Do Students Use Semantics When Solving Parsons Puzzles? – A Log-Based Investigation

Amruth N. Kumar [0000-0002-1951-3995]

[1] Ramapo College of New Jersey, Mahwah, NJ 07430, USA
amruth@ramapo.edu

Abstract. Parsons puzzles are jigsaw puzzles wherein students are given a program in scrambled order and tasked with reassembling the program in its correct order. Do students use program semantics when solving Parsons puzzles? The answer to this question has implications for the use of Parsons puzzles as a pedagogic tool. In order to answer this question, we considered semantics at the level of statements and control-flow. We analyzed the data collected by a Parsons puzzle tutor over 5 semesters and measured the extent to which students' puzzle-solving behavior conformed with the use of statement-level and control-flow semantics. We found that students used statement-level semantics to assemble up to 73% of the lines in a puzzle and control-flow semantics to assemble up to 47% of the lines. They used statement-level semantics more than control-flow semantics and more on some puzzles than others. Whenever we found a significant difference between C++ and Java students, C++ students used semantics more than Java students. Finally, we did not find an increase in the use of semantics with increased practice.

Keywords: Parsons puzzle, Puzzle-solving strategy, Program semantics.

1 Introduction

In a Parsons puzzle [10], first proposed as an engaging way to learn programming. the student is given a program in scrambled order and asked to reassemble it in its correct order. Some studies have found that Parsons puzzles are effective for learning programming [1,2,3]. Researchers have looked into what helps students solve Parsons puzzles (e.g., [2,4,6,7,8,9]). But, to date, no research has been done on why the puzzles are effective. For example, do students use program semantics when solving the puzzles? We attempted to answer this question. We considered semantics at two levels:

1. **Statement-level semantics:** Do students reassemble lines that appear next to each other in a program one after the other? For example, do students pick line 21 to assemble after assembling line 20 (or vice versa)? A student who does not use statement-level semantics might assemble the program in seemingly random order: line 35 after line 20, line 8 after line 35, and so on.

2. **Control-flow semantics:** Do students reassemble sections of code in a program in the order in which control flows through the sections when the program is executed? For example, do students assemble input section before output section? The order among the sections of code is a partial order, e.g., both if-clause and else-clause in an if-else statement are conditionally executed before any section that appears after the if-else statement. But the two clauses themselves may be assembled in either order, as long as one clause is assembled completely before the other clause is attempted.

Note that statement-level semantics determines which lines should be assembled together within each section of code (e.g., input section, compute section, or output section). Control-flow semantics builds upon statement-level semantics by superimposing an order among the sections of code (e.g., input section must precede compute section). The hypotheses of our study were:

1. RQ1: Students use statement-level semantics when solving Parsons puzzles;
2. RQ2: Students use control-flow semantics when solving Parsons puzzles;
3. RQ3: The use of statement-level and control-flow semantics increases with practice.

2 Computing the Use of Semantics

We define **action sequence** as the order of the lines in the puzzle to which a student applies permissible actions during the puzzle-solving process. For example, in [4, 1, 4, 3, 2, 3] the student assembles 4th line first, but decides to reorder it after assembling the 1st line. Assuming the student is required to solve the puzzle completely and correctly, **solution sequence** is the order in which the lines in the puzzle are placed in their *final/correct* location. For example, solution sequence corresponding to the above action sequence is [1, 4, 2, 3]. The two sequences differ when a student revises the order before arriving at the correct solution to the puzzle. For our study, we analyzed the solution sequences of students.

