Comprehending Finite Maps for Algorithmic Debugging of Higher-Order Functional Programs

Olaf Chitil and Thomas Davie

University of Kent, UK

16th July 2008
Algorithmic Debugging: Faulty Equation in Tree

\[
\text{main} = "os" \times
\]

\[
\text{foldr} \ \text{insert} \ [] \ "sort" = "os"
\]

\[
\text{foldr} \ \text{insert} \ [] \ "ort" = "o"
\]

\[
\text{foldr} \ \text{insert} \ [] \ "rt" = "r"
\]

\[
\text{foldr} \ \text{insert} \ [] \ "t" = "t"
\]

\[
\text{foldr} \ \text{insert} \ [] \ "" = ""
\]

\[
\text{sort} = \text{foldr} \ \text{insert} \ []
\]

\[
\text{insert} \ 's' \ "o" = "os"
\]

\[
\text{insert} \ 'o' \ "r" = "o"
\]

\[
\text{insert} \ 'r' \ "t" = "r"
\]

\[
\text{insert} \ 't' \ "" = "t"
\]

\[
's' > 'o' = \text{True}
\]

\[
's' > 'o' = \text{True}
\]

\[
'o' > 'r' = \text{False}
\]

\[
'r' > 't' = \text{False}
\]

\[
'o' > 'r' = \text{False}
\]
main = "os" ×

foldr insert [] "sort" = "os"
sort = foldr insert [] ✓

foldr insert [] "ort" = "o"

foldr insert [] "rt" = "r"

foldr insert [] "t" = "t"

foldr insert [] "" = ""

insert 's' "o" = "os"

's' > 'o' = True

insert 's' "" = "s"

insert 'o' "r" = "o"

' appar = 'ap"ar" = False

insert 'o' "r" = "o"

'o' > 'r' = False

insert 'r' "t" = "r"

'r' > 't' = False

insert 't' "" = "t"
Algorithmic Debugging: Faulty Equation in Tree

main = "os"  

foldr insert [] "sort" = "os"  

foldr insert [] "ort" = "o"  

foldr insert [] "rt" = "r"  

foldr insert [] "t" = "t"  

foldr insert [] "" = ""  

sort = foldr insert []  

insert 's' "o" = "os"  

insert 'o' "r" = "o"  

insert 'r' "t" = "r"  

insert 't' "" = "t"  

's' > 'o' = True  

'o' > 'r' = False  

'r' > 't' = False

Olaf Chitil (Kent, UK)
Algorithmic Debugging: Faulty Equation in Tree

```
main = "os"  ×

foldr insert [] "sort" = "os"  ×

foldr insert [] "ort" = "o"  ✓

foldr insert [] "rt" = "r"

foldr insert [] "t" = "t"

foldr insert [] "" = ""

sort = foldr insert []  ✓

insert 's' "o" = "os"  ✓

's' > 'o' = True

insert 'o' "r" = "o"

insert 'r' "t" = "r"

'o' > 'r' = False

insert 't' "" = "t"

'r' > 't' = False

Olaf Chitil (Kent, UK) Debugging Higher-Order Programs 16th July 2008 3 / 24"
main = "os"  

foldr insert [] "sort" = "os"  

foldr insert [] "ort" = "o"  

foldr insert [] "rt" = "r"  

foldr insert [] "t" = "t"  

foldr insert [] "" = ""  

sort = foldr insert []  

insert 's' "o" = "os"  

insert 's' "" = "s"  

'\textbackslash s' > '\textbackslash o' = True  

insert 'o' "r" = "o"  

insert 'r' "t" = "r"  

insert 't' "" = "t"  

'\textbackslash o' > '\textbackslash r' = False  

'\textbackslash r' > '\textbackslash t' = False
Algorithmic Debugging: Faulty Equation in Tree

\[
\text{main} = "os" \times
\]

\[
\text{foldr insert [] } "sort" = "os" \times \text{sort} = \text{foldr insert []} \checkmark
\]

\[
\text{foldr insert [] } "ort" = "o" \times \text{insert 's' } "o" = "os" \checkmark
\]

\[
's' > 'o' = \text{True} \quad \text{insert 's' } "" = "s"
\]

\[
\text{foldr insert [] } "rt" = "r" \checkmark
\]

\[
\text{foldr insert [] } "t" = "t" \times 
\text{insert 'r' } "t" = "r" \times 
'o' > 'r' = \text{False}
\]

\[
\text{foldr insert [] } "" = "" \times 
\text{insert 't' } "" = "t" \times 
'r' > 't' = \text{False}
\]
Algorithmic Debugging: Faulty Equation in Tree

