Chang, M. et al. (Eds.) (2019). Proceedings of the 27th International Conference on Computers in Education. Taiwan: Asia-Pacific Society for Computers in Education

Effects on Fostering Computational Thinking by Externalizing a Solution with Construction of a Problem-Solving Model

Kazuaki KOJIMA[a*] & Kazuhisa MIWA[b]
[a]*Learning Technology Laboratory, Teikyo University, Japan*
[b]*Graduate School of Informatics, Nagoya University, Japan*
*kojima@lt-lab.teikyo-u.ac.jp

Abstract: Literature on education has paid considerable attention to computational thinking (CT), thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent. Many attempts to develop CT in problem-solving education have been made in K-12 education. This study proposes an approach to foster CT in curricula for general undergraduate students: students construct rule-based computational models of problem solving. We empirically investigated effects of construction of a rule-based model on externalizing a solution in problem-solving in terms of problem decomposition, one aspect of CT. Undergraduate students described knowledge required to solve a simple problem in the format of rule-based models before and after they constructed models of the problem for a production system. Results indicate that model construction improved student decomposition in knowledge externalization of the solution.

Keywords: Computational thinking, learning by modeling, production system

1. Introduction

Literature on education has paid considerable attention to computational thinking (CT), which refers to thinking like a computer scientist. In fact, CT has great influence not only in the natural sciences of physics, chemistry, and biology, but also in psychology, economics, literature, and psychiatry. In other words, the activity of computer scientists is now spreading into broad disciplinary areas.

Although CT has not yet been clearly defined, most researchers seem to agree on the definition by Cuny, Snyder, and Wing (2010): *thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent* (*e.g.*, Aho, 2012; Brennan, & Resnick, 2012; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). Such skill must not be reserved only for experts in information engineering, science, and other academic areas, but instead, must be applicable by anyone who engages in problem solving. In this respect, various efforts to implement CT training in education should be emphasized. Above all, many attempts to develop CT in problem-solving education have been made in K-12 education (*e.g.*, Barr, & Stephenson, 2011; Bocconi, Chioccariello, Dettori, Ferrari, Engelhardt, Kampylis, & Punie, 2012; Grover, & Pea, 2013; Yadav et al. 2014). Such CT training generally adopts programming as a tool to represent problem solutions in learning activities. Popular systems used in CT training are graphical programming environments and web-based simulation authoring tools (Grover, & Pea, 2013), which are easy to use for learners who are not information-engineering students.

Here, we propose an approach to foster CT in curricula for general undergraduate students. In fact, in an article on CT, Jeannette M. Wing (2006) insists that professors in computer science should teach university freshmen subjects such as “a way to think like a computer scientist” in all departments, not just the department of computer science. In our approach, students experience modeling like a computer scientist. Actually, they construct rule-based computational models of problem solving. Scientists in cognitive science have used computational models as research tools for in-depth

understanding of the human mind. Such model construction is expected to improve skills for externalizing knowledge required in problem solving, and that must foster some components of CT.

The current study investigated effects of construction of a rule-based model on the externalization of knowledge required to solve a simple problem. We empirically confirmed how undergraduate students externalize knowledge of a problem solution and whether construction of a computational model improves students' knowledge externalization. We particularly focused on decomposition of a problem solution as one aspect of CT, which is described in the next session.

The remainder of this paper is structured as follows. Section 2 briefly explains part of our approach's theoretical background. Section 3 describes the empirical investigation's method for confirming the research questions. Finally, Section 4 provides study results and discusses them.

2. Theoretical Background

Many researchers have continuously discussed various aspects of CT. Although CT does not yet have a clear definition, most researchers agree on some of its main components, for example, decomposition of problems (modularizing) and recognition and generalization of patterns, abstractions, and automation (algorithms) (*e.g.*, Barr, & Stephenson, 2011; Grover, & Pea, 2013; Krauss, & Prottsman, 2017).

In our approach, students construct computational models that can simulate problem solving for a production system. Therefore, they must disassemble solution processes in problem solving into separate operations and set conditions to adapt the operations to proper states. To implement operations in a computational model, students might cut certain details of the operations derived. Finally, students implement production rules under the specifications so that the production system reproduces the solution. These steps are regarded to involve the following four CT components proposed by Krauss and Prottsman (2017): *decomposition*, *pattern recognition*, *abstraction*, and *automation*.

