Yang, J. C. et al. (Eds.) (2018). Proceedings of the 26th International Conference on Computers in Education. Philippines: Asia-Pacific Society for Computers in Education

Proposal of an Adaptive Programming-Learning Support System Utilizing Structuralized Tasks

Kento KOIKE[a*], Takahito TOMOTO[b], Tomoya HORIGUCHI[c] & Tsukasa HIRASHIMA[d]
[a]*Graduate School of Engineering, Tokyo Polytechnic University, Japan*
[b]*Faculty of Engineering, Tokyo Polytechnic University, Japan*
[c]*Graduate School of Maritime Sciences, Kobe University, Japan*
[d]*Graduate School of Engineering, Hiroshima University, Japan*
*m1865005@st.t-kougei.ac.jp

Abstract: The authors have examined properties of programming tasks and have studied a framework for structuring tasks based on their properties. In this paper, we examine what is learned from general–special and whole–partial relations among tasks. We also consider a method for presenting tasks in a method that is adaptive to learners' comprehending situation using this framework. Under this framework, we propose a system that presents adaptive tasks based on the method of utilizing the task sequence.

Keywords: Programming education, stepwise understanding, structural understanding, structured task, graph of microworlds.

1. Introduction

The demand for programming education has increased in recent years, as evidenced by new curriculum standards in Japan (Kanemune, 2018). Meanwhile, there is increased need for frameworks that efficiently support programming learning. In particular, frameworks for implementing computer support include intelligent tutoring systems (ITSs) and learning management systems (LMSs) (e.g., Enomoto, Miyazawa, Miyadera, & Morimoto, 2018; Kanemune, Nakatani, Mitarai, Fukui, & Kuno, 2004; Shinkai & Sumitani, 2008; Tomoto & Akakura, 2017).

In ITS and LMS applications, the difficulty and order of presented tasks cannot be disregarded. Koedinger & Aleven (2007) described the relation between task difficulty and the amount of task support, which they refer to as the "assistance dilemma." If tasks are difficult when learners are provided with less support, making it harder for learners to solve tasks by themselves. If tasks are easy when learners are provided with too much support, however, thus learners do not consider them fully. Koedinger et al. thus concluded that addressing the assistance dilemma required stepwise presentation based on the learner model. To that end, Enomoto et al. (2018) proposed a method for determining the difficulty of programming tasks from correct answer rates among all learners and making adjustments for individual learners. Through this method, we can objectively observe and adjust the difficulty of each task.

Although such methods allow determining the order of tasks presented to a learner, they cannot determine what to learn through the sequence of tasks, in other words, the overall learning goals (Hirashima, Niitsu, Hirose, Kashihara & Toyoda, 1994; Horiguchi, Tomoto & Hirashima, 2015). If the task sequence is meaningful, it becomes possible to diagnose which aspect of the whole concept is currently being learned. It has also been reported that properly deciding learning content and sequence promotes learning (Scheiter & Gerjets, 2002). One approach toward making task sequences meaningful is to focus on both within- and between-task structure (e.g., Hirashima, Kashihara & Toyoda, 1995; Horiguchi & Hirashima, 2005).

To determine appropriate problems for learning programming, it is first necessary to analyze the problem to be solved in ordinary programming, then structure the properties necessary

for solving the problem. Properties required for programming problem-solving ability can be determined by structuring properties included in programming problem-solving. Furthermore, by giving learners tasks generated using a task framework that includes structured properties, we can contribute to problem-solving ability for programming.

The authors have previously discussed properties of tasks in programming and the structuring of these tasks (Koike, Tomoto, Horiguchi & Hirashima, 2018) with the goal of sequencing tasks based on within- and between-task structure. Based on our previous investigations of the properties and structure of programming tasks, this paper describes how to adapt to learners' comprehension state when presenting tasks to them. We then propose a learning support system for doing so.

2. Structuralizing Programming Tasks

As mentioned in Section 1, adapting to learners' comprehension state when presenting tasks requires defining what the programming task is. Doing this requires structuring in which the programming task properties are interpretable. In this chapter, therefore, we describe programming-task properties that the authors have investigated for structuring (Koike et al., 2018).

2.1 Related Research

Watanabe, Tomoto, Fujimori, & Akakura (2017) structured programming tasks by categorizing properties in task nodes as (i) purpose of the code, (ii) actual processing (code), or (iii) learner tasks for confirming task understanding. Watanabe et al. also constructed task sequences by relating task nodes as general–special or whole–partial relations. General–special relations associate original general tasks with the same purpose with a special task that reduces factors to be considered by adding constraints on actual code. In whole–partial relations, whole and partial tasks with actual code have an inclusion relation by which different purposes are related. Through these relations, Watanabe et al. aimed at improving learner understanding by having learners shift from special to general situations, or shift from partial to whole tasks.

