。作業の効率化を考えた場合、整備をする人は、自分の担当項目が 20、30 あったらまとめて順番にチェックしてアップするのが、効率が良いと想像できるが、それらの設計仕様の擦り合わせが必要と思った。また、タブレットがどれぐらいの大きさか、数値をどう入力していくのか、端末からの誤入力とか変な数値が入るか、端末でどれくらいの文字の大きさで見えるのか、どういう風に数字を入力していくのかというところが絡むと思うが、想像つかない。あと細かい字を切り替えるのも大変そうと思い、どういう風にやるのかと思った。
さらに、ベースの CSV ファイルで考えると、マイクロソフトは 365 でエクセルと共有して編集できる。
色々な人が複数でアクセスしてデータ入力しても対応できる部分があれば、そういうものを入れて、必要なことを共有しながら、同時に自分が担当することを入れることもできると思ったので、マニュアルで、その説明の部分が共有できていないからいろいろな質問が出ていると感じる。今からでも修正が可能であればご検討願いたい。

● 当初、このシステムを開発するにあたって、キントーンのアプリのみを開発するのか、あるいは端末アプリも両方開発するのかが、議題に上がって、その時に出たのが整備事業所によっては、作業環境にネットワーク環境があるとは限らないので、そういったところでも入力できるようなシステムを作ってほしいということ。端末アプリの方は基本的にオフラインでも、実行できることを前提に開発した。文字の大きさについては、Windows の標準機能を用いればユーザーの好みに応じたサイズに変更することが可能となっている。

◎ 現場で使われる方々の意見により、いいものにしていただきたい。
要望を吸い上げ、それにシステムが対応するってことをやっていただければよいと思ったので、後で出来上がった時に、使いにくいとか、紙の方便利だということにならないように目標をつけて対応すれば良いと思われる。

● 要望があれば、対応できるものは順次対応していきたいと思うが、作業量によって開発期間が変わってくるので、ケースバイケースで対応したい。

○ 端末アプリを使わないという選択をするサービスステーションがあってもいいのか？
例えば、整備する台数が年間に少しかない SS とかは。

● 物件管理システムは現在皆さんに使用されていると思われる。
手書きをなるべく減らし、紙のチェックシートを使ってパソコンに打ち込むと間違いや入力ミスが起きる可能性が大きいので、新管理システムで運用していただきたいと思っている。

2. GMDSS の整備の DX 化の進捗状況について

資料 24DX1-4 に基づき、海上技術安全研究所の平方氏より、開発する新システムの概要について説明があった。新システム制作におけるポイントは次のとおり。

① 現行管理システムの運用を基本に「維持すること」「見直すこと」を整理する。
② Web アプリ（キントーンアプリ）をベースに整備記録の作成
③ データベースの実績を調査し、「データ品質の劣化」、「整備記録作成ミス」を改善。
④ 「最低限の機能」を盛り込み、「将来の拡張機能」を整理する。

現在、作業が遅れているが、タブレットアプリの制作を中心に、機能確認、操作性確認を進め、修正を繰り返し実施している状況。

3. GMDSS 救命設備の整備記録作成に必要なデータの測定器の開発について

資料 24DX1-5 に基づき、西日本フジクラの菅氏より、開発する測定器についての説明があった。開発する測定器の概要は次のとおり。

① GMDSS 測定器で測定したデータをパソコン等に転送する方法を検討。
② 送出するデータ形式は汎用性のある CSV 形式で測定データを転送。
③ 現行の測定機器を改造して対応する方法の検討。
④ 新規に開発する測定器の検討。

委員長コメント（◎）

◎ テーマとして、測定結果を改ざんされないということもあるのかなと思う。
むしろ CSV でやるよりも、画像データを読み込む方がいいのかもしれないと思った。計測データを CSV であれば書換えられることもあるので、どうやって改ざんを防ぐのかというのは課題になると思う。最近、自動車関係で検査の不正とかがあるので、改ざんできないということが重要かと思われる。改ざん防止を担保している検査方法となれば、売りになると思う。

4. AI 活用技術に関する調査研究について

資料 24DX1-6 に基づき、オフィス S.K.Y の加瀬氏より、AI 活用技術に関する調査研究について説明があった。内容は次のとおり。

① 整備記録記載におけるヒューマンエラー調査

・膨脹式救命いかだ整備技術指導書（電子データ）を入力とし、自動処理または半自動処理によってAI技術が活用できるか（前段階）調査を行う。
・GMDSS救命整備指導書は、新管理システムの入力となるエクセルファイルについて、入力時のヒューマンエラー（綴りミスや表現の揺れ）について調査を行う。

