Formal Concept Analysis of Programming Operation Using Tangible Tool

Tatsuo Motoyoshi *, Hiroyuki Masuta *, Ken’ichi Koyanagi *, Toru Oshima *, and Hiroshi Kawakami **

Abstract: This paper describes the structural analysis of the programming operation of the tangible programming tool P-CUBE, which is a tangible block-type programming tool that was developed for use by beginners, including those with visual impairments. An experiment was conducted to compare P-CUBE and conventional programming software platforms, and it was demonstrated that P-CUBE encourages the development of beginners’ understanding of the structure of a program. The differences between the programming operations of P-CUBE and the programming software are considered to be a factor causing differences in users’ understanding of programming structures. The structures of the programming operations were analyzed using formal concept analysis, and differences between the logical frameworks in the operation of P-CUBE and a conventional software platform were demonstrated.

Key Words: tangible, programming, structure of operations, formal concept analysis.

1. Introduction

The number of people starting to learn programming in elementary school has been increasing recently as society has become increasingly information-intensive. Because of the increase in programming education, some programming software packages that can be operated intuitively by children have been developed. Although such software allows users to intuitively create a program, these systems are often effective only for those who are accustomed to operating personal computers (PCs). Furthermore, because users depend on visual information, such as that displayed on a monitor, those suffering from visual impairments are unintentionally excluded as potential learners. To expand the accessibility of learning programming to beginners who are unaccustomed to operating PCs, including those with visual impairments, we have developed P-CUBE, a tangible [1] block-type programming education tool [2]. P-CUBE is intended to teach fundamental programming concepts [3],[4] at the beginner level through simple operations. In a previous study, we experimentally compared P-CUBE and conventional programming software and demonstrated that P-CUBE encourages beginners to understand the structure of a program [5]. The differences between the programming operations of P-CUBE and those of the conventional programming software are considered to be a factor causing differences in users’ understanding of programming structures. Therefore, the structures in the programming operation of both P-CUBE and a tile-type programming software were analyzed using formal concept analysis (FCA) [6],[7]. From the results of the analysis, the differences between the attribute implications [7] in the concepts of the programming operations were ascertained.

The framework of this paper is as follows. The system operation of P-CUBE and its effectiveness as a programming education tool for beginners are discussed in Sections 2 and 3, respectively. Section 4 describes the structural analysis of the programming operation of P-CUBE. Section 5 discusses the results of the study and draws conclusions.

2. Tangible Programming Tool P-CUBE

P-CUBE is a tangible programming tool for beginners, including those with visual impairments. This section introduces the design of P-CUBE and its programming operation.

2.1 Design

P-CUBE consists of a program mat and programming blocks with radio-frequency identification (RFID) system chips. The users can create programs for a mobile robot by placing programming blocks on the program mat based on the program structure [2]. Figure 1 shows the P-CUBE design. When using P-CUBE, the users can easily make a program without handling any mechanical equipment.

2.2 Programming Operation of P-CUBE

The programming procedure of P-CUBE is as follows.

1. A programming block is selected.
2. The programming block is placed on the program mat.
3. Steps 1 and 2 are repeated until the program is complete.
4. The position information of the programming blocks is read and transferred to a mobile robot via a PC.
5. The program is executed.

Figure 2 shows a flow chart of a line tracing program and the corresponding positions of the programming blocks on the pro-

Fig. 1 Design of P-CUBE.
Fig. 2 Example of program created with programming blocks and corresponding flowchart.

Fig. 3 Programming software.

Fig. 4 Experimental flow.

Fig. 5 Examples of programming tasks.

program mat. The positional relationships among the programming blocks of a program correspond to the structure of the program elements in the flowchart describing the program. These elements allow a user to learn the fundamental structures of a program, such as sequential processing, loops, and conditional branches.

3. Effectiveness of P-CUBE

Two evaluation experiments were conducted to investigate the effectiveness of P-CUBE. The first experiment evaluated how effectively programming beginners were able to learn using P-CUBE. The second experiment evaluated how P-CUBE influences the depth of a learner’s understanding of program structure in a collaborative learning environment.

