Natural Language Interface for Goal-Oriented Knowledge Graphs Using Retrieval-Augmented Generation

Kosuke Yano[1], Yoshinobu Kitamura[2] and Kazuhiro Kuwabara[2]
[1]*Graduate School of Information Science and Engineering, Ritsumeikan University, Ibaraki, Osaka, 567-8570, Japan*
[2]*College of Information Science and Engineering, Ritsumeikan University, Ibaraki, Osaka, 567-8570, Japan*

Keywords: Function Decomposition Tree, Retrieval-Augmented Generation, Large Language Model.

Abstract: A search method leveraging Retrieval-Augmented Generation (RAG) for goal-oriented knowledge graphs is proposed, with a specific focus on function decomposition trees. A function decomposition tree represents hierarchically functions of artifacts or actions of human with explicit descriptions of purposes and goals. We developed a schema to convert the trees into RDF, enabling structured and efficient searches. Through RAG technology, a natural language interface converts user's inputs into SPARQL queries, retrieving relevant data and subsequently presenting them in an accessible and chat-based format. Such a flexible, and purpose-driven searches enhance usability in complex knowledge graphs. We demonstrate the tool effectively retrieves actions, intentions, and dependencies using an illustrative and a real-world example of function decomposition trees.

1 INTRODUCTION

This paper proposes a natural language search system for goal-oriented knowledge graphs called function decomposition trees (Kitamura and Mizoguchi, 2003), (Kitamura et al., 2004). The proposed search system utilizes a chat-based search interface powered by Retrieval-Augmented Generation (RAG) technology, incorporating a Large Language Model (LLM) to operate on function decomposition trees converted to RDF.

A function decomposition tree is a descriptive method, originally proposed for functional knowledge of artifacts in the engineering design field (Kitamura et al., 2004). It has been extended to procedural knowledge of human actions called CHARM (Convincing Human Action Rationalized Model) in the health care field (Nishimura et al., 2013). Its feature is explicit descriptions of purposes or goals of actions. For example, the purposes or goals of "wearing a face-mask" is "preventing from catching a cold" or "preventing secondary infections to other people." The traditional methodologies like flowcharts, BPML and IDEF3 are fundamentally procedure-oriented. In such procedure-oriented notations, the sequence of steps of actions is emphasized, which often leads to the implicit nature of the action's purpose and rationale. The application of CHARM suggests that it enhances the efficiency of learning support and promotes flexible nursing practices through a better understanding of goals. Moreover, function decomposition trees and CHARM have been applied in various fields including auditing of accounting (Taki et al., 2023).

Many studies have explored systems that allow searching knowledge graphs using natural language. For example, FREyA (Damljanovic et al., 2012) is a natural language interface that generates SPARQL queries from user's input. This research aims to develop a search tool that processes natural language inputs and generates responses in natural language based on the search results, specifically focusing on function decomposition trees. The research comprises two main components: developing a tool for universally converting function decomposition trees to Resource Description Framework (RDF) data, and the development of a chat-based search tool for querying the RDF-converted function decomposition tree data.

First, to convert function decomposition trees into RDF format, it is essential to define a framework that outlines the vocabulary used in RDF, known as Resource Description Framework Schema (RDFS). This involves identifying the key concepts that make up the function decomposition tree and describing a RDFS schema based on these definitions. Subsequently, we

Yano, K., Kitamura, Y. and Kuwabara, K.
Natural Language Interface for Goal-Oriented Knowledge Graphs Using Retrieval-Augmented Generation.
DOI: 10.5220/0013245700003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 975-982
ISBN: 978-989-758-737-5; ISSN: 2184-433X

will utilize the schema to develop a tool that converts function decomposition tree data into RDF format.

Second, to develop the search tool for function decomposition tree data, we will focus on Retrieval-Augmented Generation (RAG) technology within Large Language Model (LLM). RAG generates database search queries from natural language statements and uses the search results to formulate natural language responses. We will first generate SPARQL queries to search function decomposition tree data based on user's inputs in natural language. Next, the tool will execute the SPARQL query and retrieve the results. Finally, it will utilize the results to generate responses using LLM. This process aims to achieve the search for function decomposition tree data through natural language. We demonstrate the proposed tool through an illustrative example in the healthcare field, followed by a real-world application in the auditing domain of accounting (Taki et al., 2023).