In that study, however, even when specific processes correspond with a specific purpose, there is no inherent process for achieving a certain purpose; multiple possibilities exist. A large example is rearrangement of arrays. Normally, array rearrangement can be achieved by various processes, such as bubble or merge sorts. In small examples, it is possible to swap the values of two variables using "a = b, b = c, c = a" or "t = ary [i], ary [i] = ary [j], ary [j] = t." There are thus multiple processes that achieve the same purpose and Watanabe et al.'s definition cannot distinguish between them, because doing so requires clarification of relations between multiple tasks for the same purpose. To clarify this relation, it is necessary to consider differences in behavior between processes. If premised on gradual task presentation, it is desirable that previously learned tasks are included in the task to be learned next (Scheiter & Gerjets, 2002). Considering this, learned tasks should have an inclusion relation with the next task to be learned, and following this inclusion relation will ultimately be the overall task goal. Therefore, identifying the objective of the overall task is necessary to identify the objective of the partial task. However, when the partial task depends on the whole task in this way, it is not possible to reuse the processing of a certain task in general for other tasks. Therefore, to reuse the processing of a certain task, it is necessary to describe what function the task has as a part.

Therefore, in this research we aimed at making programming tasks more reusable by separately discussing their function, behavior, and processing properties.

2.2 Properties of Programming Tasks

To structure programming tasks and clarify their properties, we first consider how the programming task is usually solved. In the general problem-solving process, an initial state is given first, and the learner must achieve the target end state through manipulation using a series of operations. When comparing this with problem-solving processes in programming learning, we consider the initial state of a variable as given input, and the target end state after manipulation of variables through programming code as output (Fig. 1). Here, the difference between pre- and post-processing states

can be interpreted as behavior brought about by the processing. Therefore, the behavior exhibited in the process can be understood through the difference between the initial and end states of the variable. Behavior can also be interpreted as a function according to the human interpretation. The definitions and properties of processing, behaviors, and functions are explained below in reference to Fig. 2.

2.2.1 Processing

The processing in this research is a manipulation sequence, namely, a set of commands that causes a difference between the initial and end states. Thus, the smallest operation sequence is a single command. Commands include control statements such as assignment statements, “if” statements, and “for” statements.

Task processing in Fig. 2 is described using pseudocode, because this research focuses on general understanding that does not depend on a specific programming language. In that figure, variables beginning with “$” are arguments, and variables starting with “%” are local variables. Other statements are merely descriptive and not intended to refer to concrete function calls, etc.

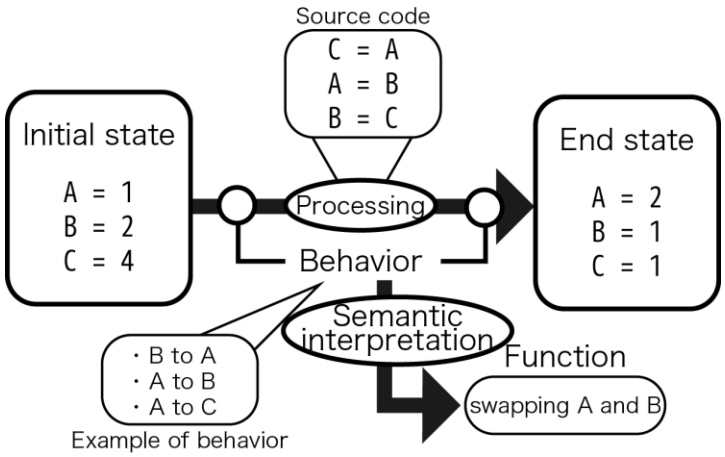

Figure 1. The problem-solving process in programming

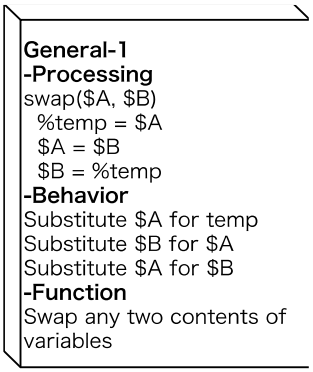

Figure 2. Example of a task

2.2.2 Behavior

“Behavior” is the processing that results in differences between the initial and end states of variables, in other words the behavior of the operation sequence. Variables here include iterators such as those used in “for” statements. In other words, we do not distinguish specific scopes (such as global or local) of variable states.

In Fig. 1, three constraints (such as "B to A", "A to B", and "A to C") are always satisfied between the initial and end state. From this also, the behavior itself is described as a set of constraints

that can be read from the change of variables between input and output. In addition, in Fig. 2,%temp is assigned to $B as processing, but the value of $temp is $A by "%temp = $A" which is the previous processing. Therefore, $A is finally assigned to $B.

