“So my program doesn’t run!” Definition, origins, and practical expressions of students’ (mis)conceptions of correctness

Y. Ben-David Kolikant* and M. Mussai

We studied students’ conceptions of correctness and their influence on students’ correctness-related practices by examining how 159 students had analyzed the correctness of error-free and erroneous algorithms and by interviewing seven students regarding their work. We found that students conceptualized program correctness as the sum of the correctness of its constituent operations and, therefore, they rarely considered programs as incorrect. Instead, as long as they had any operations written correctly students considered the program ‘partially correct’. We suggest that this conception is a faulty extension of the concept of a program’s grade, which is usually calculated as the sum of points awarded for separate aspects of a program. Thus school (unintentionally) nurtures students’ misconception of correctness. This misconception is aligned with students’ tendency to employ a line by line verification method – examining whether each operation is translated as a sub-requirement of the algorithm – which is inconsistent with the method of testing that they formally studied.

Keywords: correctness; conceptions; verification practices; local perspective; novice programmer

1. Introduction

Knowing the craft of correctness verification is essential to being a successful programmer. Programmers regularly examine and critique programs of their own and those of their peers. For a computer scientist the importance of correctness and correctness-related practices is expressed in the description of computer scientists in the report of the ACM K–12 curriculum task force: “A computer scientist is concerned with the robustness, the user-friendliness, the maintainability, and above all the correctness of computer solutions to business, scientific, and engineering problems” (Tucker, 2003, p. 12). Accordingly, the task force recommended that K–12 students be introduced to correctness practices of testing, debugging, and forming loop invariants.

Similarly, in Israel, where this study was conducted, all computer science (CS) high-school students study the same curriculum (Gal-Ezer & Harel, 1999), which as early as the first introductory course devotes an entire chapter to explaining the meaning of correctness. According to the CS textbook used (Ginat, 1998), which is compatible with the CS curriculum, a correct program (or an algorithm) is defined as a program that produces the expected output – meaning all the expected output and nothing but the
expected output – for every valid input (Ginat, 1998). The same definition was given by Joni and Soloway (1986) to what they referred to as “Working programs.”

Thus, the decision as to whether a given program is correct or not is absolute: it is either “yes” or, if an error exists in the program, “no.” Students are also introduced to other measures of program quality, such as efficiency, modularity, and readability. Additionally, with respect to correctness-related practices, they are taught to verify correctness by testing programs, a procedure consisting of identifying input examples that span all the valid input space and then testing these examples, namely verifying that the program’s execution produces the expected output for these input examples. Testing takes place throughout the entire CS program.

However, as expressed by Edwards (2004, p. 26), “Despite our best efforts as educators, student programmers continue to develop misguided views about their programming activities.” Students do not easily adopt the thought habits (Gries, 2002) or work habits of programmers and instead choose to employ inadequate local or line by line practices (for a review on students’ practices see Robins, Rountree, & Rountree, 2003).

In a previous work (Ben-David Kolikant, 2005) it was suggested that students have misconceptions of correctness that underlie their decision to employ inadequate correctness-related practices. These conceptions were explained as faulty extensions of students’ experience as computer users. In this paper, based on an extended experiment, we suggest an additional source of students’ misconceptions of correctness, namely school. Specifically, we suggest that the traditional practices of grading programs in school, where programs are viewed in a line by line perspective, led students to the conception that correctness is a relative feature. Furthermore, we found that students conceptualized program correctness as the sum of the correctness of each part in the program. This faulty extension is in agreement with students’ viewpoints regarding programming as a local activity, as described in the literature (see, for example, Fluery, 1993; Winslow, 1996), which is what reasonably explains its longevity. Thus, it is educators’ “best efforts” that (unintentionally) nurture students’ misconceptions that underlie their inadequate work habits.

2. Background

2.1. Misconceptions

The theoretical framework underlying this study is constructivism, according to which (a) learning is a process of knowledge construction, not of knowledge recording or absorption, (b) learning is knowledge dependent, i.e. people use current knowledge to construct new knowledge, and (c) learning is highly tuned to the situation in which it takes place (Resnick, 1989). Smith, diSessa, and Roschelle (1993/1994) asserted that knowledge is a huge system comprising many pieces that are diversely interconnected. Learning is a continuous process of refinement and is an extension of our inherent knowledge. Students cope with new situations using their existing knowledge, and naturally they may misuse pieces of knowledge that were productive in past similar situations, expecting them to be productive in the current situation as well. In other words, misconceptions are generated. Understanding why students learn to produce alternative conceptions is beneficial for designing an effective means of instruction (see, for example, Robins et al., 2003).