3.1 Programming Training Experiment

A programming training experiment was conducted to verify the effectiveness of P-CUBE as a learning tool for programming beginners [5].

3.1.1 Overview of experimental procedure

In this experiment, P-CUBE was compared with the tile-type programming software BeautoBuilder2 and the code-type programming software Arduino Sketch. The subjects were 13 programming beginners (21–24 years old). The programming software used in the experiment is shown in Fig. 3.

Training and test phases were defined in the experiment. In the training phase, the subjects were instructed to complete two fundamental programming tasks by using one of the three programming tools (P-CUBE, BeautoBuilder2, or Arduino Sketch). In the test phase, the subjects were instructed to complete one task using each of the three programming tools. The programming tasks given in the experiment are as follows.

Task 1 Make a program in which the mobile robot moves forward for 4 s, turns right for 3 s, turns left for 3 s, and moves backward for 4 s.

Task 2 Make a program in which the mobile robot traces a black line using one infrared (IR) sensor.

Task 3 Make a program in which the mobile robot traces a black line using two IR sensors.

A flow diagram of the experiment is shown in Fig. 4. Tool 1 is the programming tool used in the training phase. Figure 5 shows two of the programming tasks given in the experiment.

Subjects were separated into three groups $P_{tr}$, $T_{tr}$, and $C_{tr}$ based on the programming tool used in the training phase. In the training phase, groups $P_{tr}$, $T_{tr}$, and $C_{tr}$ used P-CUBE, BeautoBuilder2, and Arduino Sketch, respectively.

3.1.2 Results

Table 1 shows the time required for tasks in the programming training experiment. The time required for the training phase of $P_{tr}$ was shorter than those of the other groups. When the code-type tool (Arduino Sketch) was used in the test phase, the completion time of $P_{tr}$ was shorter than those of the other groups, and there was significant differences between $P_{tr}$ and $C_{tr}$ [5]. $P_{tr}$ subjects are thus considered able to smoothly switch to using the code-type tool and understand the fundamental program structure better than the other groups’ subjects. Thus, it was demonstrated that P-CUBE encourages beginners to understand the structure of a program.

3.2 Collaborative Learning Experiment

An experiment was conducted to verify the effectiveness of P-CUBE in a collaborative learning environment [8].

3.2.1 Overview of experimental procedure

In this experiment, P-CUBE was compared with BeautoBuilder2 in terms of their effectiveness as training tools. In the
Table 1 Time required for tasks.

<table>
<thead>
<tr>
<th>Sub. No.</th>
<th>Group</th>
<th>Training term (Total)</th>
<th>Test term (Code type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P&lt;sub&gt;1&lt;/sub&gt;</td>
<td>P&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>5 min 41 s</td>
<td>8 min 31 s</td>
</tr>
<tr>
<td>P&lt;sub&gt;2&lt;/sub&gt;</td>
<td>P&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>4 min 50 s</td>
<td>11 min 35 s</td>
</tr>
<tr>
<td>P&lt;sub&gt;3&lt;/sub&gt;</td>
<td>P&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>13 min 30 s</td>
<td>8 min 36 s</td>
</tr>
<tr>
<td>P&lt;sub&gt;4&lt;/sub&gt;</td>
<td>P&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>11 min 9 s</td>
<td>7 min 20 s</td>
</tr>
<tr>
<td>P&lt;sub&gt;5&lt;/sub&gt;</td>
<td>P&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>11 min 16 s</td>
<td>11 min 23 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;1&lt;/sub&gt;</td>
<td>T&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>5 min 43 s</td>
<td>14 min 5 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;2&lt;/sub&gt;</td>
<td>T&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>14 min 29 s</td>
<td>15 min 35 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;3&lt;/sub&gt;</td>
<td>T&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>32 min 9 s</td>
<td>16 min 51 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;4&lt;/sub&gt;</td>
<td>T&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>5 min 42 s</td>
<td>7 min 56 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;5&lt;/sub&gt;</td>
<td>T&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>31 min 30 s</td>
<td>9 min 32 s</td>
</tr>
<tr>
<td>C&lt;sub&gt;1&lt;/sub&gt;</td>
<td>C&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>21 min 32 s</td>
<td>11 min 40 s</td>
</tr>
<tr>
<td>C&lt;sub&gt;2&lt;/sub&gt;</td>
<td>C&lt;sub&gt;Beg&lt;/sub&gt;</td>
<td>20 min 25 s</td>
<td>15 min 17 s</td>
</tr>
<tr>
<td>P&lt;sub&gt;6&lt;/sub&gt;</td>
<td>Ave.</td>
<td>9 min 17 s</td>
<td>9 min 29 s</td>
</tr>
<tr>
<td>T&lt;sub&gt;6&lt;/sub&gt;</td>
<td>Ave.</td>
<td>17 min 22 s</td>
<td>12 min 9 s</td>
</tr>
<tr>
<td>C&lt;sub&gt;6&lt;/sub&gt;</td>
<td>Ave.</td>
<td>20 min 59 s</td>
<td>13 min 26 s</td>
</tr>
</tbody>
</table>