2.2.3 Function

A function is a human interpretation, and describes what can be achieved through behaviors. Sasajima et al. stated that functions are not interpretations of behavior without excess or deficiency, but rather interpretations of a certain behavior (Sasajima, Kitamura, Ikeda & Mizoguchi, 1996). They furthermore suggest that when a given function is regarded as part of an overall function, the function has meaning as a part only when there exists a function one level higher than the given function. For example, if the overall function is swapping two variables, the “temp” variable has meaning, but the swapping of the two variables itself has no meaning. This handling of such functions is advantageous in that the positioning of the function is certain within the system, so there will be no erroneous function positioning. However, when a target system and its function are positioned, it becomes difficult to reuse the function in other target systems. Therefore, in this research we consider functions as not necessarily connecting with goals on a one-to-one basis, and we position functions on the assumption of possible overall goals for each function. For example, in Figure 2 at the behavior level, it just repeats the assignment and there are no meanings to these three assignments. By semantically interpreting this, we can define a function called “swap”.

2.2.4 Relationship between Processing, Behavior, and Function

The relation between processing, behavior, and function can be described using the example of sorting. For example, simple sorting and selective sorting are different processes, but their behaviors are similar in that both rearrange an array. Also, when considering each function, they become the same in terms of their interpretation as “rearranging an array.” In this way, the properties of processing, behavior, and function can be independently related.

3. Relationship between Tasks

It is possible to compare tasks by relating them based on their processing property, as defined in Chapter 2. When tasks can be compared, they can be related according to the differences between them. This also makes it possible to determine which tasks have higher and lower difficulty in comparison to tasks that learners have solved. That is, if tasks are related, it is possible to adapt to the comprehending state of learners and prompt their transition to another task.

In this research, we define general–special and whole–partial relations based on differences between these tasks (Fig. 3). In Fig. 3, whole–part relations are shown with black arrows starting at the part side and ending at the whole side. General–special relations are graphed as white arrows starting at the special side and ending at the general side.

3.1 General–special Relation

Focusing on Moderate-1 and Moderate-2 in Fig. 3, we can see that the iterator for array *a* uses variables $I and $J. In this way, we associate the special task in the general–special relation in the sense that it can be handled in a more specific situation and the general task in the sense that it can be handled in a more general situation. Learners can learn how to make general-purpose tasks by shifting tasks linked to this relation.

3.2 Whole–partial Relation

Between Special-1 and Special-2 in Fig. 3, it can be seen that swap-1 includes processing of sort-1. Therefore, Special-2 is a whole task including Special-1, and Special-1 is an included partial task.

The whole–partial relation associates tasks in such inclusion relations. As a result, the learner experiences reusing partial tasks learned while shifting from partial to whole tasks.

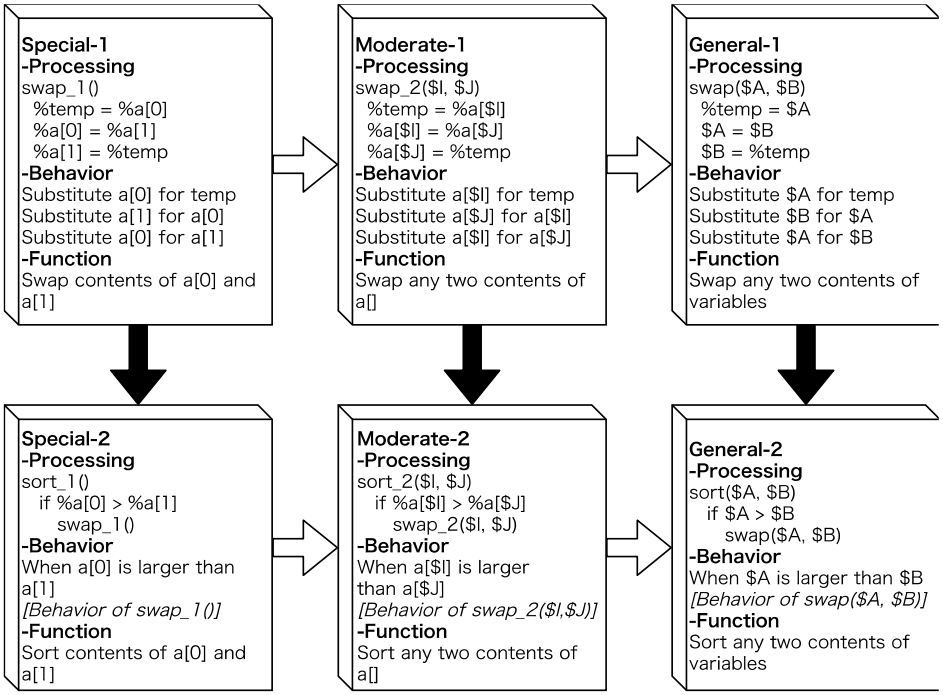

Figure 3. Example task relations

4. Functionalization of Tasks

The authors have previously tried to construct a task sequence based on the discussion in Sections 2 and 3 (Koike et al., 2018). Therefore, in this chapter we propose a task sequence for expansion and correction activities as a viewpoint from which to newly construct tasks based on past task sequences.

