TraceDiff: Debugging Unexpected Code Behavior Using Trace Divergences

Ryo Suzuki, Gustavo Soares, Andrew Head, Elena Glassman, Ruan Reis, Melina Mongiovi, Loris D’Antoni, Björn Hartmann
Increasing Demand for Introductory CS Courses

source: https://www.geekwire.com/2014/analysis-examining-computer-science-education-explosion/
Teachers’ Personalized Attention **Does Not Scale**

**Traditional** Classroom

**Massive** Classroom
(1,000-2,000 students)
Automatic Programming Feedback

Massive Classroom (1,000-2,000 students)
def accumulate(combiner, base, n, term):
    if n == 1:
        return base
    else:
        return combiner(term(n),
                        accumulate(…, n-1, …))

>>> accumulate(add, 0, 5, identity)
15  # 0 + 1 + 2 + 3 + 4 + 5

>>> accumulate(mul, 2, 3, square)
72  # 2 * 1^2 * 2^2 * 3^2
```python
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
        else:
            return combiner(term(n),
                             accumulate(…, n-1, …))
```
def accumulate(combiner, base, n, term):
    if n == 1:
        return base
    else:
        return combiner(term(n), accumulate(combiner, base, n-1, term))

>>> accumulate(add, 0, 5, identity)

Test Case

Feedback

x 14
o 15
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
    else:
        return combiner(term(n),
                        accumulate(…, n-1, …))

accumulate(add, 0, 5, identity)

x 14 # 0 + 1 + 2 + 3 + 4 + 5
o 15 # 0 + 1 + 2 + 3 + 4 + 5
```python
def accumulate(combiner, base, n, term):
    if n == 1:
        return base
    else:
        return combiner(term(n), accumulate(combiner, base, n-1, term))

def add(a, b):
    return a + b

def identity(x):
    return x

accumulate(add, 0, 5, identity) # 0 + 1 + 2 + 3 + 4
>>> 15
```

Expected:
```
accumulate(add, 0, 5, identity) # 0 + 1 + 2 + 3 + 4
>>> 15
```
```python
def accumulate(combiner, base, n, term):
    if n == 1:
        return base
    else:
        return combiner(term(n), accumulate(combiner, base, n - 1, term))

def add(a, b):
    return a + b

def identity(x):
    return x

accumulate(add, 0, 5, identity)
```