Table 2 Operation durations and test scores.

<table>
<thead>
<tr>
<th>Sub. No.</th>
<th>P&lt;sub&gt;L&lt;/sub&gt;</th>
<th>P&lt;sub&gt;R&lt;/sub&gt;</th>
<th>T&lt;sub&gt;S&lt;/sub&gt;</th>
<th>T&lt;sub&gt;L&lt;/sub&gt;</th>
<th>P&lt;sub&gt;S&lt;/sub&gt;</th>
<th>P&lt;sub&gt;T&lt;/sub&gt;</th>
<th>P&lt;sub&gt;F&lt;/sub&gt;</th>
<th>Test score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
<td>119</td>
<td>100</td>
<td>689</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>187</td>
<td>288</td>
<td>5</td>
<td>641</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>119</td>
<td>287</td>
<td>187</td>
<td>134</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>232</td>
<td>265</td>
<td>0</td>
<td>577</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>127</td>
<td>329</td>
<td>81</td>
<td>1093</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>209</td>
<td>303</td>
<td>541</td>
<td>565</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>92</td>
<td>458</td>
<td>154</td>
<td>708</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>146</td>
<td>834</td>
<td>-</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Ave.</td>
<td>150.9</td>
<td>293.3</td>
<td>149.3</td>
<td>661.3</td>
<td>6.43</td>
<td>6.43</td>
<td>5.38</td>
<td>8.00</td>
</tr>
</tbody>
</table>

A mobile robot has two IR sensors and one touch sensor. The robot can trace the black line on the white mat and pass the door opening and closing at regular intervals. Choose the correct instruction to each blank box of the mobile robot’s program flowchart.

![Flow diagram of the experiment](Image)

Fig. 7 Experimental flow.

3.2.2 Results

We recorded the average durations of programming tool operation and the number of correct answers as the test score. Table 2 shows the operation durations and test scores in the collaborative learning experiment. We measured the time of touching the operation tool as the operation durations from the video data. The average durations of programming tool operation were 293.3s (SD=99.9), 150.9s (SD=57.8), 661.3s (SD=257.7), and 149.3s (SD=170.7) for P<sub>L</sub>, P<sub>R</sub>, T<sub>S</sub>, and T<sub>L</sub>, respectively. Operation durations of subjects who operated BeautoBuilder2 were longer than those of subjects who operated P-CUBE in the experiment. This indicates that the subjects’ understanding of the program structure did not depend on the programming tool used in the training phase. In the collaborative learning process, P-CUBE created fewer operation duration differences than BeautoBuilder2. There are no test score differences between the partners who operated P-CUBE in the experiment. This indicates that the subjects’ understanding of the program structure did not depend on the duration of their P-CUBE operation. Conversely, the level of understanding of the subjects who operated BeautoBuilder2 for less time than their partners was lower than that of the subjects who operated the software for a longer duration. It is assumed that the subjects who operated P-CUBE for less time were able to understand the fundamental program structure by watching their partners operate the tool.