② ヒューマンエラー防止のためのAI技術活用調査

・前段階調査をへて、適した自然言語処理によるAI技術を選択し、膨脹式救命いかだの指導書の電子データ、GMDSS救命整備のエクセルファイルを入力し、前者に対してはオントロジーの自動生成の可能性、後者に対してはヒューマンエラー（表現の揺れや綴りミスなど）を吸収して、将来のオントロジー作成のためのデータの整理の自動化の可能性の試作プログラム作成を通した検証を行う。

議題５　今後のスケジュールについて

事務局より、資料24DX1-7及び参考資料2に基づき、今後の委員会の開催予定、各開発及び調査の実施予定及び新管理システムに関するタブレットの操作説明会の日程等についての概要説明があった。

特に意見等はなく、今後のスケジュールは確認された。

事務局からのコメントとして、次の発言があった。

整備記録の管理システムは９月末で終わりとなり、１０月からは新管理システムが始まるので、早急に新システムに移行できるように準備を進めている。色々な課題が出てきているが、とりあえず今年度いっぱいで、きちっと動くようにしたい。

来年度は管理システムだけではなく、違う要求をしようと思っており、IoT関係等、検査技術とかをやってみようかと考えている。

ご意見をいただければ予算の範囲もあるが、検討したいと思っている。

その他　事務局より、第２回委員会の日程について提案があり、１２月１１日（水）の１４：００からの開催予定となった。

別　紙

2024 年度 第 1 回 DX 委員会出席者名簿

	氏　　名	所　　属	役　　職	出欠
1	清水　悦郎	国立大学法人東京海洋大学 学術研究院	海洋電子機械工学部門 教授　博士（工学）	出席
2	平方　勝	(国研) 海上技術安全研究所	デジタルトランスフォーメーションプロジェクトサブリーダー　上席研究員	出席
3	西　紀美男	R.F.D ジャパン㈱	技術部　技術部長	出席
4	日高　健治	㈱ 泉屋商店	代表取締役	出席
5	金田俊太郎	金田商事 ㈱	代表取締役	出席
6	島田　雅司	島田燈器工業 ㈱	代表取締役社長	出席
7	村上　博史	㈱ シモセン	代表取締役	出席
8	綱田　幹人	綱田工業 ㈱	代表取締役社長	出席
9	玉城　敏幸	㈱ 中幸船具店	代表取締役	出席
10	熊沢　泰生	ニチモウ ㈱	海洋営業部長兼研究開発室長	欠席
11	栄　俊樹	日本無線 ㈱	マリンシステム品質保証部船用通信システム品質保証グループ長	Web
12	板倉　拓也	藤倉コンポジット ㈱	引布加工品事業部　事業部長	欠席
13	岡本　大正	船田産業 ㈱	代表取締役社長	欠席
14	園本　竜也	古野電気 ㈱	舶用機器事業部営業企画部 部長補佐	出席
15	上原　浩巳	㈱マリン・インターナショナル	代表取締役社長	出席
16	黒森　博志	三菱電機ディフェンス＆スペーステクノロジーズ ㈱	東部事業部　電子技術部　次長	出席
17	湯浅　成人	湯浅工業 ㈱	代表取締役	欠席
18	小森愛一郎	㈱ 横浜通商	横浜支店長	出席
19	平瀬　利明	国土交通省	海事局検査測度課　専門官	出席
開発担当者	小沢　匠	(国研) 海上技術安全研究所	産業システム系 物理システム研究グループ研究主任	出席
	加瀬　究	㈱ オフィス S.K.Y	技術部長	出席
	菅　哲郎	西日本フジクラ ㈱	取締役　資材部長	出席
事務局	濱田　哲	(一社) 日本船舶品質管理協会	専務理事	出席
	大谷　雅実	〃	常務理事	出席
	池上　敦	〃	業務部長	出席
	芦田　研二	〃	指導技師（GMDSS 等）	欠席
	庄司陽二郎	〃	指導技師（膨脹式救命いかだ）	出席

