LLP Laboratório de Linguagens de Programação

HaskellFL: A Tool for Detecting Logical Errors in Haskell

Vanessa Vasconcelos
Mariza Bigonha

Motivation

Functional Programming

Scala

Clojure

Swift

Kotlin

ERLANG

elixir

Racket

elm

Haskell

Functional Programming

- Functional Programming is building software via:
  - Function composition: create new functions by composing others
  - Pure functions: every time it is called, it produces the same result
  - No shared state: no global values
  - Limited side effects: limited iteration with external world
  - Immutability: once a variable is created, its value cannot be changed

Logical Errors

- Logical errors: they do not cause the program to crash or simply not work at all, they cause it to return a wrong output

```
float average(float a, float b)
{
    return a + b / 2;     /* should be (a + b) / 2 */
}
```

Problem Definition

Problem Definition

- Challenges in understanding and taking advantage of the functional paradigm
- Much time spent at debugging

Why Haskell

```haskell
add a b = a + b
```

- Purely functional language
- Pure functions: Haskell, calling add with the same a and b will always return the same value
- Impure functions: C++, moveX modifies pos state

```cpp
class Pos {
    private:
        int x;
        int y;

    public:
        Pos(int x, int y) {
            this->x = x;
            this->y = y;
        }

        void moveX(int inc) {
            this->x = this->x + inc;
        }
};
```

```cpp
Pos pos = Pos(0,0);
pos.moveX(1); // 1 0
pos.moveX(1); // 2 0
pos.moveX(1); // 3 0
```


LLP Laboratório de Linguagens de Programação

Why Haskell

- Used in functional programming introductory classes
- Several companies use Haskell in internal products or research

Compilers

Compilers

Advertising, Spam Filtering

Finance

Hardware

Goals

Goals

- Project and implement a tool, containing a Haskell interpreter for a subset of Haskell 2010 grammar
- Implement two fault localization techniques
- Build a Haskell test suite covering the chosen Haskell grammar's subset

Haskell Grammar Subset

HaskellFL Grammar Subset

- In: functions, case, if then else, guards, pattern matching, abstract data types, let and where, lambda function
- Out: do notation, list comprehension, type declaration

```
lista = [x*2 | x <- [1..10]]
```

```
greetAndSeeYou :: IO ()
greetAndSeeYou = do name <- nameReturn
                    putStrLn ("See you, " ++ name ++ "!")
```

```
type PhoneBook = [(String,String)]
```

HaskellFL Grammar Subset

```haskell
data TriangleType = Equilateral
                  | Isosceles
                  | Scalene
                  | Illegal
                  deriving (Eq, Show)

filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter f (x:xs)
  | f x       = x : (filter f xs)
  | otherwise = filter f xs
```

```haskell
describeList :: [a] -> String
describeList xs = "The list is " ++ case xs of [] -> "empty."
                                               [x] -> "a singleton list."
                                               xs -> "a longer list."
```

```haskell
nub :: (Eq a) => [a] -> [a]
nub [] = []
nub (x:xs) = x : nub (filter (\y -> y /= x) xs)
```

```haskell
quickSort :: (Ord a) => [a] -> [a]
quickSort [] = []
quickSort (x:xs)  = quickSort smaller ++ [x] ++ quickSort larger
  where smaller = filter (\y -> y <= x) xs
        larger  = filter (\y -> y > x) xs
```

```haskell
scanl :: (a -> b -> a) -> a -> [b] -> [a]
scanl f q (x:xs)      = if xs == []
    then [q]
    else q : (scanl f (f q x) xs)
```

```haskell
foldl f z []     = z
foldl f z (x:xs) = let z' = f z x
                   in foldl f z' xs
```

LLP Laboratório de Linguagens de Programação

Fault Localization

Example 1 - Mid

```
1.module Main where
2.  mid x y z = if y < z
3.    then if x < y
4.          then y
5.          else if x < z
6.                then y -- BUG
7.                else z
8.    else if x > y
9.          then y
10.         else if x > z
11.               then x
12.               else z
```

```
mid 3 3 5 = 3
mid 1 2 3 = 2
mid 3 2 1 = 2
mid 5 5 5 = 5
mid 5 3 4 = 4
mid 2 1 3 = 1
```

