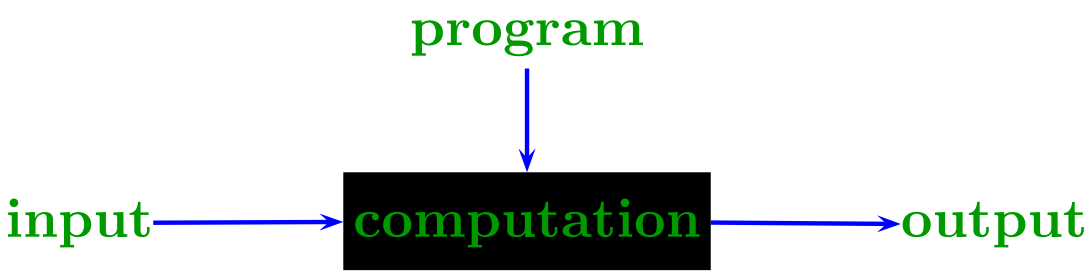
# A Theory of Tracing Pure Functional Programs

**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.**

```
elem :: Int -> [Int] -> Bool
elem x xs = or (map (==x) xs)
```

```
elem 42 [1..]
 ⇝ or (map (== 42) [1..])
 ⇝ or (map (== 42) (1:[2..]))
 ⇝ or (False : map (== 42) [2..])
 ⇝ or (map (== 42) [2..])
 ⇝ ...
```

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

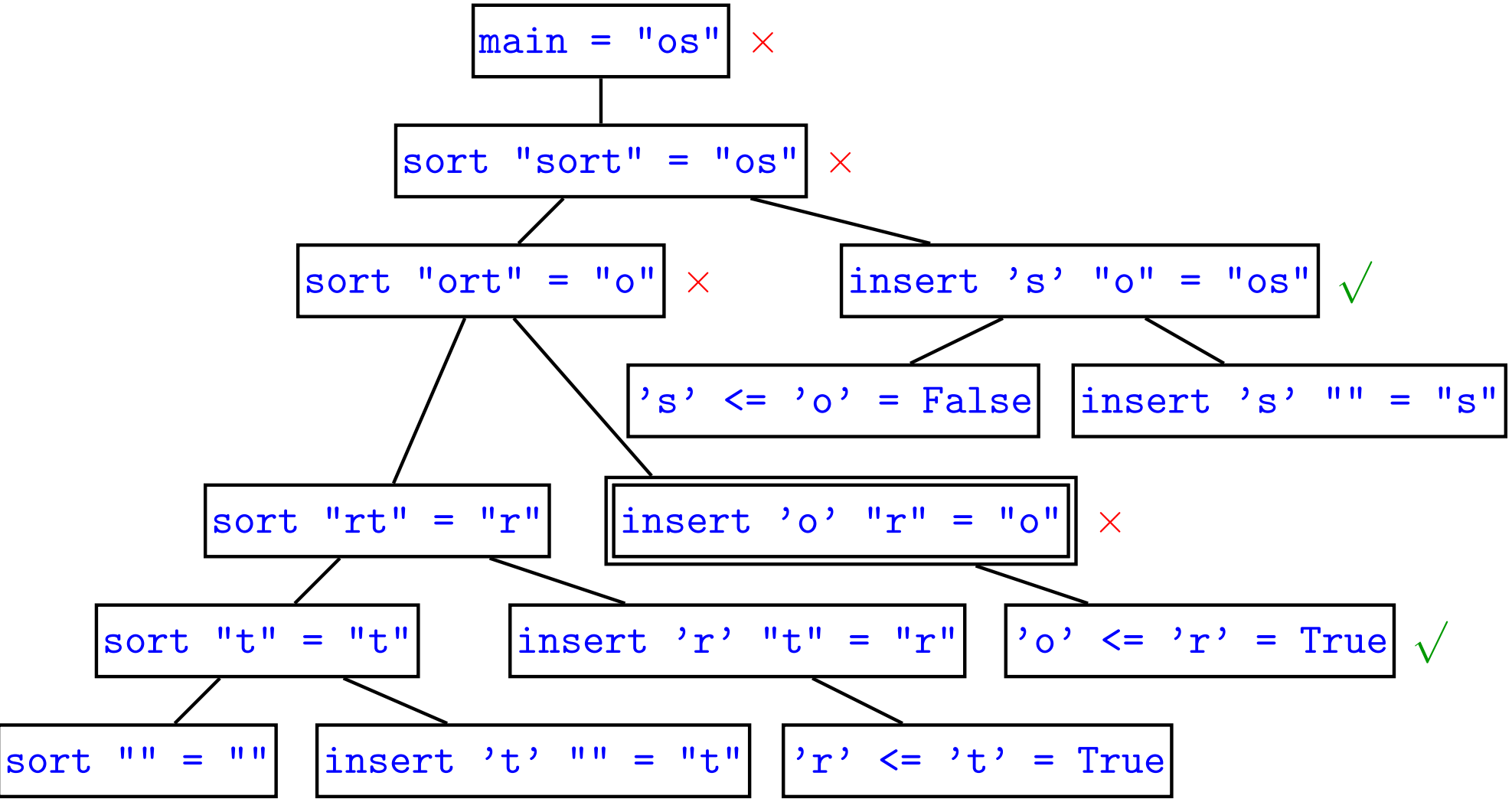
# Algorithmic Debugging

```
insert :: Ord a => a -> [a] -> [a]

insert x [] = [x]
insert x (y:ys) =
  if x > y then y : insert x ys
           else x : ys

sort :: Ord a => [a] -> [a]
sort [] = []
sort (x:xs) = insert x (sort xs)

main = print (sort "sort")
```

