Locating Type Errors *Speedily*

with

Delta Debugging

Joanna Sharrad

University of Kent

jks31@kent.ac.uk

Olaf Chitil

University of Kent

oc@kent.ac.uk
Example

Ill-typed case statement:

```haskell
1| f x = case x of
2|   0 -> [0]
3|   1 -> 1
```

Non type-variable argument in the constraint: Num [a2]

Example from: Chen and Erwig. 2014. Counter-factual typing for debugging type errors.
Input raw source code (1,2). Recursion over Program File (2,3,2). Output result of Relevant Difference (2,4).

Result

2. Gramarye

3. Error Message

1. Source Code

Delta Debugging

4. Source Code

Modification

Blackbox Compiler

Example

This code has a type error.

```
1| f x = case x of
2|    0 -> [0]
3|    1 -> 1
```

Example from: Chen and Erwig. 2014. Counter-factual typing for debugging type errors.
Example

Applying Delta Debugging:

```haskell
f x = case x of
  0 -> [0]
  1 -> 1
```
Example

Applying Delta Debugging:

1| f x = case x of
2|     0 -> [0]
3|

PASS (Compiles)
Application of the Moiety Algorithm

Pre-processing to avoids line-splits causing unresolveds

```
1 | f x = case x of
2 |    0 -> [0]
3 |    1 -> 1
4 | plus :: Int -> Int -> Int
5 | plus = (+)
6 | fib x = case x of
7 |    0 -> f x
8 |    1 -> f x
9 |    n -> fib (n-1) `plus` fib (n-2)
```
## Application of the Moiety Algorithm

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 -&gt; [0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>parse error on input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application of the Moiety Algorithm

1 -> 1

parse error on input
Application of the Moiety Algorithm

```
1| 2| 3| 4| plus :: Int -> Int -> Int
5| 6| 7| 8| 9|
```

not parse error on input

\((3,4)\)
Application of the Moiety Algorithm

1| 2| 3| 4| 5| 6| 7| 8| 9 |

plus = (+)  

not parse error on input  

(3,4) (4,5)
Application of the Moiety Algorithm

```
1|  
2|  
3|  
4|  
5|  
6| fib x = case x of  
7|  
8|  
9| not parse error on input  
  (3,4) (4,5) (5,6)
```
Application of the Moiety Algorithm

```
0 -> f x
```

Parse error on input

\[(3,4) (4,5) (5,6)\]
Application of the Moiety Algorithm

Final Moieties (splitting points):

(3,4) (4,5) (5,6)

1, f x = case x of
2,   0 -> [0]
3,   1 -> 1
4, plus :: Int -> Int -> Int
5, plus = (+)
6, fib x = case x of
7,   0 -> f x
8,   1 -> f x
9,   n -> fib (n-1) `plus` fib (n-2)
Input raw source code (1,2).
Pre-check to remove 'Parse Errors on Input' (2).
Recursion over Configuration (3,4,3).
Output result of Relevant Difference (3,5).
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

1| \( f \ x = \text{case } x \text{ of} \)
2| \( \quad 0 \rightarrow [0] \)
3| \( \quad 1 \rightarrow 1 \)
4| \( \text{plus} :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \)
5| \( \text{plus} = (+) \)
6| \( \text{fib} \ x = \text{case } x \text{ of} \)
7| \( \quad 0 \rightarrow f \ x \)
8| \( \quad 1 \rightarrow f \ x \)
9| \( \quad n \rightarrow \text{fib} \ (n-1) \ `\ plus` \ \text{fib} \ (n-2) \)
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

FAIL (Type Error)

UNRESOLVED

```
1| f x = case x of
2|   0 -> [0]
3|   1 -> 1
4| plus :: Int -> Int -> Int
5| plus = (+)
6|
7|
8|
9|

1| fib x = case x of
2|   0 -> f x
3|   1 -> f x
4|   n -> fib (n-1) `plus` fib (n-2)
```
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

1 2 3 4 plus :: Int -> Int -> Int
2 3 4 plus = (+)
5 6 7 8 9

FAIL (Type Error) PASS (Compiles)
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

1| \( f \ x = \text{case } x \text{ of} \\
2| \quad 0 \rightarrow [0] \\
3| 4| 5| 6| 7| 8| 9| 1| 2| 3| 4| 5| 6| 7| 8| 9|

PASS (Compiles)   Parse Error on Input
# Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

```haskell
1| f x = case x of
2|   0 -> [0]
3|
4|
5|
6|
7|
8|
9|
```

```
1|
2|
3|   1 -> 1
4| plus :: Int -> Int -> Int
5| plus = (+)
6|
7|
8|
9|
```

PASS (Compiles)  
Parse Error on Input
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

Configuration Splitting Points:

{{1},{2},{3},{4},{5}}

1|
2|
3| 1 -> 1
4| plus :: Int -> Int -> Int
5| plus = (+)
6|
7|
8|
9|

Parse Error on Input
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

Configuration Splitting Points:

{[1],[2],[3],[4],[5]}
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

Configuration Splitting Points:

{[1],[2,3],[4],[5]}

1| 0 -> [0]
2| 1 -> 1
3| plus :: Int -> Int -> Int
4| plus = (+)
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

Configuration Splitting Points:

\{[1],[2,3],[4],[5]\}

\[\]

```
1  f x = case x of
2   0 -> [0]
3   1 -> 1
4  plus :: Int -> Int -> Int
5  plus = (+)
```

```
## Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

### Configuration Splitting Points:

\{[1,2,3],[4],[5]\}

```haskell
f x = case x of
  0 -> [0]
  1 -> 1

plus :: Int -> Int -> Int
plus = (+)
```

FAIL (Type Error)
Applying the Good-Omens Algorithm

On request ‘parse error on input’ removal

```
1| f x = case x of
2|   0 -> [0]
3|   1 -> 1
4| 1| plus :: Int -> Int -> Int
5| 2| plus = (+)
6| 3|
7| 4| 5| 6| 7| 8| 9|
FAIL (Type Error) PASS (Compiles)
```
Evaluation

- Evaluated with a scalability data-set based on Pandoc
  - 80 type errors, 2 placed in each of 40 chosen modules
  - Modules have between 32 to 2305 lines of code
- Comparison with our moiety debugger Elucidate
- Can we reduce the debugger run-time?
Reduce the run-time of the
Testing the Agnostic ability of the debugger

- 11 test programs in both Haskell and Ocaml
- Blackboxes: GHC and Ocamlc

Promising initial results show debugger is agnostic

Correctly located 8 identical type errors in both languages
Future Work

- Heuristics to choose pre-processing or request
- An empirical study with real programmers
- More evaluation of the debuggers agonistic behavior
Thank You

- Shown an agnostic type error debugger using a Blackbox compiler, Delta Debugging, Moiety and Good-Omens Algorithms
  - Increased the average speed by **1.5 minutes**
  - Best **38 minutes** faster, worst **8 minutes** slower
  - Agnostic debugging a success needing more research