In task sequences that the authors have examined thus far, from the viewpoint of structuring, the function in the task is implicitly treated as having arguments. However, if reuse of exactly the same processing is desired, for example when using the exact same process in ten places, the processing requires no arguments. In this case, it is more convenient to functionalize the process. Adding arguments to the function improves its versatility at the cost of convenience in special circumstances. In this way, understanding argumentation for processing as a general–special relation explains formation of the function. Therefore, it is possible to contrast whole–partial relations with general–special relations. By contrasting these relations, learners can understand how general–special relations play a role in learning about general versatility.

From the above discussion, there is room for reconsideration of handling arguments in tasks. Since arguments were not considered until now, we implicitly regarded tasks as functions, though we did not specify them as functions. Therefore, in this paper we can discuss the versatility and specialty of tasks by newly adding a description of their function. We can furthermore grasp task sequences in two dimensions, taking whole–partial and general–special relations as axes. In other words, in whole–partial relations it is possible to learn the importance of understanding the behavior of partial tasks, and to reuse partial tasks by repeating them for multiple whole tasks. On the other hand, we consider that by experiencing multiple special tasks related to the same general task in general–special relations, it is possible to learn the importance of providing special tasks with flexibility to make them more general-purpose. Therefore, in the following we will describe a method for actually utilizing the system by following the two-dimensional task sequence.

5. How to Present Structured Tasks in a System

In this section, we discuss how to use a system developed by the authors (Fig. 4) to present tasks based on a newly constructed task sequence (Fig. 5) (Koike & Tomoto, 2018).

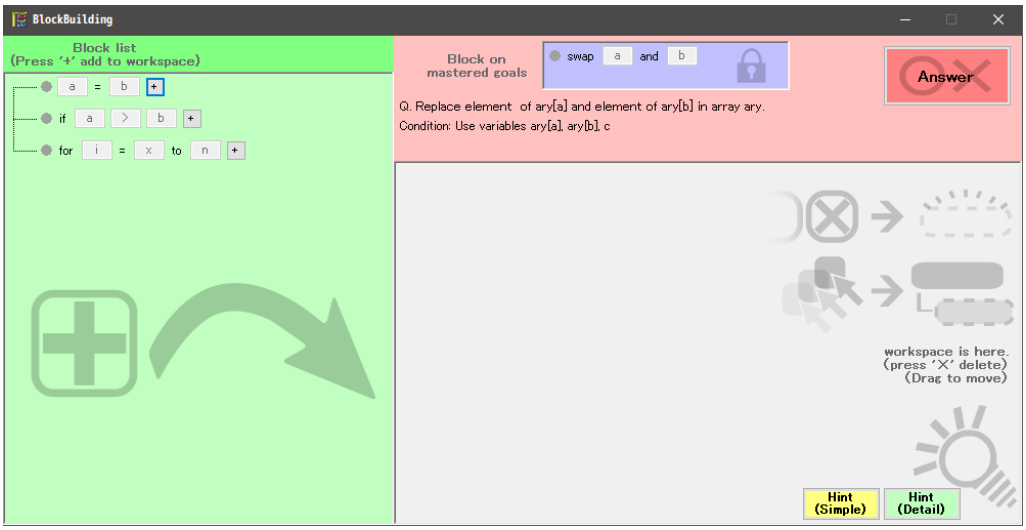

Figure 4. User interface of the system (Koike & Tomoto, 2018)

5.1 Presentation of Tasks in a Conventional System

In a learning support system developed by the authors (Fig. 4), the goal is to understand chunks of code as parts in a program (Koike & Tomoto, 2018). In this system, programs are constructed according to the problem and constraints displayed at the upper right. In addition, previously constructed programs are retained in the list on the left as reusable parts, and larger parts are newly constructed using them. In other words, learners are to understand two things through use of this system: (1) reuse of partial tasks in other tasks in whole–partial relations, and (2) increasing versatility by shifting from special to general tasks in general–special relations.

Until now, however, presented tasks were generated based on task sequences subjectively set by the authors, and these did not adapt to the learner's comprehending state. Task sequences determined by the authors included all tasks related through whole–partial relations, in a sequence without branches. This does not allow the experience of reusing partial tasks within two or more whole tasks. Also, since there is no general–special relation, it is difficult for the learner to learn more general task structuring. We thus consider a method for adaptively presenting tasks by using the relations defined in this paper. Also, when handling tasks in the system based on the framework, a correct answer are prepared on the system that is limited to one out of multiple correct answers.

