Towards Generic Programming with Sized Types

Termination of Generic Programs

Andreas Abel

July 3, 2006
MPC'06, Kurressaare, Estonia

Teaser

Termination of Higher-Order Functions

- Recursive function applied to arguments:

$$f(x) = \ldots f(a) \ldots f(b) \ldots$$

- Check that $a < x$, $b < x$.
- What about a function passed to its argument?

$$f(g) = \ldots g(f) \ldots$$

- How to establish termination of f??

Generic Programs and Termination

- Typical generic program: traversal
- Observation: *structurally recursive*
- How to substantiate this observation?
- Are all generic programs total on total input?
- Are all generic programs even primitive recursive (catamorphisms)?

Outline

Contents

1 Example: Generic Finite Maps

Finite Maps for List-shaped Keys

- R. Hinze (JFP 2000): Generalizing Generalized Tries
- Key type: `[a]`, value type: `v`

```
data MapList f v
   = Leaf
   | Node (Maybe v) (f (MapList f v))
```

- If `f w` $\cong$ `a` $\rightarrow_{\mathsf{fin}}$ `w`, then `MapList f v` $\cong$ $[$`a`$]$ $\rightarrow_{\mathsf{fin}}$ `v`.
- Looking up a list-shaped key:

```
lookupList :: (forall w. a -> f w -> Maybe w) ->
              [a] -> MapList f v -> Maybe v
lookupList lo l      Leaf        = Nothing
lookupList lo []     (Node mv m) = mv
lookupList lo (a:as) (Node mv m)
    = lo a m >>= lookupList lo as
```

Merging Finite Maps

- Merging (possibly undefined) values.

```
comb :: (v -> v -> v)
     -> Maybe v -> Maybe v -> Maybe v
```

- Merging finite maps:

```
mergeList ::
  (forall w. (w -> w -> w) -> f w -> f w -> f w)
  -> (v -> v -> v)
  -> MapList f v -> MapList f v -> MapList f v
mergeList mergeF c Leaf t = t
```

```
mergeList mergeF c t Leaf = t
mergeList mergeF c (Node m1 t1) (Node m2 t2) =
  Node (comb c m1 m2)
       (mergeF (mergeList mergeF c) t1 t2)
```

Termination of Merging

- Is `mergeList` terminating on all inputs?

```
mergeList mergeF c (Node m1 t1) (Node m2 t2) =
  Node (comb c m1 m2)
       (mergeF (mergeList mergeF c) t1 t2)
```

- Consider:

```
mf m t1 t2 = m (Node Nothing t1) (Node Nothing t2)

run = mergeList mf fst (Node Nothing t1)
                       (Node Nothing t2)
```

- But: `mf` cannot be assigned the type

```
forall w. (w -> w -> w) -> f w -> f w -> f w
```

2 Termination via Sized Types

Sized Inductive Types

- Type constructor

$$\frac{a : \mathsf{ord} \qquad F : * \xrightarrow{+} *}{\mu^a F : *}$$

- $\mu^{a+1} F = F\,(\mu^a F)$.
- $\mu^a F$ contains trees of height $< a$.
- Semantically, a is an ordinal.
- Syntactically, $a ::= \imath + n \mid \infty$.

Sized Lists

- Example: $\mathsf{List}^a A = \mu^a F$ where $F\,X = 1 + A \times X$.
- $\mathsf{List}^a A$ contains lists of length $< a$.
- Definable data constructors:

$$\begin{array}{lcl} \mathsf{nil} & : & \forall A \forall \imath.\, \mathsf{List}^{\imath+1} A \\ \mathsf{cons} & : & \forall A \forall \imath.\, A \to \mathsf{List}^{\imath} A \to \mathsf{List}^{\imath+1} A \end{array}$$

Type-Based Termination

- Terminating recursion $\mathsf{letrec}\ f = t$:

$$\frac{\imath : \mathsf{ord},\ f : \mu^{\imath} F \to C(\imath) \vdash t : \mu^{\imath+1} F \to C(\imath+1)}{\mathsf{fix}^{\mu}(\lambda f.\, t) : \mu^a F \to C(a)}$$

