Generalized Iteration and Coiteration for Higher-Order Nested Datatypes

Mendler rules!

Andreas Abel
(joint work with Ralph Matthes and Tarmo Uustalu)

Slide 1

FoSSaCS 2003
Warsaw, Poland
April 8, 2003

Work supported by:
PhD Prg. Logic in Computer Science, Munich (DFG)
EU TYPES Project

Powerlists

Slide 2

- lists of length 2^n
- perfectly balanced leaf-labelled binary trees
- in Haskell:

```
data PList a = Zero a
             | Succ (PList (a, a))

l0 = Zero 0
l1 = Succ (Zero (0,1))
l2 = Succ (Succ (Zero ((0,1),(2,3))))
l3 = Succ (Succ (Succ (Zero (((0,1),(2,3)),((4,5),(6,7))))))
```

- logarithmic access time

Summing up a Powerlist (First Try)

Slide 3

- compute the sum of all elements in a powerlist

```
sum :: PList Integer -> Integer

sum (Zero i) = i         -- i :: Integer
sum (Succ l) = sum ???   -- l :: PList (Integer, Integer)
```

- need to generalize function `sum`

Summing up a Powerlist (Second Try)

Slide 4

- Use polymorphic recursion:

```
sum' :: (a -> Integer) -> PList a -> Integer

sum' f (Zero a) = f a
sum' f (Succ l) = sum' (\ (a,b) -> f a + f b) l

sum :: PList Integer -> Integer
sum = sum' id
```

- `sum'` is terminating and total.

Nested Datatypes

Slide 5

- some Haskell datatypes

```
PList a = Zero a | Succ (PList (a,a))

List  a = Nil | Cons a (List a)
Bush  a = Nil | Cons a (Bush (Bush a))

Lam   a = Var a | App (Lam a) (Lam a) | Abs (Lam (Maybe a))
```

- *regular* datatype: type argument `a` to `List` constant in recursion
- *non-regular* or *nested* datatypes (`PList`, `Bush` and `Lam`): type argument changes in recursion

Summing up a Bush

Slide 6

- "Bushy lists" [e.g. Bird *et. al.*]:

```
data Bush a = Nil | Cons a (Bush (Bush a))

sum' :: (a -> Integer) -> Bush a -> Integer

sum' f Nil        = 0
sum' f (Cons a b) = f a + sum' (sum' f) b

sum :: Bush Integer -> Integer
sum = sum' id
```

- Contribution: `sum'` is *iterative*, hence total.
- Method: `sum'` definable in F^ω.

System F^ω

Slide 7

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

$$\begin{array}{lll} \kappa 0 & := & * & \text{types} \\ \kappa 1 & := & * \to * & \text{type transformers} \\ \kappa 2 & := & (* \to *) \to * \to * & \text{transformers of type transformers} \end{array}$$

- Constructors $F : \kappa$, in particular types $A : *$

$$\begin{array}{lll} F & ::= & X \mid \lambda X.G \mid F\,G \\ & \mid & \forall X^\kappa.\, A \mid \exists X^\kappa.\, A \mid A \to B \mid 1 \mid A \times B \mid 0 \mid A + B \end{array}$$

- Objects (terms) $t : A$

Nested Datatypes in System F^ω

Slide 8

- recap the datatypes

```
List  a = Nil | Cons a (List a)
PList a = Zero a | Succ (PList (a,a))
Lam   a = Var a | App (Lam a) (Lam a) | Abs (Lam (Maybe a))
```

- regular datatypes are fixpoints of kind $*$ [$\mu : (* \to *) \to *$].

$$\mathsf{List} = \lambda A.\, \mu(\lambda X.\ 1 + A \times X)$$

- nested datatypes are fixpoints of kind $\kappa 1$ [$\mu : (\kappa 1 \to \kappa 1) \to \kappa 1$].

$$\begin{array}{lll} \mathsf{PList} & = & \mu(\lambda F.\, \lambda A.\ A + F(A \times A)) \\ \mathsf{Lam} & = & \mu(\lambda F.\, \lambda A.\ A + F A \times F A + F(1 + A)) \end{array}$$

Mendler Iteration for Regular Datatypes

- Inductive types with Mendler-style iteration in System F.