```
main = "os"  

foldr insert [] "sort" = "os"  ×  sort = foldr insert []  √

foldr insert [] "ort" = "o"  ×

insert 's' "o" = "os"  √

's' > 'o' = True

insert 's' "" = "s"

foldr insert [] "rt" = "r"

insert 'o' "r" = "o"  ×

foldr insert [] "t" = "t"

insert 'r' "t" = "r"  √

'o' > 'r' = False

foldr insert [] "" = ""

insert 't' "" = "t"

'r' > 't' = False

foldr insert [] "" = ""
```

Olaf Chitil (Kent, UK)
Debugging Higher-Order Programs
16th July 2008 3 / 24
Algorithmic Debugging: Faulty Equation in Tree

main = "os" ×

foldr insert [] "sort" = "os" × sort = foldr insert [] √

foldr insert [] "ort" = "o" ×

insert 's' "o" = "os" √

′s′ > ′o′ = True

insert 's' "" = "s"

foldr insert [] "rt" = "r"

foldr insert [] "t" = "t"

insert 'r' "t" = "r" ×

′o′ > ′r′ = False √

insert 't' "" = "t"

foldr insert [] "" = ""

′r′ > ′t′ = False
Fault located!

Faulty computation: \texttt{insert 'o' "r" = "o"}

Program

\begin{verbatim}
main :: String
main = sort "sort"

sort :: Ord a => [a] -> [a]
sort = foldr insert []

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f a [] = a
foldr f a (x:xs) = f x (foldr f a xs)

insert :: Ord a => a -> [a] -> [a]
insert x [] = [x]
insert x (y:ys) = if x > y then y : (insert x ys) else x:ys
\end{verbatim}
Representation of a Functional Value

As Applicative term

\[
\text{parse} \\
\quad (\text{pSucc (flip ($))} \triangleright\!	riangleleft (\text{pSucc (const 1)} \triangleright\!	riangleleft \text{pSym '1'}) \triangleright\!	riangleleft \\
\quad (\text{pSucc id} \triangleright\!\langle\!\rangle \text{pSucc combine} \triangleright\!	riangleleft \\
\quad (\text{pSucc (const 0)} \triangleright\!\langle\!\rangle \text{pSym '0'} \triangleright\!\langle\!\rangle \text{pSucc (const 0)} \triangleright\!\langle\!\rangle \text{pSym '1'}) \triangleright\!	riangleleft \\
\quad (\text{pSucc id} \triangleright\!\langle\!\rangle \text{pSucc combine} \triangleright\!	riangleleft \\
\quad (\text{pSucc (const 0)} \triangleright\!\langle\!\rangle \text{pSym '0'} \triangleright\!\langle\!\rangle \text{pSucc (const 0)} \triangleright\!\langle\!\rangle \text{pSym '1'}) \triangleright\!	riangleleft \\
\quad \text{")})\)
\]

"101"

= [4] ?

Program fragment

\[
\text{type Parser a = String -> [(a, String)]}
\]
\[
\text{parse :: Parser a -> String -> [a]}
\]
As Applicative term

parse
  (pSucc (flip ($)) <*> (pSucc (const 1) <*> pSym '1') <*> (pSucc id <|> pSucc combine <*> (pSucc (const 0) <*> pSym '0' <|> pSucc (const 0) <*> pSym '1') <*> (pSucc id <|> pSucc combine <*> (pSucc (const 0) <*> pSym '0' <|> pSucc (const 0) <*> pSym '1') <*> (pSucc id <|> pSucc combine <*> (pSucc (const 0) <*> pSym '0' <|> pSucc (const 0) <*> pSym '1'))))

"101"
= [4] ?

As Finite map

parse {"101" → [(_,"01"),(_,"1"),(4,[])]} = [4]
Function Dependency Tree: Functions as Finite Maps