Analysis of Merging

$$\begin{array}{lcl} \mathsf{Map}\langle\mathsf{List}\rangle^a\, F\, V & = & \mu^a\, (\lambda X.\, 1 + (1+V) \times F\, X) \\ \mathsf{Bin}\, V & = & V \to V \to V \\ \mathsf{comb} & : & \forall V.\, \mathsf{Bin}\, V \to \mathsf{Bin}\, (1+V) \\ \mathsf{merge}\langle\mathsf{List}\rangle & : & \forall F.\, (\forall W.\, \mathsf{Bin}\, W \to \mathsf{Bin}\, (F\, W)) \to \\ & & \quad \forall V.\, \mathsf{Bin}\, V \to \forall \imath.\, \mathsf{Bin}\, (\mathsf{Map}\langle\mathsf{List}\rangle^{\imath}\, F\, V) \\ \mathsf{merge}\langle\mathsf{List}\rangle & := & \lambda merge_F \lambda c.\ \mathsf{fix}^{\mu}\, \lambda merge. \\ & & \quad \mathsf{comb}\ (\lambda(mv_1, t_1) \lambda(mv_2, t_2). \\ & & \quad\quad (\mathsf{comb}\ c\ mv_1\ mv_2,\ merge_F\ merge\, t_1\, t_2)) \\ \text{with} \quad merge & : & \mathsf{Bin}\, (\mathsf{Map}\langle\mathsf{List}\rangle^{\imath}\, F\, V) \\ \mathsf{comb}\ (\lambda \ldots) & : & \mathsf{Bin}\, (\mathsf{Map}\langle\mathsf{List}\rangle^{\imath+1}\, F\, V) \\ t_1,\ t_2 & : & F\, (\mathsf{Map}\langle\mathsf{List}\rangle^{\imath}\, F\, V) \end{array}$$

3 Generic Programming (Romanticism)

Generic Programming (Romanticism)

- Kinds

$$\kappa ::= * \mid \kappa \to \kappa'$$

- Grammar of types

$$\begin{array}{lcl} F, G & ::= & C \mid X \mid \lambda X.F \mid F\, G \mid \mu F \\ C & ::= & 1 \mid + \mid \times \mid \mathsf{Int} \mid \ldots \end{array}$$

- Data types definable

$$\begin{array}{lcl} \mathsf{List} & : & * \to * \\ \mathsf{List} & := & \lambda A.\, \mu(\lambda X.\, 1 + A \times X) \end{array}$$

Generic Equality

- Type of generic equality (kind-indexed type)

$$\begin{array}{lcl} \mathsf{Eq}\langle A : * \rangle & = & A \to A \to \mathsf{Bool} \\ \mathsf{Eq}\langle F : \kappa \to \kappa' \rangle & = & \forall G : \kappa.\, \mathsf{Eq}\langle G : \kappa \rangle \to \mathsf{Eq}\langle F\, G : \kappa' \rangle \end{array}$$

- Instance: equality for lists

$$\mathsf{Eq}\langle \mathsf{List} : * \to * \rangle = \forall A : *.\quad \begin{array}{l} (A \to A \to \mathsf{Bool}) \to \\ (\mathsf{List}\, A \to \mathsf{List}\, A \to \mathsf{Bool}) \end{array}$$

Generic Equality

- Generic equality (type-indexed value)

$$\mathsf{eq}\langle F : \kappa \rangle : \mathsf{Eq}\langle F : \kappa \rangle$$

- User-defined clauses

$$\begin{array}{ll} \mathsf{eq}\langle 1 : * \rangle\ ()\ () & = \mathsf{true} \\ \mathsf{eq}\langle + : * \to * \to * \rangle\ eqA\ eqB\ (\mathsf{inl}\ a)\ (\mathsf{inl}\ a') & = eqA\ a\ a' \\ \mathsf{eq}\langle + : * \to * \to * \rangle\ eqA\ eqB\ (\mathsf{inr}\ b)\ (\mathsf{inr}\ b') & = eqB\ b\ b' \\ \mathsf{eq}\langle + : * \to * \to * \rangle\ eqA\ eqB\ \_\ \_ & = \mathsf{false} \\ \mathsf{eq}\langle \times : * \to * \to * \rangle\ eqA\ eqB\ (a, b)\ (a', b') & = eqA\ a\ a'\ \&\&\ eqB\ b\ b' \\ \mathsf{eq}\langle \mathsf{Int} \rangle\ n\ n' & = n == n' \end{array}$$