Our preliminary study (Kojima, & Miwa, 2018) confirmed that model construction improved the pattern recognition in externalizing knowledge required for a simple problem when decomposition of the problem was supported. Students learned to describe more appropriate conditions for knowledge to solve the problem after they had experienced construction of a rule-based model. The current study focused on the skill of decomposition. Mvalo and Bates (2018) studied students' use of decomposition as a CT component in problem-solving tasks to design and troubleshoot computer networks using simulation software. In the study, focus group interviews to undergraduate and postgraduate students confirmed that they used decomposition in the tasks. However, this study reported only results of qualitative research in the domain of information engineering. To accommodate our goal to foster CT for general undergraduate students, we must precisely investigate how non-information-engineering students perform decomposition of knowledge. Therefore, we examined the following two research questions.

RQ1 How do undergraduate students externalize knowledge required to solve a problem within the format of rule-based models in terms of decomposition?

RQ2 Does construction of a computational model for the problem improve students' decomposition in knowledge externalization of the solution?

We have developed a framework for learning in which general undergraduate students create models of human cognitive processes. This framework uses a production system for novices called DoCoPro (an anywhere production system) (Miwa, 2008; Miwa, Nakaike, Morita, & Terai, 2009; Nakaike, Miwa, Morita, & Terai, 2009). We adapt learning by constructing models to a learning framework for fostering CT.

Figure 1 shows a screenshot of DoCoPro. A student creates a model of solving a problem by inputting the initial states in the working memory in the left frame, editing if-then format rules in the right frame, and simulating problem-solving processes by executing the model with the controller in the upper frame. For allowing novices to experience model construction, DoCoPro limits its constructs. Students have only to learn about if-then rules, matching, and some built-in functions (e.g., functions to test whether two values are equal and to add an assertion to the working memory). It has no functions to perform simulation effectively and to represent human cognitive functions helpful in scientific research. Instead, it helps students examine rules through trial and error by providing functions to test the rules in a variety of ways. Furthermore, DoCoPro has instructional texts that help students learn how to create a

model with an example of a simple block-stacking problem in addition to basic concepts of a production system.

