Testing using Dependent Types

Fredrik Lindblad

Chalmers University of Technology, Göteborg, Sweden

TYPES'06

$$reverse\ (reverse\ xs)\ =\ xs$$

$$reverse\ (reverse\ xs)\ =\ xs$$

OK

Constrained Input

for ranked binary trees

$$\mathit{empty} : \mathit{RTree}\ X$$
$$\mathit{node} : X \to \mathit{Nat} \to \mathit{RTree}\ X \to \mathit{RTree}\ X \to \mathit{RTree}\ X$$

define what it is to be heap

$$\mathit{IsHeap}\ h = \mathit{Leftist}\ h \wedge \mathit{WellRanked}\ h \wedge \mathit{HOP}\ h$$

implement element insertion

$$\mathit{insHeap} : X \to \mathit{RTree}\ X \to \mathit{RTree}\ X$$

preserves heap property

$$\mathit{IsHeap}\ h \to \mathit{IsHeap}\ (\mathit{insHeap}\ x\ h)$$

Constrained Input

for ranked binary trees

$$empty : RTree\ X$$
$$node : X \rightarrow Nat \rightarrow RTree\ X \rightarrow RTree\ X \rightarrow RTree\ X$$

define what it is to be heap

$$IsHeap\ h = Leftist\ h \wedge WellRanked\ h \wedge HOP\ h$$

implement element insertion

$$insHeap : X \rightarrow RTree\ X \rightarrow RTree\ X$$

preserves heap property

$$IsHeap\ h \rightarrow IsHeap\ (insHeap\ x\ h)$$

tricky

Constrained Input, contd.

Idea: Use a proof search tool for a dependent functional programming language to perform testing.

Constrained Input, contd.

Idea: Use a proof search tool for a dependent functional programming language to perform testing.

Heap enumeration

$$\exists h : \mathit{RTree\ Nat}.\ \ \mathit{IsHeap}\ h$$

Constrained Input, contd.

Idea: Use a proof search tool for a dependent functional programming language to perform testing.

Heap enumeration

$$\exists h : \mathit{RTree\ Nat}.\ \ \mathit{IsHeap}\ h$$

$$?_h : \mathit{RTree\ Nat}, \quad ? : \mathit{IsHeap}\ ?_h$$

Constrained Input, contd.

Idea: Use a proof search tool for a dependent functional programming language to perform testing.

Heap enumeration

$$\exists h : \mathit{RTree\ Nat}.\ \ \mathit{IsHeap}\ h$$

$$?_h : \mathit{RTree\ Nat}, \quad ? : \mathit{IsHeap}\ ?_h$$

$$(\mathit{empty},\ \ldots)$$
$$\ldots$$
$$(\mathit{node}\ 0\ (s\ 0)\ (\mathit{node}\ (s\ ?)\ (s\ 0)\ \mathit{empty}\ \mathit{empty})\ \mathit{empty},\ \ldots)$$
$$\ldots$$

Higher-Order Terms

Testing may involve witnessing a function

$$\exists f.\ \exists x.\ R\ x\ (f\ x)$$

Higher-Order Terms

Testing may involve witnessing a function

$$\exists f.\ \exists x.\ R\ x\ (f\ x)$$

$$?_f : A \to B, \quad ?_x : A, \quad ? : R\ ?_x\ (?_f\ ?_x)$$

Higher-Order Terms

Testing may involve witnessing a function

$$\exists f.\ \exists x.\ R\ x\ (f\ x)$$

$$?_f : A \to B, \quad ?_x : A, \quad ? : R\ ?_x\ (?_f\ ?_x)$$

Function synthesis

$$\exists f.\ \forall x.\ R\ f\ (f\ x)$$

$$?_f : A \to B, \quad ? : (x : A) \to R\ x\ (?_f\ x)$$

Function Synthesis

Instead of co-evolving function and proof for the specification:

- Given $\exists f.\ \forall x.\ R\ x\ (f\ x)$
- Produce candidate functions, f_i
- Accumulate input/output pairs, (x_i, y_i)

$$
\begin{array}{ccccccc}
\longrightarrow & & \longrightarrow & & \exists f.f\ x_0 = y_0 \wedge \cdots \wedge f\ x_{i-1} = y_{i-1} \Rightarrow f_i & & \longrightarrow \\
\uparrow & & & & & & \downarrow \\
 & \longleftarrow & y_i \Leftarrow \exists y.R\ x_i\ y & \longleftarrow & x_i \Leftarrow \exists x.\neg(R\ x\ (f_i\ x)) & \longleftarrow &
\end{array}
$$

Function Synthesis, contd.

