Feedback Generation for Performance Problems in Introductory Programming Assignments

Florian Zuleger, TU Wien
Berkeley, 11.03.14

Joint work with
Sumit Gulwani (MSR), Ivan Radiček (TU Wien)
Feedback generation for programming assignments

Main focus has been on **functional correctness**

We focus on **performance problems** of functionally correct programs

<table>
<thead>
<tr>
<th>s</th>
<th>secret implementation result</th>
<th>your result</th>
<th>Output/Exception</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>...</td>
<td>&quot;aa&quot;</td>
<td>Mismatch</td>
<td>Your puzzle method raised an exception unexpectedly.</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>&quot;aa&quot;</td>
<td>&quot;aa&quot;</td>
<td>IndexOutOfBoundsException</td>
<td>index was outside the bounds of the array.</td>
</tr>
<tr>
<td>&quot;aa&quot;</td>
<td>&quot;aaaa&quot;</td>
<td>&quot;aaaa&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;cb&quot;</td>
<td>&quot;ccbb&quot;</td>
<td>&quot;ccbc&quot;</td>
<td>Mismatch</td>
<td>Your puzzle method produced the wrong result.</td>
</tr>
<tr>
<td>&quot;aaa&quot;</td>
<td>&quot;aaaaaa&quot;</td>
<td>&quot;aaaaaa&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivation:
Observations from Pex4Fun

→ *Small* number of different *strategies*
  – Global high-level insight of a solution
  – Require different feedback

→ Same strategy can have *many* different *implementations*
  – Low-level choices
  – Not relevant to feedback
Different Strategies: Counting

Problem: Are the strings anagrams?

Strategy: For every character in string, count and compare number of characters (O(n^2))

Feedback: Calculate number of characters in pre-processing phase
Different Strategies: Sorting

Strategy: Sort and compare strings \((O(n^2))\)

Feedback: Instead of sorting, compare number of characters

```csharp
01 bool Puzzle(string s, string t) {
02     var sa = s.ToCharArray();
03     var ta = t.ToCharArray();
04     Array.Sort(sa);
05     Array.Sort(ta);
06     return sa.SequenceEqual(ta);
07 }
```
Different Implementations

```csharp
01 bool Puzzle(string s, string t) {
02    foreach (Char ch in s) {
03        if (count(s, ch) != count(t, ch)) {
04            return false;
05        }
06    }
07    return true;
08 }
09 }
10 int count(String s, Char c) {
11    int number = 0;
12    foreach (Char ch in s) {
13        if (ch == c) {
14            number++;
15        }
16    }
17    return number;
18 }
```

```csharp
Counting-Manual
```

```csharp
Counting-Linq
```
Different Implementations

```csharp
01 bool Puzzle(string s, string t) {
    var sa = s.ToCharArray();
    var ta = t.ToCharArray();
    Array.Sort(sa);
    Array.Sort(ta);
    return sa.SequenceEqual(ta);
}
```

```csharp
01 string SortedString(string s) {
    Char[] myArr = s.ToCharArray();
    for(uint i = 1; i < myArr.Length; ++i)
        for(uint j = i; j > 0 && myArr[j-1] > myArr[j]; --j) {
            Char swap = myArr[j];
            myArr[j] = myArr[j-1];
            myArr[j-1] = swap;
        }
    return new String(myArr);
}
```

`Sorting-Library`

```csharp
01 bool Puzzle(string s, string t) {
    var sa = s.ToCharArray();
    var ta = t.ToCharArray();
    Array.Sort(sa);
    Array.Sort(ta);
    return sa.SequenceEqual(ta);
}
```

`Sorting-Manual`
Problem Statement

How to **distinguish** between **different strategies**, while **ignoring implementation details**?

- **Different strategies – Same complexity**
  - IDEA: Performance analysis

- **Same strategy – Implementation differences**
  - IDEA: Syntactic pattern matching
Main idea: Specify Strategies by Key Values

Execution on s=”aab” and t=”aba”:

(2,a) (12,a) (12,a) (12,b) (3,2) (2,a) ... 

(2,a) (3,a) (3,a) (3,b) (4,2) (2,a) ...

Same values traces!
More Key Values

Execution on s="aba" and t="baa":

(5, aab)

(13, aab)
Matching Implementation against Specification

Inputs

Implementation

Specification

Execution

Trace

Trace

Trace Embedding?

MATCH!
Specifications
with Observe Statements

```csharp
01 bool Puzzle(string s, string t) {
02     foreach (Char ch in s) {
03         observe(ch);
04         int count = 0;
05         foreach (Char c in s) {
06             observe(c);
07             if (ch == c) {
08                 count++;
09             }
10         }
11         observe(count);
12     }
13 }
```

Only specified values are recorded

Execution on s=”aba” and t=”baa”:

(3, a) (6, a) (6, b) (6, a) (11, 2) (3, b) ...

Counting-Specification
Traces of Implementations

```csharp
01 bool Puzzle(string s, string t) {
02     foreach (Char ch in s) {
03         if (count(s, ch) != count(t, ch)) {
04             return false;
05         }
06     }
07     return true;
08 }
09 }
10 int count(String s, Char c) {
11     int number = 0;
12     foreach (Char ch in s) {
13         if (ch == c) {
14             number++;
15         }
16     }
17     return number;
18 }
```

All values are recorded

Execution on s=”aba” and t=”baa”:

(2, a) (11, 0) (12, a) (14, 1) (12, b) (12, a) (14, 2) (3,2) ...
Definition: Trace Embedding

Specification trace (A):
(3, a) (6, a) (6, b) (6, a) (11, 2) ...

Implementation trace (B):
(2, a) (11, 0) (12, a) (14, 1) (12, b) (12, a) (14, 2) (3,2) ...

Trace Embedding
Is there an injective mapping of program locations s.t. A is a subsequence of B?

Here: (3,2) (6,12) (11,3)

Intuition:
→ Same values in same order
→ Same values appear on matching locations
Deciding Trace Embedding

Specification trace (A):
(3, a) (6, a) (6, b) (6, a) (11, 2) ...

Implementation trace (B):
(2, a) (11, 0) (12, a) (14, 1) (12, b) (12, a) (14, 2) (3, 2) ...

Deciding Trace Embedding is NP complete.

We have a heuristic algorithm that is fast in practice.
Methodology

- Inefficient Specification
- Efficient Specification

Implementation

No feedback at the moment.

Good job!

New unmatched specification!

Refine existing specification or create a new one.

NO MATCH!

Match!
Experiments

Observations from Pex4Fun Data:

- Large number of inefficient implementations for most assignments (90% for Anagram)
- 62% assignments with more than one inefficient strategy

Performance: On most examples <0.5s

Teacher effort: #specifications/ #implementations = 3%
for ~2000 solutions to 3 problems

Precision: 0 false negatives / 5 false positives
Thank you!