• Test-cases based feedback
  • Hard to relate failing inputs to errors

• Manual feedback by TAs
  • Time consuming and error prone
Scalability Challenges (>100k students)

Bigger Challenge in MOOCs
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e]*e)
    return deriv

def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e]*e)
    return deriv

Teacher’s Solution  Grading Rubric
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e] * e)
    return deriv

def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
    else:
        deriv.append(poly[e] * e)
    return deriv
Technical Challenges

Large space of possible corrections
Minimal corrections
Dynamically-typed language

Constraint-based Synthesis to the rescue
Running Example
computeDeriv

Compute the derivative of a polynomial

\[
poly = [10, 8, 2] \quad \# f(x) = 10 + 8x + 2x^2
\]
\[
\Rightarrow [8, 4] \quad \# f'(x) = 8 + 4x
\]
Teacher’s solution

```python
def computeDeriv(poly):
    result = []
    if len(poly) == 1:
        return [0]
    for i in range(1, len(poly)):
        result += [i * poly[i]]
    return result
```
Demo
Simplified Error Model

• `return a` ➜ `return {0, ?a}`
• `range(a_1, a_2)` ➜ `range(a_1+1, a_2)`
• `a_0 == a_1` ➜ `False`
Autograder Algorithm
Algorithm

Rewriter

Translator

Solver

Feedback

.py ---- .py ---- .sk ---- .out ----
Algorithm: Rewriter
Rewriting using Error Model

\[
\text{range}(0, \text{len}(\text{poly}))
\]

\[
\text{range}(\{0, 1\}, \text{len}(\text{poly}))
\]

default choice

\[a \rightarrow a+1\]
Rewriting using Error Model

\[
\text{range}(0, \text{len}(\text{poly}))
\]

\[
\text{range}([0, 1], \text{len}(\text{poly}))
\]

\[a \rightarrow a+1\]
Rewriting using Error Model

```
range(0, len(poly))
```

```
range({0, 1}, len({poly, poly+1}))
```

$a \rightarrow a+1$
Rewriting using Error Model

\[
\text{range}(0, \text{len}(\text{poly}))
\]

\[
\text{range}\left(\{0, 1\}, \{\text{len}(\{\text{poly}, \text{poly+1}\}), \\text{len}(\{\text{poly}, \text{poly+1}\})+1\}\right)
\]
def computeDeriv(poly):
    deriv = []
    zero = 0
    if ({len(poly) == 1, False}):
        return {deriv, [0]}
    for e in range({0,1}, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e] * e)
    return {deriv, [0]}

Problem: Find a program that minimizes cost metric and is functionally equivalent with teacher’s solution
Algorithm: Translator

Rewriter → Translator → Solver → Feedback

.py  →  .sk
void main(int x) {
    int k = ??;
    assert x + x == k * x;
}

void main(int x) {
    int k = 2;
    assert x + x == k * x;
}
Translation to Sketch

(1) Handling python’s dynamic types

(2) Translation of expression choices
(I) Handling Dynamic Types

```c
struct MultiType{
  int type;
  int ival;  // bit bval;
  MTString str;  // MTList lst;
  MTDict dict;  // MTTuple tup;
}
```
Python Constants using **MultiType**

1 →

1

- int
- bool

1

- list

2 →

2

- int
- bool

2

- list

[1, 2] →

len=2, lVals = { *, * }

LIST
Python Exprs using MultiType

\[ x + y \rightarrow \]

\[
\begin{array}{c|c|c|c}
5 & \text{INT} & \text{bool} & \text{INT} \\
\text{int} & \text{list} & \text{list} & \text{list} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
10 & \text{INT} & \text{bool} & \text{INT} \\
\text{int} & \text{list} & \text{list} & \text{list} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
15 & \text{INT} & \text{bool} & \text{INT} \\
\text{int} & \text{list} & \text{list} & \text{list} \\
\end{array}
\]
Python Exprs using MultiType

\[ x + y \rightarrow \]

\[
\begin{array}{c|c|c|c}
\text{int} & \text{LIST} & \text{bool} \\
[1,2] & \text{list} & \\
\end{array}
\quad + 
\quad
\begin{array}{c|c|c|c}
\text{int} & \text{LIST} & \text{bool} \\
[3] & \text{list} & \\
\end{array}
\quad = 
\begin{array}{c|c|c|c}
\text{int} & \text{LIST} & \text{bool} \\
[1,2,3] & \text{list} & \\
\end{array}
\]
Python Expressions using **MultiType**