Generic Equality

- Hardwired clauses

$$\begin{array}{ll} \mathsf{eq}\langle F\, G \rangle & = \mathsf{eq}\langle F \rangle\ \mathsf{eq}\langle G \rangle \\ \mathsf{eq}\langle \lambda X.F \rangle & = \lambda eqX.\mathsf{eq}\langle F \rangle \\ \mathsf{eq}\langle X \rangle & = eqX \\ \mathsf{eq}\langle \mu F \rangle & = \mathsf{fix}\ \mathsf{eq}\langle F \rangle \\ & \quad \mathsf{where}\ \mathsf{fix}\ f = f\,(\mathsf{fix}\, f) \end{array}$$

- Example

$$\begin{array}{lcl} \mathsf{eq}\langle \mathsf{List} \rangle & = & \mathsf{eq}\langle \lambda A.\, \mu(\lambda X.\, 1 + A \times X) \rangle \\ & = & \lambda eqA.\ \mathsf{fix}\,(\lambda eqX.\, \mathsf{eq}\langle + \rangle\ \mathsf{eq}\langle 1 \rangle\ (\mathsf{eq}\langle \times \rangle\ eqA\ eqX)) \end{array}$$

- Recursive version

$$\mathsf{eq}\langle \mathsf{List} \rangle\ eqA = \mathsf{eq}\langle + \rangle\ \mathsf{eq}\langle 1 \rangle\ (\mathsf{eq}\langle \times \rangle\ eqA\ (\mathsf{eq}\langle \mathsf{List} \rangle\ eqA))$$

Instantiating Generic Equality

- Type of $\mathsf{eq}\langle\mathsf{List}\rangle$

$$\mathsf{Eq}\langle\mathsf{List}:*\to*\rangle = \forall A:*.\quad (A\to A\to\mathsf{Bool})\to (\mathsf{List}\,A\to\mathsf{List}\,A\to\mathsf{Bool})$$

- Computing $\mathsf{eq}\langle\mathsf{List}\rangle$

$$\begin{aligned}\mathsf{eq}\langle\mathsf{List}\rangle\ eqA\ (\mathsf{inl}\,())\ (\mathsf{inl}\,()) &= \mathsf{true}\\ \mathsf{eq}\langle\mathsf{List}\rangle\ eqA\ (\mathsf{inr}\,(a,x))\ (\mathsf{inr}\,(a',x')) &= (eqA\ a\ a')\\ &\quad \&\&\ (\mathsf{eq}\langle\mathsf{List}\rangle\ eqA\ x\ x')\end{aligned}$$

- Structurally recursive!

4 Generic Programming with Sized Types

Sized-Type Indexed Values

- Sized-Type Indexed Values

$$\begin{array}{lcll}\mathsf{poly}\langle C\rangle & = & \textit{user-defined} & \\ \mathsf{poly}\langle X\rangle & = & x & \\ \mathsf{poly}\langle\lambda\iota F:\mathsf{ord}\to\kappa\rangle & = & \mathsf{poly}\langle F\rangle & \\ \mathsf{poly}\langle\lambda X F:\kappa_1\to\kappa_2\rangle & = & \lambda x.\,\mathsf{poly}\langle F\rangle & \text{where } \kappa_1\neq\mathsf{ord}\\ \mathsf{poly}\langle F\,G\rangle & = & \mathsf{poly}\langle F\rangle\,\mathsf{poly}\langle G\rangle & \\ \mathsf{poly}\langle\mu^a F\rangle & = & \mathsf{fix}^\mu\,\mathsf{poly}\langle F\rangle & \end{array}$$

Sized Type-Indexed Types

- Sized Type-Indexed Types

$$\begin{array}{lcll}\mathsf{Type}\langle C\rangle & = & \textit{user-defined} & \text{for } C\in\{1,+,\times,\mathsf{Int},\mathsf{Char},\dots\}\\ \mathsf{Type}\langle X\rangle & = & X & \\ \mathsf{Type}\langle\lambda X F\rangle & = & \lambda X.\,\mathsf{Type}\langle F\rangle & \\ \mathsf{Type}\langle F\,G\rangle & = & \mathsf{Type}\langle F\rangle\ \mathsf{Type}\langle G\rangle & \\ \mathsf{Type}\langle\mu^a F\rangle & = & \mu^a\ \mathsf{Type}\langle F\rangle & \end{array}$$

Conclusions

Conclusions

- Sized types can certify termination for *functions that are passed to their arguments*

- Sized types interact well with:

  - higher-order functions
  - rank-2 polymorphism

- Conjecture: (Romantic) generic grograms are structurally recursive

- . . . from the perspective of sized types.