5.2 Adaptive Task Presentation using Task Sequences

In this section, we consider a method for implementing and practicing task sequences (Fig. 5) newly constructed in the system described above. In Fig. 5, the start points of black arrows indicate parts, and end points indicate the whole, thus showing the whole–partial relation. The start points of white arrows are special and end points are general, thus showing special–general relations.

We first consider how to implement a single task. By using the "function" label of a task, it is possible to produce a sentence "create [function name]" as a system problem statement. When doing so, in system processing of the variable name, we have to decide what constraints are imposed upon making the function. Then, using tasks, it is possible to generate sentences such as "using [% appended variable]" or "using [used function]" as a task constraint. Furthermore, considering the judgment of correct answers in the system, it is possible to compare the operation described in the "processing" label of the task with the answer from the learner. In this way, implementation is

possible by applying the task in this paper to the system. For example, in the case of General-2, “use A and B as input variables and use swap functions” can be generated as a constraint on the problem statement “make a program that swaps the values of input A and input B.”

By implementing tasks in the system, it is possible to present tasks in consideration of task sequences. For example, when a learner solves the Special-2 task in Fig. 5, Moderate-2 exists as a task for more generically learning this concept, and Special-3a exists as a task for learning a new concept. This makes it possible to diagnose positioning of the solved problem in the task sequences, specifically which task should come next. Therefore, if the goal is learning a new concept, the system can selectively present tasks demonstrating whole–partial relations, or if the goal is to learn abstract (generic) concepts, the system can selectively present tasks demonstrating general–special relations. For example, when shifting from Moderate-1 to Moderate-2 to learn the whole–partial relation, the system will assist in learning Moderate-2 by assigning Special-2 to learners for whom Moderate-2 would be difficult. Alternatively, to learn general–special relations, when a learner who shifted from Special-1 to Moderate-2 next shifts to Moderate-3b, the system would give Moderate-3a to learners for whom Moderate-3b would be difficult. To further simplify the task, Special-3a might be given to assist learning. In this way, when one task cannot be solved, another can be presented as an auxiliary task, and task branching that could not be performed in previous task sequences can be represented within the system.

In these task sequences, if a sequence is composed of general tasks like those in Fig. 5, special tasks are generated in the sequence by fixing and diversifying arguments as instances. By doing this, we consider that learning via task sequences can be generalized to general tasks by learning multiple special tasks based on a certain general task.

As the number of concepts to be learned increases, the number of tasks that can be expressed to assist stepwise learning increases exponentially. Therefore, the example in Fig. 5 is not intended to exhaustively express all task sequences.