2.2. Students’ inadequate correctness-related practices

Much work in the literature of CS education is devoted to understanding students’ work habits regarding correctness. Edwards (2003, 2004) found that students perceived the
activity of testing as boring, not creative, and only ancillary in comparison with other programming activities. In addition, students did not have clear techniques for testing. Edwards suggested that students be taught the craft of testing throughout the course of the CS program. That way the students would internalize the idea of taking responsibility for the quality of their program and learn to design test cases for programs that they had developed.

Fluery (1993) compared the beliefs of novices and expert programmers regarding various aspects of software development and found that experts were driven from a holistic viewpoint regarding programming, whereas novices had a local viewpoint. Specifically regarding debugging, it was found that experts felt that programmers should strive to obtain a holistic understanding of the program they are to debug. Correspondingly, they appreciated programs that were easy to comprehend and thus appreciated modularity and readability.

In contrast, novices perceived debugging in a local manner, i.e. as a process composed of independent steps of locating mistakes and fixing them. Thus, they over-emphasized the value of locating mistakes. They preferred programs that enabled them to easily recognize the bugs and thus perceived modular programs as not easily traced and debugged. Some novices assumed that there was only one bug and after it was located and fixed the debugging process stopped. In fact, some novices had the propensity to check a program with only one input example.

2.3. Previous work on students’ misconceptions of correctness

Only a small amount of work has been done on students’ perceptions of correctness, perhaps because of the belief expressed by Joni and Soloway (1986, p. 96) that “everyone will agree that a non-working program is incorrect” is common in the field. Nonetheless, Ben-David Kolikant (2005) suggested that students perceived correctness as relative. This claim was based on an experiment where students were given three programs that “worked,” in the sense that they produced the expected output for any given input, yet they were incorrect because for parts of the input space there was additional output, which was not expected. Many students decided that the programs were relatively correct and a noteworthy number of them were indecisive regarding whether the programs were correct or not.

Moreover, Ben-David Kolikant and Pollack (2004) found that the students’ norm was to tolerate the existence of errors in their programs. These students were satisfied with programs that “worked in general” or worked for “many input examples” but were not necessarily error free for the entire input space.

Students’ perceptions were explained as a ramification of their informal experience as users and end-use programmers. Students habitually worked with user-friendly software artifacts. These artifacts encouraged a work style termed “bricolage” or “tinkering,” meaning the building of new software artifacts by hands-on exploration and by trying software products at the interface level. This habit influenced their tendency to overly rely upon the computer output and also to be satisfied when they had made the computer work for a specific local need, rather than the entire input space (Ben-David Kolikant & Ben-Ari, 2008; see also Ben-Ari, 2001).

2.4. Research goals and questions

Nonetheless, the above explanation was based on programs that students claimed (mostly) work – and thus are (mostly) correct – because they produced the expected output. In this
paper we have sought to expand our understanding of students’ conceptions of correctness by examining students’ “red lines,” namely when students decide that a program is incorrect. We were specifically interested in exploring students’ decisions regarding non-working programs, i.e. programs that provide no output or partial output to all (or to a noticeable part of) the input space. If students’ decisions are solely governed by users’ perspectives, then we would expect that they would decide that these programs are incorrect because they do not “work for many input examples.” Any other decision by the students would indicate new faulty connections to students’ existing knowledge, namely new misconceptions; therefore, it was important to trace the process by which students made their decision.

Our research questions were therefore formulated as follows.

• How do students conceptualize (in)correctness?
• How did students’ misconceptions originate, i.e. what situations involving students’ misconceptions were found to be similar and what existing knowledge and associated practices were thus extended into the correctness-related situations?
• What factors nurture the longevity of students’ misconceptions, i.e. by what connections are these misconceptions linked to the students’ body of knowledge?

3. Methods

3.1. Questionnaire

One hundred and fifty-nine high-school students from six schools (eight classes) in Jerusalem were given a questionnaire. This population consisted of 74 Grade 10 students, 40 Grade 11 students, and 45 Grade 12 students. The questionnaire included six algorithms and the goals that were to be achieved. Four algorithms included both a logic error and a secondary error. By logic error we mean an error that concerns the “heart” of the algorithmic solution. By secondary error we mean errors that are ancillary to the design of the solution, such as assigning an initial value of zero to a counter. The other two algorithms were error-free, which enabled us to inspect students’ decisions regarding both cases.

For each algorithm the students were asked to perform the following tasks.

Task A. Describe the error, if any.
Task B. Give the algorithm a grade ranging from 0 to 10 (where 10 is the highest grade).
Task C. Decide whether the algorithm is correct on a yes/no scale.
Task D. Describe in detail how you checked the algorithm.

The students responded anonymously and voluntarily and did not receive any incentives. The questionnaires were completed and submitted by the students during one of their CS lessons. There were no time limits. The teachers did not see the students’ responses.