**Result**

`accumulate(add, 0, 5, identity) ≠ 0 + 1 + 2 + 3 + 4 + 5`  
>>> 14

**Expected**

`accumulate(add, 0, 5, identity) ≠ 0 + 1 + 2 + 3 + 4 + 5`  
>>> 15

- call `accumulate(add, 0, 5, identity)`
- `n = 5`
- call `accumulate(add, 0, 4, identity)`
  - `n = 4`
  - call `accumulate(add, 0, 3, identity)`
    - `n = 3`
    - call `accumulate(add, 0, 2, identity)`
      - `n = 2`
      - call `accumulate(add, 0, 1, identity)`
        - `n = 1`
        - call `accumulate(add, 0, 0, identity)`
          - `n = 0`
```
### 1. Highlight Differences

<table>
<thead>
<tr>
<th>Result</th>
<th>Expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>accumulate(add, 0, 5, identity) # $0 + 1 + 2 + 3 + 4$</td>
<td>accumulate(add, 0, 5, identity) # $0 + 1 + 2 + 3 + 4$</td>
</tr>
<tr>
<td>&gt;&gt;&gt; 14</td>
<td>&gt;&gt;&gt; 15</td>
</tr>
</tbody>
</table>

- call accumulate(add, 0, 5, identity)
- n = 5
  - call accumulate(add, 0, 4, identity)
    - n = 4
      - call accumulate(add, 0, 3, identity)
        - n = 3
          - call accumulate(add, 0, 2, identity)
            - n = 2
              - call accumulate(add, 0, 1, identity)
                - n = 1
                  - accumulate(add, 0, 1, identity) returns 0
    - n = 2
      - accumulate(add, 0, 2, identity) returns 2
    - n = 3
      - accumulate(add, 0, 3, identity) returns 5
    - n = 4
      - accumulate(add, 0, 4, identity) returns 9
    - n = 5
      - accumulate(add, 0, 5, identity) returns 14
- call accumulate(add, 0, 0, identity)
  - n = 0
    - accumulate(add, 0, 0, identity) returns 0
  - n = 1
    - accumulate(add, 0, 1, identity) returns 1
    - n = 2
      - accumulate(add, 0, 2, identity) returns 3
    - n = 3
      - accumulate(add, 0, 3, identity) returns 6
2. Interactively Explore
3. Abstract Values Into Expressions
Motivation
Program Synthesis Feedback

(e.g. Singh [PLDI’13], D’Antoni [TOCHI’15], Rolim [ICSE’17])
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
        else:
            return combiner(term(n),
                            accumulate(..., n-1, ...))
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
        else:
            return combiner(term(n),
                            accumulate(..., n-1, ...))
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
        else:
            return combiner(term(n),
                            accumulate(…, n-1, …))

Line 2 needs to be changed
In line 2, change n = 1 to n = 0
Program Synthesis Feedback

Line 2 needs to be changed

In line 2, check “n”

In line 2, change n = 1 to n = 0
Program Synthesis Feedback

- Pointing
  - Line 2 needs to be changed
  - In line 2, check “n”

- High-level Hints

- Bottom-out
  - In line 2, change n = 1 to n = 0
Diagnose the cause of error: 19%
Suggest to run code in PythonTutor: 17%
Explain incorrect behavior: 16%
Remind relevant resources: 16%
Provide example usage: 11%
Point out location: 14%
Suggest concrete fix: 7%

132 posts on piazza
Diagnose the cause of error: 19%
Suggest to run code in PythonTutor: 17%
Explain incorrect behavior: 16%
Remind relevant resources: 16%
Provide example usage: 11%
Point out location: 14%
Suggest concrete fix: 7%

132 posts on piazza
Diagnose the cause of error: 19%
Suggest to run code in PythonTutor: 17%
Explain incorrect behavior: 16%
Remind relevant resources: 16%
Provide example usage: 11%
Point out location: 14%
Suggest concrete fix: 7%

132 posts on piazza
Diagnose the cause of error: 19%
Suggest to run in PythonTutor: 17%
Explain incorrect behavior: 16%
Remind relevant resources: 16%
Provide example usage: 11%
Point out location: 14%
Suggest concrete fix: 7%
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
    else:
        return combiner(term(n),
                        accumulate(…, n-1, …))
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
    else:
        return combiner(term(n),
                        accumulate(…, n-1, …))

accumulate(add, 0, 5, identity)

x 14 # 0 + 1 + 2 + 3 + 4 + 5
o 15 # 0 + 1 + 2 + 3 + 4 + 5
Line 2 needs to be changed
In line 2, change $n = 1$ to $n = 0$

$$\text{accumulate}(\text{add}, 0, 5, \text{identity})$$

$\times 14 \# 0 + 1 + 2 + 3 + 4 + 5$

$\circ 15 \# 0 + 1 + 2 + 3 + 4 + 5$
```
def accumulate(combiner, base, n, term):
    if n == 1:
        if n == 0:
            return base
        else:
            return combiner(term(n),
                            accumulate(…, n-1, …))
else:
    return combiner(term(n),
                    accumulate(…, n-1, …))
```

Our Goal

```
accumulate(add, 0, 5, identity)
```

`x 14 # 0 + 1 + 2 + 3 + 4 + 5`

`o 15 # 0 + 1 + 2 + 3 + 4 + 5`
Implementation
Incorrect submission

1. Synthesis

Closest correct program

Execute & Record Trace

Filter & Highlight Trace Difference

Result

```
accumulate(add, 0, 5, identity) # 0 +
>>> 10
```

- call accumulate(add, 0, 5, identity)
- total = 0
- > total = 2

Expected

```
accumulate(add, 0, 5, identity) # 0 +
>>> 15
```

- call accumulate(add, 0, 5, identity)
- total = 0
- > total = 1
Incorrect submission

1. Synthesis

Closest correct program

2. Execute & Record Trace

Result

 accumulator(add, 0, 5, identity) # 0 +
 >>> 10

 ✓ call accumulate(add, 0, 5, identity)
 ✓ total = 0
 ✓ total = 2

3. Filter & Highlight Trace Difference

Expected

 accumulator(add, 0, 5, identity) # 0 + :
 >>> 15

 ✓ call accumulate(add, 0, 5, identity)
 ✓ total = 0
 ✓ total = 1
In the image, a process flow is depicted involving steps for handling incorrect submissions in a programming context.

1. **Synthesis**
   - Incorrect submission
   - Closest correct program

2. **Execute & Record Trace**
   - Example code execution:
     - `accumulate(add, 0, 5, identity) # 0 +
       >>> 10`
     - `call accumulate(add, 0, 5, identity)`
     - `total = 0`
     - `total = 2`

3. **Filter & Highlight Trace Difference**
   - Expected code execution:
     - `accumulate(add, 0, 5, identity) # 0 +
       >>> 15`
     - `call accumulate(add, 0, 5, identity)`
     - `total = 0`
     - `total = 1`
Learning Code Transformation (e.g., Refazer [ICSE’17])

Example from student submissions (e.g. Student 1)