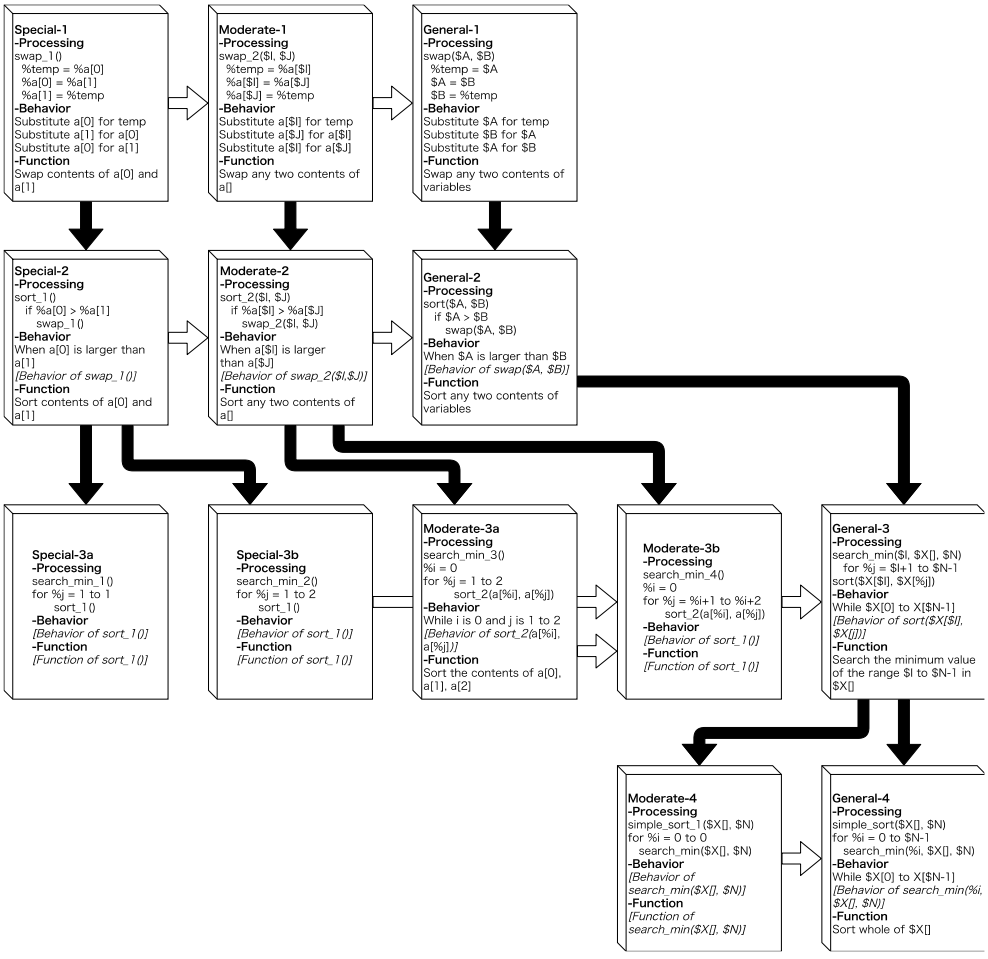

Figure 5. Example of a task sequence using the system

6. Proposal of an Adaptive Support System for Learning Programming

The proposed system, which is based on a system previously developed by the authors (Koike & Tomoto, 2018), performs adaptive task presentation using task sequences. To do that, we first need to express the generalized task in the system, as described in Section 5.2. For example, (1) the problem statement might be “create [function name],” (2) the constraint might be “using [% appended variable]” or “using [used function]”, and (3) judgement of the correct answer might be based on comparison of the contents of [processing] and the learner-provided answer. If such a task can be expressed in the system, it is possible to present the task as task sequences. For example, if the goal is to learn the versatility of the task as in Fig. 6, the system might present problems in order 1, 2, 3, ..., N, based on the task sequence. In this case, learners who have solved 1 and 2 are expected to notice that processing in 1 and 2 have the same structure. By solving 3, learners can learn 3 as a generic solution for 1 and 2. Finally, learners will reach task N, through which they will acquire the most versatile structure of the swap function. In this example, Problem 1 and Problem 2 can handle only variables and arrays. However problem N can handle multiple cases.

And then, we need to connect each task for whole-part relation in the system. For example, if the goal is to learn the new concept or the reusability of the task as in Fig. 7, even to learn like when learning the generalized task.

With the two structures shown in Fig. 6 and Fig. 7, it is possible to present the whole task of "Special-" tasks, "Moderate-" tasks or "General-" tasks. When such a task can be presented, if learners cannot solve "General-" tasks then they can learn from "Special-" task to generalization. If learners cannot solve the whole task of "General-" then firstly they can learn the whole task of "Special-" tasks.

By utilizing such a framework in the system, it is possible to present task sequences that are adaptive to learners' comprehending.