Task A enabled us to monitor whether students were aware of the errors in the algorithm. Task B enabled us to explore students’ grading scales, in particular what errors they considered as crucial. In Task C students were forced to think in a dichotomous manner regarding correctness. The relations among the tasks were of importance too. In particular, we were interested in exploring how, if at all, the detection of a logic error is expressed in the score given and in the decision regarding the correctness of the algorithm. We were also interested in exploring the relationship between Task A and Task D in order
to determine whether a certain verification method is more effective in recognizing the errors than other methods.

Table 1 describes the given algorithms. The third and the sixth algorithms are correct. The other algorithms do not work, in the sense that they produce no output or produce incorrect output for all the input space of the problem. We deliberately chose non-working algorithms in order to determine whether the concept of relative correctness, found by Ben-David Kolikant (2005), also applied in these cases. The complete algorithms are provided in Appendix 1.

3.2. Interviews

The interviews were conducted after we had received the results of the questionnaires in order to clarify the phenomena discovered in the responses to the questionnaires, such as (a) many students did not answer Task C in the questionnaire, in which they were asked to decide on the algorithms’ correctness on a yes/no scale and (b) many students’ responses to Task D – about how they checked their program – were in the form of “I did it in my head” or “I read it.” Therefore, we gave the questionnaire to a ninth class, a Grade 12 class in an elite high-school, which was not part of the original questionnaire sample. The students’ responses to these tasks were examined on the spot and thereafter we chose our seven interviewees.

In the questionnaire one of the seven interviewees had responded that two of the four incorrect programs were correct. Another interviewee had identified all four incorrect programs as being incorrect. The remaining five students interviewed did not respond to Task C. Of these, two who believed they had found a logic error were very lenient in their grades, one student penalized the programs very firmly for their logic errors and the other

<table>
<thead>
<tr>
<th>No.</th>
<th>Goal</th>
<th>Logic error and matching output</th>
<th>Secondary errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exchanging the values of two variables using a third variable</td>
<td>Overriding a value of one variable</td>
<td>An initial value was not assigned to one of the variables</td>
</tr>
<tr>
<td>2</td>
<td>Finding the location of a given value in an array</td>
<td>Overriding a value in a loop</td>
<td>The printing instructions were omitted</td>
</tr>
<tr>
<td>3</td>
<td>Examining whether the sum of two given numbers is equal to a third given number.</td>
<td>No error</td>
<td>No error</td>
</tr>
<tr>
<td>4</td>
<td>Examining whether the sum of three given numbers is equal to a fourth given number, using a loop</td>
<td>Printing the wrong output in addition to the correct output</td>
<td>An initial value was not assigned to the accumulating variable</td>
</tr>
<tr>
<td>5</td>
<td>Counting the number of even numbers between 0 and a given number</td>
<td>Wrong output (summing instead of counting)</td>
<td>The initial value of the counter is 1 instead of 0</td>
</tr>
<tr>
<td>6</td>
<td>Finding the maximum value of three given numbers</td>
<td>No error</td>
<td>No error</td>
</tr>
</tbody>
</table>
two did not find a logic error. In addition, only one of the seven students used input samples to test the programs.

At the beginning of each interview, the students were requested to analyze an additional algorithm (see Appendix 2). The algorithm was supposed to count the number of students who failed an examination and to calculate the average grade of a given 10 student class. The algorithm performed both tasks correctly, however, we placed the variable in which the average was calculated in the loop and therefore it overrode each iteration, but it eventually ended up with the correct value. We decided to give a correct program because we were interested in exploring students’ correctness-related practices and we assumed that deciding that a program is correct requires more work than deciding that it is incorrect, which can be done by merely indicating one input example for which the program does not produce the expected output. However, we anticipated that some students would mistakenly decide that the algorithm was incorrect because they would notice the overridden variable but would not notice the final value of the variable. Their responses would then be treated as if they had found a logic error.

The students were encouraged to think out loud as they worked on this algorithm. Then we asked them to explain what they did and to justify their method. Finally, we asked them about the correctness of the six algorithms, how they graded the algorithms, and their standards for deciding whether the algorithms were correct or incorrect. Each interview lasted about an hour.

3.3. Methods of analysis: Questionnaire

Students’ responses to Task A, where they were asked to describe the errors in the algorithms, were ranked from 0 to 3. A score of 3 was given when the student recognized the logic error in the program; a score of 2 was given when the student did not recognize the logic problem but found (if there was any) a secondary error; a score of 1 was given when the student did not find an error (or did not write anything); a score of 0 was given when the student mistakenly identified a non-existing error.