```python
def product(n, term):
    total, k = 1, 1
    while k<=n:
        total = total * k
        total = total * term(k)
        k = k+1
    return total
```
Learning Code Transformation (e.g., Refazer [ICSE’17])

Example from student submissions (e.g. Student 1)

Learn code transformation from examples

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

Insert

\[
\text{<exp>} \times \text{<name>} \rightarrow \text{<exp>} \times \text{term(}<\text{name}>)\]
Learning Code Transformation (e.g., Refazer [ICSE’17])

Example from student submissions (e.g. Student 1)

Learn code transformation from examples

Apply code transformation (e.g., Student 2)
Incorrect submission

1 Synthesis

Closest correct program

2 Execute & Record Trace

3 Filter & Highlight Trace Difference

Result

```python
accumulate(add, 0, 5, identity) # 0 +
```

```python
>>> 10
```

- call accumulate(add, 0, 5, identity)
- total = 0
- total = 2

Expected

```python
accumulate(add, 0, 5, identity) # 0 +
```

```python
>>> 15
```

- call accumulate(add, 0, 5, identity)
- total = 0
- total = 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

n: 3
n: 3

i: 1
i: 1

total: 0
total: 10
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1

n: 3
total: 0, 1
i: 1, 2
k: 1

n: 3
total: 10, 11
i: 1, 2
k: 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

n: 3
total: 0, 1
i: 1, 2
k: 1, 4

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1

n: 3
total: 10, 11
i: 1, 2
k: 1, 4
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
```python
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1
```

```python
def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
```

- n: 3
- total: 0, 1, 5, 14
- i: 1, 2, 3, 4
- k: 1, 4, 9

- n: 3
- total: 10, 11, 15, 24
- i: 1, 2, 3, 4
- k: 1, 4, 9
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1

n: 3
total: 0, 1, 5, 14
i: 1, 2, 3, 4
k: 1, 4, 9

n: 3
total: 10, 11, 15, 24
i: 1, 2, 3, 4
k: 1, 4, 9
Result