**Freja by Henrik Nilsson**

```
sort "sort" = "os" ?  n

insert 's' "o" = "os" ?  y

sort "ort" = "o" ?  n

insert 'o' "r" = "o" ? n

'o' <= 'r' = True ?  y

Error located:
  second equation of `insert`,
  taking else branch.
```

# The Evaluation Dependency Tree for Algorithmic Debugging



# 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"
```

**Hood by Andy Gill**

# Redex Trails

```
Output: ----------------------------------------------------
os\n

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?**

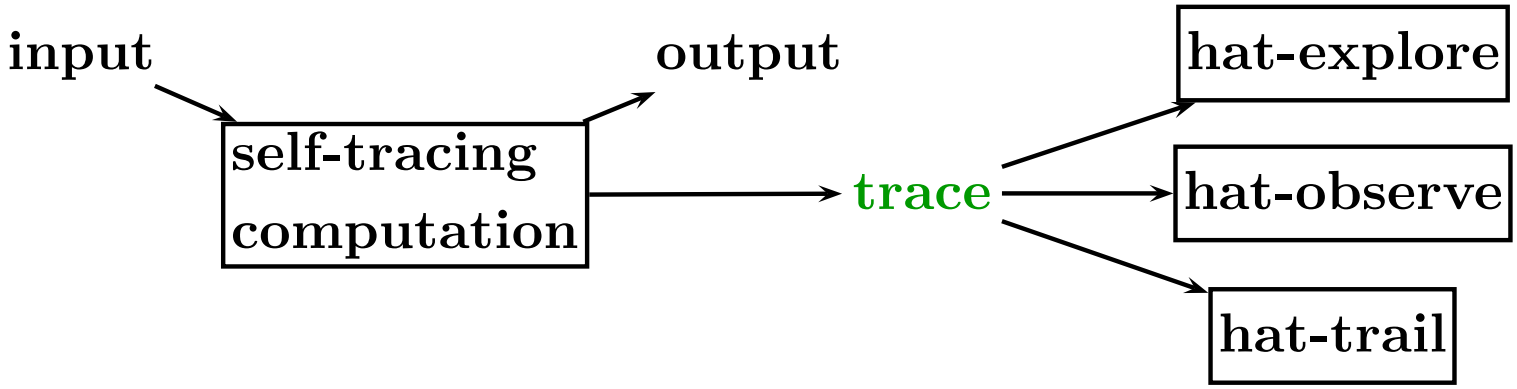
**Original Hat by Colin Runciman and Jan Sparud**

# 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**
  - `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.**