Typing rules are encoded as **constraints**
(2) Translation of Expression Choices

```
{[\text{green}, \text{red}]} \rightarrow
MultiType\ modifyMTi([\text{green}, \text{red}]) {

}
```
harness main(int N, int[N] poly){

    MultiType polyMT = MTFromList(poly);

    MultiType result\_1 = computeDeriv\_teacher(polyMT);
    MultiType result\_2 = computeDeriv\_student(polyMT);

    assert MTEquals(result\_1, result\_2);
}

Translation to Sketch (Final Step)
harness main(int N, int[N] poly){
    totalCost = 0;
    MultiType polyMT = MTFromList(poly);

    MultiType result1 = computeDeriv_teacher(polyMT);
    MultiType result2 = computeDeriv_student(polyMT);

    ............
    if(choice_k) totalCost++;
    ............
    assert MTEquals(result1,result2);
    minimize(totalCost);
}
Algorithm: Solver

Rewriter ➔ Translator ➔ Solver ➔ Feedback

Solver: .sk ➔ Feedback: .out
Solving for minimize(x)

Sketch Uses **CEGIS** – multiple **SAT** calls

**Binary search** for x – no reuse of learnt clauses

**MAX-SAT** – too much overhead

**Incremental linear search** – reuse learnt clauses
Incremental Solving for minimize(x)

\((P, x)\) → Sketch → \((P_1, x=7)\) → Sketch → \(\{x<7\}\)

UNSAT → Sketch → \(\{x<4\}\) → Sketch → \((P_2, x=4)\)
Algorithm: Feedback

Rewriter → Translator → Solver → Feedback

.out
Feedback Generation

Correction rules associated with Feedback Template

Extract synthesizer choices to fill templates
Evaluation
Autograder Tool for Python

Currently supports:

- Integers, Bool, Strings, Lists, Dictionary, Tuples

- Closures, limited higher-order fn, list comprehensions
Benchmarks

Exercises from first five weeks of 6.00x and 6.00

**int**: prodBySum, compBal, iterPower, recurPower, iterGCD

**tuple**: oddTuple

**list**: compDeriv, evalPoly

**string**: hangman 1, hangman 2

**arrays (C#)**: APCS dynamic programming (Pex4Fun)
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>evalPoly-6.00</td>
<td>13</td>
</tr>
<tr>
<td>compBal-stdin-6.00</td>
<td>52</td>
</tr>
<tr>
<td>compDeriv-6.00</td>
<td>103</td>
</tr>
<tr>
<td>hangman2-6.00x</td>
<td>218</td>
</tr>
<tr>
<td>prodBySum-6.00</td>
<td>268</td>
</tr>
<tr>
<td>oddTuples-6.00</td>
<td>344</td>
</tr>
<tr>
<td>hangman1-6.00x</td>
<td>351</td>
</tr>
<tr>
<td>evalPoly-6.00x</td>
<td>541</td>
</tr>
<tr>
<td>compDeriv-6.00x</td>
<td>918</td>
</tr>
<tr>
<td>oddTuples-6.00x</td>
<td>1756</td>
</tr>
<tr>
<td>iterPower-6.00x</td>
<td>2875</td>
</tr>
<tr>
<td>recurPower-6.00x</td>
<td>2938</td>
</tr>
<tr>
<td>iterGCD-6.00x</td>
<td>2988</td>
</tr>
</tbody>
</table>
Average Running Time (in s)
Feedback Generated (Percentage)
Feedback Generated (Percentage)
Feedback Generated (Percentage)
Feedback Generated (Percentage)
<table>
<thead>
<tr>
<th>TestSet</th>
<th>Generated Feedback</th>
<th>Percentage</th>
<th>AvgTime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13365</td>
<td>8579</td>
<td>64.19%</td>
<td>9.91</td>
</tr>
</tbody>
</table>
Why low % in some cases?

- Completely Incorrect Solutions
- Unimplemented Python Features
- Timeout
  - comp-bal-6.00
- Big Conceptual errors
Big Error: Misunderstanding APIs

- eval-poly-6.00x

```python
def evaluatePoly(poly, x):
    result = 0
    for i in list(poly):
        result += i * x ** poly.index(i)
    return result
```
Big Error: Misunderstanding Spec

• hangman2-6.00x

def getGuessedWord(secretWord, lettersGuessed):
    for letter in lettersGuessed:
        secretWord = secretWord.replace(letter,'_')
    return secretWord
A technique for automated feedback generation
Error Models, Constraint-based synthesis

Provide a basis for automated feedback for MOOCs

Towards building a Python Tutoring System

Thanks! rishabh@csail.mit.edu