Students’ responses to Task B ranged from 0 to 10. We chose to omit cases of no response because there were quite a few. The responses to Task C were coded into a variable with four possible values, “no response,” correct (“yes”), incorrect (“no”), and “partially correct.” The value “no response” was added because many students did not respond to this task. We also added the value “partially correct” because some of the students added this option in handwriting.

For Task D we classified students’ responses into the following categories: (1) reports stating that the work was done in “their head” with no further explanation or reports stating that they read the program without making any calculations; (2) testing with at least one input example; (3) other ways that were reported; (4) no response.

Finally, correlations were calculated for students’ responses to the four tasks, according to the types of variables. Spearman’s correlations were calculated to examine the relationship between the grades and the error found (Task A). Lambda (λ) correlations (Liebetrau, 1983) were calculated in order to explore the relationship among the rest of the tasks, where nominal variables were used. We also compared the performances of Grades 10, 11, and 12 in order to examine whether students improved as they proceeded with their CS studies. One-way ANOVA tests and post hoc Scheffe tests were conducted for the different grade levels in order to examine how CS experience gained in school actually influenced students’ responses. The same tests were conducted when students were
grouped according to their classes, to determine whether a certain teacher had influenced their responses.

3.4. Methods of analysis: Interviews

We sought to reveal the following.

- Students’ procedures for checking the algorithm as well as their justification.
- Students’ definition of correct, incorrect, and relatively correct programs, as well as the sources of students’ misconceptions of correctness, expressed in students’ explanations of Tasks A, B, and C.
- The interrelationship between students’ conceptions and their verification method.

All the interviews were tape-recorded and transcribed. The two authors first analyzed each transcription verbatim, individually, utilizing the constant comparative method (Glaser & Strauss, 1967). Then, at several meetings, we jointly identified the common patterns and themes that emerged from our individual analyses. A discusstonal theory building strategy was part of the analysis process (Glaser & Strauss, 1967), i.e. the conceptual categories and their properties were discussed and the theory was represented in a continuous theoretical discussion.

4. Findings

4.1. Questionnaires

4.1.1. Task A

Table 2 presents the distribution of students’ responses to Task A, in which they were asked to indicate the errors in the algorithms. The number of students who found the logic errors ranged from 30% (program 4) to 60% (program 5). Furthermore, a significant number of students mistakenly thought that they had found errors that actually did not exist and, in addition, a quarter to one-third of the students did not find errors in the erroneous programs. The students’ performance was quite poor, given the simplicity of the programs and the obvious errors. The students’ low performance is consistent with the results regarding the low performance of CS students reported in the literature (see, for example, Lister et al., 2004; McCracken et al., 2001). There was no statistically significant difference among the three grades.

<table>
<thead>
<tr>
<th>Programs</th>
<th>No</th>
<th>Rank 3 Logic error</th>
<th>Rank 2 Secondary error</th>
<th>Rank 1 No error</th>
<th>Rank 0 Non-existing error</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erroneous</td>
<td>1</td>
<td>57%</td>
<td>11%</td>
<td>24%</td>
<td>8%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49%</td>
<td>6%</td>
<td>36%</td>
<td>9%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>30%</td>
<td>26%</td>
<td>38%</td>
<td>6%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>60%</td>
<td>4%</td>
<td>31%</td>
<td>4%</td>
<td>100%</td>
</tr>
<tr>
<td>Error-free</td>
<td>3</td>
<td>0%</td>
<td>0%</td>
<td>96%</td>
<td>5%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0%</td>
<td>0%</td>
<td>83%</td>
<td>17%</td>
<td>100%</td>
</tr>
</tbody>
</table>
4.1.2. Task B

Table 3 presents the means and standard deviations (SD) of students’ responses to Task B, in which they were asked to grade the algorithms according to their correctness. The standard deviation is greater when students graded an erroneous program than when they graded error-free programs. This greater deviation can be explained by the lack of a consensus among students regarding what constitutes fatal and secondary errors. However, their other explanations were checked too, such as the influence of students’ formal experience, as well as the influence of their teachers’ grading criteria. A one-way ANOVA test of the responses from the eight classes’ revealed no significant difference; yet significant differences were found for three out of the four incorrect algorithms when we grouped students according to grade level and compared them [algorithm 2: $F(2,137) = 4.294$, $P = 0.016$; algorithm 4: $F(2,138) = 7.656$, $P = 0.001$; algorithm 5: $F(2,140) = 6.540$, $P = 0.02$]. A post hoc Scheffe test indicated a significant difference between the Grade 12 students and the Grade 11 students. For example, the twelfth graders were harsher in assigning grades than the other students, possibly because their experience with large-scale programs had changed their perspective regarding small-scale programs. The responses of the Grade 10 students were not significantly different from the other two grades.