```
main = "os"

sort = {"sort" -> "os"}

foldr { 's' "o" ↦ "os", 'o' "r" ↦ "o", 'r' "t" ↦ "r", 't' "" ↦ "t" } [] "sort" = "os"

foldr { 's' "o" ↦ "os", 'o' "r" ↦ "o", 'r' "t" ↦ "r", 't' "" ↦ "t" } [] "ort" = "o"

foldr { 's' "o" ↦ "os", 'o' "r" ↦ "o", 'r' "t" ↦ "r", 't' "" ↦ "t" } [] "rt" = "r"

foldr { 's' "o" ↦ "os", 'o' "r" ↦ "o", 'r' "t" ↦ "r", 't' "" ↦ "t" } [] "t" = "t"

foldr { 's' "o" ↦ "os", 'o' "r" ↦ "o", 'r' "t" ↦ "r", 't' "" ↦ "t" } [] "" = ""

insert 's' "o" = "os" insert 'o' "r" = "o" insert 'r' "t" = "r" insert 't' "" = "t"

' s' > ' o' = True insert ' s' "" = "s" ' o' > ' r' = False ' r' > ' t' = False
```

Olaf Chitil (Kent, UK) Debugging Higher-Order Programs 16th July 2008 9 / 24
Compositional Tree enables Algorithmic Debugging

```
main = "os"
foldr insert [] "sort" = "os"
sort = foldr insert []
```

Soundness of Algorithmic Debugging

If parent equation is incorrect and all child equations are correct, then program equation of parent is faulty.
Soundness of Algorithmic Debugging

If parent equation is incorrect and all child equations are correct, then program equation of parent is faulty.
An intended semantics is a binary relation $\sqsubseteq$ on terms.

1. Reflexivity:
   \[ M \sqsubseteq M \]

2. Transitivity:
   \[ M \sqsubseteq N \land N \sqsubseteq O \implies M \sqsubseteq O \]

3. Closure:
   \[ M \sqsubseteq N \implies M O \sqsubseteq N O \land O M \sqsubseteq O N \]

4. Least element:
   \[ M \sqsubseteq \{\} \]

5. Application:
   \[ \{N_1 \mapsto M_1, \ldots, N_k \mapsto M_k\} \] \( N_i \sqsubseteq M_i \)

6. Abstraction:
   \[ ON_1 \sqsubseteq M_1 \land \ldots \land ON_k \sqsubseteq M_k \implies O \sqsubseteq \{N_1 \mapsto M_1, \ldots, N_k \mapsto M_k\} \]
Another Fragment of the Tree for Finite Maps

\[
\text{sort} = \{"sort" \mapsto "os"\}
\]

**Program fragment**

\[
\text{sort} = \text{foldr} \ \text{insert} \ []
\]

**sub-proof 1**

\[
\text{insert 's' "o" } \sqsupseteq \ "os" \land \ldots \ \{\text{children}\}
\]

\[
\Rightarrow \ \{\text{abstraction property}\}
\]

\[
\text{insert } \sqsupseteq \ \{'s' "o" \mapsto "os", \ldots\}
\]

**sub-proof 2**

\[
\text{foldr} \{\ldots\} [] "sort" \sqsupseteq "os" \ {\text{child}}
\]

\[
\Rightarrow \ \{\text{abstraction property}\}
\]

\[
\text{foldr} \{\ldots\} [] \sqsupseteq \{"sort" \mapsto "os"\}
\]

\[
\sqsupseteq \ \{\text{program equation of sort}\}
\]

\[
\text{sort}
\]

\[
\sqsupseteq \ \{\text{sub-proof 1}\}
\]

\[
\text{foldr} \ \{'s' "o" \mapsto "os", \ldots\} []
\]

\[
\sqsupseteq \ \{\text{sub-proof 2}\}
\]

\[
\{"sort" \mapsto "os"\}
\]
main = "os"

foldr insert [] "sort" = "os"

sort = foldr insert []

insert 's' "o" = "os"

Program

main = sort "sort"
sort = foldr insert []
Applicative Terms: application appears in definition of parent