2024 年度第２回「船用品整備品質管理高度化技術開発委員会」議事要旨

1. 日　時　　2024 年 12 月 11 日（水）14:00～16:20
2. 場　所　　朝日生命須長ビル　９階会議室（Web 会議 併用）
　　（東京都中央区日本橋馬喰町 2-2-6）
3. 出席者　　別紙参照
4. 議 題
　　1） 前回委員会議事要旨（案)の確認
　　2） 船用品の整備に関するデジタル化（以下、DX 化）事業の中間報告
　　3） ２０２５年度の事業計画（案）について
　　4） その他
5. 配布資料
　資料 24DX2-1　第１回 船用品整備品質管理高度化技術開発委員会議事要旨（案）
　資料 24DX2-2　膨脹式救命いかだの整備の DX 化の中間報告
　資料 24DX2-3　GMDSS 救命設備の整備の DX 化の中間報告
　資料 24DX2-4　GMDSS 救命設備の整備記録作成に必要なデータ測定器の開発の中間報告
　資料 24DX2-5　AI 活用技術に関する調査研究の中間報告
　資料 24DX2-6　２０２５年度の事業計画（案）について
　参考資料　１　２０２４年度 船用品整備品質管理高度化技術開発委員会 名簿
　参考資料　２　新物件管理システム説明会実施状況

6. 議事要旨

清水委員長が議長を務め、委員の出欠の状況及び配布資料の確認を行ったあとに議事が進行された。

議題１　第１回 船用品整備品質管理高度化技術開発委員会議事要旨 (案)の確認

事務局より、資料 24DX2-1 に基づき、第１回船用品整備品質管理高度化技術開発委員会議事要旨 (案)の概要説明があった。

特に意見等はなく、第１回船用品整備品質管理高度化技術開発委員会議事要旨 (案)は確認された。

議題２　技術開発の中間報告についての報告

1. 膨脹式救命いかだの整備の DX 化の中間報告について

資料 24DX2-2 に基づき、海上技術安全研究所の小沢氏より、膨脹式救命いかだの整備の DX 化について中間報告、今後の予定についての説明があり、その後、事業者からのバグ報告/修正要望等についての対応状況について報告があった。

各委員からの主な質問（○）及び説明者からの回答（●)、委員長コメント（◎）

○　説明の補足、確認の意味で発言があった。

・検査の方法において、整備記録の漏洩等の圧力試験の値の小数点二位以下の記載については、救命設備ということから安全側の評価となるよう項目ごとに切捨て/切上げの処理を分けている。
・漏洩試験において、初期の圧力より終わりの圧力の方が高くなる場合には、整備技術指導書に従い再度試験を実施することとしている。
・メーカーとしての確認ですが、次回検査の時期の記載については、特定の日までということにはなっていないので、空白としなければならない。

○ 初期の試験圧力値よりも終わりの値が上がる場合はエラーとなるのか。

○ 実際に初期の試験圧力値よりも終わりの値が上がることは良くあることで、再試験となる。なお、エラーは、不合格という意味ではなく、試験をもう一度実施するという意味である。

○ 次回検査日については、1年9か月から3年3か月の間に実施するので、範囲が広いことから、整備事業者が記入することができないので空欄となる。
ライセンスファイルについて、マニュアル等に記載してあるのか。

● 簡易マニュアルに記載しているので参照してほしい。

○ 細かい修正、訂正依頼について、リクエストはできるのか。

● リクエストに対応できるものはケースバイケースで実施していきたい。なお、要望等はポータルサイトを通じてご意見等をいただきたい。

○ 法令等で改正があった場合、アプリのメンテナンスはどのようになるのか。

● 品管からのオーダーで変更することになる。

○ 整備記録の変更に関わる届け出はどのようになるのか。

● 全国的な取扱いになると思われるので、本省で品管と確認しながら取扱いを進めたい。

○ 整備記録の旧バージョンから添付ファイルを見ることができないか。

● 最新バージョンでは前回整備記録読み取り時に印刷画面が出力されるようにシステムを改修している。

○ 総括表の文字が小さく見えづらいので修正が可能か。

○ 整備記録等の文字の大きさを全体的に大きくできるのであれば見やすくなるのでありがたい。

● 備考欄の文字の大きさは要望によりフォントサイズを変更できるように改修したが、整備記録は記載する項目が決まっており、フォントサイズを大幅に変更すると次ページに跨がるなどフォーマットが崩れる懸念がある。そのため、文字を大きくするにも限界があることを理解してほしい。なお、総括表については一枚に表示するいかだの件数を少なくすることによって文字を大きくできると思われるので、今後検討していきたい。

◎ いろいろと要望を挙げてもらって、より良いものに仕上がるようにしていただきたいと思う。

◎ キントーンのデータベースの更新により、使用が制限されたことがある。原因を教えて欲しい。

● データベースに記録する整備記録の情報に変更があった場合、新しい入力項目へその内

容を入力する必要がある。この時、既存のデータは３０万件もあるため、データベースの更新に時間がかかり、その間キントーンの一部機能の使用が制限される。運用が安定すればこのようなことは起きないと思われる。