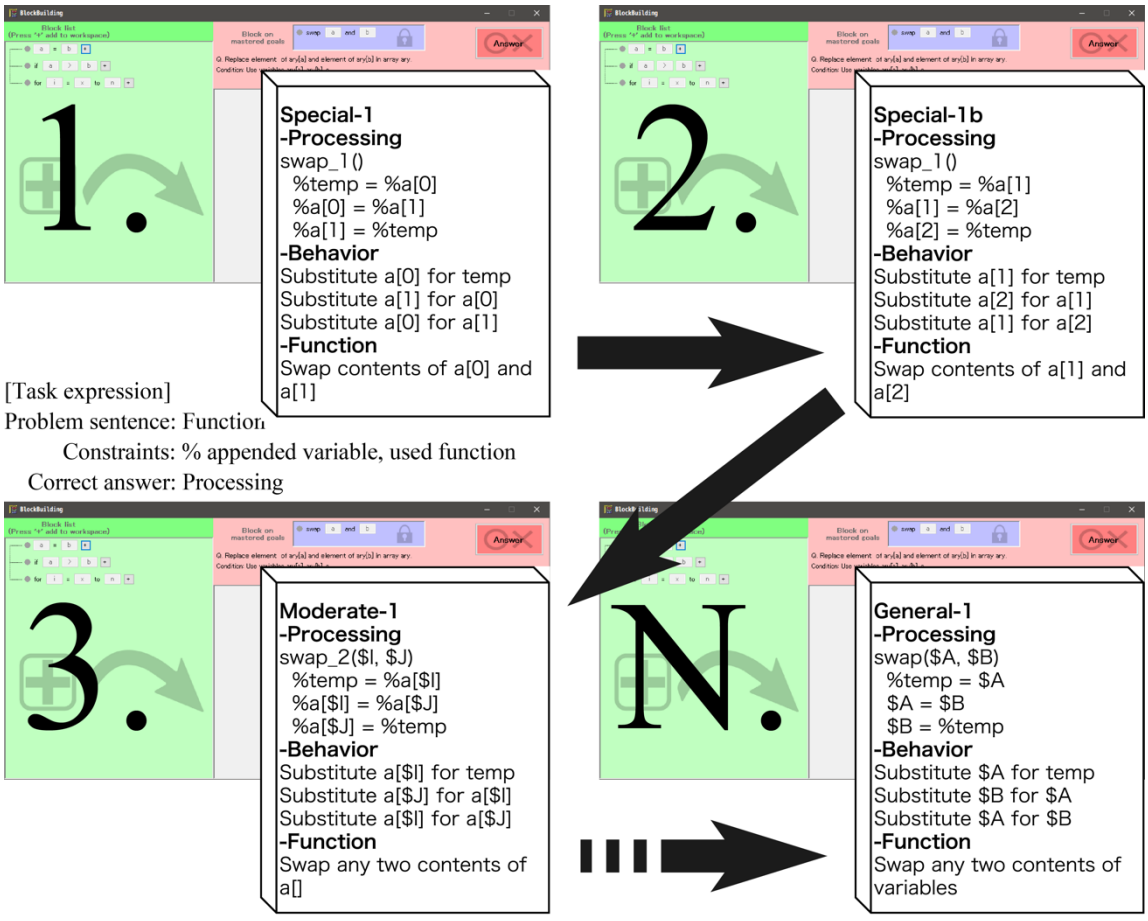

Figure 6. Example of task presentation using the general-special relations in the system

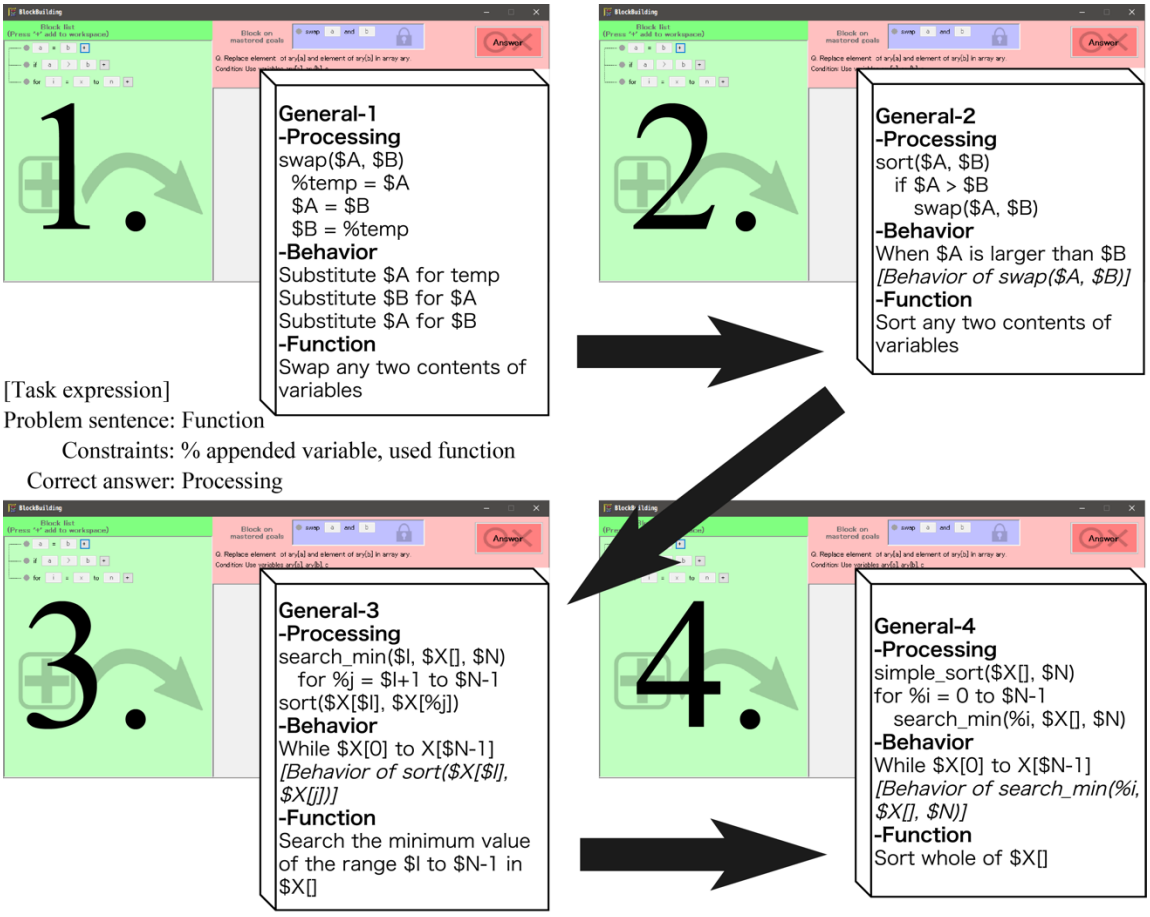

Figure 7. Example of task presentation using the whole-part relations in the system

7. Conclusion

This paper organized task sequences that could be applied in practice. Clarifying what is learned through whole–partial and general–special relations provided a viewpoint for evaluation of task sequence on two axes.