2 RELATED WORK

2.1 Function Decomposition Tree

A function decomposition tree is a framework used to hierarchically decompose functuions or actions into ways of sub-functions or sub-actions, representing them in a tree structure as a form of knowledge graph. Figure 1 illustrates an example of function decomposition tree aimed at handling a cold.

The function decomposition tree is composed of *function nodes*, *way nodes*, and *relation nodes*, which are depicted by ellipses, squares, and rectangles with arrows, respectively. The *function node* represents a *function*, to be realized by artifacts or actions to be performed by humans. The *way node* represents a *way of function/action achievement*, indicating the approach used to achieve a function or an action. In a tree, a *function node* is decomposed into *way nodes*, which are further decomposed into *function nodes*, thereby recursively detailing the action. Figure 1 illustrates that handling a cold can be achieved through either "Prophylaxis way" or "symptomatic way". This hierarchical structure effectively demonstrates the different pathways available for achieving the desired outcome. In short, it represents goal-method relationship between actions.

In addition, *function nodes* can also specify the *functional category*. *Functional category* indicates the nature of a function or an action. For instance, in Figure 1, the action "Prevent virus invasion into the body" is categorized as a prevention function, which is shown in purple. On the other hand, the counter functions are shown in yellow.

Relation nodes describe the dependencies between *way nodes*. Some *ways*, when selected, can impact other *ways*. For example, in Figure 1, choosing the "Mask-wearing way" as a way node for "Prevent virus invasion into the body" also fulfills the action "Prevent secondary infections", as indicated by the relation node "Interlocking Selection."

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) combines Large Language model (LLM) with external database search systems (Gao et al., 2023). Traditional LLMs face challenges when processing queries that fall outside their training data, often resulting in hallucinations and difficulties in tracking reasoning processes. RAG addresses these issues by incorporating knowledge from external databases, thereby enhancing accuracy and reliability. In RAG, user queries trigger searches in an external database, and responses are generated based on the search results. In RAG, vector databases are typically utilized as a reference. In contrast, this study enables indirect searches of function decomposition trees by referencing RDF-converted function decomposition tree data. This approach requires the generation of SPARQL queries from natural language inputs.

The technology for converting text to SPARQL has been extensively researched (Damljanovic et al., 2012), (Kovriguina et al., 2023), (Avila et al., 2024). These studies have primarily focused on general RDF datasets; however, this research specifically targets RDF representations tailored to function decomposition trees for SPARQL generation.

3 METHODS

In this research, we implemented a natural language search tool for function decomposition tree data. Initially, we identified the key concepts underlying the function decomposition tree and developed a schema tailored specifically for it. Using this schema, we then developed a tool to convert function decomposition trees into an RDF dataset. Finally, leveraging the converted function decomposition tree data, we implemented a chat-based search interface utilizing a Retrieval-Augmented Generation (RAG) for efficient querying of the function decomposition tree.

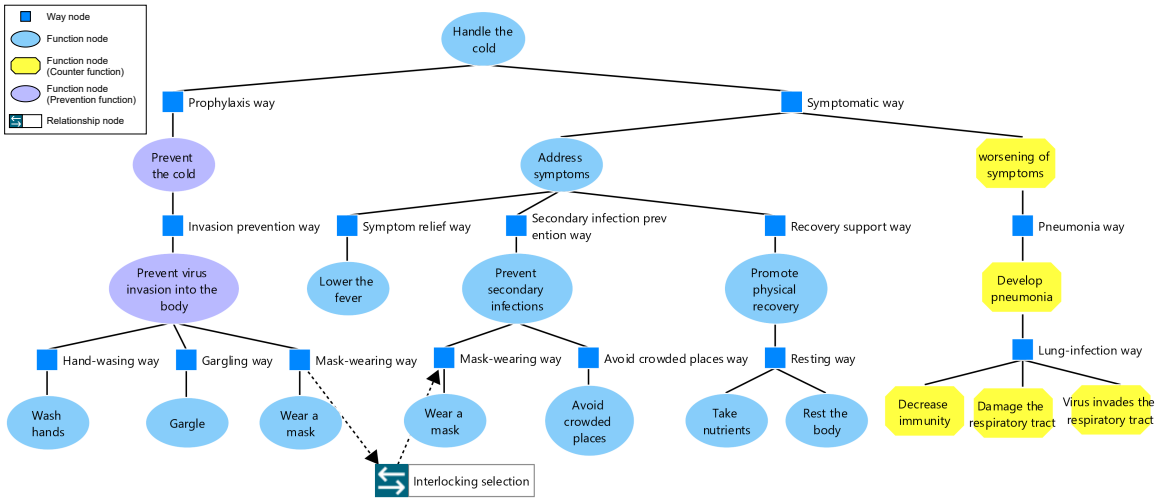

Figure 1: Example of function decomposition tree about handling a cold.