◎　新システム（キントーン）には、船舶所有者のコントロール（入力項目）がない。との意見について

●　実際の整備記録を踏襲してシステムを構築しているため、整備記録に記載しない船舶所有者については入力用のコントロールを用意していない。以前のシステムは船舶所有者の項目があったが、今回はないのかという意味と思われるが、以前と同様に船舶リストには記載用のコントロールを用意しているので問題はないと思われる。

◎　キントーンの整備記録において、船舶番号だけでなく船名でも検索できるようにしてほしい。との意見について。

●　旧システムでは、船名検索としていたが、同じ船名が多数存在し、また、海保の巡視船等は管区が変更するたびに船名も変更される。よって、船名での検索では間違いが多数生じるので、一生変わることのない船舶番号で管理されていれば、適切なデータの提供ができることから、ご理解を求めているところです。

◎　変更した経緯をていねいに説明し、ご理解を得られるようにしていただきたい。

○　整備記録の様式は１０月から使用されているが、様式等に変更が生じた場合は、どのような対応となるのか。

●　役所の意向により、変更が必要となれば品管より海技研へシステム改修を依頼することになる。

2. GMDSS の整備の DX 化の中間報告について

資料 24DX2-3 に基づき、海上技術安全研究所の平方氏より、GMDSS の整備の DX 化について中間報告及び今後の予定についての説明があり、その後、整備事業者からの質問、要望等の対応についての報告があった。主な対応は次のとおり。

① キントーン「使用測定器マスター」に製造番号を追加して欲しいとの要望に対し，製造番号を追加すると同時に，キントーン内で整備記録を作成する際に使用測定器の製造番号を参照できるようにした

② キントーン「船舶リスト」に登録されている既存船の情報について，整備事業所では編集できるようにして欲しい旨要望があったのに対し，整備事業所で編集・申請できるようにした．また，キントーン作業で，航行水域と航行区域の情報を参照できないミスは修正した．

③ EPIRB 自動離脱装置もキントーン「EPIRB マスター」から参照できるようにしてほしい旨の要望に対して，参照できるように修正した．

④ キントーン内で検査種別（和文・英文）の入力を選択式にして欲しい旨の要望に対して，選択式に修正した．同時に，クライアントアプリ（エクセル版）の総括表の入力について整合をとった．

⑤ キントーン作業で，整備責任者も「GMDSS 整備技術者技術者マスター」から参照

できるようにしてほしい旨の要望に対して，参照できるように修正した．

⑥ クライアントアプリ（エクセル版）で，EPIRB の試験回数（1 回（セルフテスト）から 3 回（従来）を想定）に応じて平均値処理して，クライアントアプリ（エクセル版）整備記録に記載（反映）するようにした．

⑦ キントーン上において，試験項目入力箇所に数値が入力されていないのに，判定の欄に「否」と表示されていた不具合を改修した．点検・試験結果入力は，当面の間，クライアントアプリ（エクセル版）で作成を依頼した．キントーン上は，データの格納場所とした．

⑧ キントーンの改修に伴い，キントーンから書き出される csv データの項目順番が変更された．それに伴い，クライアントエクセルの全面的な改修を行った．

⑨ その他，キントーンからクライアントエクセルへの参照ミス，クライアントエクセル内でのミス（参照ミス，数式ミス等）を改修した．クライアントアプリ（エクセル版）において，EPIRB の整備記録様式を改訂版様式に修正した．

○ 整備記録の英文及び前回の記録が見られるようになる時期の目途について、どのような状況か。

● イパーブ以外は英文があるが、新型イパーブについては、英文用語として使用されていないため、正式名称が無い状況。勝手に英文表記を決めるわけにもいかないので、公的機関等で調整して決めていただきたいと思っている。

○ GMDSS の整備記録は、和文は新様式で英文は旧様式になっているので、早急に対応していただきたい。

● GMDSS の整備記録データの移行を早急に実施する予定。
現在、作業が遅れているが、タブレットアプリの制作を中心に、機能確認、操作性確認を進め、修正を繰り返し実施している状況。

3. GMDSS 救命設備の整備記録作成に必要なデータの測定器の開発について

資料 24DX2-4 に基づき、西日本フジクラの菅氏より、開発する測定器の中間報告及び今後の予定について説明があった。

① 対象となる測定器
現行型は、SET-501（製造番号 202720 以降品）、SET-501V、STT-502 の 3 種類。
新型は、SET-502 で対応予定。

