Automatic feedback for student mistakes using examples

Björn Hartmann
Gustavo Soares
Loris D’Antoni
Personalized Education

• CCC CAPE 2016 Workshop Vision:
  “Personalization can meet the demands of heterogeneous backgrounds and different learning styles, and ensure engagement and retention.”

• Enabled by online learning platforms (MOOCs and on-campus blended classes) that allow tailoring problem selection and feedback generation to individual students’ needs.
Example: AutomataTutor
(D'Antoni et al, TOCHI 2015; deployed at 20 universities)

Draw the DFA accepting the language:

\{ s \mid ‘ab’ \text{ appears in } s \text{ exactly } 2 \text{ times} \}

Student’s Solution:

![DFA Diagram]

Feedback: Your DFA accepts the language

\{ s \mid ‘ab’ \text{ appears in } s \text{ at least } 2 \text{ times} \}
Challenge: How to include information about errors that are specific to the problem, but general enough to capture many different student instances of the error?
Problem-Specific Hints for Code

• Autograder for Introductory Programming (Singh, Gulwani, Solar-Lezama): Instructors must manually write down error model.

• Our approach: Assume many students are working on the same problem set. Learn hints from student data.
Setting: CS61a – Intro CS at UC Berkeley

• ~1500 students/semester
• Autograding infrastructure already in place: students submit problems to grading server (http://okpy.org), which runs a test suite, logs code and status, and returns feedback to student
• >200GB of plain-text python source code from last few semesters
• Next: DS8 Intro to Data Science – needs more instrumentation
Approach

• Focus on code that runs but fails test suite.
• Capture (incorrect $\Rightarrow$ correct) source pairs from students (cf. Hartmann et al., CHI 2010)
• Extract a reusable transformation script from each source pair using synthesis.
• If script succeeds to fix new program, turn template into conceptual hint.
Write a function `product(n, term)` that returns `term(1) * ... * term(n)`

```
def product(n, term):
    i = 1
    product = 1
    while i <= n:
        product *= term(i)
        i += 1
    return product
```

When should you call the passed-in function “term” in your while loop?
A technique for **learning transformations using examples**

Transformation script synthesis

**Input-output Example** → **DSL syntax** → **Synthesis Algorithms** → **Ranking** → **Top Ranked scripts**
A technique for learning transformations using examples

Transformations script synthesis

Input-output Example

- DSL syntax
- DSL semantics
- Synthesis Algorithms
- Ranking

Top Ranked scripts

Microsoft PROSE (FlashMeta) as the framework for program synthesis

[Polozov and Gulwani, OOPSLA’15]
DSL example
**DSL example**

\[
\text{edit} = \text{Insert}(\text{node}, \text{ConstNode}(\text{CallExpression}, \text{Children}(
\text{LeafConstNode}(\text{NameExpression-term}), \text{SingleChild}(
\text{ConstNode}(\text{Arg}, \text{SingleChild}(
\text{ReferenceNode}(\text{node}, \text{NameExpression}(\text{"i"}))))), 1)
\]
\( edit = \text{Insert}(\text{node}, \text{ConstNode}(\text{CallExpression}, \text{Children}(\text{LeafConstNode}(\text{NameExpression-term}), \text{SingleChild}(\text{ConstNode}((\text{Arg}, \text{SingleChild}(\text{ReferenceNode}(\text{node}, \text{NameExpression}(\text{"i"}))))), 1)) \)
**DSL example**

```
edit = Insert(node, ConstNode(CallExpression, Children(LeafConstNode(NameExpression-term), SingleChild(ConstNode(Arg, SingleChild(ReferenceNode(node, NameExpression("i"))))))), 1)

selectedNodes = Selected(\x => Match(x, AugmentedAssignStatement {NameExpression NameExpression}), InOrderSort(ast))
```
**EditMap**($\lambda node \Rightarrow edit, selectedNodes$) Where