```
main = "os"
foldr insert [] "sort" = "os"
sort = foldr insert []
insert 's' "o" = "os"
```

Program

```
main = sort
sort = foldr insert []
```

```
main = "os"
sort = {"sort" → "os"}
foldr {...} [] "sort" = "os"
insert 's' "o" = "os"
... ... ...
```
Applicative Terms: application appears in definition of parent

```
main = "os"
foldr insert [] "sort" = "os"
sort = foldr insert []
insert 's' "o" = "os"
```

Finite Map: function symbol appears in definition of parent

```
main = "os"
sort = {"sort" ↦ "os"}
foldr {...} [] "sort" = "os"
insert 's' "o" = "os"
```
**Program fragment**

```haskell
main = sort "t"

sort = foldr insert []

foldr f a [] = a

foldr f a (x:xs) = f x (foldr f a xs)

insert x [] = [x]
```
**Program fragment**

```haskell
main = sort "t"

sort = foldr insert []

foldr f a [] = a

foldr f a (x:xs) = f x (foldr f a xs)

insert x [] = [x]
```
Program fragment

```haskell
main = sort "t"
sort = foldr insert []
foldr f a [] = a
foldr f a (x:xs) = f x (foldr f a xs)
insert x [] = [x]
```
Basis of Trees: The Trace

Program fragment

\[
\text{main} = \text{sort} \ "t" \\
\text{sort} = \text{foldr} \ \text{insert} \ [] \\
\text{foldr} \ f \ a \ [] = a \ \\
\text{foldr} \ f \ a \ (x:xs) = f \ x \ (\text{foldr} \ f \ a \ xs) \\
\text{insert} \ x \ [] = [x]
\]
**Program fragment**

main = sort "t"

sort = foldr insert []

foldr f a [] = a

foldr f a (x:xs) = f x (foldr f a xs)

insert x [] = [x]
Basis of Trees: The Trace

main

sort

foldr

insert

foldr f a [] = a
foldr f a (x:xs) = f x (foldr f a xs)
insert x [] = [x]
New nodes for right-hand-side, connected via result pointer.

Only add to graph, never remove.

Sharing ensures compact representation.
Each reduction edge gives rise to a tree node.

Tree structure based on node parent:
- applicative: parent of reduction node (application)
- finite map: parent of function symbol (left-most)

Most evaluated form of node: always follow reduction edges
- finite map: show nodes representing functions differently
Finite map for a node: find all applications of node.

\[
\text{fMap}_G(\text{node of sort}) = \{ 't' : [] \mapsto 't':[] \}
\]

\[
\text{fMap}_G(\text{node of insert}) = \{ 't' \mapsto \{ [] \mapsto 't':[] \} \} = \{ 't' \mapsto [] \mapsto 't':[] \}
\]
main = "t"

sort = {"t" ↦ "t"}

foldr { 't' [] ↦ "t"} [] "t" = "t"

insert 't' [] = "t"

foldr { 't' [] ↦ "t"} [] [] = []
Well-Definedness of Finite Maps

Self-application

main = g id
id x = x
g h = (h h) 4

fMap_I(node of id) = \{fMap_I(node of id) \mapsto \{4 \mapsto 4\}, 4 \mapsto 4\}
Well-Definedness of Finite Maps

Self-application

\[
\begin{align*}
\text{main} &= \text{g id} \\
\text{id} \, x &= x \\
g \, h &= (h \, h) \, 4
\end{align*}
\]

\[f_{\mathcal{I}}(\text{node of } \text{id}) = \{f_{\mathcal{I}}(\text{node of } \text{id}) \mapsto \{4 \mapsto 4\}, 4 \mapsto 4\}\]

\[f_{\mathcal{J}}(\text{node of } \text{id}) = \{\{4 \mapsto 4\} \mapsto \{4 \mapsto 4\}, 4 \mapsto 4\}\]
Conclusions

- A finite map is a useful alternative representation of a functional value.
- Trace provides framework for both applicative terms and finite maps.
  - formal definition
  - soundness proof
  - implementation