② データ転送方法
現行測定器から検査データを PC 内通信ソフトへ送信する試作プログラムを開発し、通信ソフトで CSV ファイルに変換できることを確認。

③ ハードウェア
現行型パソコン/プリンターは排他接続となる為､外付切替器とデータ出力スイッチにて対応するプロトタイプを作成し、動作確認中。

④ ソフトウェア

SET-501・STT-502 に関してはプロトタイプを作成し、動作確認中。

4. AI 活用技術に関する調査研究について

資料 24DX2-5 に基づき、オフィス S.K.Y の加瀬氏より、AI 活用技術に関する調査研究についての中間報告があった。内容は次のとおり。

① 整備記録記載におけるヒューマンエラー調査

・膨脹式救命いかだ整備技術指導書（電子データ）を入力とし、自動処理または半自動処理によって AI 技術が活用できるか（前段階）調査を行った。

・GMDSS 救命整備指導書は、新管理システムの入力となるエクセルファイルについて、入力時のヒューマンエラー（綴りミスや表現の揺れ）について調査を行った。

② ヒューマンエラー防止のための AI 技術活用調査

・前段階調査をへて、適した自然言語処理による AI 技術を選択し、膨脹式救命いかだの指導書の電子データ、GMDSS 救命整備のエクセルファイルを入力し、前者に対してはオントロジーの自動生成の可能性、後者に対してはヒューマンエラー（表現の揺れや綴りミスなど）を吸収して、将来のオントロジー作成のためのデータの整理の自動化の可能性の試作プログラム作成を通した検証を行った。

議題３　２０２５年度の事業計画（案）について

事務局より、資料 24DX2-6 に基づき、２０２５年度の事業計画（案）についての説明があった。

特に意見等はなく、２０２５年度の事業計画（案）は確認された。

その他　事務局より、第３回委員会の日程について提案があり、３月１２日（水）の１４：００からの開催予定となった。

別 紙

2024年度 第2回 DX委員会出席者名簿

（敬称略）

	氏　名	所　属	役　職	出欠
1	清水　悦郎	国立大学法人東京海洋大学 学術研究院	海洋電子機械工学部門 教授　博士（工学）	出席
2	平方　勝	（国研）海上技術安全研究所	デジタルトランスフォーメーション プロジェクトサブリーダー上席研究員	出席
3	西　紀美男	アール・エフ・ディー・ジャパン㈱	技術部　技術部長	出席
4	日高　健治	（株）泉屋商店	代表取締役	欠席
5	金田俊太郎	金田商事（株）	代表取締役	出席
6	島田　雅司	島田燈器工業（株）	代表取締役社長	出席
7	村上　博史	（株）シモセン	代表取締役	2名 出席
8	綱田　幹人	綱田工業（株）	代表取締役社長	出席
9	玉城　敏幸	（株）中幸船具店	代表取締役	出席
10	熊沢　泰生	ニチモウ（株）	海洋営業部　部長 兼　研究開発室　室長	欠席
11	栄　俊樹	日本無線（株）	マリンシステム品質保証部船用通信システム品質保証グループ長	欠席
12	板倉　拓也	藤倉コンポジット（株）	引布加工品事業部　事業部長	出席
13	岡本　大正	船田産業（株）	代表取締役社長	Web
14	園本　竜也	古野電気（株）	舶用機器事業部　営業企画部 部長補佐	出席
15	上原　浩巳	（株）マリン・インターナショナル	代表取締役社長	出席
16	黒森　博志	三菱電機ディフェンス＆スペーステクノロジーズ（株）	東部事業部　電子技術部　次長	出席
17	湯浅　成人	湯浅工業（株）	代表取締役	欠席
18	小森愛一郎	（株）横浜通商	横浜支店長	出席
官関庁係	平瀬　利明	国土交通省	海事局検査測度課　専門官	出席
開発担当者	小沢　匠	（国研）海上技術安全研究所	産業システム系 物理システム研究グループ研究主任	出席
	加瀬　究	（株）オフィスS.K.Y	技術部長	出席
	菅　哲郎	西日本フジクラ（株）	取締役　資材部長	出席
事務局	濱田　哲	（一社）日本船舶品質管理協会	専務理事	出席
	大谷　雅実	同上	常務理事	出席
	池上　敦	同上	業務部長	出席
	竹原　隆	同上	上席技師	Web
	芦田　研二	同上	指導技師（GMDSS等）	出席
	庄司陽二郎	同上	指導技師（膨脹式救命いかだ等）	出席