Delta Debugging Type Errors with a Blackbox Compiler

Joanna Sharrad
University of Kent, UK
jks31@kent.ac.uk

Olaf Chitil
University of Kent, UK
oc@kent.ac.uk

Meng Wang
University of Bristol, UK
meng.wang@bristol.ac.uk

Example

Insert an element into an ordered list:

```
1.   insert x [] = [x]
2.   insert x (y:ys) | x > y = y : insert x ys
3.                  | otherwise = x : y : ys
```

Example from : Stuckey, P., Sulzmann, M., Wazny, J. 2004. Improving type error diagnosis.

Example

This code has a type error.

```
1.   insert x [] =  x
2.   insert x (y:ys) | x > y = y : insert x ys
3.                  | otherwise = x : y : ys
```

Example from : Stuckey, P., Sulzmann, M., Wazny, J. 2004. Improving type error diagnosis.

Example

GHC 8.4.3 type error message:

```
Insert.hs:2:27: error:
    • Occurs check: cannot construct the infinite type: a ~ [a]
    • In the expression: y : Main.insert x ys
    In an equation for ‘Main.insert’:
        Main.insert x (y : ys)
        | x > y = y : Main.insert x ys
        | otherwise = x : y : ys
    • Relevant bindings include
    ys :: [a] (bound at Insert.hs:2:13)
    y :: a (bound at Insert.hs:2:11)
    x :: a (bound at Insert.hs:2:8)
    insert :: a -> [a] -> [a] (bound at Insert.hs:1:1)
```

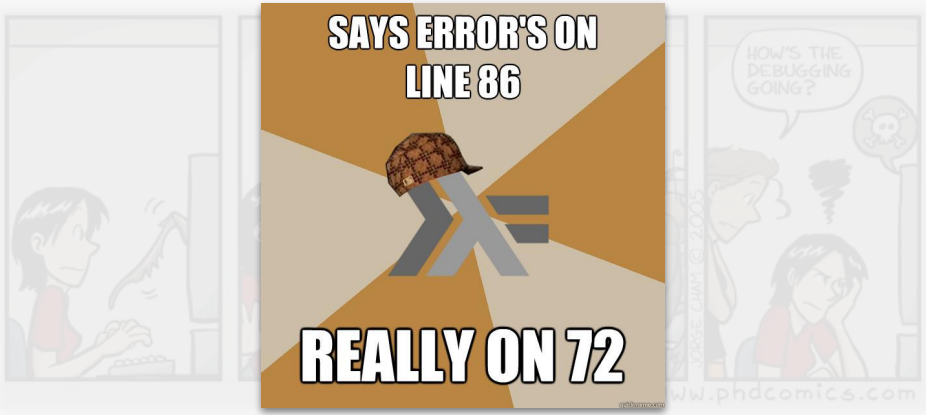
What is the issue?

We all experience type errors, causing hours of frustration…

What is the issue?

We all experience type errors, causing hours of frustration…

…made worse by the compiler reporting the wrong line...



An illustration of our method

Two programs to work with:

```
1.
2.
3.
```

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.                 | otherwise = x : y : ys
```

An illustration of our method

Add lines to one and remove from the other:

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1.
2.
3.                    | otherwise = x : y : ys
```

An illustration of our method

Type-check the programs:

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1.
2.
3.                    | otherwise = x : y : ys
```

FAIL

An illustration of our method

Type-check the programs:

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

FAIL

```
1.
2.
3.                    | otherwise = x : y : ys
```

UNRESOLVED

An illustration of our method

New base programs to modify:

```
1.
2.
3.
```

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

An illustration of our method

Add and remove line two:

```
1.
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1. insert x [] = x
2.
3.
```

An illustration of our method

Type-check the programs:

```
1.
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1. insert x [] = x
2.
3.
```

PASS

An illustration of our method

Type-check the programs:

```
1.
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

PASS

```
1. insert x [] = x
2.
3.
```

PASS

An illustration of our method

Our new base line programs:

```
1.
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

An illustration of our method

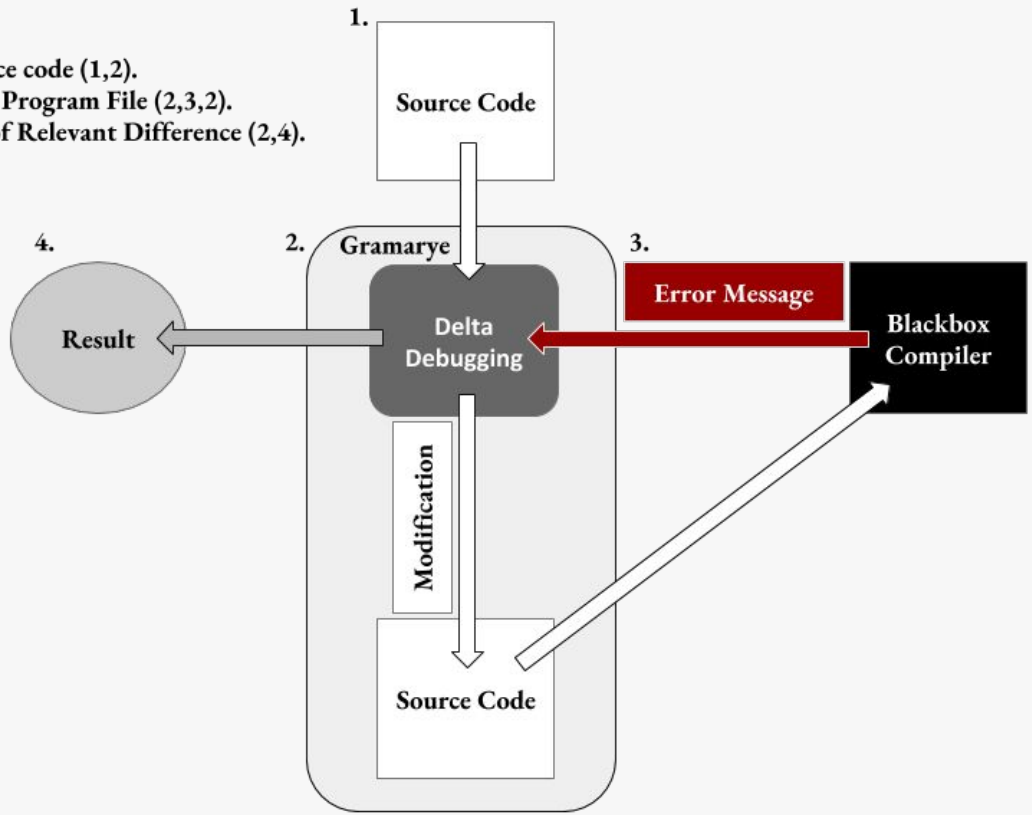
Algorithm terminates:

```
1.
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

```
1. insert x [] = x
2. insert x (y:ys) | x > y = y : insert x ys
3.
```

Relevant difference is line **1**

Input raw source code (1,2).
Recursion over Program File (2,3,2).
Output result of Relevant Difference (2,4).



Isolating Delta Debugging

Andreas Zeller. 2009. Why Programs Fail.

Automates scientific problem solving

- Iterates over two programs - Modifying and Testing
- Binary Chop to provide what to modify
- Isolates the difference between the two

A Black Box Compiler

Accessing standard output from the compiler

- No modification to the existing compiler
- Method allows option of extending to other compilers
- Used only as a type checker

Manipulating Source Code

Directly changing the source of the program

- Adding and Removing whole lines
- No need for our own parser
- Keeps programmers' layout intact

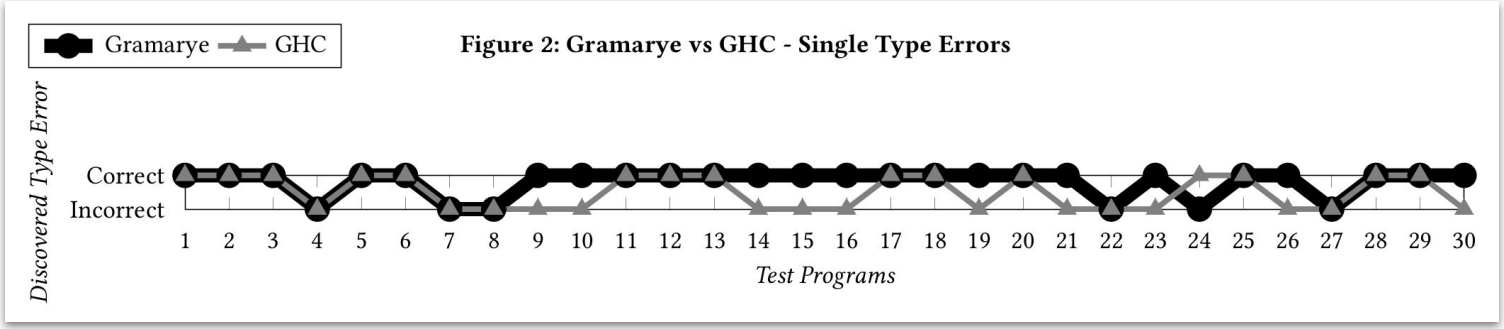
Evaluation

- The inaccurate reporting of the line number of a type error
- Compared the method to the outcome of GHC
- 30 programs with single errors, 870 with multiple errors
  - Collected by Chen and Erwig from previous papers
  - Contain an Oracle - we know the line of the error

Locating Single Type Errors

Show improvement in locating single errors?

- If line number matched oracle, it was accurate
- **80%** success rate vs GHC at **50%**



Locating Multiple Type Errors

Show improvement in locating multiple errors?

- Two non-connected functions containing type errors
- **68%** success rate vs GHC at **71%**

```
addList ls s = if s `elem` ls then ls else s : ls
v5 = addList "a" ["b"]

sumLists = sum2 . map sum2
sum2 [] = []
sum2 (x:xs) = x + sum2 xs
```

Precisely Locating Type Errors

Return a smaller set of locations?

- **53%** accurate on single type errors vs GHC at **40%**
- **43%** accurate on multiple type errors vs GHC at **15%**

Program	Gramarye	GHC	Program	Gramarye	GHC	Program	Gramarye	GHC
1	✓	✓	11	×	✓	21	✓	×
2	×	✓	12	✓	✓	22	×	×
3	✓	×	13	✓	✓	23	✓	×
4	×	×	14	×	×	24	×	×
5	×	✓	15	×	×	25	✓	✓
6	×	✓	16	✓	×	26	✓	×
7	×	×	17	✓	×	27	×	×
8	×	×	18	✓	✓	28	✓	✓
9	✓	×	19	×	×	29	×	✓
10	✓	×	20	✓	✓	30	×	✓

Future Work

- Investigate the outlier for Single Type Errors
- Evaluate the non-determinism of the algorithm’s choices
- Move to larger examples

Thank You

- Type error debugging tool that employs: Delta Debugging, Blackbox compiler, and direct Source Code manipulation.
  - **80%** successful in locating single type errors
  - **68%** successful in locating multiple type errors
  - **53%** and **43%** precise accuracy locating type errors

https://github.com/JoannaSharrad/gramarye