4. Structural Analysis of Programming Operations

In the programming training experiment, P<sub>0</sub> subjects were considered to be able to more smoothly switch to code-type tool usage and understand the fundamental program structure better than the other groups’ subjects. In the collaborative learning experiment, there were no differences between the test scores of P<sub>S</sub> and those of P<sub>L</sub>. We assume that handling operation...
differences between P-CUBE and the programming software influence the level of understanding of the program structure. To visualize the structural difference of handling operations, the theoretical structure of the programming operation of P-CUBE was analyzed and compared with that of the tile-type software. The analysis focused on the logical relationship between the operating elements and motions in the programming operation and program structure, and the potential inclusion relations in the programming operations. Consequently, the two programming tools are defined as follows: The BeautoBuilder2 motion tile has a function which sets motion duration. The mobile robot executed the designated series of movements, the motions and durations of which were predetermined by the sequential and conditional branch programs. The mobile robot traced a black line using two equipped IR sensors.

4.3 Structural Analysis

4.3.1 Formal concept analysis

FCA is applied as a knowledge-discovery procedure in areas of software and educational engineering [10],[11]. In this paper, FCA was adopted to describe the theoretical structure of the programming operations.

FCA is a data analysis technique based on a triplet $K = (G, M, I)$ called a context. A context table with logical attributes can be represented by a triplet $K = (G, M, I)$, where $I$ is a binary relationship between $G$ and $M$. Elements of $G$ are called objects and correspond to table rows, and elements of $M$ are called attributes and correspond to table columns. For $g \in G$ and $m \in M$, $g \leq m$ indicates that the object $g$ has the attribute $m$ [12]. FCA produces a concept lattice and attribute implications. A concept lattice represents a hierarchical structure of human-like concepts. Attribute implications describe a particular dependency of attributes in the concept. We applied these outputs to the analysis of human operations and system awareness, and verified the efficiency of FCA as a method for visualizing implicit user rules [13]. More specifically, we utilize FCA to visualize implicit structures in the programming operations of each tool.

4.3.2 Formal context

The objects and attributes considered in this study are defined as follows.

**Objects**: Operating elements which were followed by the subject’s eyes.

**Attributes**: Structures of the program, Elemental motions

The operating elements are the items that are followed by the subject’s eyes while making a program, such as a programming block or a tile tool, as described in Section 4.1. The elements of the two programming tools are defined as follows: The BeautoBuilder2 motion tile has a function which sets motion duration.

$IF$: If block (P-CUBE)

$LP$: Loop block (P-CUBE)

$MO$: Motion block (P-CUBE)

$TM$: Timer block (P-CUBE)

$IF$: If tile (BeautoBuilder2)

$LP$: Loop tile (BeautoBuilder2)

$MO$: Motion tile (BeautoBuilder2)

The possible program structures are conditional branch ($CB$), loop ($Lo$), motion ($M$), and duration ($Du$). The elemental motions are the therbligs described in Section 4.2.

4.3.3 Context table

In this section, the context table of each programming task is defined. This paper focuses on the processes of making sequential and conditional branch programs. The mobile robot executed the designated series of movements, the motions and durations of which were predetermined by the sequential program. While executing the conditional branch program, the mobile robot traced a black line using two equipped IR sensors.

The symbol “$\times$” in the context table indicates that the operating element is used for constructing the structure of the program, or the operating element requires elemental motions.
Table 3 Context table for sequential program made using P-CUBE.

<table>
<thead>
<tr>
<th>CB</th>
<th>Lo</th>
<th>Mt</th>
<th>Du</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Context table for sequential program made using BeautoBuilder2.

<table>
<thead>
<tr>
<th>CB</th>
<th>Lo</th>
<th>Mt</th>
<th>Du</th>
<th>M1′</th>
<th>M2′</th>
<th>M3′</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Context table for conditional branch program made using P-CUBE.

<table>
<thead>
<tr>
<th>CB</th>
<th>Lo</th>
<th>Mt</th>
<th>Du</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>LP</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>MO</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>TM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6 Context table for conditional branch program made using BeautoBuilder2.