Form.	$\mu_{\kappa 0}$	:	$(\kappa 0 \to \kappa 0) \to \kappa 0$
Intro.	$\mathsf{in}_{\kappa 0}$	:	$F\,(\mu_{\kappa 0} F) \to \mu_{\kappa 0} F$
Elim.	$\mathsf{MIt}_{\kappa 0}$	:	$(\forall X.\,(X \to G) \to F\,X \to G) \to \mu_{\kappa 0} F \to G$
Comp.	$\mathsf{MIt}_{\kappa 0}\,s\,(\mathsf{in}_{\kappa 0}\,t) \longrightarrow_\beta s\,(\mathsf{MIt}_{\kappa 0}\,s)\,t$		

Slide 9

- Note: *no positivity/monotonicity* required for F!
- Reduction close to general recursion.

$$\mathsf{fix}\,s\,t \longrightarrow_\beta s\,(\mathsf{fix}\,s)\,t$$

- Universally quantified type variable X ensures termination.
- Archetype of *type-based termination*.

Generalization of MIt to higher kinds

- Pointwise inclusion:

$$F \subseteq G := \forall A.\,F\,A \to G\,A$$

- Mendler iteration for kind $\kappa 1 = * \to *$.

Slide 10

Form.	$\mu_{\kappa 1}$	:	$(\kappa 1 \to \kappa 1) \to \kappa 1$
Intro.	$\mathsf{in}_{\kappa 1}$	:	$F\,(\mu_{\kappa 1} F) \subseteq \mu_{\kappa 1} F$
Elim.	$\mathsf{MIt}_{\kappa 1}$	:	$(\forall X^{\kappa 1}.\,X \subseteq G \to F\,X \subseteq G) \to \mu_{\kappa 1} F \subseteq G$
Comp.	$\mathsf{MIt}_{\kappa 1}\,s\,(\mathsf{in}_{\kappa 1}\,t) \longrightarrow_\beta s\,(\mathsf{MIt}_{\kappa 1}\,s)\,t$		

Programming with MIt

Slide 11

- Summing up a powerlist: $\mu F = \mathsf{PList}$, $GA = \mathsf{Integer}$.

$$\begin{aligned} \mathsf{sum} &:= \mathsf{MIt}\ldots \\ \mathsf{sum} &: \mu F \subseteq G \\ &= \forall A.\ \mathsf{PList}\,A \to \mathsf{Integer} \end{aligned}$$

- Cannot work: elements of powerlist of generic type A.
- Next try: Let $HA = \mathsf{Integer}$.

$$\begin{aligned} \mathsf{sum} &: \mu F \circ H \subseteq G \\ &\leftrightarrow \mathsf{PList}\,\mathsf{Integer} \to \mathsf{Integer} \end{aligned}$$

- Cannot work either!

Right Kan extension

Slide 12

- Need more general function:

$$\begin{aligned} \mathsf{sum}' &: \forall A.\ \mathsf{PList}\,A \to (A \to \mathsf{Integer}) \to \mathsf{Integer} \\ &= \mathsf{PList} \subseteq \lambda A.\,(A \to \mathsf{Integer}) \to \mathsf{Integer} \\ &= \mathsf{PList} \subseteq \lambda A.\,(A \to H\ ?\ ) \to G\ ? \\ &= \mathsf{PList} \subseteq \lambda A.\forall B.\,(A \to HB) \to GB \\ &= \mathsf{PList} \subseteq \mathsf{Ran}_H\,G \end{aligned}$$

- Right Kan extension:

$$\mathsf{Ran}_H\,GA := \forall B.\,(A \to HB) \to GB$$

Summing up a powerlist (Implementation)