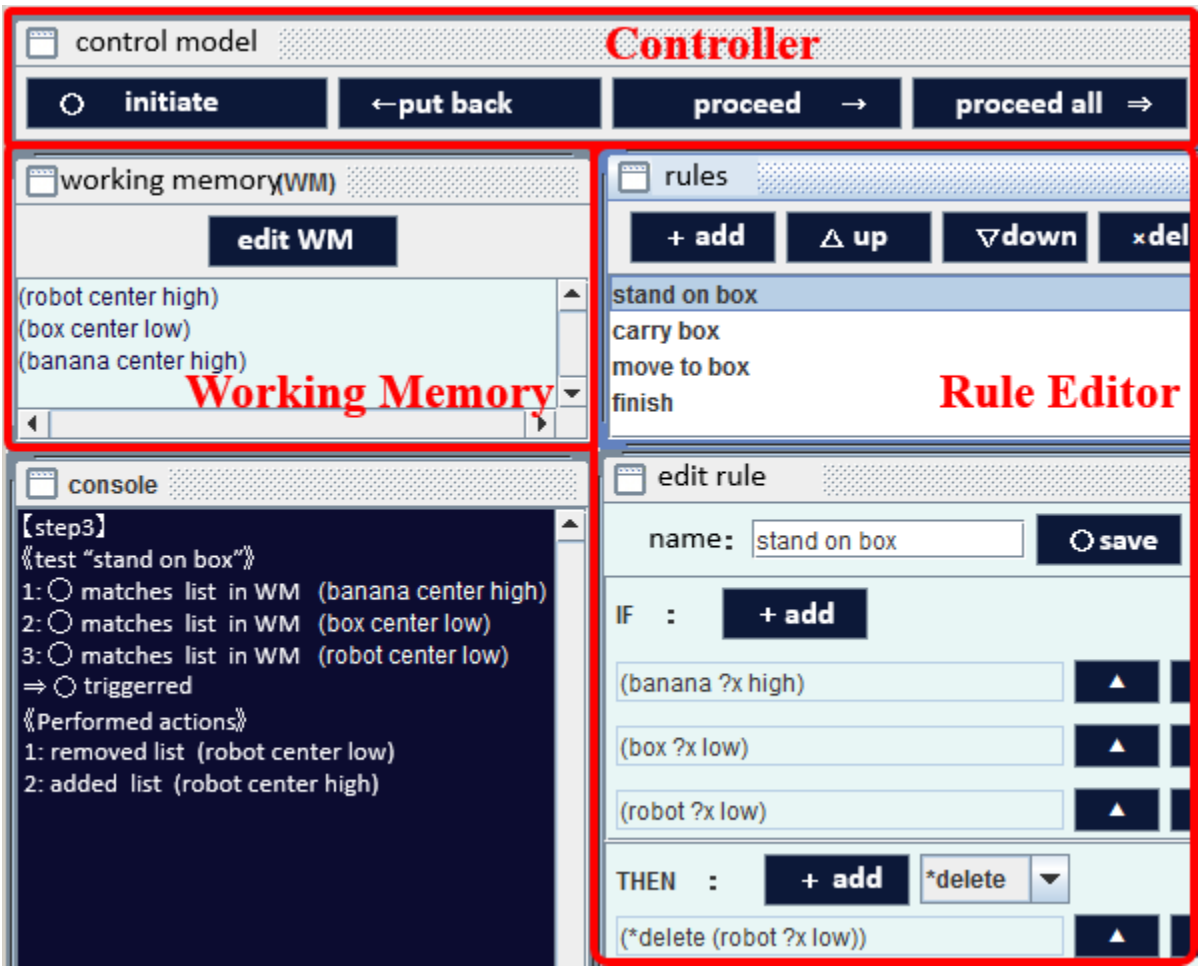

Figure 1. Part of Screenshot of DoCoPro.

3. Method

3.1 Procedures & Materials

We conducted lectures in which undergraduate students constructed models of problem solving in a cognitive science class. Although most students had experienced programming in other classes, they had not experienced training from experts in information engineering. In the cognitive science class, two lectures were conducted for model construction. In the first lecture, students learned about a production system with instructions from the lecturer and the first part of the instructional texts of DoCoPro. They actually experienced creating an if-then rule with DoCoPro.

Students were then presented a robot and banana problem, an altered version of the famous toy problem monkey and banana. Figure 2 illustrates initial and goal states of the problem. We used this quite simple problem because it requires problem decomposition but even novices are expected to succeed in model construction. When this problem was presented to students, they received Figure 2 and the following descriptions about the solution: "To have the robot get the banana, have the robot move to the same position as the banana. The robot can move into a high position by standing on the box. The robot can also carry the box." Representation of the initial state (a) for DoCoPro was also presented as follows.

(robot door low)
(box window low)
(banana center high)

Students were asked to describe if-then rules necessary to solve this problem with natural sentences. We refer to this task as a pretest. Students were instructed to design general rules adaptable to various initial states. They completed the remainder of the instructional texts before the second lecture, which was 2 weeks after the first lecture.

In the second lecture, each student engaged in constructing a model of the robot and banana problem with DoCoPro. Students were again instructed to design general rules, and compose a model that can appropriately function regardless of the order of rules. After model construction, students again described the rules of the problem by using sentences (a posttest).

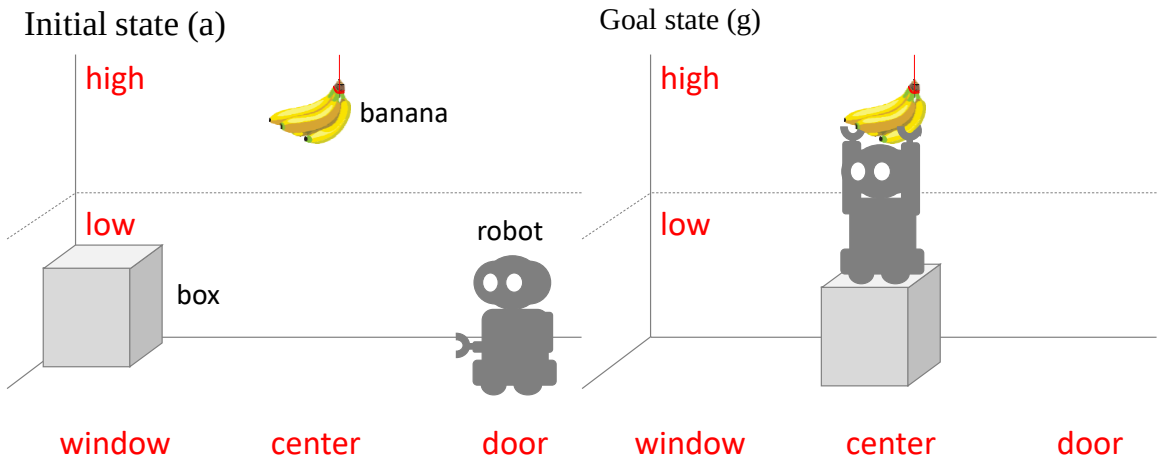

Figure 2. Initial and goal states in the robot and banana problem.