4.1.3. Task C

Table 4 presents students’ responses to Task C, in which they were asked to decide whether or not the algorithms were correct. Interestingly, most of the students chose not to respond to this question. Furthermore, the number of students who did not perform this task was significantly higher than the number of students who did not respond to Task B, in which they were asked to grade the algorithms (Table 3). Moreover, more students chose to

<table>
<thead>
<tr>
<th>Program</th>
<th>No.</th>
<th>Mean ± SD</th>
<th>n</th>
<th>Response rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erroneous</td>
<td>1</td>
<td>3.9 ± 2.7</td>
<td>146</td>
<td>92%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.1 ± 3.3</td>
<td>139</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6.7 ± 3.1</td>
<td>140</td>
<td>88%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.9 ± 3.4</td>
<td>142</td>
<td>89%</td>
</tr>
<tr>
<td>Error-free</td>
<td>3</td>
<td>9.7 ± 0.9</td>
<td>148</td>
<td>93%</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>9.2 ± 1.6</td>
<td>143</td>
<td>90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>No.</th>
<th>Correct</th>
<th>Incorrect</th>
<th>Relatively correct</th>
<th>No response</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erroneous</td>
<td>1</td>
<td>7%</td>
<td>36%</td>
<td>1%</td>
<td>56%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21%</td>
<td>15%</td>
<td>1%</td>
<td>62%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>19%</td>
<td>11%</td>
<td>3%</td>
<td>67%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4%</td>
<td>21%</td>
<td>2%</td>
<td>73%</td>
<td>100%</td>
</tr>
<tr>
<td>Error-free</td>
<td>3</td>
<td>55%</td>
<td>0%</td>
<td>0%</td>
<td>45%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>47%</td>
<td>1%</td>
<td>0%</td>
<td>52%</td>
<td>100%</td>
</tr>
</tbody>
</table>
perform this task for programs 3 and 6, the error-free programs, and not for the erroneous programs. This suggests that students felt comfortable in deciding that a program is correct when no error was found; however, once they recognized an error they were not sure whether that meant that the program was incorrect. The data obtained from the interviews (described below) strengthens this hypothesis.

4.1.4. Task D

Table 5 presents students’ responses regarding Task D, in which they were asked to describe how they checked the programs. Interestingly, many students did not respond to this task at all. The most common answer among those who did respond was of the type “I read it” and “I checked it in my head,” and there was no written evidence indicating that input examples were tested. The highest proportion of students who reported testing or who wrote down their test results was only 26.4% for program 1, even though this method was studied at school. No statistical difference among the classes or grades was detected.

4.2. Correlations

The relations between students’ responses to Task A and Task B are presented in Table 6, where we grouped students according to their responses to Task A and calculated the mean and the standard deviation of the scores to the programs given by each group in response to Task B (\(n\) indicates the number of students in each group). Upon examining Table 6 one can see that the scores given to the incorrect programs by students who found

<table>
<thead>
<tr>
<th>Program</th>
<th>No.</th>
<th>Categories of students’ responses</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>By heart/reading</td>
<td>Testing</td>
</tr>
<tr>
<td>Erroneous</td>
<td>1</td>
<td>25%</td>
<td>26%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25%</td>
<td>18%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23%</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>21%</td>
<td>11%</td>
</tr>
<tr>
<td>Error-free</td>
<td>3</td>
<td>24%</td>
<td>18%</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>19%</td>
<td>17%</td>
</tr>
</tbody>
</table>

Table 5. The distribution of students’ responses to Task D (\(n = 159\)).

<table>
<thead>
<tr>
<th>No.</th>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(n)</td>
<td>Mean ± SD</td>
<td>(n)</td>
<td>Mean ± SD</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>5.7 ± 1.6</td>
<td>31</td>
<td>4.4 ± 3.5</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>7.4 ± 2.8</td>
<td>40</td>
<td>8.2 ± 2.7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4.8 ± 3.2</td>
<td>48</td>
<td>8.8 ± 2.1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>4.6 ± 3.7</td>
<td>35</td>
<td>7.4 ± 3.3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8.3 ± 1.8</td>
<td>142</td>
<td>9.8 ± 0.9</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
<td>7.3 ± 2.5</td>
<td>116</td>
<td>9.7 ± 0.9</td>
</tr>
</tbody>
</table>

Table 6. The correlation between students’ responses to Tasks A and B.