3.1 Components in a Function Decomposition Tree

As the elements of the function decomposition tree, we defined two key components: *functional categories* and *relation*. *Functional categories* indicate the types of functions or action nodes possess, while *relation* represents the relationships between *way nodes*.

Nodes representing functions within a function decomposition tree are defined as *Systemic processes*, which are characterized by an initial state and an end state, with the end state serving as the goal state. Such processes can be categorized into two sub-classes based on their goals: *intended goal processes* and *unintended goal processes*. The proposed method leverages these two types of processes to define function decomposition trees.

3.1.1 Functional Category

Functional categories refer to the various types of function or action nodes within the function decomposition tree. Each functional category is treated as a specialization of function nodes. The *counter function* and the *prevention function* as described below are definded as subclasses of the *Systemic Process*.

3.1.2 Functional Category: *Counter Function*

A function decomposition trees may contain not only the actions that should be taken described, but also the failure events and side-effects associated with the execution of these actions. These elements are classified into *counter functions*. For instance, in Figure 1, the subtree rooted at the node labeled “worsening of symptoms” shown in yellow exemplifies this type of action. By documenting failure events and side effects, the intent behind actions becomes clearer, potentially leading the prevention of defects (Kitamura et al., 2004). Similarly to regular actions shown in blue, failure events or side effects are decomposed the results of higher-level nodes into causes at lower-level nodes. However, unlike regular actions, failure events or side effects are not intended to be executed. We defined *counter function* as a subclass of *unintended goal processes* within RDFS.

3.1.3 Functional Category: *Prevention Function*

Some actions in a function decomposition tree are aimed at prevention. They are categorized into *prevention function*. For instance, in Figure 1, the subtree rooted at the purple node labeled “prevent a cold” exemplifies this type of action. These preventive actions are distinct from typical actions, as they focus on averting specific outcomes rather than achieving them.

In the context of processes aimed at prevention, we can decompose the concept of *intended goal processes* into two categories: *action* and *function*. *Action* refer to processes executed by living organisms, while *function* are carried out by inanimate entities. Both categories can be oriented towards prevention. We define *prevention actions* as a subclass of *action*, and *prevention function* as a subclass of *function*. Since their overaching concepts of *actions* and *functions* are not differentiated in a funtion decomposition tree, *prevention action* and *prevention function* are treated as the same type of *functional category*. However, these concepts are defined as separate entities in an RDF to support efficient querying.

3.1.4 Dependencies Between *Ways*

In a function decomposition tree, certain *ways* are dependent on others for their selection state. This dependency is categorized into two primary relationship types: *interlocking selection relation* and *exclusive relation*. The *interlocking selection relation* relationship occurs when the selection of one way necessitates the selection of another. Conversely, the *exclusive relation* relationship ensures that if one way is selected, the other is precluded. The *exclusive relation* is further subdivided into *sibling relation* and *contradictory relation*. A *sibling relation* occurs when both *ways* share the same parent node, whereas a *contradictory relation* arises when the two *ways* are distant from each other within the tree structure. Both *Interlocking selection Relation* and *Exclusive relation* are represented in the function decomposition tree using relationship nodes. For instance, in Figure 1, the node connecting the two *Mask-wearing ways* with a dotted line represents an *Interlocking selection Relation* relationship node. In the proposed method, we define the dependencies between *ways* by treating relationship nodes as RDF properties. These nodes are described as sub-properties of each dependency type, allowing for a structured representation of the dependencies within the function decomposition tree.

3.1.5 Schema

Based on the components of the function decomposition tree defined in Section 3.1, we described schema in the RDFS. Figure 2 graphically represents the schema of a function decomposition tree in RDFS. For simplification, the label and ID are described with the identical content; thus, the label is omitted. Additionally, the descriptions of `rdfs:Class` and `rdf:Property` in some parts of Figure2 is omitted for simplification, because the descriptions of `rdf:type` can be inferred through the use of `rdfs:subClassOf` and `rdfs:subPropertyOf`.