<table>
<thead>
<tr>
<th>CB</th>
<th>Lo</th>
<th>Mt</th>
<th>Du</th>
<th>M1′</th>
<th>M2′</th>
<th>M3′</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFt</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPt</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOt</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sequential program**  When making a sequential program using P-CUBE, the user positions the Motion and Timer blocks on the program mat, as shown in Task 1 in Fig. 5. Conversely, the user can make a sequential program with BeautoBuilder2 simply by handling a mouse. Tables 3 and 4 are the context tables for P-CUBE and BeautoBuilder2, respectively.

**Conditional branch program**  When making a conditional branch program using P-CUBE, the user positions If, Loop, and Motion blocks on the program mat, as shown in Fig. 5. Timer blocks are not used, because the instructions do not require the motion to be sustained for any duration in this program. Tables 5 and 6 are the context tables for P-CUBE and BeautoBuilder2, respectively. The only object in the context table for BeautoBuilder2 is a mouse because it is the only tool a user handles when operating that program.

4.3.4 Concept lattice  This section discusses the concept lattice of each concept. Figure 8 shows the concept lattice of a user who made a sequential program using BeautoBuilder2; Fig. 9 shows the concept lattice of a user who made a conditional branch program using the same application. Figure 10 shows the concept lattice of a user who made a sequential program using P-CUBE; Fig. 11 shows the concept lattice of a user who made a conditional branch program using the same application. We used the Concept Explorer [14],[15] program to build the concept lattices. Each node in the concept lattice represents a concept which indicates the correspondence between objects and attributes. Attributes are added from the top to the bottom of the concept lattice, and objects are added from the bottom to the top of the concept lattice. When increasing attributes from upper layers, upper half of the node indicated in gray. When increasing objects from lower layers, lower half of the node indicated in black. For example, the second top node of the concept lattice in Fig. 10, the Motion block MO and the Timer block TM corresponds elemental motions M1, M2, M3, M4.

Elemental motions of P-CUBE are arranged in a hierarchy. Motions M1–M4 are placed in upper layer; i.e., all operations corresponds to program structures which are designated in the program task contain M1–M4. Motion M5 is contained in the operation which corresponds to some program structures. In contrast, the elemental motions of BeautoBuilder2 are not arranged in a hierarchy, and motions M′1–M′3 are placed in the
same layer.

4.3.5 Attribute implications

This section discusses the attribute implications of each concept. We used the Concept Explorer [14],[15] program to perform the analysis. First, for the sequential program made using P-CUBE, the following seven attribute implications ($I_{p1} – I_{p7}$) were output by FCA.

$I_{p1}$: $M_t \rightarrow M_1, M_2, M_3, M_4$

$I_{p2}$: $M_1 \rightarrow M_2, M_3, M_4$

$I_{p3}$: $M_2 \rightarrow M_1, M_3, M_4$

$I_{p4}$: $M_3 \rightarrow M_1, M_2, M_4$

$I_{p5}$: $M_4 \rightarrow M_1, M_2, M_3$

$I_{p6}$: $M_5 \rightarrow D_u, M_1, M_2, M_3, M_4$

$I_{p7}$: $D_u \rightarrow M_1, M_2, M_3, M_4, M_5$

The antecedents of $I_{p1}$ and $I_{p7}$ respectively contain $M_t$ and $D_u$, each of which refers to one of the program structures. These attribute implications are consequence relations between the program structure and motions. $I_{p7}$ contains the attribute $M_5$ in addition to $M_1$–$M_4$. FCA also yielded the following eight attribute implications ($I_{p8} – I_{p16}$) for the conditional branch program made using P-CUBE.

