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

\[ M = \begin{align*} x \mid c \mid f x \end{align*} \]

- Patterns

\[ P = \begin{align*} x \mid cp_1...p_n \end{align*} \]

where the arity of \( c \) is \( n \)

- Rewriting rules

\[ fp_1...p_n = M \]
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
An EDT is generated from an ART

Example

main = True

id True = True
not True = True

tr

trl

trt

trr

tr
Correctness of Algorithmic debugging

Faulty nodes

No

\[ f \ldots = R \]

\[ g_1 \ldots = R_1 \]
\[ g_2 \ldots = R_2 \]
\[ \ldots \]
\[ g_n \ldots = R_n \]

Yes Yes Yes

No

Correctness

- If the equation of a faulty node is \( f a_1 \ldots a_n = R \), then the definition of the function \( f \) in the program is faulty
The difficulties

- suitable reduction principle
- more general induction hypothesis

For a faulty node $m$, $fa_1...a_n \not\sim_I R$. We define $\text{reduct}(mt)$ and $\text{mef}(mt) = R$. We are going to prove $fa_1...a_n \rightarrow_P \text{reduct}(mt) \sim_I \text{mef}(mt)$.

In order to prove $\text{reduct}(mt) \sim_I \text{mef}(mt)$, we prove a more general result $\text{reduct}(n) \sim_I \text{mef}(n)$ 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