3.2 Conversion to RDF Data

Based on the schema created in Section3.1.5, a program was developed to convert function decomposition tree data into the Turtle format, one of the formats used in RDF. In function decomposition trees, procedures and methods within actions are broken down into a tree structure, with each node described in text form. These textual descriptions were described using the `rdfs:label` property in a converted RDF dataset. This enabled users to perform RDF searches using text-to-SPARQL, allowing natural language input to be queried. However, this approach required exact text matches, leading to search failures when user input differed from the descriptions in the function decomposition tree, even if the meanings were similar or nearly identical. To address this issue, we introduced an additional `:keyword` property alongside `rdfs:label`. By employing LLM, keywords were extracted from the content of `rdfs:label` and recorded under the `:keyword` property. This enhancement allowed for RDF searches based on keywords, significantly reducing search problems caused by variations in textual expressions.

3.3 Searching for RDF Data Using a Chat-Based Interface with RAG

We developed a chat-based interface for querying a function decomposition tree. User-provided natural language input is transformed into a SPARQL query through the application of a large language model (LLM). We employed OpenAI's gpt-4o-mini model for this transformation. The results obtained from executing the SPARQL query are subsequently used to formulate a response via an LLM. We identified three unique patterns in users' queries, detailed below, and designed prompts specifically to address each input pattern. In the following discussion, we use the function decomposition tree illustrated in Figure 1 as an example.

3.3.1 Decomposition of *Actions* and *Ways*

In the first pattern, the tool infers and responds to the user specific actions or *ways* within the function decomposition tree. The procedure begins by extracting keywords from the natural language sentence provided by the user. These keywords are then utilized to perform a filtering process on the `:keyword` as described in Section 3.2. This filtering helps in narrowing down the search to identify the node that is considered to be the closest match to the user's input. Next, a SPARQL query is generated to search for *function nodes* or *way nodes* that indicate the approach to achieving the identified node. The results from executing this query are then used to construct the response. This SPARQL query functions to identify sub-nodes that exist beneath a specific node within a function decomposition tree.

Let us consider the input: `What are the ways to prevent virus invasion?` as illustrated in Figure 3. The tool extracts keywords from the input, specifically "prevent," "virus," and "invasion," and uses these to filter and search for relevant nodes. Subsequently, it identifies the subordinate nodes under the matched node and returns results such as "Mask-

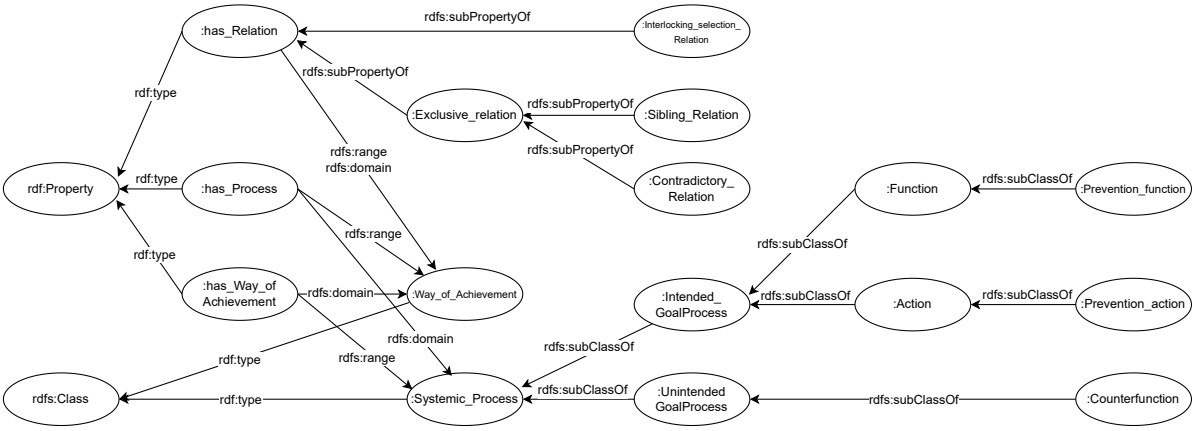

Figure 2: RDF Schema of function decomposition tree.

