

Translucent Abstraction

Algebraic Datatypes with Safe Views

Meng Wang

University of Oxford

Algebraic Datatypes is Transparent and Open

data *List a = Nil | Cons a (List a)*

Pattern Matching:

append :: (List a, List a) → List a

append (Nil, ys) = ys

append (Cons x xs, ys) = Cons x (append (xs, ys))

Equational Reasoning:

append (Cons 1 Nil, Cons 2 Nil)

= Cons 1 (append (Nil, Cons 2 Nil))

= Cons 1 (Cons 2 Nil)

Algebraic Datatypes is Transparent and Open

data *List a = Nil | Cons a (List a)*

Pattern Matching:

append :: (*List a, List a*) → *List a*

append (*Nil, ys*) = *ys*

append (*Cons x xs, ys*) = *Cons x (append (xs, ys))*

Equational Reasoning:

append (Cons 1 Nil, Cons 2 Nil)
= *Cons 1 (append (Nil, Cons 2 Nil))*
= *Cons 1 (Cons 2 Nil)*

Algebraic Datatypes is Transparent and Open

data *List a = Nil | Cons a (List a)*

Pattern Matching:

append :: (*List a, List a*) → *List a*

append (Nil, ys) = ys

append (Cons x xs, ys) = Cons x (append (xs, ys))

Equational Reasoning:

$$\begin{aligned} & \text{append (Cons 1 Nil, Cons 2 Nil)} \\ = & \text{Cons 1 (append (Nil, Cons 2 Nil)} \\ = & \text{Cons 1 (Cons 2 Nil)} \end{aligned}$$

Algebraic Datatypes is Transparent and Open

type *Queue a = List a*

emptyQ = Nil

enQ :: a → Queue a → Queue a

enQ x xs = append xs (Cons x Nil)

deQ :: Queue a → Queue a

deQ (Cons x xs) = xs

Algebraic Datatypes is Transparent and Open

type *Queue a = (List a, List a)*

emptyQ = Nil

enQ :: a → Queue a → Queue a

enQ x xs = append xs (Cons x Nil)

deQ :: Queue a → Queue a

deQ (Cons x xs) = xs

Translucent Views

- Firstly introduced by Wadler (POPL 1987) and is an on-going research topic
- Translucency = Encapsulation + Openness

```

data List a = Nil | Cons a (List a)
view List a = Lin | Snoc (List a) a
  to Nil                                = Lin
  to (Cons x Nil)                       = Snoc Nil x
  to (Cons x (Snoc xs y))               = Snoc (Cons x xs) y
  from Lin                               = Nil
  from (Snoc Nil x)                     = Cons x Nil
  from (Snoc (Cons x xs) y)             = (Cons x (Snoc xs y))

```

Translucent Views

- Firstly introduced by Wadler (POPL 1987) and is an on-going research topic
- Translucency = Encapsulation + Openness

data $List\ a = Nil \mid Cons\ a\ (List\ a)$

view $List\ a = Lin \mid Snoc\ (List\ a)\ a$

$to\ Nil = Lin$

$to\ (Cons\ x\ Nil) = Snoc\ Nil\ x$

$to\ (Cons\ x\ (Snoc\ xs\ y)) = Snoc\ (Cons\ x\ xs)\ y$

$from\ Lin = Nil$

$from\ (Snoc\ Nil\ x) = Cons\ x\ Nil$

$from\ (Snoc\ (Cons\ x\ xs)\ y) = (Cons\ x\ (Snoc\ xs\ y))$

Translucent Views

data $List\ a = Nil \mid Cons\ a\ (List\ a)$

view $List\ a = Lin \mid Snoc\ (List\ a)\ a$

$to\ Nil = Lin$

$to\ (Cons\ x\ Nil) = Snoc\ Nil\ x$

$to\ (Cons\ x\ (Snoc\ xs\ y)) = Snoc\ (Cons\ x\ xs)\ y$

$from\ Lin = Nil$

$from\ (Snoc\ Nil\ x) = Cons\ x\ Nil$

$from\ (Snoc\ (Cons\ x\ xs)\ y) = (Cons\ x\ (Snoc\ xs\ y))$

$init\ (Snoc\ xs\ x) = xs$

$reverse\ Nil = Nil$

$reverse\ (Cons\ x\ xs) = Snoc\ (reverse\ xs)\ x$

Equational Reasoning

$$\begin{aligned}
 \text{to Nil} &= \text{Lin} \\
 \text{to (Cons x Nil)} &= \text{Snoc Nil x} \\
 \text{to (Cons x (Snoc xs y))} &= \text{Snoc (Cons x xs) y} \\
 \text{from Lin} &= \text{Nil} \\
 \text{from (Snoc Nil x)} &= \text{Cons x Nil} \\
 \text{from (Snoc (Cons x xs) y)} &= (\text{Cons x (Snoc xs y)}) \\
 \text{reverse Nil} &= \text{Nil} \\
 \text{reverse (Cons x xs)} &= \text{Snoc (reverse xs) x}
 \end{aligned}$$

$$\begin{aligned}
 &\text{reverse (Cons 1 (Cons 2 Nil))} = \text{Snoc (reverse (Cons 2 Nil)) 1} \\
 &= \text{Snoc (Snoc Nil 2) 1} = \text{Snoc (Cons 2 Nil) 1} \\
 &= \text{Cons 1 (Snoc Nil 2)} = \text{Cons 2 (Cons 1 Nil)}
 \end{aligned}$$

Translucent Views

- Firstly introduced by Wadler (POPL 1987) and is an on-going research topic
- Translucency = Encapsulation + Openness

type *Queue a = (List a, List a)*

view *Queue a = List a*

to = append

from = ?

Bidirectional Transformation comes to the rescue?