We define **semantic sequences** as the possible orders in which a student might solve a puzzle using either statement-level or control-flow semantics. A puzzle may have several semantic sequences. In order to compute a student's use of semantics when solving a puzzle, we compared the student's solution sequence against each semantic sequence for the puzzle as follows:

- Statement-level semantics is the sum of the number of lines in the solution sequence that appear in the same order in the semantic sequence. For example, if the solution sequence is [3,4,1,2,6,5] and a semantic sequence for the puzzle is [1,2,3,4,5,6], the use of statement-level semantics is 4 corresponding to the subsequences [3,4] and [1,2].
- Control-flow semantics is computed by giving credit to only the subsequences in the solution sequence that appear in the same relative order as in a semantic sequence. In the earlier example, the numerical value returned is 2, corresponding to

the subsequence [3,4] – the subsequence [1,2] is not credited because it should have appeared before the subsequence [3,4]. If the solution sequence is [3,4,1,2,5,6] instead, the numerical value returned is 4, corresponding to the subsequences [3,4] and [5,6] which are in correct relative order.

So, computation of statement-level semantics credits all matching subsequences, even if they are out of order, whereas control-flow semantics credits only matching subsequences that are in correct relative order, even if non-contiguous. The degree of use of semantics of a student on a puzzle is the maximum of comparing the student's solution sequence against all the semantic sequences for the puzzle.

3 The Study

For this study, we used epplets (epplets.org), a suite of software tutors on Parsons puzzles available freely online for educational use [5]. The tutors present Parsons puzzles, which students solve using drag-and-drop actions. Students are required to solve each puzzle completely and correctly before going on to the next puzzle. A puzzle containing n lines can be solved with n actions. If a student takes more than 10% redundant actions to solve a puzzle, the tutors schedule additional similar puzzles for the student to solve. The tutors log the sequence of actions taken by students to solve the puzzles.

For this study, we analyzed the data collected by the tutor on if-else statements. The first two puzzles presented by the tutor were on the following programs:

1. A program to read two numbers and print the smaller value among them. This puzzle was on the concept of a single if-else statement in the program.
2. A program to read numerical grade, convert it to letter grade and print it. This puzzle was based on the concept of cascading nested if-else statements, i.e., nesting in either if-clause or else-clause, but not both.

The tutor was used by introductory programming students, both majors and non-majors, as after-class assignments. For this study, we used the data collected by the tutor over five semesters: Fall 2016 – Fall 2018. Data was included in the study from only the students who gave permission for their data to be used for research purposes. When students used the tutor multiple times, data was included only from their first use of the tutor so that results were not affected by practice effect. Both C++ and Java students used the tutor. Where possible, we will separate our analysis by language.

When solving puzzles using the tutor, students were instructed to completely and correctly reassemble each puzzle. They were not instructed to use any particular strategy to solve the puzzles. Students could either solve a puzzle completely and correctly or bail out by clicking on Quit button. If students attempted to submit an incomplete solution, the tutor asked them to complete assembling the unassembled lines in the puzzle. If students attempted to submit a complete but incorrect solution, the tutor highlighted the first misplaced line in the assembled program and suggested whether

the line had to be moved earlier or later in the program. Each puzzle contained two distracters. The student was expected to delete them.

Once the puzzle-solving session ended, the tutor logged the click-stream data of the session, including every permissible operation applied by the student to solve each puzzle. This was the data we used to compute the action sequence and solution sequence of each student for each puzzle. Permissible actions applied to distracters were not included in the solution sequence since distracters were not part of the correctly assembled program.

4 The Results

The first puzzle in if-else tutor contained 16 lines of code and included a single if-else statement. The mean statement-level and control-flow semantics scores for the puzzle, along with 95% confidence interval, separated by programming language, are shown in Table 1.

Table 1. Mean scores on the if-else puzzle no. 1 containing 16 lines of code.

Puzzle 1	N	Statement-level semantics √	Control-flow semantics √
C++	98	10.37 ± 0.41	6.57 ± 0.51
Java	302	9.72 ± 0.20	5.79 ± 0.30

Note that mean statement-level semantics score was around 10 lines out of 16 for both the languages, and mean control-flow semantics score ranged from 5.79 to 6.57 lines. ANOVA analysis of the two scores as dependent variables and programming language as the fixed factor yielded a significant main effect for language on both the scores: statement-level semantics [$F(1,409) = 10.56$, $p = 0.001$] and control-flow semantics [$F(1,409) = 6.64$, $p = 0.01$]. In both the cases, the scores as shown in Table 1 were statistically significantly higher for C++ than Java. In the tables, statistically significant differences are marked with √.

