Verification of Iteration Schemes for Nested Datatypes in Coq

Dulma Rodriguez

Andreas Abel   Ralph Matthes

University of Munich LMU

Workshop TYPES 2006

Nottingham, England

April 20, 2006

Nested Datatypes

- Regular Datatype: $\mathsf{List}\ A$

$$\begin{aligned} \mathsf{nil} &: \mathsf{List}\ A \\ \mathsf{cons} &: A \longrightarrow \mathsf{List}\ A \longrightarrow \mathsf{List}\ A \end{aligned}$$

- Nested Datatype: $\mathsf{PList}\ A$ (2^n elements)

$$\begin{aligned} \mathsf{zero} &: A \longrightarrow \underbrace{\mathsf{PList}\ A}_{2^0} \\ \mathsf{succ} &: \underbrace{\mathsf{PList}\ (A \times A)}_{2^n} \longrightarrow \underbrace{\mathsf{PList}\ \ A}_{2^{n+1}} \end{aligned}$$

Summing up a powerlist

- $\mathsf{sum} \quad : \quad \mathsf{PList\ Nat} \longrightarrow \mathsf{Nat}$

- $\mathsf{sum}' \quad : \quad \forall A.(A \longrightarrow \mathsf{Nat}) \longrightarrow \mathsf{PList}\ A \longrightarrow \mathsf{Nat}$

  $\mathsf{sum}'\ f \quad (\mathsf{zero}\ a) = f\ a$

  $\mathsf{sum}'\ f \quad (\mathsf{succ}\ l) = \mathsf{sum}'\ (\lambda(a_1, a_2).f\ a_1 + f\ a_2)\ l$

- $\mathsf{sum} \quad := \quad \mathsf{sum}'\ \mathsf{id}$

Iteration Schemes

- Terminating recursion schemes
- Special form: Iteration
  - Conventional Iteration
    - Eliminators
    - Based on initial algebras semantics
  - Iteration in style of Mendler (1987)
    - Style of general recursion
    - Typed based termination
- “Iteration and Coiteration Schemes for Higher-Order and Nested Datatypes”, Abel, Matthes and Uustalu, 2005.

The Systems and Their Interrelationship

Regular types

$$\begin{tikzcd}
\underline{\mathsf{It}}^{\omega} \arrow[r, bend left] \arrow[dr, dashed] & \mathsf{MIt}^{\omega} \arrow[l, bend left] \arrow[d] \\
 & \mathsf{F}^{\omega}
\end{tikzcd}$$

Nested types

$$\begin{tikzcd}
\mathsf{It}^{\omega}_{=} \arrow[r] \arrow[d] & \mathsf{MIt}^{\omega}_{=} \arrow[d] \\
\mathsf{GIt}^{\omega} \arrow[r, bend left] & \mathsf{GMIt}^{\omega} \arrow[l, bend left] \arrow[d] \\
\underline{\mathsf{It}}^{\omega} \arrow[r, bend left] & \mathsf{MIt}^{\omega} \arrow[l, bend left] \arrow[d] \\
 & \mathsf{F}^{\omega}
\end{tikzcd}$$

Iteration Systems in Coq

- Shallow embedding

-
```
Module Type MIt_Type.
 ...
 Parameter in:      (* general constructor *)
 Parameter MIt:     (* iterator *)
 Axiom MIt_red: forall s t, MIt(s)(in t) = s (MIt s) t.
 ...
End.
```

Embeddings in Coq

Embedding of MIt$^\omega$ in F$^\omega$ $\Longrightarrow$ plain module of type `MIt_Type`

Embedding of system A in system B $\Longrightarrow$ functor `A_B` of type `A_Type` and argument `(B : B_Type)`

```
Module GMIt_MIt (M: MIt_Type) : GMIt_Type.
  Definition GMIt := ... M.MIt ...
  Lemma GMIt_red : ...
  Proof.
    ...
    rewrite MIt_red.
    reflexivity.
  Qed.
End.
```

Conclusions

- The embeddings were verified in Coq.
- Clear separation
  - System specification as a Module Type
  - System implementation as a shallow embedding
- Executable specification
- This project opens the field for experimentation with truly nested datatypes (Matthes, MPC’06).