Test cases/Lines	1	2	3	4	5	6	7	8	9	10	11	12	P/F
3 3 5		●	●		●	●							P
1 2 3		●	●	●									P
3 2 1		●						●	●				P
5 5 5		●						●		●		●	P
5 3 4		●	●		●		●						P
2 1 3		●	●		●	●							F

LLP Laboratório de Linguagens de Programação

Methods

- **Tarantula**: entities that are primarily executed by failed test cases are more likely to be faulty than those primarily executed by passed test cases

- **Ochiai**: coefficient known from the biology domain, it is more sensitive to potential fault locations in failed runs than to activity in passed runs

$$Tarantula(s) = \frac{\frac{failed(s)}{totalfailed}}{\frac{failed(s)}{totalfailed} + \frac{passed(s)}{totalpassed}} \qquad Ochiai(s) = \frac{failed(s)}{\sqrt{totalfailed(failed(s) + passed(s))}}$$

LLP Laboratório de Linguagens de Programação

Example 1 - Mid

Test cases/Lines	1	2	3	4	5	6	7	8	9	10	11	12	P/F
3 3 5		●	●		●	●							P
1 2 3		●	●	●									P
3 2 1		●						●	●				P
5 5 5		●						●		●		●	P
5 3 4		●	●		●		●						P
2 1 3		●	●		●	●							F
Tarantula	0.00	0.50	0.63	0.00	0.71	**0.83**	0.00	0.00	0.00	0.00	0.00	0.00	
Ochiai	0.00	0.41	0.5	0.00	0.58	**0.71**	0.00	0.00	0.00	0.00	0.00	0.00	

```
1.module Main where
2.  mid x y z = if y < z
3.    then if x < y
4.          then y
5.          else if x < z
6.                then y -- BUG
7.                else z
8.    else if x > y
9.          then y
10.         else if x > z
11.               then x
12.               else z
```