The second puzzle contained 36 lines of code and included nested if-else statements. The scores on this puzzle are listed in Table 2. The difference between the languages was again statistically significant for both the scores. Again, C++ scores were significantly higher than Java scores.

Table 2. Mean scores on the nested if-else puzzle no. 2 containing 36 lines of code.

Puzzle 1	N	Statement-level semantics √	Control-flow semantics √
C++	77	18.95 ± 1.10	9.52 ± 1.05
Java	167	15.86 ± 0.70	8.25 ± 0.67

If a student solved puzzle no. 1 with redundant actions, the tutor presented two more puzzles involving a single if-else statement. These follow-up puzzles nos. 3 and 4 were on the following programs:

- A program to read a number and print whether it is odd or even.

- A program to read sound in decibels, and print whether it is loud (if over 85 decibels) or tolerable.

Table 3 lists the mean scores of students on these follow-up puzzles nos. 3 and 4. The difference between C++ and Java was statistically significant on both the scores for puzzle no. 3, but on neither score for puzzle no. 4.

Table 3. Mean scores on the follow-up if-else puzzles nos. 3 and 4 (13 lines each).

Puzzle	Language	N	Statement-level semantics	Control-flow semantics
3 √	C++	31	8.07 ± 0.62	6.00 ± 0.78
	Java	134	7.25 ± 0.31	4.61 ± 0.39
4	C++	30	7.57 ± 0.59	4.97 ± 0.72
	Java	125	7.42 ± 0.30	4.65 ± 0.36

If a student solved puzzle no. 2 with redundant actions, the tutor presented two more puzzles on nested if-else statements. These follow-up puzzles nos. 5 and 6 were on the following programs:

- A program to read the month and print the corresponding season: Winter (January-March), Spring (April-June), Summer (July-September) and Fall otherwise).
- A program to read a location in degrees and print the corresponding quadrant: First (1-90), Second (91-180), Third (181-270) and Fourth otherwise.

Table 4 lists the mean scores of students on these follow-up puzzles nos. 5 and 6. The difference between the languages was statistically significant on statement-level semantics score for puzzle no. 5, and on both the scores for puzzle no. 6. Again, we found that the scores were significantly higher for C++ than Java.

Table 4. Mean scores on the follow-up nested if-else puzzles nos. 5 and 6 (27 lines each).

Puzzle	Language	N	Statement-level semantics	Control-flow semantics
5	C++	30	15.73 ± 1.11	8.83 ± 1.15
	Java	111	13.32 ± 0.58	7.59 ± 0.60
6 √	C++	25	16.36 ± 1.18	9.88 ± 1.31
	Java	85	14.38 ± 0.65	7.92 ± 0.72

Did the scores of students increase with practice? We considered only the students who had solved all three puzzles: puzzle no. 1 and the two follow-up puzzles nos. 3 and 4 that were similar to it. Since the first puzzle had 16 lines whereas puzzles 3 and 4 had only 13 lines, we normalized the scores as percentages of the number of lines in the puzzle, as shown in Table 5. The change in score percentage from one puzzle to the next was not statistically significant except for statement-level semantics, which *decreased* from puzzle 1 to puzzle 3 for Java students. In all the other cases, solving multiple puzzles did not lead to significant change, let alone increase in the score percentage.

We had expected the scores to increase with practice due to practice effect. The tutor itself did not provide any feedback to help students increase their use of local or control-flow semantics while solving puzzles. What we found was that without such explicit support / feedback from the tutor, the puzzle-solving strategies of students did not change over the course of solving three similar puzzles.

Table 5. Normalized scores on the first three if-else puzzles nos. 1, 3 and 4 as percentages.