3.2 Data Analysis

We assessed operations in rules of models described by students in pre- and posttests. The best sequence of actions (rules) that can solve this problem includes the following four separate operations.

1. Robot moves to box (change the horizontal position of the robot to that of the box)
2. Robot carries box under banana (change the horizontal positions of the robot and box to that of the banana)
3. Robot stands on box (change the vertical position of the robot to high)
4. Robot gets banana (finish)

According to what operations were described and how they were incorporated in rules, we checked whether each student model had each of the following features.

- Normative Decomposition (ND) included four operations, and each operation was incorporated in different rules. A model with five rules comprising the four rules and one additional rule to finish the reasoning was also regarded as ND because it is natural to separate a functional rule only to halt problem-solving processes.
- Enhancing Operations (EO) included one or two rules incorporating operations that can successfully enhance a model. Actually, they were “robot moves to banana in a low position” and “robot gets down from box.”
- Surplus Decomposition (SD) included one or two operations that must not be separated from an operation in the best sequence. Actually, they were “robot lifts box” and “robot put box.” They are not independent because they are necessarily performed right before/after “robot carries box.”
- Lacking Operations (LO) means omitting one or more operations in the best sequence.
- Invalid Decomposition-Combining (ID-C) incorporated multiple separate operations into a single rule as an action in the best sequence. In a model with this feature, for example, one rule included combined operations such as “robot moves to box and carries it under banana.”
- Invalid Decomposition-Decombining (ID-D) broke an operation in the best sequence into two or more rules, except the case to break the rule to finish into “robot gets banana” and “halt the reasoning.” In a model with this feature, multiple rules included the same operation with different

parameters such as "robot moves to window (if box is at window side)" and "robot moves to center (if box is at center)."

- Invalid Operations (IO) included one or more operations that violated problem conditions or that could not be interpreted. In violated operations, for example, the box was supposed to move autonomously without the robot, and the robot can reach the banana without carrying the box itself. In another example, one rule directly describes the goal state (g) without the operations on the best sequence. Such a model cannot reproduce the solution process.

ND is regarded as a successful model, and SD, LO, ID-O, ID-D, and IO as failed because any of them can prevent a model from reaching the goal state with the best sequence through inappropriate decomposition or excessive operations. Although these seven features were independently assessed, ND is exclusive from the five failed features. EO can be held simultaneously with any others, and the five failed features can be held simultaneously with each other.

To examine RQ1 in terms of decomposition, we checked whether students described models of ND or those with features of the five failed models in the pretest. For RQ2, we compared pre- and posttests.

4. Results and Discussion

Figure 3 indicates proportions of students whose models had each of the seven features in the pre- and posttests. As the graph shows, NDs were few, and many student models had features of the five failed features in the pretest, whereas ND increased, and LO and IO decreased in the posttest. We compared numbers of each feature between the two tests by using the chi-square test; results indicated significant differences in ND ($\chi^2(1) = 26.01$, $p < .01$), EO ($\chi^2(1) = 3.94$, $p < .05$), LO ($\chi^2(1) = 46.10$, $p < .01$) and IO ($\chi^2(1) = 26.01$, $p < .01$). No significant differences were found in SD ($\chi^2(1) = 1.47$, *n.s.*), ID-C ($\chi^2(1) = 0.79$, *n.s.*) and ID-D ($\chi^2(1) = 2.66$, *n.s.*). These results confirmed that models with ND increased and those with lack of operations decreased after model construction with DoCoPro.

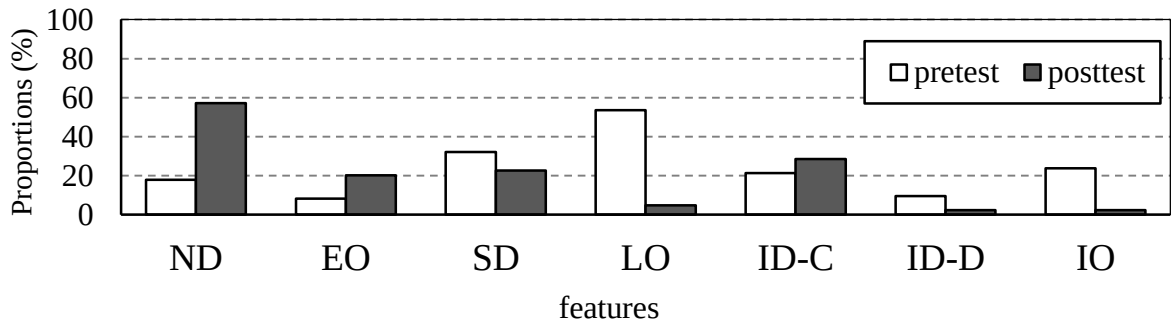

Figure 3. Proportions of students using each feature in pre- and posttests.