$totalfailed = 1$ $\quad$ $totalpassed = 5$

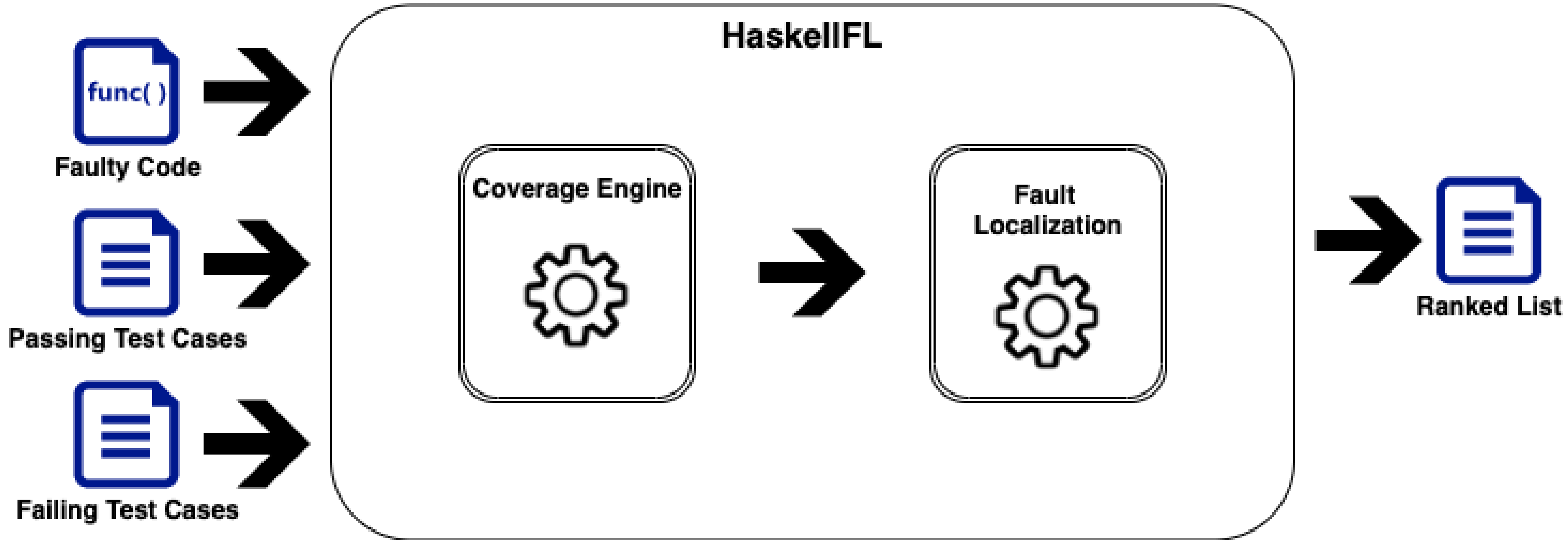
$failed(6) = 1$ $\quad$ $passed(6) = 1$

$$Ochiai(6) = \frac{failed(6)}{\sqrt{totalfailed(failed(6) + passed(6))}} = \frac{1}{\sqrt{2}} \approx 0.71$$

$$Tarantula(6) = \frac{\frac{failed(6)}{totalfailed}}{\frac{failed(6)}{totalfailed} + \frac{passed(6)}{totalpassed}} = \frac{1}{1 + \frac{1}{5}} = \frac{5}{6} \approx 0.83$$

HaskellFL

HaskellFL Architecture



HaskellFL Output

```haskell
1.  module Main () where
2.
3.      dropWhileClone p [] = []
4.      dropWhileClone p (x:xs)
5.          | p x       = dropWhileClone p xs
6.          | otherwise = x:xs
7.
8.      isSpace s = if s == "" -- BUG
9.          then True
10.         else False
11.
```

```
"## HaskellFL ##"
"Tarantula: "
Line = 9, Score = 1.0
Line = 5, Score = 1.0
Line = 8, Score = 0.556
Line = 10, Score = 0.5
Line = 6, Score = 0.5
Line = 3, Score = 0.5
```

```
"## HaskellFL ##"
"Ochiai: "
Line = 8, Score = 0.745
Line = 5, Score = 0.745
Line = 3, Score = 0.707
Line = 10, Score = 0.632
Line = 9, Score = 0.632
Line = 6, Score = 0.632
```