By doing this, we developed a task sequence on the premise of applied in practice and discussed a utilization method of tasks linked through whole–partial and general–special relations. We also proposed a system that presents adaptive tasks based on the method of utilizing the task sequence.

In the future, we will consider methods for generating variations on understanding tasks within tasks and for detailing functional representations of tasks. Furthermore, in this framework, tasks were specialized and partialized based on the meaning. Thus, we would like to deeply discuss how to define its meaning on the link.

Acknowledgements

This study was partially funded by Grants-in-Aid for Scientific Research (C) (18H11586) and (B) (17H01839) in Japan.

References

Enomoto, M., Miyazawa, Y., Miyadera, Y., & Morimoto, Y. (2018). Adaptive Programming-Learning Support System Using a Fill-in-the-blank Worksheet Based on Item Response Theory. *Transactions of*

Japanese Society for Information and Systems in Education, *35*(2), 175-191. http://doi.org/10.14926/jsise.35.175 (in Japanese).

Hirashima, T., Kashihara, A., & Toyoda, J. I. (1995). A Formulation of Auxiliary Problems and Its Evaluations. Proc. of AI-ED95, 186-193.

Hirashima, T., Niitsu, T., Hirose, K., Kashihara, A., & Toyoda, J. I. (1994). An indexing framework for adaptive arrangement of mechanics problems for ITS. *IEICE TRANSACTIONS on Information and Systems*, *77*(1), 19-26.

Horiguchi, T., & Hirashima, T. (2005). Graph of Microworlds: A Framework for Assisting Progressive Knowledge Acquisition in Simulation-based Learning Environments. In AIED (pp. 670-677).

Horiguchi, T., Tomoto, T., & Hirashima, T. (2015). The Effect of Problem Sequence on Students' Conceptual Understanding in Physics. In S. Yamamoto (Ed.), Human Interface and the Management of Information. Information and Knowledge in Context (pp. 313–322). Cham: Springer International Publishing.

Kanemune, S. (2018). Programming Tools and Education in New Curriculum. *Transactions of Japanese Society for Information and Systems in Education*, *35*(2), 104–110. http://doi.org/10.14926/jsise.35.104 (in Japanese).

Kanemune, S., Nakatani, T., Mitarai, R., Fukui, S., & Kuno, Y. (2004). Design and Implementation of Object Sharing for Dolittle Language. *Journal of Information Processing*, *45*(5), 81. Retrieved from http://ci.nii.ac.jp/naid/110002712352/en/ (in Japanese).

Koedinger, K. R., & Aleven, V. (2007). Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psychology Review*, *19*(3), 239-264.

Koike, K., & Tomoto, T. (2018). Proposal of Expandable Modular Statements for Structural Understanding on Programming and Development of Leaning Support System. *Transactions of Japanese Society for Information and Systems in Education*, *35*(2), 215-220. http://doi.org/10.14926/jsise.35.215 (in Japanese).

Koike, K., Tomoto, T., Horiguchi, T., & Hirashima, T. (2018). Proposal of a Framework for Stepwise Task Sequence in Programming. *International Conference on Human Interface and the Management of Information*, 266-277.

Sasajima, M., Kitamura, Y., Ikeda, M., & Mizoguchi, R. (1996). A representation language for behavior and function: FBRL. *Expert systems with applications*, *10*(3-4), 471-479.

Scheiter, K., & Gerjets, P. (2002). The impact of problem order: Sequencing problems as a strategy for improving one's performance. In *24th Annual Conference of the Cognitive Science Society* (pp. 798–803).

Shinkai, J., & Sumitani, S. (2008). Development of Programming Learning Support System Emphasizing Process. *Japan Journal of Educational Technology*, *31*, 45-48. http://doi.org/10.15077/jjet.KJ00004964344 (in Japanese)

Tomoto, T., & Akakura, T. (2017). Report on Practice of a Learning Support System for Reading Program Code Exercise. *International Conference on Human Interface and the Management of Information*, 85-98.

Watanabe, K., Tomoto, T., Fujimori, S., & Akakura, T. (2017). Design and Development of the Function to Set Auxiliary Problems in the Process of Program Meaning Deduction. *IEICE Technical Report*, *116*(517), 85-88. Retrieved from https://ci.nii.ac.jp/naid/40021161228/en/ (in Japanese).