$I_{p8}$: $M_t \rightarrow M_1, M_2, M_3, M_4$

$I_{p9}$: $M_1 \rightarrow M_2, M_3, M_4$

$I_{p10}$: $M_2 \rightarrow M_1, M_3, M_4$

$I_{p11}$: $M_3 \rightarrow M_1, M_2, M_4$

$I_{p12}$: $M_4 \rightarrow M_1, M_2, M_3$

$I_{p13}$: $M_5 \rightarrow M_1, M_2, M_3, M_4$

$I_{p14}$: $C_B \rightarrow M_1, M_2, M_3, M_4, M_5$

$I_{p15}$: $L_o \rightarrow M_1, M_2, M_3, M_4, M_5$

The antecedent of $I_{p1}$ contains $M_t$. The antecedent of $I_{p7}$ contains $C_B$. The antecedent of $I_{p15}$ contains $L_o$. These attribute implications contain attributes which correspond to the program structure. $I_{p8}$ and $I_{p15}$ contain the attribute $M_5$ in addition to $M_1$–$M_4$, which constitute the consequent of $I_{p1}$. When using P-CUBE, consequence relations between a program structure and motions will vary according to the program structure. Furthermore, $M_5$ is contained solely by the antecedent of the attribute implication $I_{p15}$ which indicates the consequence relation among motions alone.

On the other hand, $I_{s1} – I_{s5}$ are the attribute implications for the sequential program made using P-CUBE.

$I_{s1}$: $M_5' \rightarrow M_t, D_u, M_1', M_2'$

$I_{s2}$: $M_t \rightarrow D_u, M_1', M_2', M_3'$

$I_{s3}$: $D_u \rightarrow M_t, M_1', M_2', M_3'$

$I_{s4}$: $M_1' \rightarrow M_t, D_u, M_2', M_3'$

$I_{s5}$: $M_2' \rightarrow M_t, D_u, M_1', M_3'$

FCA also yielded the following attribute implication ($I_{p8}$) for the conditional branch program made using BeautoBuilder2.

$I_{p8}$: $1 \rightarrow M_1', M_2', M_3'$

Although $I_{s2}$ and $I_{s3}$ are consequence relations between a program structure and motions, the same motions are present in the consequent. Furthermore, there are no particular attribute implications which indicate the consequence relation solely among motions like $I_{p8}$.

4.4 Discussion

Concept lattice The concept lattice shows the hierarchy structure of elemental motions. In P-CUBE operations, $M_5$ is grouped into a different layer than $M_1$–$M_4$. This means that motion $M_5$ is the operation which corresponds to specific program structures such as conditional branch, loop, or duration. In contrast, $M_1$–$M_4$ are regular motions which correspond to all program structures. All $M_1'$–$M_3'$ motions which are operation elements of BeautoBuilder2 are grouped into the same layer. $M_1'$–$M_3'$ are regular programming operation motions regardless of the program structure.

The result shows that the concept lattice has the potential to visualize the realization of programming operations. We hypothesize that the difference in layer placement of the motion has a potential influence on the depth of a learner’s understanding of the program structure.

Attribute implications When using BeautoBuilder2, attribute implications which indicate consequence relations between a program structure and motion elements have the same motion elements in the consequent. There are no consequence relations solely among motion elements. Thus, it is assumed that users cannot realize a structure difference in the handling operation produced by the program structure. Furthermore, since the antecedents of $I_{p1}$ is the empty set, we concluded that users cannot classify the program structure through the programming operation.

When using P-CUBE, there are attribute implications $I_{p8}$, $I_{p7}$, $I_{p1}$, $I_{p15}$, and $I_{p16}$ which indicate consequence relations between the program structure and motions. Elemental motion Assembly ($M_5$) is contained solely in the consequent of $I_{p1}$, $I_{p15}$, and $I_{p8}$. $I_{p7}$ means that the elemental motion Assembly ($M_5$) is needed in addition to motions $M_1$–$M_4$ when making a sequential program that requires a motion duration ($D_u$). $I_{p7}$ and $I_{p8}$ have similar meanings to that of $I_{p1}$. $I_{p7}$ means that the elemental motion Assembly ($M_5$) is needed in addition to motions $M_1$–$M_4$ when making a conditional branch program ($C_B$). $I_{p8}$ means that the elemental motion Assembly ($M_5$) is needed in addition to motions $M_1$–$M_4$ when making a conditional branch program that requires a loop structure ($L_o$). If a user must instruct the robot to perform repetitive processing ($L_o$) or the program has a conditional branch structure ($C_B$), then the user must assemble multiple blocks, which is not the case when making other program structures. In addition, $I_{p15}$, which indicates consequence relations solely between elemental motions, means that an operating element that requires the motion Assembly ($M_5$) must also involve motions $M_1$–$M_4$ to perform the operation. It is assumed that a user realizes that Assembly ($M_5$) indicates that a higher motion is needed when making a program with a specific structure. We consider it likely that this recognition encourages the development of the user’s understanding of the program structure. These attribute implications are considered as a consequence relation between motions.

