A Template-Based Technique for Automatic Generation of Programming Word Problems

Sonja Linghui Shan, Michael Horsch, Roy Ka-Wei Lee
Department of Computer Science

ABSTRACT

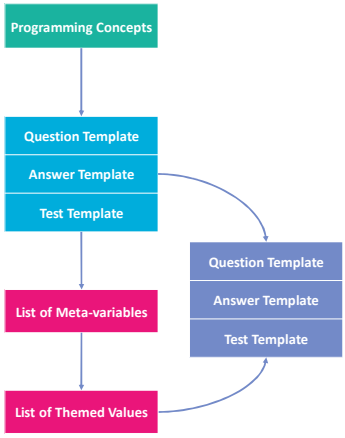
- Programming word problems (PWPs) are questions presented in written text expecting programming language solutions.
- We employed a template-based approach to **automatically generate** PWPs, which includes creating the natural language text questions, the programming language solutions, and test cases.
- To achieve variations in question context, we use meta-variables to create PWPs that differ in their textual themes but exemplify similar programming concepts.
- The approach allows us to generate PWPs much more quickly than writing them individually.

MOTIVATION

- PWPs are one of the most used question types for student practice and assessment in Computer Science.
- To ensure the effectiveness of PWPs, instructors need to provide students with new PWPs as previously-used questions can circulate among students.
- Thus, for educational purposes, there is a demand for a constant supply of fresh PWPs which includes the questions, the corresponding sample answers, and test cases to check student answers.
- Additionally, many research problems in Natural Language Processing rely on machine learning models. These models need a large supply of PWPs as input. Each sample in the input is a PWP question and its corresponding answer.
- Manual generation of PWPs are labour-intensive and can be error-prone.
- Considering the high demand and the difficulty of manual generation, our research explores a potential solution.

MAP

The following diagram shows the flow of generating a new PWP using our software.



META-VARIABLE GRAMMAR

- Meta-variables are "holes" in the templates that are replaced with themed values in the final synthesized PWP.
- The name of a meta-variable indicates its purpose and the type of replacement values it can take.
- Consider the example from the methods section:
  - <name list1> is the name of a list variable.
  - <ppt txt1> is a property described in text.
  - <ppt int2> is a property described with an integer.

METHODS

The following example shows the steps in generating a new PWP, including the question, sample answer, and test cases, using our system:

a. We start by deciding on the programming concepts this PWP is meant to embody.

Programming Concepts	Use of built-in functions
	List offset and operations

b. Based on the programming concepts, we develop a set of templates to address the question, answer, and test cases.

Question Template	Assume that you are given a list named <name list1>. The length of the list is <ppt int1>. Print the <ppt txt1> element in the list.
Answer Template	print(<name list1>[<ppt int2>])
Test Template	def test_1(): <name list1> = [2, [1], 1.33, "0", (4, 5), {}] Student Code* assert <name list1>[<ppt int2>] == Output from Student Code*

c. A list of meta-variables, unique to this set of templates, are extracted by our software.
d. We input theme-specific values to replace the meta-variables with.

Meta-variables and Themed Values	<name list1>	baitNum
	<ppt int1>	6
	<ppt txt1>	Second
	<ppt int2>	1

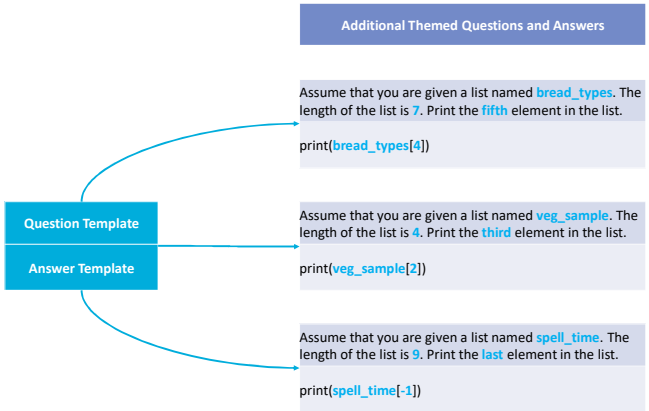
e. Our software then combines the templates with the list of meta-variable replacement values to create a new PWP with its corresponding answer and test cases.

Themed Question	Assume that you are given a list named baitNum. The length of the list is 6. Print the second element in the list.
Themed Answer	print(baitNum[1])
Themed Test	def test_1(): baitNum = [2, [1], 1.33, "0", (4, 5), {}] Student Code* assert baitNum[1] == Output from Student Code*

* Student answers are automatically extracted and inserted into the themed test cases to generate a grading script.

RESULTS

- We authored **52** templates, covering various CMPT 140/141 topics, and **4** sets of textual themes. Combining them we generated **208** new PWPs, each of which includes a question, a suggested solution, and several test cases.
- Our software is effective in generating new PWPs since given our approach:
  - The fixed template embodies the fundamental programming concepts.
  - The varied meta-variable replacement values support the different textual contexts we need for a new PWP.
- By substituting the meta-variables with different theme-specific values, we can easily synthesize various PWPs using a single template.
- For example, combining the template from the methods section with other theme-specific values, we generated the following different PWP questions and solutions. The accompanying test cases are omitted given their similarities to what is shown in the methods section.



FUTURE WORKS

- To the best of our knowledge, our research is the first attempt in automatic generation of PWPs using a template-based approach. Our software is effective, however, the fixed nature of the templates can lead to a limited number of variations and at times awkward phrasing.
- Pending approval from the Research Ethics Board, we intend to recruit first year computer science undergraduates to answer these machine-generated questions and explore the following questions based on their feedback:
  - Do machine-generated questions have the same variations in difficulty levels as manually-generated ones?
  - Do students find the machine-generated questions excessively confusing in terms of phrasing or other aspects?
  - How does student performance on the machine-generated questions differ from their performance on manually-generated ones?
  - Can the machine-generated questions can be effectively used as a pedagogical instrument?
- Related to PWPs' use in machine learning, we want to collect the variations in correct programming language answer to the same textual question.
- We use black-box testing for student answers, and we believe their answers will help to diversify our collection of PWP question-and-answer pairs.

DISCUSSION

- We encountered several pronounced challenges during our project.
- To balance complexity and flexibility of the templates we needed to systematically control the placement of meta-variables.
- We devised a grammar system to keep track of the meta-variables for their purposes and the kind of replacement values they are allowed to take.
- The templates for test cases were difficult to compose.
  - Both the flexibility incorporated in the question templates, and the specific values from different themes need to be accommodated by the test cases.
  - Problems including the use of membership and relational operators highlight this issue.
- Our test cases attempt to address potential opportunistic behaviour in assessment.
  - For example, extra testing is added by our software for recursion problems to ensure that recursion is used to execute the task.

ACKNOWLEDGEMENTS

At the University of Saskatchewan, we acknowledge we are on Treaty Six Territory and the Homeland of the Métis. We pay our respect to the First Nation and Métis ancestors of this place and reaffirm our relationship with one another.

Funding for this project was provided by the Natural Sciences and Engineering Research Council and the University of Saskatchewan.

NSERC CRSNG

University of Saskatchewan