Ranked List

Demo

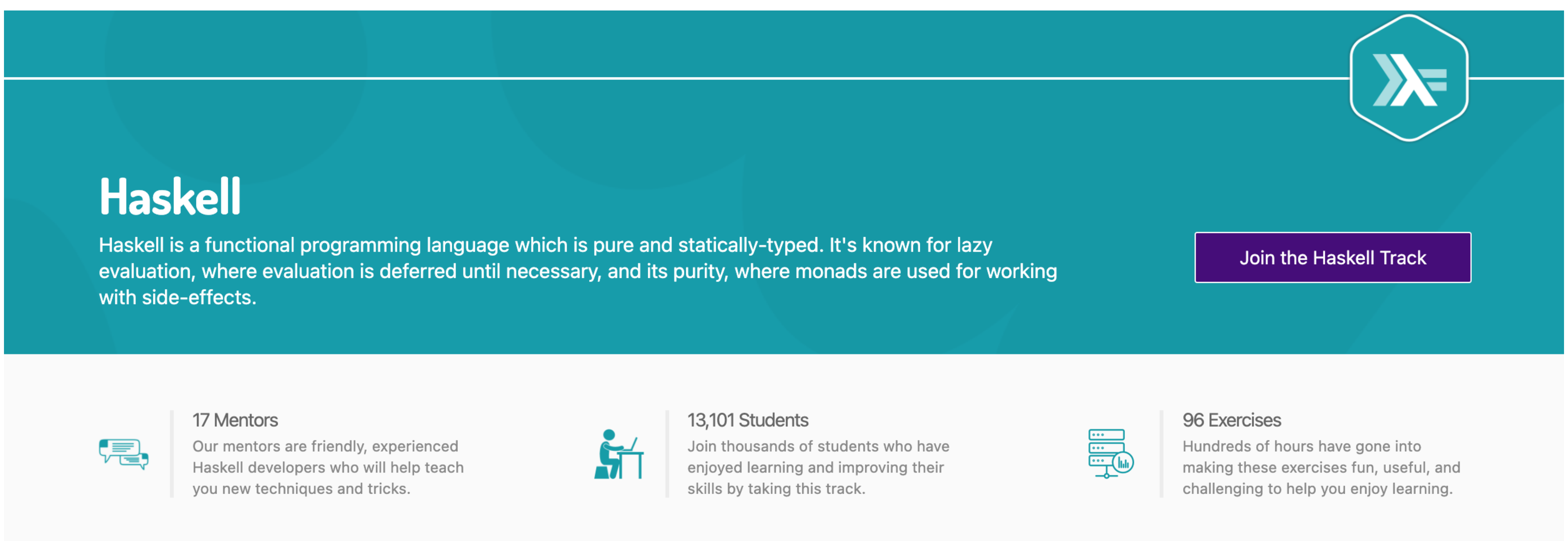
Demo


vanessa@Vanessas-MacBook-Pro:~/Downloads/HaskellFL
vanessa@Vanessas-MacBook-Pro
~/Downloads/HaskellFL
main ● ?
2023
18:58:58

Test Suite

Test Suite

- 24 problems
- Submissions from students in the Functional Programming class at UFMG
- Two versions of mid function
- Submissions for Exercism’s Haskell track available on GitHub



LLP Laboratório de Linguagens de Programação

Test Suite

Program	#Tests	Ranking	
		Tarantula	Ochiai
mid (Version 1)	6	5	2
mid (Version 2)	6	1	1
dropWhileClone	10	3	1
dropWhile	9	1	1
break (Version 1)	5	1	1
break (Version 2)	8	1	1
toTuples	10	1	1
remdupsReducer	7	1	1
joinr	12	1	1
separateTuplesByType	7	1	1
flip	5	1	1
unzip	3	1	1
maxSumLength	11	1	1
binary-search-tree	8	2	2
grade-school	7	1	1
luhn	6	2	2
raindrops	8	1	1
resistor-color-duo	7	1	1
robot-simulator	9	1	1
roman-numerals	8	1	1
simple-linked-list	6	1	1
space-age	7	1	1
sum-of-multiples	7	3	1
triangle	8	6	5

Results

Results - EXAM Score

- Indicates the percentage of program elements that a developer would have to inspect until finding the bug

```haskell
1.  module Main () where
2.
3.      dropWhileClone p [] = []
4.      dropWhileClone p (x:xs)
5.         | p x       = dropWhileClone p xs
6.         | otherwise = x:xs
7.
8.      isSpace s = if s == "" -- BUG
9.          then True
10.         else False
11.
```

```
"## HaskellFL ##"
"Ochiai: "
Line = 8, Score = 0.745
Line = 5, Score = 0.745
Line = 3, Score = 0.707
Line = 10, Score = 0.632
Line = 9, Score = 0.632
Line = 6, Score = 0.632
```