List sorting

- Cheat: Correct search depth given, no iterated deepening.

$$f : (X : S_{<}) \to [|X|] \to [|X|]$$

$$\forall X : S_{<}.\forall xs : [|X|].Sorted\ (f\ xs) \wedge Perm\ xs\ (f\ xs)$$

$$\begin{array}{rcl}
x_i & = & [0], [1,0], [0,1], [1,0,1], [0,0,1], [1,1,0] \\
f_6 & = & \lambda X\ x \to elimList\ x\ (\lambda y \to y) \\
& & (\lambda y\ z\ w\ y' \to w\ (\\
& & \quad elimList\ y'\ (y :: []) \\
& & \quad (\lambda a\ b\ c \to elimBool\ (X.(<)\ a\ y)\ (a :: c)\ (y :: a :: b)) \\
& &))\ []
\end{array}$$

Alternatives when Writing Properties

Boolean function

- Quick computation
- Constant size proof term

Alternatives when Writing Properties

Boolean function

- Quick computation
- Constant size proof term

Inductive definition

- Decomposing properties, $M(\vec{?}) \rightsquigarrow M_1(\vec{?}) \wedge M_2(\vec{?})$, checked in parallel
- Linear size proof term

Alternatives when Writing Properties

Boolean function

- Quick computation
- Constant size proof term

Inductive definition

- Decomposing properties, $M(\vec{?}) \rightsquigarrow M_1(\vec{?}) \wedge M_2(\vec{?})$, checked in parallel
- Linear size proof term

bool. fcn.

$$\begin{aligned} &\mathit{perm} : (X : S_=) \to [|X|] \to [|X|] \to \mathit{Bool} \\ &\mathit{True}\ (\mathit{perm}\ X\ \mathit{xs}\ \mathit{ys}) \end{aligned}$$

ind. def.

$$\begin{aligned} &P_0 : \mathit{Perm}\ []\ [] \\ &P_1 : \mathit{Delete}\ x\ \mathit{ys}\ \mathit{zs} \to \mathit{Perm}\ \mathit{xs}\ \mathit{zs} \to \mathit{Perm}\ (x :: \mathit{xs})\ \mathit{ys} \\ &D_0 : \mathit{Delete}\ x\ (x :: \mathit{ys})\ \mathit{ys} \\ &D_1 : \mathit{Delete}\ x\ \mathit{ys}\ \mathit{zs} \to \mathit{Delete}\ x\ (y :: \mathit{ys})\ (y :: \mathit{zs}) \\ &\mathit{Perm}\ \mathit{xs}\ \mathit{ys} \end{aligned}$$

Proof Search Tool

- Meta variables and explicit substitutions
- Sequent calculus style treatment of implication and tuple elimination
- Delayed instantiation, equality constraints
- Prolog like search algorithm

Summary

- Idea: Perform testing using a proof search tool for a depentently typed functional language
- Instead of “generate, then test” – incremental construction of terms guided by input constraints and specification
- Hopes: Less need for customized generators and randomness
- Problems which are not completely testable could be attacked.
- Ability to construct simple higher-order terms

Summary

- Idea: Perform testing using a proof search tool for a depentently typed functional language
- Instead of “generate, then test” – incremental construction of terms guided by input constraints and specification
- Hopes: Less need for customized generators and randomness
- Problems which are not completely testable could be attacked.
- Ability to construct simple higher-order terms

Future

- Measure efficiency for standard testing problems
- Explore the usefulness of handling non-testable testing situations