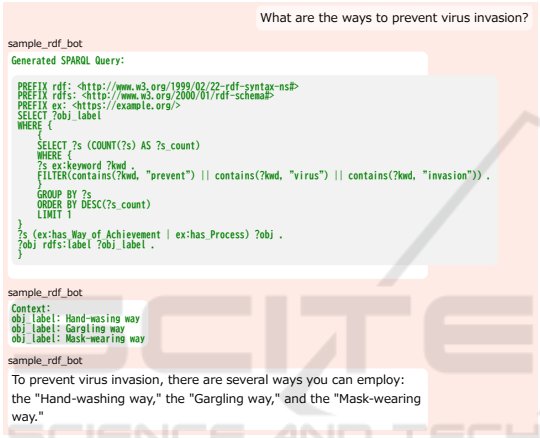

Figure 3: Decomposing an action.

wearing way," "Gargling way," and "Hand-washing way." Finally, using these results, the tool presents the following response to the user: `To prevent virus invasion, there are several ways you can employ: the "Hand-washing way," the "Gargling way", and the "Mask-wearing way."` This approach enables the tool to output actions related to the identified ways if the matched nodes are *way nodes*. Conversely, if the matched nodes are *action nodes*, the tool can suggest *ways* to achieve specific actions.

3.3.2 Identifying Purpose of *Actions*

In the second pattern, the tool infers and presents to the user a purpose or a goal of a action or function associated with a specific action or a way within the function decomposition tree. The procedure begins similarly to Section 3.3.1, where keywords are extracted from the natural language sentence provided by the user. These keywords are then used to filter nodes via the `:keyword` property, enabling the tool to identify the node that best matches the user's input. Subsequently, a SPARQL query is generated to search for *function node*s that are positioned above the specified node, which serves as the goal in either *function nodes* or *way nodes*. The results of executing this query are then used to construct the response. This SPARQL query is designed to locate action nodes that exist above a given node within the function decomposition tree.

Let us suppose a user inputs: `What is the purpose of wearing a mask?` as illustrated in Figure 4. The tool extracts "wear" and "mask" as keywords and uses filtering to locate the relevant node. It then searches for action nodes above this node in the tree (Figure 1), resulting in the output: `The purpose of wearing a mask is to "Prevent virus invasion into the body." The act of preventing virus invasion has a "prevention Function."` This approach allows the tool to output function nodes located above the node corresponding to the user's input. Additionally, it can present the type of function that the function node aims to achieve.

3.3.3 Identifying Dependencies Between *Ways*

In the third pattern, the tool identifies and presents dependencies between *ways* within the function decomposition tree to the user. The procedure begins by extracting keywords from a natural language sentence provided by the user, following the procedures similar to those in Sections 3.3.1 and 3.3.2. These keywords are then used to filter nodes to identify the *way node* that best matches the user's input. Next, properties are searched for where the node in question serves as the subject and another *way node* acts as the object. Finally, the results are utilized to generate a response. As defined in Section 3.1.5, a relation node is characterized by the *has_Relation* property,

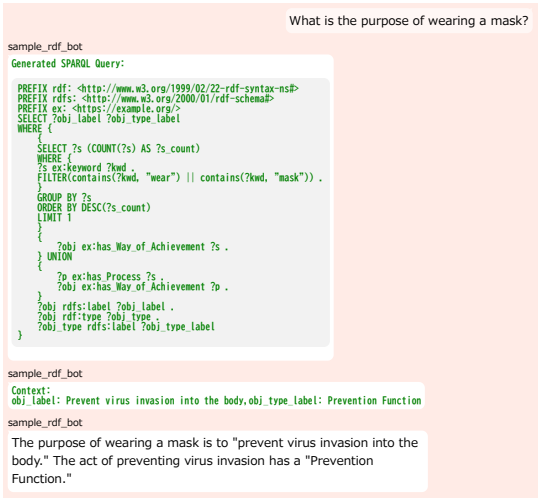

Figure 4: Searching for purpose of actions.