The attribute implications in the concepts of the programming operations differ among the programming tools. The operation of P-CUBE as a tangible tool was shown to have a dependency relationship between the program structure and the elemental motions. The presence of dependency relationships are considered to have a potential influence on the depth of a learner’s understanding of the program structure.

5. Discussion and Conclusion

In this study, P-CUBE, a tangible block-type programming education tool, was developed as an easily operable fundamen-
tual programming teaching aid. Experiments were conducted to demonstrate the effectiveness of this tool in improving users’ understanding of the structure of a program in comparison with other programming software platforms. These experiments demonstrated that P-CUBE encourages the development of beginners’ understanding of the structure of a program. Furthermore, when subjects were instructed to work in pairs, it was demonstrated that their understanding of the program structure did not depend on the operation duration of P-CUBE.

Conversely, the level of understanding of subjects who operated the programming software for less time than their partners was lower than that of the subjects who had the longer operation duration. This indicates that subjects who operated P-CUBE for less time were still able to understand the fundamental structure of the programs by watching their partners. We conclude that there are logical structure differences between P-CUBE operations and the programming software which influence user understanding.

To determine the factors causing the differences in understanding, the theoretical structures of the programming operations of P-CUBE and the programming software were analyzed using FCA. The analysis results indicate that the concept lattice has the potential to aid visualization of the realization of programming operations. In addition, the potential inclusion relations in the programming operations were demonstrated by the attribute implications. There are differences between the attribute implications of the different programming tools. The tangible tool P-CUBE shows dependency between the program structure and elemental motions. Conversely, BeautoBuilder2 shows no attribute implications that correspond to the program structure. The presence of dependency relationships is considered to have a potential influence on the level of a user’s understanding of fundamental program structures. FCA has the potential to aid visualization of the implicit inclusion relationship in manual-handling tasks. The operation difference has an effect on the depth of users’ system awareness [16]. Accordingly, this analytical method can considered useful for verifying the effectiveness of time-consuming operations.

In the future, we intend to further evaluate the learning effects of P-CUBE and compare them with conventional PC software platforms. In addition, we plan to evaluate the effectiveness of P-CUBE in programming instruction for users with visual impairments.

Acknowledgments

This work was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 26870483. The authors would like to thank Pastel Studio staff at Toyama Prefectural University.

References


Ken’ichi KOYANAGI (Member)

He received his Ph.D. degree in engineering from Osaka University, Japan in 2005. From 2001 to 2006, he was a Research Associate at Osaka University. Since 2006, he has been with the Toyama Prefectural University where he is currently an Associate Professor. His research interests include development of mechatronics devices and force display systems using functional materials, passive type force display, rehabilitation application of force display systems, and precision control of direct-drive motors using functional dampers.

Toru OSHIMA

He received his B.S., M.S., and Ph.D. degrees from Tokyo Denki University, Japan, in 1981, 1983, and 1987, respectively. He worked as researcher of the Tokyo Metropolitan Prosthetic and Orthotic Research Institute from 1986 to 1990. In 1990, he joined Toyama Prefectural University, Japan, where he is currently a Professor of the Department of Intelligent Systems Design Engineering. His main fields of research are Robotics and Biomechanics.

Hiroshi KAWAKAMI (Member)

He received his Bachelor, Master, and Dr. Eng. degrees from Kyoto University, Japan, in 1987, 1989, and 1993 respectively. He started his career at Okayama University as an assistant professor in 1989. In 1998, he joined Kyoto University, where he was an Associate Professor of Graduate School of Informatics, and is currently a Professor of Unit of Design. His research interests include systems design, where he has proposed FUBEN-EKI that stands for designing systems based on appreciating "benefits of inconvenience."