Language	Semantics	Puzzle 1	Puzzle 3	Puzzle 4
C++	Statement-level	61.67	61.54	58.21
(N=30)	Control-flow	39.38	45.90	38.21
Java	Statement-level	58.80	55.13 √	57.18
(N=120)	Control-flow	33.13	35.83	35.96

5 Discussion

We analyzed the data of students solving six if-else puzzles in this study. The measure of statement-level semantics was 55-62% on the three if-else puzzles and 44-61% on the three nested if-else puzzles. This confirmed our hypothesis RQ1 that students used statement-level semantics when solving Parsons puzzles.

The measure of control-flow semantics was 33-46% on if-else puzzles and 23-37% on nested if-else puzzles. This confirmed our hypothesis RQ2 that students used control-flow semantics when solving at least part of the Parsons puzzles. Students used statement-level semantics more than control-flow semantics, and more on some puzzles than others.

We observed that whenever the difference between C++ and Java scores was significant, C++ scores were greater than Java scores. A simple explanation is that this was a comparison between two independent samples – C++ students never used Java puzzles and vice versa. So, the difference may be attributable to differences between the two groups of students, such as in their level of prior preparation. Since we did not collect students' course performance data, we could not verify this hypothesis.

Statement-level and control-flow semantics scores did not increase with practice. This refuted our third hypothesis RQ3. The tutors did not provide any feedback to promote the use of semantics. In the absence of such explicit support, the puzzle-solving strategy of students did not change, much less improve, in spite of the potential of practice effect.

In this study, we considered only complete and correct student solutions. We did not consider partial solutions, which may provide additional insights into the use of semantics when solving Parsons puzzles. We used solution sequence representation that ignores intermediate deliberations of students. So, our analysis did not capture any trial-and-error puzzle-solving behaviors.

Acknowledgments: Partial support for this work was provided by the National Science Foundation under grants DUE-1432190 and DUE-1502564.

References

1. Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new Exam Question: Parsons Problems. In Proceedings of the Fourth International Workshop on Computing Education Research (ICER '08). ACM, New York, NY, USA, 113-124. DOI=http://dx.doi.org/10.1145/1404520.1404532.
2. Barbara J. Ericson, James D. Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18). ACM, New York, NY, USA, 60-68. DOI: https://doi.org/10.1145/3230977.3231000
3. Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. 2017. Solving Parsons Problems Versus Fixing and Writing Code. In Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17). ACM, New York, NY, USA, 20-29. DOI: https://doi.org/10.1145/3141880.3141895.
4. Juha Helminen, Petri Ihantola, Ville Karavirta, and Lauri Malmi. 2012. How do Students Solve Parsons Programming Problems?: An Analysis of Interaction Traces. In Proceedings of the ninth annual international conference on International computing education research (ICER '12). ACM, New York, NY, USA, 119-126. DOI: https://doi.org/10.1145/2361276.2361300.
5. Amruth N. Kumar. 2018. Epplets: A Tool for Solving Parsons Puzzles. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). ACM, New York, NY, USA, 527-532. DOI: https://doi.org/10.1145/3159450.3159576.
6. Amruth N. Kumar. 2019. Mnemonic Variable Names in Parsons Puzzles. In Proceedings of the ACM Conference on Global Computing Education (CompEd '19). ACM, New York, NY, USA, 120-126. DOI: https://doi.org/10.1145/3300115.3309509
7. Amruth N. Kumar. 2019. Helping Students Solve Parsons Puzzles Better. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19). ACM, New York, NY, USA, 65-70. DOI: https://doi.org/10.1145/3304221.3319735
8. Amruth N. Kumar. 2017. The Effect of Providing Motivational Support in Parsons Puzzle Tutors. In Proceedings of Artificial Intelligence in Education. (AI-ED 2017), Wuhan, China, June 2017, 528-531. DOI= https://doi.org/10.1007/978-3-319-61425-0_56
9. Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals Help Students Solve Parsons Problems. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 42-47. DOI: https://doi.org/10.1145/2839509.2844617.
10. Dale Parsons and Patricia Haden. 2006. Parson's Programming Puzzles: A fun and Effective Learning Tool for First Programming Courses. In Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE '06), Denise Tolhurst and Samuel Mann (Eds.), Vol. 52. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 157-163.