Tracing and Debugging of Lazy Functional Programs

A Comparative Evaluation of Three Systems

Olaf Chitil, Colin Runciman and Malcolm Wallace

University of York

Tracing

program $\longrightarrow$

$\longrightarrow$ output

input $\longrightarrow$

computation

Available Tracing & Debugging Systems

Freja

Henrik Nilsson

Hat

Jan Sparud, Colin Runciman, Malcolm Wallace, Olaf Chitil

Hood

Andy Gill

Freja

compiler for a subset of Haskell

```
main = show $
  let xs = [4*2,3+6]
  in (head xs, last xs)

head (x:xs) = x


last (x:xs) = last xs
last [x]    = x
```

`main` $\Rightarrow (8, \perp)$	`no`
`4*2` $\Rightarrow 8$	`yes`
`head [8,?]` $\Rightarrow 8$	`yes`
`last [8,?]` $\Rightarrow \perp$	`no`
`last [?]` $\Rightarrow \perp$	`no`
`last []` $\Rightarrow \perp$	`yes`

`Erroneous reduction:` `last [?]` $\Rightarrow \perp$

Hat

modified nhc98 + browser

```
main = print $
  let xs = [4*2,3+6] :: [Int]
  in (head xs, last xs)

head (x:xs) = x

last (x:xs) = last xs
last [x]    = x
```

```
• last []
| last (3+6:[])
| last (8:3+6:[])
| ▽ 4*2
⊥ main
```

Hood

Haskell library

```
import Observe

main = printO $
  let xs = [4*2,3+6]
  in (head xs
     ,last (observe "arg" xs))

head (x:xs) = x

last = observe "last" last'

last' (x:xs) = last' xs
last' [x]    = x
```

```
-- arg
  _ : _ : []

-- last
  { \(_ : _ : []) -> throw <Exception>
  , \(_ : []) -> throw <Exception>
  , \[] -> throw <Exception>
  }
```

Comparison in Principle: Commonalities

- phases
  1. collection of information at run-time
  2. browsing of information
- suitable for all kinds of erroneous behaviour
  - wrong output
  - abortion with error message
  - non-termination
- source-level traces
- observable program behaviour is preserved

Comparision in Principle: Differences

	basic entity	trace structure	trace describes
Freja	reduction	tree	full computation
Hat	redex	graph	full computation
Hood	value	independent observations	values of expressions

	implementation
Freja	modified abstract machine
Hat	Haskell program transformation
Hood	Haskell library

Comparision in Principle: Differences in Shown Evaluation

```
let xs = [4*2,3+6]
in (head xs, last xs)
```

Freja `last [8,?]` $\Rightarrow \perp$

Hat `last (8:3+6:[])`

Hood

```
-- arg
  _ : _ : []
```

Evaluation in Practise

Readability of Expressions

```
last [8,?] ⇒ ⊥        | last []              -- last
last [?] ⇒ ⊥          | last (3+6:[])          { \(_ : _ : []) -> throw <...>
last [] ⇒ ⊥           | last (8:3+6:[])        , \(_ : []) -> throw <...>
                                               , \[] -> throw <Exception>
                                               }
```

- expressions mostly of moderate size
- mostly only demanded values needed
- functions:
  - ▷ good names desirable for Freja and Hat
  - ▷ λ-abstractions bad in Freja and Hat

    `<lambda#4> [?] ⇒ ⊥` `(\) (3+6:[])`

Readability of Expressions: Free Variables

```
tableRead
  "y"
  (TableImp
    (newTableFunction
     where
       newIndex = "x",
       newEntry = 1,
       oldTableFunction = implTableEmpty))
=>
Just 1
```

```
tableRead "y" (TableImp newTableFunction)
▽ tableInsert "x" 1 (TableImp implTableEmpty)
```

```
-- moveval
  { \ 8 -> Draw }
  { \ 8 -> Win }
```

Locating an Error

With each system we located all errors

	largest example	Guidance & Strategies	Useability
Freja	10-30 questions	Freja asks the questions!	principle easy
Hat	0-6 parents	good starting-point	steep learning curve
Hood	0-3 annotations	absolute freedome	similarity to print, trace

Locating an Error: Wrong Subexpressions

```
translateStatement
  (TableImp
    (newTableFunction
     where
       newIndex = "y",
       newEntry = 2,
       oldTableFunction = newTableFunction
                          where
                            newIndex = "x",
                            newEntry = 1,
                            oldTableFunction = implTableEmpty))
  ?
  (Assignment "x" (Compound (Variable "x") Minus (Variable "y")))
=>
_Tuple_2 [Lod 1,Lod 1,Sb,Sto 1] 4
```

Locating an Error

- used information:
  - ▷ use the source, Luke!
  - ▷ Hood observations are independent; observed mostly functions
  - ▷ unexpected sharing exposed only by Hat
- reduction of information through trusting at run-time desirable
- run-time: Freja wins
- good compile-time warnings and run-time error messages make some tracing unnecessary

Language Constructs

```
primes = sieve [2..]

main = print primes
```

- `primes` is not a redex
- Freja asks about `primes` first
- Hat has no parent for `primes`

Hat has additional redexes for guards, `case` and `if`.

Modification of the Program

```
instance Observable Statement where

  observer (Assignment var expr) =
    send "Assignment" (return Assignment << var << expr)
  observer (If test seq1 seq2) =
    send "If" (return If << test << seq1 << seq2)
  observer (While test seq) =
    send "While" (return While << test << seq)
```

In Hood

- considerable modifications necessary
- modifications may introduce new errors

\+ observations may be left in program

Conclusions and Future Work

Freja Hat Hood

- determine strategies for locating errors
- major current limitations:
  - stylised abstractions
  - input / output
- more experiments needed
  are introduced errors typical?
- usefulness for other purposes than debugging?