$$OchaiBest = \frac{1}{10} = 10\%$$

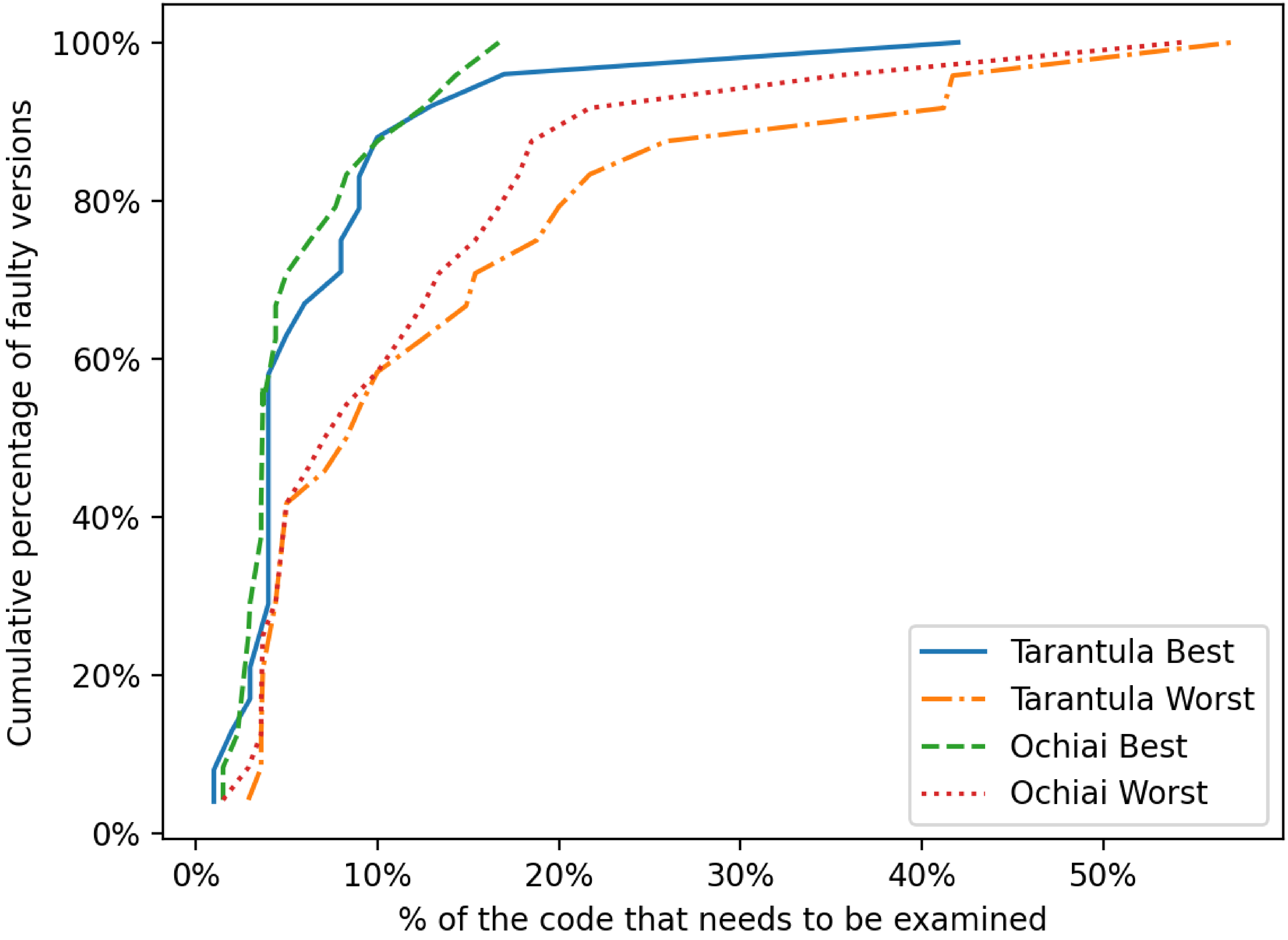
$$OchaiWorst = \frac{2}{10} = 20\%$$

Results - EXAM Score

- Indicates the percentage of program elements that a developer would have to inspect until finding the bug

EXAM Score	Tarantula Best	Tarantula Worst	Ochiai Best	Ochiai Worst
(0-4.9)%	58.33%	33.33%	66.67%	33.33%
(5-9.9)%	25.00%	20.83%	16.67%	20.83%
(10-14.9)%	8.33%	12.50%	12.50%	16.67%
(15-19.9)%	4.17%	8.33%	4.17%	16.67%
(20-24.9)%	0.00%	8.33%	0.00%	4.17%
(25-29.9)%	0.00%	4.17%	0.00%	0.00%
(30-34.9)%	0.00%	0.00%	0.00%	0.00%
(35-39.9)%	0.00%	0.00%	0.00%	4.17%
(40-44.9)%	4.17%	8.33%	0.00%	0.00%
(45-49.9)%	0.00%	0.00%	0.00%	0.00%
(50-54.9)%	0.00%	0.00%	0.00%	4.17%
(55-59.9)%	0.00%	4.17%	0.00%	0.00%

Results


100%
80%
60%
40%
20%
0%
Cumulative percentage of faulty versions
0%
10%
20%
30%
40%
50%
% of the code that needs to be examined
Tarantula Best
Tarantula Worst
Ochiai Best
Ochiai Worst

Conclusion

Contributions

- We created an interpreter for a Haskell grammar subset
- HaskellFL tool and our test suite are available as an open source project at https://github.com/VanessaCristiny/HaskellFL
- HaskellFL located the errors using Tarantula and Ochiai methods examining very few lines for the majority of our test suite
- Our results showed that Ochiai presented better results than Tarantula

Future Work

- Extend the grammar to include do notation and list comprehensions
- Implement mutation-based fault localization algorithms
- Actually repair the code

Q&A