*Note: Programs 1, 2, 4, and 5 are erroneous, whereas programs 3 and 6 are error-free.*
no error (Rank 1) are lower than the scores given by students who found ancillary errors (Rank 2), which in turn are lower than the scores given by students who found a logic error (Rank 3). This trend is statistically significant, with negative correlations between the errors that students found (Task A) and the scores they assigned to all the programs (Task B) [program 1: Spearman ($r_s$) = $-0.187$, $P < 0.05$; program 2: Spearman ($r_s$) = $-0.532$, $P < 0.01$; program 3: Spearman ($r_s$) = $-0.454$, $P < 0.01$; program 4: Spearman ($r_s$) = $-0.629$, $P < 0.01$; program 5: Spearman ($r_s$) = $-0.472$, $P < 0.01$; program 6: Spearman ($r_s$) = $-0.687$, $P < 0.01$). From this trend we conclude that students distinguish between logic and ancillary errors and reduce the program’s grade accordingly.

No statistically significant correlations between other tasks for programs were detected, with two exceptions, both exceptions relating to program 4: a significant correlation was found between Task A and Task D for program 4 ($\lambda = 0.166$, $P < 0.001$) and also between Task A and Task B ($\lambda = 0.72$, $P < 0.05$). On the basis of this statistical analysis we concluded that students’ decisions regarding the correctness of the programs (Task C) were not dominated by students’ detection of errors in the program (Task A), which essentially means that when students detected an error (even if it was a logic error) it was not automatically translated into the decision that the program was incorrect.

### 4.3. Students’ (mis)conception of “relative correctness” as obtained from the interviews

A prominent theme in the interviewees’ discourse was the notion of “partial correctness.” The second author, henceforth referred to as “M,” who conducted the interviews, asked each student to decide whether the algorithm described above (see Appendix 2) was correct or incorrect. As discussed earlier, the algorithm correctly calculates the average on the final iteration, but it also calculates an incorrect average in all earlier iterations. Two students declared that they did not find any errors and thus decided that the program was correct. Two other students decided that the program was incorrect as soon as they noticed that the variable that eventually contained the average was overridden. The remaining three students found the same supposed error but suggested that the program was partially correct. The following segment is taken from an interview with one of the students, S1, right after he found the alleged error.

M: Ah. . . . So is the algorithm correct or incorrect? What do you think?
S1: I’d give it [a score of] 5 because it prints the number of the students that failed. It does not find the average score of the class, so it fulfils 50% of the requirements.
M: So is the algorithm correct or incorrect?
S1: It is partially correct.
M: What does partially correct mean?
S1: It does part of what it was required to do. Half.

Interestingly, S1’s prompt answer was worded in terms of scores; S1 told M what score he thought the program should get (line 2), although he was not asked about it (line 1). He reasoned that a score of 5 indicated that the program fulfills half of its requirements. When asked again about the program’s correctness (line 3) he chose to use the term “partially correct,” although this term was not previously used by M. This term was further elaborated by S2, who used it in a similar way: “Partially. It is not completely correct, but it is correct in principle.”
The strong relationship between partial correctness and grading was explicitly expressed in the following conversation with S3. This excerpt begins with M asking the student why he did not respond to Task C in the questionnaire for any of the programs. S3 explained that grading is identical to deciding on correctness because grading indicates the level of the correctness of programs (line 4):

M: I noticed that in any of the programs you did not write whether they were correct or incorrect. Is there any particular reason?
S3: [Looking at his questionnaire] I did.
M: No. You gave only grades.
S3: It’s the same. Zero is for an incorrect program and so forth. It is like levels.

Furthermore, S4 explained that her conception of correctness as relative prevented her from responding to Task C, in which she needed to respond in an absolute manner of yes or no (line 1).

S4: The programs are partially correct. The grade is not 10. So you can’t say that the program is either correct or incorrect. The grade is something in between, so I cannot say it is correct, but . . . [pausing].
M: Is there such a thing as “partially correct”?
S4: No. This is why I did not answer [Task C] . . . there is no such concept but I have this concept.
M: So what do you mean by partially correct?
S4: That part of the program is correct. I wrote some operations wrong. I forgot some operations. But it does not mean the program is incorrect.
M: Is there a case where you would decide that a program is certainly incorrect?
S4: No. Well . . . maybe if one operation . . . no . . . maybe if everything is totally wrong then, of course, [a score of] 0 out of 10.

Note that S4’s decision regarding program correctness is based upon its code constituents. Partial correctness means that some of the operations were written incorrectly (line 5). Incorrect programs are programs where everything is wrong and they are exactly the same programs that receive a score of 0 (line 7). This straightforward translation between scores and correctness was expressed by S3 too, in the above excerpt (line 4), as well as by S1 (line 2).