Slide 13

$$\begin{array}{lll}
\mathsf{sum}' & : & \forall A.\ \mathsf{PList}\ A \to (A \to \mathsf{Integer}) \to \mathsf{Integer} \\
\mathsf{sum}' & := & \mathsf{MIt}_{\kappa 1}\ \lambda sum^{\forall A.\ X\,A \to (A \to \mathsf{Integer}) \to \mathsf{Integer}} \\
& & \quad \lambda t^{A + X(A \times A)} \\
& & \quad \lambda f^{A \to \mathsf{Integer}} \\
& & \quad \mathsf{case}\ t\ \mathsf{of} \\
& & \qquad \mid \mathsf{inl}\, a^A \Rightarrow f\ a \\
& & \qquad \mid \mathsf{inr}\, l^{X(A \times A)} \Rightarrow sum\ l\ (\lambda p^{A \times A}.\ f\,(\mathsf{fst}\, p) + f\,(\mathsf{snd}\, p))
\end{array}$$

$$\begin{array}{lll}
\mathsf{sum} & : & \mathsf{PList}\ \mathsf{Integer} \to \mathsf{Integer} \\
\mathsf{sum} & := & \lambda l.\ \mathsf{sum}'\ l\ \mathsf{id}
\end{array}$$

Generalizing “⊆”

Slide 14

- Since we need Kan extensions to program anything reasonable, why not hardwire them into the system?
- *Parameterized inclusion*

$$\begin{array}{rcl}
F \leq^H G & := & \forall A \forall B.\, (A \to HB) \to FA \to GB \\
& \leftrightarrow & \forall A.\, FA \to \forall B.\, (A \to HB) \to GB \\
& = & F \subseteq \mathsf{Ran}_H\, G
\end{array}$$

Generalized Mendler Iteration

Slide 15

- Inductive constructors with generalized Mendler iteration.

Form.	$\mu_{\kappa 1}$	:	$(\kappa 1 \to \kappa 1) \to \kappa 1$
Intro.	$\mathsf{in}_{\kappa 1}$	:	$F\,(\mu_{\kappa 1} F) \subseteq \mu_{\kappa 1} F$
Elim.	$\mathsf{GIt}_{\kappa 1}$	:	$(\forall X^{\kappa 1}.\, X \leq^H G \to F\, X \leq^H G) \to \mu_{\kappa 1} F \leq^H G$
Comp.	$\mathsf{GIt}_{\kappa 1}\, s\, f\, (\mathsf{in}_{\kappa 1}\, t) \longrightarrow_\beta s\, (\mathsf{GIt}_{\kappa 1}\, s)\, f\, t$		

- MIt is a special case.
- Scales to arbitrary kinds.

Embedding into F^ω

Slide 16

- Inductive types with Mendler iteration can be defined in System F^ω.
- Idea: obtain def. of μ from type of the eliminator MIt:

$$\begin{array}{lcl} \mathsf{MIt}_{\kappa 0} & : & \forall F \forall G.\, (\forall X.\, (X \to G) \to F\, X \to G) \to \mu_{\kappa 0} F \to G \\ & \leftrightarrow & \forall F.\, \mu_{\kappa 0} F \to \forall G.\, (\forall X.\, (X \to G) \to F\, X \to G) \to G \end{array}$$

$$\mu_{\kappa 0} F \quad := \quad \forall G.\, (\forall X.\, (X \to G) \to F\, X \to G) \to G$$

- Encode the r.h.s. of the computation rule in the def. of in:

$$\begin{array}{lcl} \mathsf{MIt}_{\kappa 0} & := & \lambda s \lambda r.\, r\, s \\ \mathsf{in}_{\kappa 0} & := & \lambda t \lambda s.\, s\, (\mathsf{MIt}_{\kappa 0}\, s)\, t \end{array}$$

- Works similar for MIt and GIt for higher ranks.

Coinductive Constructors

Slide 17

`\dualize`

Related and further work on nested datatypes

Slide 18

- Matthes: CSL 01 (rank-2)
- Bird, Meertens, Paterson, Gibbons *et al.*: Nested datatypes (in Haskell), specialized gfold.
- Hinze, Okasaki: Efficient algorithms using nested datatypes.
- A., Matthes (FICS'03): Mendler-style primitive recursion for higher ranks.
- Further work: nested datatypes in the *type-based termination* setting of Hughes/Pareto/Sabry, Barthe *et. al.*, A.

$$\frac{i, g : \mu^i F \leq^H G \vdash t : \mu^{i+1} F \leq^H G}{\mathsf{fix} g.t : \forall i.\, \mu^i F \leq^H G}$$