Lazy Algebraic Types in Isabelle HOL

Andreas Lochbihler

Digital Asset (Switzerland) GmbH

Pascal Stoop

ETH Zurich

Motivation

Lazy evaluation would be nice to have in Isabelle

- Computing with codatatypes

```
codatatype 'a stream = SCons 'a ('a stream)
```

- Data-driven programming

```
to_list :: ('a, 'b) rbt ⇒ ('a * 'b) list
```

Motivation

Lazy evaluation would be nice to have in Isabelle

- Computing with codatatypes

```
codatatype 'a stream = SCons 'a ('a stream)
```

- Data-driven programming

```
to_list :: ('a, 'b) rbt ⇒ ('a * 'b) list
```

Requirements

- ⊕ Work with the existing code generator and all target languages
- ⊕ Transparent to definitions and proofs

Motivation

Lazy evaluation would be nice to have in Isabelle

- Computing with codatatypes

```
codatatype 'a stream = SCons 'a ('a stream)
```

- Data-driven programming

```
to_list :: ('a, 'b) rbt ⇒ ('a * 'b) list
```

Requirements

⊕ Work with the existing code generator and all target languages

⊕ Transparent to definitions and proofs

`HOL-Library.Code_Lazy`

Suspension ADT `lazy`

```
delay :: (unit ⇒ 'a) ⇒ 'a lazy
force :: 'a lazy ⇒ 'a
```

Wadler et al. 1998:
"even with difficulty"

Suspension ADT `lazy`

Wadler et al. 1998:
“even with difficulty”

```
delay :: (unit ⇒ 'a) ⇒ 'a lazy
force :: 'a lazy ⇒ 'a
```

Types

```
datatype 'a list
     = Nil
     | Cons 'a ('a list)
```

$\longrightarrow$

```
datatype 'a list
     = Lazy_list (('a lazy_list) lazy)
  and 'a lazy_list
     = Nil_lazy
     | Cons_lazy 'a ('a list)
```

Suspension ADT lazy

Wadler et al. 1998:
“even with difficulty”

```
delay :: (unit ⇒ 'a) ⇒ 'a lazy
force :: 'a lazy ⇒ 'a
```

Types

```
datatype 'a list
     = Nil
     | Cons 'a ('a list)
```

→

```
datatype 'a list
     = Lazy_list (('a lazy_list) lazy)
  and 'a lazy_list
     = Nil_lazy
     | Cons_lazy 'a ('a list)
```

Functions

```
unpack (Lazy_llist susp) = susp
```

```
app xs ys = case xs of
     Nil ⇒ ys
   | Cons x xs' ⇒
       Cons x (app xs' ys)
```

→

```
app xs ys = case force (unpack xs) of
     Nil_lazy ⇒ ys
   | Cons_lazy x xs' ⇒
       Lazy_list (delay (λ_ ⇒
         Cons_lazy x (app xs' ys)))
```

Demo

Available in Isabelle2018-RC1

`HOL-Library.Code_Lazy`

Available in Isabelle2018-RC1

`HOL-Library.Code_Lazy`

Pattern-matching elimination independently usable:

- case_of_simps
- Code_Target_Nat