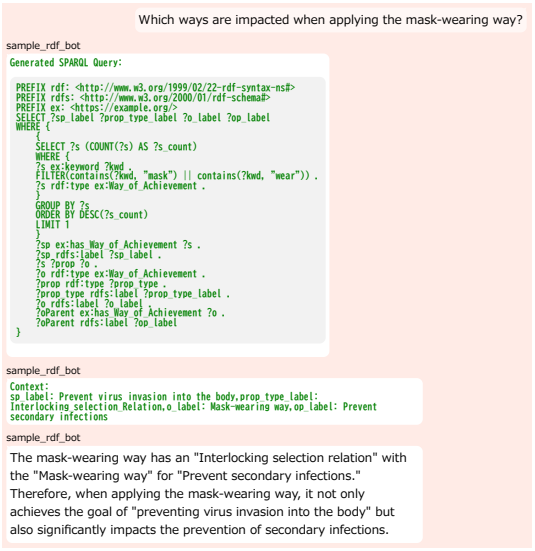

Figure 5: Searching for the implication of selecting the *way*.

which includes *way nodes* as both the subject and predicate. Consequently, this SPARQL query is designed to search for relation nodes associated with a particular way within a function decomposition tree. Let us consider the input: `Which ways are impacted when applying the mask-wearing way?` as illustrated in Figure 5. The tool then checks if the identified *way nodes* have any associated relational nodes. In this example, an "Interlocking selection" relation node exists, linking to the "Mask-wearing way." Using this connection, the tool generates the response: `The mask-wearing way has an "Interlocking selection relation" with the "Mask-wearing way" for "Prevent secondary infections." Therefore, when applying the mask-wearing way, it not only achieves the goal of "preventing virus invasion into the body" but also significantly impacts the prevention of secondary infections.` This response clarifies that using the mask-wearing way for infection prevention also helps prevent secondary infections in others. In this pattern, the tool provides the user with insights into additional outcomes that can be achieved when selecting a specific way to accomplish a particular action.

4 REAL-WORLD EXAMPLE

To demonstrate our approach, we present a real-world example involving the function decomposition tree about audit procedures in accounting (Taki et al., 2023). The function decomposition tree about auditing procedures in accounting is based on the document about asset accounting procedures (Financial Services Agency, 2002) published by the Financial Services Agency in Japan. The number of nodes in the tree is 247 and the complex connections among these nodes make manual searches exceedingly difficult. Since function decomposition tree is documented in Japanese, we conducted searches in Japanese. Below, we provide examples of operations for the three patterns discussed in Section 3.3.

4.1 Decomposition of *Actions* and *Ways*

As an example of the first pattern ("Decomposition of Actions and Ways") described in Section 3.3.1, we show how to search for the ways of performing *grouping* in accounting audits (Figure 6). Here, *grouping* refers to the process of dividing assets into independent units.

Let us suppose that a user inputs the query: `What are the ways for grouping assets?` The tool identifies keywords within the input sentence and utilizes them to filter nodes that align with the user's query. It then examines the sub-nodes linked to these relevant nodes and generates a response grounded in the search findings. In this example, the generated response is: `The ways to group assets include "when there is goodwill," "when there is no goodwill with shared assets," and "when there are shared assets."`

In the reference document, this information is dispersed across the sections titled "Handling of Goodwill," "Grouping of Assets", and "Treatment of Shared Assets", making it difficult to manually locate. By utilizing this tool, users can efficiently access and view multiple ways of achieving the *grouping* at once.

Figure 6: Searching for the ways of grouping assets.

4.2 Identifying Purpose of *Actions*

To show an example of the second pattern ("Identifying Purpose of Actions") described in Section 3.3.2, we search for the purpose of comparing current and projected future cash flows (Figure 7). Let us suppose that a user inputs a query: `What is the purpose of comparing the total amount of assets and the undiscounted future cash flows?` The tool extracts keywords from the input sentence and uses them to filter nodes that match the user's query. Then, the tool searches the super-nodes associated with the relevant nodes and formulates a response based on the search results. In this example, the generated response is: `The purpose of comparing the total amount of assets and the undiscounted future cash flows is "to recognize impairment losses."`

These details, as outlined in the reference document (Financial Services Agency, 2002), are described as procedural steps, making it challenging for users to understand the underlying purpose. By using this tool, users can efficiently search for the purpose behind their actions.

4.3 Identifying Dependencies Between *Ways*

In this section, as the example of the pattern described in Section 3.3.3, we search for implications of selecting the way in the function decomposition tree (Figure 8). For a user's query: `What are implications of selecting the book value allocation way?`, the tool extracts keywords from the input sentence and uses them to filter nodes that match the user's query. If the node that matches the user's query has *relation-node*, the tool searches the node that is connected to

Figure 7: Searching for the purpose of comparing cash flows.