Furthermore, note that the students’ notion of relative correctness is based on a theme that program correctness should be calculated as the sum of the correctness of its constituent parts. Thus, as long as there is a grain of correctness – one or more operations written correctly (as elaborated by S4 in lines 5 & 7) – the program’s grade is greater than 0, and straightforwardly, the program is not considered as incorrect, however, the program is also not considered to be correct (line 1). The decision that a program is incorrect is reserved for the rare occasion where no evidence of correctness is found and thus the program receives a grade of 0.

4.3.1. Students’ approach to grading

The perspective of a program as the sum of independent constituents was repeated when we asked students to explain how they graded a program. The idea was that a program
should be graded not as a whole but rather as the sum of the grades of the operations that
constitute it. This idea was evident in all the transcripts. Here we exemplify it with two
excerpts:

S4: I divide the program into several parts and grade each part. For example, two-thirds of the
program gets a grade of 6 or 7. [In general] Points should be given on operations that were
constructed right, separately.

S2: Think of a box that you have to fill in with stuff. If you put in all the stuff, then you get 100
[points] and if you put in part – like an error that is not essential, such as assigning a zero
value to a variable – then remove some of the points.

4.3.2. Methods of verification

Only one student tested the program. The other six checked their program by reading the
source, as explained by S3 and S4:

S3: I read it [the program]. Step after step. I read it. And the requirements. . . . I roughly know
how I would have written it so I read and compared it.

M: Did you use any input examples?

S4: No. I just read it.

In another part of each interview these six students, who verified correctness by reading
the programs “in their head,” were asked about their views regarding the relative efficacy
of different verification methods. Four asserted that testing is a better way to verify
correctness than reading code. Of these four, two students explained that testing is time
consuming (recall that they worked in a pen and paper environment) and that in an
examination they would test the programs if they had time. One explained that she found
the error immediately so she saw no point in tracing the program’s execution.

M: Why not [testing] this time?

S6: [Laughing] Because it is long.

M: Usually, what is your habitual work process?

S6: In exams I am afraid of making mistakes. So if I have time I’ll choose numbers [input
examples] and try to trace them with tables. Here I simply saw the error right away.

Similarly, the other two students said that the programs given in the questionnaire were
short so they thought there was no need to test them. Of the two that did not assert that
testing is better than reading, one of the two claimed that he did not know any verification
methods other than reading.

5. Discussion

We found that students’ decisions regarding the correctness of programs were not
governed by the absolute definition that they had been taught (Gal-Ezer & Harel, 1999;
Ginat, 1998), according to which the existence of an error means that the program is
incorrect. In contrast, a students’ definition involves a gray area of “partial correctness”
for programs that have errors and thus cannot be considered as correct yet the students
find a “grain of correctness” in them. This grain is detected by reading through the program code seeking instruction(s) that are written correctly, i.e. the student seeks sub-tasks (even minor ones) that were supposed to be implemented.

Furthermore, there was no correlation between the students’ responses regarding a program’s correctness (Task C) and the type of error found in the program (Task A), which implies that there are no errors fatal enough to motivate students to decide that the program is incorrect, although they would probably penalize the program by subtracting points from its score. Nevertheless, the program would get a score greater than 0 as long as there was something in its code that was properly written and would thus straightforwardly be considered as partially correct. Moreover, program correctness was perceived as the sum of the correctness of its constituent operations and, therefore, a program is incorrect only if no part of the program code does what it is required to do, which rarely happens.

Thus, partial correctness is not limited to (partially) working programs, which were examined in a previous work (Ben-David Kolikant, 2005), but rather non-working programs are also considered by students to be partially correct. Additionally, students made these decisions by inspecting the code, a method that is culturally distanced from computer users, who are accustomed to working at the interface level and to whom the code is usually unavailable.

It could be that schools, or as Edwards (2004) put it, “our best efforts as educators,” unintentionally nurture students’ (mis)conceptions of correctness and correspondingly students’ inadequate correctness-related practices. More specifically, we suggest that students used knowledge gained in situations involving grading programs, mistakenly extending it to those situations where program correctness was concerned. These suggestions are based on students’ responses in the interviews, when our interviewees answered in terms of grades when asked about correctness. Furthermore, their perceptions regarding correctness as the sum of the correctness of the program parts resembled the calculation of a program grade as the sum of the grades of its constituent parts.

Students’ conception of correctness is consistent with the widespread argument in the literature that novices approach programming tasks in a local manner (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999) or, using a different wording, in a line by line manner (Winslow, 1996; see also a review by Robins et al., 2003). In fact, Fluery (1993) distinguished between novice and expert programmers by a local or holistic viewpoint regarding programming. Students’ conceptions are thus connected to a wider conception of programming. Similarly, the fact that the students chose to analyze program correctness by reading the program line by line, dealing with each line in a local manner, although they were taught the (holistic) testing method, is also an expression of students’ local viewpoint regarding programming activities.

