Proving the Correctness of Algorithmic Debugging for Functional Programs

Yong Luo and Olaf Chitil

University of Kent, UK

19th April 2006

Aims and Outline

Aims

- Model the Haskell tracer Hat
- Provide theoretical foundation
- Guide implementation

Outline

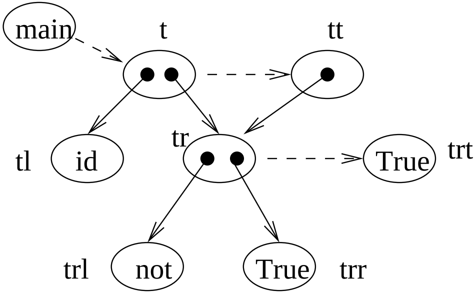
- Augmented Redex Trail (ART)
- Evaluation dependency Tree (EDT)
- Correctness of Algorithmic Debugging
- Proofs
- Future work

An example

not True = True
not False = True

id x = x

main = id (not True)



Language

- Terms

$$\begin{array}{rcl} M & = & x \\ & \mid & c \\ & \mid & f\ x \end{array}$$

- Patterns

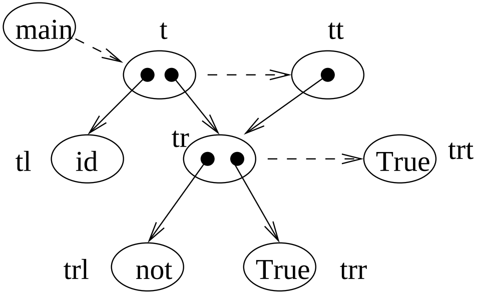
$$\begin{array}{rcl} P & = & x \\ & \mid & c p_1 \ldots p_n \end{array}$$

  where the arity of c is n

- Rewriting rules

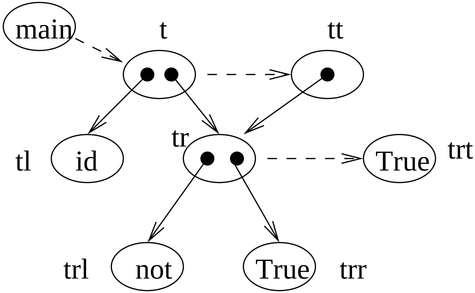
$$f p_1 \ldots p_n = M$$

Formalising ART (1)



- An ART is a graph
- Starts from "main"
- General function *graph* to add new graphs
- Sharing

Formalising ART (2)

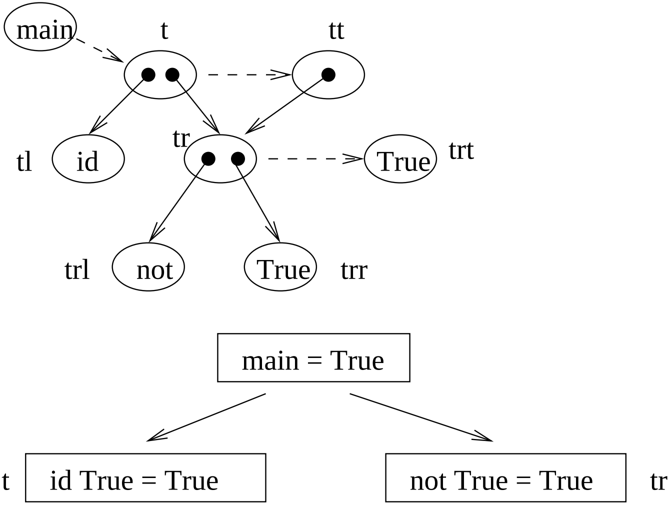


- Independence from evaluation order
- Node naming Scheme
  - not distinguish isomorphic graphs
  - given parent node implicitly

EDT

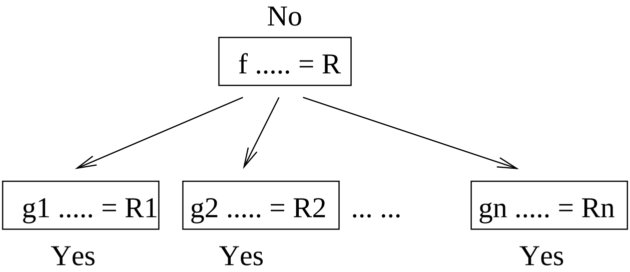
An EDT is generated from an ART

Example



Correctness of Algorithmic debugging

Faulty nodes



Correctness

- If the equation of a faulty node is $fa_1 \ldots a_n = R$, then the definition of the function f in the program is faulty

Proofs

The difficulties

- suitable reduction principle
- more general induction hypothesis

For a faulty node m, $fa_1 ... a_n \not\simeq_I R$. We define $\mathit{reduct}(mt)$ and $\mathit{mef}(mt) = R$.
We are going to prove $fa_1 ... a_n \rightarrow_P \mathit{reduct}(mt) \simeq_I \mathit{mef}(mt)$.
In order to prove $\mathit{reduct}(mt) \simeq_I \mathit{mef}(mt)$, we prove a more general result $\mathit{reduct}(n) \simeq_I \mathit{mef}(n)$ for all $n \in G$.

Future work

- Replace the unevaluated parts
- Consider different reduction strategies and add error messages to an ART when there is a pattern matching failure
- Add local rewriting rules
- Add rewriting rules for constants