A Theory of Tracing Pure Functional Programs

Olaf Chitil
University of Kent
United Kingdom
Tracing a Computation

Aims:

- locate bugs (wrong output, abortion, non-termination)
- comprehend programs
Conventional Debugging

Techniques:

- print statements
- debuggers such as gdb

Show at a point of time in computation a part of computation state.

Properties:

- expose (abstract) machine
- erroneous value often observed long after bug
Declarative Languages

Abstract machines more complex, should be hidden from programmer.

\[\text{elem} :: \text{Int} \rightarrow \text{[Int]} \rightarrow \text{Bool}\]
\[\text{elem } x\;\text{xs} = \text{or } (\text{map } (==x)\;\text{xs})\]

\[\text{elem } 42\;\text{[1..]}\]
\[\leadsto \text{or } (\text{map } (== 42)\;\text{[1..]})\]
\[\leadsto \text{or } (\text{map } (== 42)\;\text{(1:[2..])})\]
\[\leadsto \text{or } (\text{False} : \text{map } (== 42)\;\text{[2..]})\]
\[\leadsto \text{or } (\text{map } (== 42)\;\text{[2..]})\]
\[\leadsto \ldots\]

Instead take advantage of purity: no side-effect, only result.
Algorithmic Debugging

\textbf{insert :: Ord a => a -> [a] -> [a]} \textbf{sort :: Ord a => [a] -> [a]}

\begin{align*}
\text{insert } x \; [] & = [x] \\
\text{insert } x \; (y:ys) & = \\
& \quad \text{if } x > y \text{ then } y : \text{insert } x \; ys \\
& \quad \text{else } x : ys \\
\text{sort } [] & = [] \\
\text{sort } (x:xs) & = \text{insert } x \; (\text{sort } xs)
\end{align*}

\textbf{main = print (sort "sort")}

Freja by Henrik Nilsson

\begin{align*}
\text{sort } \text{"sort"} & = \text{"os"} \; ? \; n \\
\text{insert } \text{‘s’} \; \text{"o"} & = \text{"os"} \; ? \; y \\
\text{sort } \text{"ort"} & = \text{"o"} \; ? \; n \\
\text{insert } \text{‘o’} \; \text{"r"} & = \text{"o"} \; ? \; n \\
\text{‘o’} & \leq \text{‘r’} = \text{True} \; ? \; y \\
\text{Error located:} \\
& \text{second equation of ‘insert‘, taking else branch.}
\end{align*}
The Evaluation Dependency Tree for Algorithmic Debugging

main = "os" ×

sort "sort" = "os" ×

sort "ort" = "o" ×

insert 's' "o" = "os" √

's' <= 'o' = False

insert 's' "" = "s"

sort "rt" = "r"

insert 'o' "r" = "o" ×

's' <= 'o' = False

sort "t" = "t"

insert 'r' "t" = "r" ×

'0' <= 'r' = True √

sort "" = ""

insert 't' "" = "t"

'0' <= 't' = True
Source-Based Algorithmic Debugging

==== Hat-Explore 2.00 ==== Call 2/2 =============================

1. main = {IO}
2. sort "sort" = "os"
3. sort "ort" = "o"

---- Insert.hs ---- lines 5 to 10 -----------------------------------------------

if x > y then y : insert x ys
else x : ys

sort :: [Char] -> [Char]

sort [] = []

sort (x:xs) = insert x (sort xs)

Hat by Colin Runciman, Malcolm Wallace, Olaf Chitil, ...
Observation of Expressions and Functions

Observation of function sort:

- sort "sort" = "os"
- sort "ort" = "o"
- sort "rt" = "r"
- sort "t" = "t"
- sort "" = ""

Observation of function insert:

- insert 's' "o" = "os"
- insert 's' "" = "s"
- insert 'o' "r" = "o"
- insert 'r' "t" = "r"
- insert 't' "" = "t"
Redex Trails

Output: -------------------------------------------------------------

os

Trail: ------- Insert.hs line: 10 col: 25 ---------------------------

<- putStrLn "os"
<- insert 's' "o" | if True
<- insert 'o' "r" | if False
<- insert 'r' "t" | if False
<- insert 't' []
<- sort []

Go backwards: which redex created this expression?

Original Hat by Colin Runciman and Jan Sparud
Implementations

Algorithmic Debugging: Freja, Hat, Buddha
Observations: Hood, Hugs-Hood, GHood, Hat
Redex Trails: Hat

- Two phases: trace generation + trace viewing
- Trace liberates from time arrow of computation

Architecture of Hat:
Challenges

Problems:

- (In)correctness of Algorithmic Debugging
- What is tracing? Systems disagree
- Tracing of all language features
- Partial traces

Need to generalise:

- Tracing eager functional languages
- Flexible algorithmic debugging
  - $\text{factorial}(-2) = 42$ ?
- Multi-level algorithmic debugging
- Trace transformation before viewing
- Partial Traces
Summary

- Tracing techniques should take advantage of features of declarative languages.
  - Algorithmic Debugging
  - Observations
  - Redex Trails

- Implementations are currently ahead of theoretical results.