6. Conclusions

We conclude that student notions of relative or partial correctness include programs that have “a grain of correctness,” i.e. at least one or more operations that were put in a reasonable place with respect to the program’s requirements. Only programs where the students could not find any such “grain of correctness” would be considered as incorrect. We claim that this perception is a ramification of novices’ tendencies to employ a local viewpoint regarding program correctness, unlike experts, who employ a global viewpoint, as described in the literature.
Moreover, we suggest that the school’s correctness-related practices of grading actually nurtures students’ misconception of correctness. Teachers should be aware of the possibility that students’ perceive correctness as relative. Furthermore, teachers should be aware of the indirect yet powerful influence of their practices on students’ conceptions of correctness and of programming in general. The fact that the Grade 12 students were less tolerant of logic errors than the other students is encouraging. However, CS educators should educate the younger generation that an error, even a small one, means that the program is incorrect. After all, even a minor error can cause a disaster and result in a major expenditure in terms of human life. Nevertheless, further work is required to explore teachers’ beliefs underlying their pedagogical decisions, such as the belief expressed by Joni and Soloway (1986, p. 96) that “everyone will agree that a non-working program is incorrect,” which this work demonstrates is wrong.

References


Appendix 1. The six programs given in the questionnaire

**Program 1**

Input: 2 integers.
Output: the two numbers, the bigger number first followed by the smaller number.

1. Read (Min)
2. Read (Max)
3. if Min > Max
   3.1. then
      3.1.1. Max ← Min
      3.1.2. Temp ← Max
      3.1.3. Min ← Temp
4. write (‘the bigger number is’, Max, ‘the smaller number is’, Min)

Input samples for verification (you may use additional or other input samples).

a. Input: Min: 1, Max: 3 The expected output is: ‘the bigger number is 3 the smaller number is 1.’
b. Input: Min: 5, Max: 5 The expected output is: ‘the bigger number is 5 the smaller number is 5.’
c. Min: 7, Max: 2 The expected output is: ‘the bigger number is’ 7 the smaller number is 2.’

**Program 2**

The algorithm checks if there is an element in the array whose value is equal to the value of the variable Number. If so, then it assigns the value ‘true’ to Found, a Boolean variable, if not, Found is assigned ‘false.’

1. Found ← False
2. for I runs from 1 to 10 do
   2.1. if A[I] = Number
      2.1.1. then Found ← True
      2.1.2. else Found ← False

Input samples for verification (you may use additional or other input samples).

(1) Number = 5

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[i]</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(2) Number = 6

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[i]</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
Program 3

Input: 3 integer numbers.
Output: 'Yes' if the third number in the input equals the sum of the other two numbers in the input. 'No' otherwise.

1. read (X)
2. read (Y)
3. read (Z)
4. if \( Y + X = Z \)
   4.1. then write ("yes")
   4.2. else write ("no")

Input samples for verification (you may use additional or other input samples).

a. 1 2 3  
b. 2 2 2  
c. 7 5 2  
d. 15 4 6

Program 4

Three people want to use an elevator in which the total maximum weight allowed is 250 kg. The input of the following algorithm is the weight of each of these people. The output is either the message ‘go’, in case their total weight does not exceed 250 kg and ‘don’t go’ otherwise.

1. for I runs from 1 to 3 do
   1.1. read a person’s weight into X
   1.2. Sum ← Sum + X
   1.3. if Sum < 250
       1.3.1. then write ("go")
       1.3.2. else write ("don’t go");

Input samples for verification (you may use additional or other input samples).

a. 50 50 60  
b. 100 50 150  
c. 200 40 60
Program 5

Input: Num, a positive integer number.
Output: the number of even numbers from zero to Num.

1. read (Num)
2. Counter ← 1
3. for N runs from 1 to Num do
   3.1. if N is even
       3.1.1. then Counter ← Counter + N
4. write (Counter)

Program 6

Input: 3 integer numbers read into variables X, Y, and Z.
Output: the maximum value among the three.

1. read (X, Y, Z)
2. if X < Y
   2.1. then Max ← Y
   2.2. else Max ← X
3. if Max < Z
   3.1. then Max ← Z
4. write (Max)

Appendix 2. The program given in the interviews

The goal of the following algorithm is to read the grades of 10 students in an exam for computer science students, and calculate (and display as output) how many students failed the exam as well as the average grade.

1. Average ← 0
2. Failure_number ← 0
3. Sum ← 0
4. for I runs from 1 to 10 do
   4.1. read (Grade)
   4.2. if Grade < 55
       4.2.1. then Failure_number ← Failure_number + 1
   4.3. Sum ← Sum + Grade
   4.4. Average ← Sum / 10
5. write (Average, Failure)