Figure 8: Identifying dependencies of the "book value allocation way".

relation-node.

In this example, the generated response is: `The book value allocation way has a "linked selection relationship" with the way of archiving the act of "determining the grouping way for shared assets". By choosing this way, the grouping way for assets is determined.`

In the reference document (Financial Services Agency, 2002), the relationships between different ways for each action are not described. By using this tool, the relationships between the ways involved in each action can be clearly presented to the user.

5 DISCUSSION

As demonstrated in Section 4, the proposed tool is capable of efficiently retrieving specific information, even when applied to a large-scale function decomposition tree. For each of the three identified patterns, we successfully extracted and presented both *actions* and *ways* in response to users' queries, using an RDF-converted function decomposition tree. In particular, as shown in Section 4.3, implicit knowledge from the function decomposition tree can be extrated and presented to the user, offering insights not readily apparent in the reference document.

However, in some instances, the search functionality is not performed effectively. When user queries are brief, the limited number of available keywords may lead to the retrieval of incorrect nodes. In the demonstration in Section 4.2, the presence of additional keywords facilitated accurate retrieval. However, with six nodes containing both "assets" and "future cash flow," limited input keywords make it challenging to narrow down the nodes. This limitation can lead to the extraction of incorrect information. With the current tool, users need to formulate more precise or detailed queries to obtain the desired information in some cases.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a tool that provides users with goal-oriented knowledge through a natural language interface. Users can retrieve (1) finer-grained functions/actions, (2) goals/purposes, or (3) dependencies, of specified functions/actions. Such knowledge facilitates users' understanding and performance of functions of artifacts or human actions. The functional categories of the function decomposition tree, such as prevention and counter functions, particularly clarify teleological roles of functions.

It should be mentioned that the *functional categories* defined in Section 3.1 are currently only used for presentation to users, and not fully used in the search process. Furthermore, as discussed in Section 5, insufficient user's input can lead to incorrect results. Since the tool relies on extracting keywords from sentences, a limited number of keywords makes it challenging to narrow down potential answers. In future, we aim to integrate these *functional categories* more robustly into the search mechanism, complementing the existing keyword extraction approach.

ACKNOWLEDGEMENTS

This work was partially supported by JSPS KAKENHI Grant Number 24K15078.

REFERENCES

Avila, C. V. S., Vidal, V. M., Franco, W., and Casanova, M. A. (2024). Experiments with text-to-sparql based on chatgpt. In *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*, pages 277–284. IEEE.

Damljanovic, D., Agatonovic, M., and Cunningham, H. (2012). Freya: An interactive way of querying linked data using natural language. In *The Semantic Web: ESWC 2011 Workshops: ESWC 2011 Workshops, Heraklion, Greece, May 29-30, 2011, Revised Selected Papers 8*, pages 125–138. Springer.

Financial Services Agency (2002). Opinion on setting accounting standards for impairment of fixed assets. https://www.fsa.go.jp/news/newsj/14/singi/f-20020809-1/f-20020809c.pdf. Last accessed: 2024-11-07.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Kitamura, Y., Kashiwase, M., Huse, M., and Mizoguchi, R. (2004). Deployment of an ontological framework of functional design knowledge. *Advanced Engineering Informatics*, 18(2):115–127.

Kitamura, Y. and Mizoguchi, R. (2003). Organizing knowledge about functional decomposition. In *The 14th International Conference on Engineering Design*.

Kovriguina, L., Teucher, R., Radyush, D., Mouromtsev, D., Keshan, N., Neumaier, S., Gentile, A., and Vahdati, S. (2023). Sparqlgen: One-shot prompt-based approach for sparql query generation. In *SEMANTiCS (Posters & Demos)*.

Nishimura, S., Kitamura, Y., Sasajima, M., Williamson, A., Kinoshita, C., Hrao, A., Hattori, K., and Mizoguchi, R. (2013). Charm as activity model to share knowledge and transmit procedural knowledge and its application to nursing guidelines integration. *Journal of Advanced Computational Intelligence Vol*, 17(2):208–220.

Taki, H., Sogawa, S., Mura, K., and KItamura, Y. (2023). Ontological approach for modeling of a financial statement audit -a model of actions of an audit of impairment of fixed assets-. In *The 37th Annual Conference of the Japanese Society for Artificial Intelligence*, pages 2L6GS305–2L6GS305. The Japanese Society for Artificial Intelligence.