Those results indicated that in the pretest, most student models did not normatively decompose the solution of the robot and banana problem. In other words, undergraduate students could not necessarily externalize knowledge to solve the problem in terms of decomposition (RQ1). And ND increased in the posttest, indicating that construction of a model improved decomposition in knowledge externalization (RQ2). These facts reveal that construction of a rule-based model can foster one aspect of CT.

In the posttest, LO decreased. Students who omitted some operations of the best sequence in the pretest described them in the posttest. Most omitted operations were "finish," and update the position of the robot[1] in "robot moves to box." These operations may be implicit in human problem solving because people can solve this problem without awareness of them. In constructing a computational model, however, omitting such information causes error feedback. Such feedback must have improved

[1] In tests and model construction with DoCoPro, students wrote a name for each rule. Some student rules had names indicating "robot carry box," but had operations including only a description such as "add 'box is under banana.'"

student models in terms of lack of operations. This indicates that model construction can remove ambiguity in human representation of a problem's solution.

On the other hand, ID-C rather increased in the posttest although the difference was not significant. As explained above, models with this feature had duplicate operations with different parameters. Many students who described such models failed in model construction with DoCoPro in the second lecture. Their DoCoPro models fail to reach the goal state. Therefore, model construction by such students can be regarded as an inappropriate learning activity. Although we cannot precisely discuss this point due to page limitation, it alerted us to the necessity of learning support to guide appropriate model construction. Thus, one important our future work is investigating whether support for constructing an appropriate model enhances the effects and, if so, designing and implementing such support.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, 55(7), 832-835.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Inroads*, 2(1), 48-54.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). Developing computational thinking: approaches and orientations in K-12 wducation. In G. Veletsianos (Ed) *Proceedings of EdMedia 2016* (pp. 13-18). NC: AACE.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 AERA* (pp. 1-25).

Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Retrieved May 15, 2019, from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Grover, S., Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, 42(1), 38-43.

Kojima, K., Miwa, K. (2018). Preliminary study on fostering computational thinking by constructing a cognitive model. In Y.-T. Wu, N. Srisawasdi, M. Banawan, J. C. Yang, M. Chang, L.-H. Wong, & M. T. Rodrigo (Eds) *Workshop Proceedings of ICCE 2018* (pp. 265-270). Taoyuan, Taiwan: APSCE.

Krauss, J., Prottsman, K.: *Computational thinking and coding for every student: the teacher's getting-started guide*. CA: Corwin (2017)

Miwa, K. (2008). A cognitive simulator for learning the nature of human problem solving. *Journal of Japanese Society for Artificial Intelligence*, 23(6), 374-383.

Miwa, K., Nakaike, R., Morita, J., Terai, H. (2009). Development of production system for anywhere and class practice. In S. C. Kong, H. Ogara, H. C. Arnseth, C. K. K. Chan, T. Hirashima, F. Klett, J. Lee, C. C. Liu, C. K. Looi, M. Milrad, A. Mitrovic, K. Nakabayashi, S. L. Wong, & S. Yang (Eds) *Proceedings of ICCE 2009* (pp. 91-99). Jhongli, Taiwan: APSCE.

Mvalo, S., & Bates, C. (2018). Students' understanding of computational thinking with a focus on decomposition in building network simulations. In B. McLaren, R. Reilly, S. Zvacek, J. Uhomoibhi (Eds) *Proceedings of CSEDU 2018* (pp. 245-252), vol. 1. Setubal, Portugal: SciTePress.

Nakaike, R., Miwa, K., Morita, J., Terai, H. (2009). Development and evaluation of a web-based production system for learning anywhere. In S. C. Kong, H. Ogara, H. C. Arnseth, C. K. K. Chan, T. Hirashima, F. Klett, J. Lee, C. C. Liu, C. K. Looi, M. Milrad, A. Mitrovic, K. Nakabayashi, S. L. Wong, & S. Yang (Eds) *Proceedings of ICCE 2009* (pp. 127-131). Jhongli, Taiwan: APSCE.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education*, 14(1), 5.