\[
edit = \text{Insert}(node, \text{ConstNode}(\text{CallExpression}, \text{Children}(\text{LeafConstNode}(\text{NameExpression}-\text{term}), \text{SingleChild}(\text{ConstNode}(\text{Arg}, \text{SingleChild}(\text{ReferenceNode}(node, \text{NameExpression}(\text{"i"})))))), 1)
\]

\[
selectedNodes = \text{Selected}(\lambda x \Rightarrow \text{Match}(x, \text{AugmentedAssignStatement} \{ \text{NameExpression} \}, \text{InOrderSort}(ast)))
\]
@start IEnumerable<PythonAst> transformation := Apply(ast, patch);
Patch patch := Patch(editList) | ConcatPatch(editList, patch);
IEnumerable<Edit> editList := EditMap(edit, selectednodes) =
    Map(\node : PythonNode => edit, selectednodes);
IEnumerable<PythonNode> selectednodes := Selected(match, nodes) =
    Filter(\x : PythonNode => match, nodes);
bool match := Match(x, template);
Edit edit := Update(node, n) | Insert(node, n, k) | Delete(node, r);
Node r := ReferenceNode(node, template);
Node n := LeafConstNode(info) | ConstNode(info, children) | r;
IEnumerable<Node> children := SingleChild(n) | Children(n, children);
IEnumerable<PythonNode> nodes := InOrderSort(ast);

@input PythonNode ast;
@feature[Score=TemplateScore] PythonNode template;
@feature[Score=InfoScore] NodeInfo info;
@feature[Score=KScore] int k;
**Program synthesis**

**Example**

**Input**

- \( p += i \)

**Output**

- \( p += \text{term}(i) \)

---

Apply (patch)

- Insert (term(i)), 0, 1

---

Insert (term(i)), 0, 1

- Insert (node, n)

---

\( i \)

- LeafConstNode (type, value)

---

\( i \)

- ReferenceNode (value, template)

---

\( p \)

- \( += \)

- \( i \)
• Prefer ReferenceNode to ConstNode
• Prefer abstract templates to concrete templates
Can our technique generate fixes for student submissions based on previous data?

Preliminary Evaluation
Leave-one-out cross validation

- Mistake is a pair of submissions of a single student containing her correct submission and her last incorrect submission.

- Mistakes were taken from a sample of submissions clustered by test cases.

<table>
<thead>
<tr>
<th>Question</th>
<th>Mistakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>215</td>
</tr>
<tr>
<td>Repeated</td>
<td>108</td>
</tr>
</tbody>
</table>
Our technique fixed **81%** of the submissions

<table>
<thead>
<tr>
<th>Question</th>
<th>Mistakes</th>
<th>Fixed mistakes</th>
<th>Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>215</td>
<td>190 (88%)</td>
<td>8</td>
</tr>
<tr>
<td>Repeated</td>
<td>108</td>
<td>73 (68%)</td>
<td>10</td>
</tr>
</tbody>
</table>
Examples

Update(literal, 1)
Update(literal, 0)
Delete(arg, n − 1), Insert(n, arg, 0)

Insert(arg, n + 1)
Update(binaryExp, <)
Update(nameExp, j), Update(nameExp, j), Update(literal, 0)

Insert(binaryExp, term(x), 1)
Insert(assignStmt, term(x), 1)
Insert(returnStmt, term(x), 1)
Update(literal, n), Insert(returnStmt, term(n), 1)

Update(nameExp, Identity)
Update(literal, Identity)
Insert(if (n == 0) return identity, ifStmt, 0)
Update(expStmt, return expStmt)
Insert(return nameExp, SuiteStmt, 0)
Next steps
1. `def product(n, term):`
2.   `i = 1`
3.   `product = 1`
4.   `while i <= n:`
5.     `product *= i`
6.     `i += 1`
7.   `return product`

---

1. `def product(n, term):`
2.   `i = 1`
3.   `product = 1`
4.   `while i <= n:`
5.     `product *= term(i)`
6.     `i += 1`
7.   `return product`
Further studies

First deployment in CS61a summer or fall 2016

User study: controlled experiment of different hints

Comparison to Autograder
Automatic feedback for student mistakes using examples

Björn Hartmann
Gustavo Soares
Loris D’Antoni