```
accumulate(add, 11, 5, identity) # 11 + 1 + 2 + 3 + 4 + 5
```

```bash
call accumulate(add, 11, 5, identity)
term = identity
combiner = add
base = 11
n = 5
k = 5
call accumulate(add, 11, 4, identity)
n = 4
k = 4
call accumulate(add, 11, 3, identity)
n = 3
k = 3
call accumulate(add, 11, 2, identity)
```
However, showing all of traces can overwhelm students with too much information and make it difficult to grasp an overview of the behavior.
Incorrect submission

1. Synthesis

2. Execute & Record Trace

3. Filter & Highlight Trace Difference

Closest correct program
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
def accumulate(n):
    total = 0
    i = 1
    while i <= n:
        k = square(i)
        total = total + k
        i = i + 1

def accumulate(n):
    total = 10
    i = 1
    while i <= n:
        k = square(i)
        total = total + i
        i = i + 1
Without filtering

accumulate(add, 11, 5, identity) # 11 + 1 + 2 + 3 + 4 + >>> 81

- call accumulate(add, 11, 5, identity)
- term = identity
- combiner = add
- base = 11
- n = 5
- k = 5
- call accumulate(add, 11, 4, identity)
- n = 4
- k = 4
- call accumulate(add, 11, 3, identity)
- n = 3
- k = 3
- call accumulate(add, 11, 2, identity)

+ 24 lines

With filtering

accumulate(add, 11, 5, identity) # 11 + 1 + 2 + 3 >>> 81

- call accumulate(add, 11, 5, identity)
- call accumulate(add, 11, 4, identity)
- call accumulate(add, 11, 3, identity)
- call accumulate(add, 11, 2, identity)
- call accumulate(add, 11, 1, identity)
- call accumulate(add, 11, 0, identity)
- accumulate(add, 11, 0, identity) returns 11
- accumulate(add, 11, 1, identity) returns 23
- accumulate(add, 11, 2, identity) returns 36
- accumulate(add, 11, 3, identity) returns 50
- accumulate(add, 11, 4, identity) returns 65
- accumulate(add, 11, 5, identity) returns 81
Value Abstraction
Abstract Values into Expressions

Result

\[
\text{accumulate}(\text{add}, 0, 5, \text{identity}) \# 0 + 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\gg \gg 10
\]

- call `accumulate(add, 0, 5, identity)`
- total = 0
- total = 2
- total = 4
- total = 6
- total = 8
- total = 10
- `accumulate(add, 0, 5, identity)` returns 10

Result

\[
\text{accumulate}(\text{add}, 0, 5, \text{identity}) \# 0 + 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\gg \gg 10
\]

- call `accumulate(add, 0, 5, identity)`
- total = base
- total = add(1, 1)
- total = add(2, 2)
- total = add(3, 3)
- total = add(4, 4)
- total = add(5, 5)
- `accumulate(add, 0, 5, identity)` returns total
total = combiner(i, term(i))
total = combiner(i, term(i))
total = combiner(i, term(i))
total = combiner(i, term(i))
total = combiner(i, term(i))

20 → add(11, 9)
total = combiner(i, term(i))

20 \rightarrow add(11, 9) \rightarrow add(11, square(3)) \rightarrow ...
Abstract Values into Expressions

result

accumulate(add, 0, 5, identity) # 0 + 1 + 2 + :
>>> 10

> call accumulate(add, 0, 5, identity)
> total = 0
> total = 2
> total = 4
> total = 6
> total = 8
> total = 10
> accumulate(add, 0, 5, identity) returns 10

result

accumulate(add, 0, 5, identity) # 0 + 1 + 2 + :
>>> 10

> call accumulate(add, 0, 5, identity)
> total = base
> total = add(1, 1)
> total = add(2, 2)
> total = add(3, 3)
> total = add(4, 4)
> total = add(5, 5)
> accumulate(add, 0, 5, identity) returns total
Evaluation
TraceDiff ↔ PythonTutor

17 participants

4 problems (2: TraceDiff, 2: PythonTutor)

2 tasks for each problem:

- **Identify** the bug
- **Fix** the bug
RQ1: Can TraceDiff help students fix bugs **faster** than Python Tutor?

RQ2: Can TraceDiff help students identify and fix **more bugs** than Python Tutor?

RQ3: Do students perceive TraceDiff to be **more useful** for debugging tasks?
RQ1: Can students fix bugs faster?
RQ2: Can students identify and fix more bugs?

<table>
<thead>
<tr>
<th></th>
<th>TraceDiff</th>
<th>Python Tutor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify</td>
<td>82.4</td>
<td>76.5</td>
</tr>
<tr>
<td>Fix</td>
<td>70.1</td>
<td>61.8</td>
</tr>
</tbody>
</table>
Discussion

Task Selection: Simple ↔ Complex
Task Setting: **Single** ↔ **Multiple** Attempts
RQ3: Do students think the tool is more useful?

![Bar chart showing the comparison between TraceDiff and Python Tutor in terms of usefulness.

- Overall: TraceDiff 4.7, Python Tutor 6.4
- Help to identify: TraceDiff 4.8, Python Tutor 5.9
- Help to Understand: TraceDiff 4.6, Python Tutor 5.7
- Help to Fix: TraceDiff 3.9, Python Tutor 5.3
- Improve Skills: TraceDiff 4.7, Python Tutor 4.9

7-point Likert Scale: 0 to 7.
Future Work
Future Work

**Deploy** to actual programming courses and

**Evaluate** in more realistic situation
Contributions:

1. **A characterization of key design guidelines** for effective programming feedback that can be generated by state-of-the-art synthesis techniques, informed by a formative study.

2. **The implementation of hints in an interactive debugging interface**, appropriate for deployment and evaluation in a massive programming classroom.

3. **Quantitative and qualitative results of a controlled experiment with 17 students** where we compare TraceDiff with Python Tutor interface.
TraceDiff: Debugging Unexpected Code Behavior Using Trace Divergences [VL/HCC 2017]
https://ryosuzuki.github.io/trace-diff/

Add topics

219 commits  2 branches  2 releases  1 contributor  MIT

Branch: master  New pull request

ryosuzuki Update README.md

Latest commit 11f24e0 on Aug 14

data  Rebuild  2 months ago
resources  Add resources  2 months ago
Thank you