Logical properties of a modality for erasure

Nils Anders Danielsson

IFIP WG 2.1 Meeting #79,
Otterlo, January 2020

Erasure

- Lists of a given length in Agda:

```
data Vec (A : Set) : ℕ → Set where
  []    : Vec A 0
  _::_  : {n : ℕ} →
          A → Vec A n → Vec A (1 + n)
```

- With a bad implementation:
  $\Omega(n^2)$ space for lists of length n.
- It might make sense for the compiler to "erase" some data.

Erasure

With explicit erasure annotations:

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{Vec}\ (@0\ A : \mathsf{Set}) : @0\ \mathbb{N} \to \mathsf{Set}\ \mathsf{where} \\
\quad [\,] \quad\ \ : \mathsf{Vec}\ A\ 0 \\
\quad \_{::}\_ : \{@0\ n : \mathbb{N}\} \to \\
\qquad\qquad A \to \mathsf{Vec}\ A\ n \to \mathsf{Vec}\ A\ (1 + n)
\end{array}
$$

`@0`

- `@0` is used to mark arguments and definitions that should be erased at run-time.
- Agda is supposed to make sure that:
  - Things marked as erased are actually erased.
  - There is never any data missing at run-time.
- The typing rules are based on work by McBride and Atkey.
- The implementation is mainly due to Abel.

@0

ok : {@0 A : Set} → A → A
ok $x = x$

```
-- not-ok : {@0 A : Set} → @0 A → A
-- not-ok x = x

-- also-not-ok : @0 Bool → Bool
-- also-not-ok true  = false
-- also-not-ok false = true
```

Erased

A type-level variant of @0:

```
record Erased (@0 A : Set a) : Set a where
  constructor [_]
  field
    @0 erased : A
```

Monad

Erased is a monad:

$$\text{return} : \{@0\ A : \text{Set}\ a\} \to @0\ A \to \text{Erased}\ A$$
$$\text{return}\ x = [\ x\ ]$$

$$\_{\gg\!=}\_ :$$
$$\{@0\ A : \text{Set}\ a\}\ \{@0\ B : \text{Set}\ b\} \to$$
$$\text{Erased}\ A \to (A \to \text{Erased}\ B) \to \text{Erased}\ B$$
$$x \gg\!= f = [\ \text{erased}\ (f\ (\text{erased}\ x))\ ]$$

A toy application

An application

Natural numbers that...

- ...compute (roughly) like binary natural numbers at run-time.
- ...compute (roughly) like unary natural numbers at compile-time...
  - ...for some operations.

The underlying representation

Lists of bits with the least significant digit first and no trailing zeros:

```
abstract
  mutual

    data Bin' : Set where
      []        : Bin'
      _∷_⟨_⟩ : (b : Bool) (n : Bin') →
                  @0 Invariant b n → Bin'

    data Invariant : Bool → Bin' → Set where
      true-inv  : Invariant true n
      false-inv : Invariant false (b ∷ n ⟨ inv ⟩)
```

The underlying representation

Abstract:

- The representation can be changed without breaking client code.
- Does not “compute” at compile-time: The type-checker does not use definitional equalities.

The underlying representation

The representation of a given natural number is unique. An equivalence ($\approx$ bijection):

$\text{to-}\mathbb{N} : \text{Bin'} \to \mathbb{N}$

Indexed binary numbers

Binary natural numbers representing
a given natural number:

```
Bin-[_] : @0 ℕ → Set
Bin-[ n ] = Σ Bin' (λ b → Erased (to-ℕ b ≡ n))
```

Indexed binary numbers

Binary natural numbers representing
a given natural number:

$$\mathsf{Bin\text{-}[\_]} : @0\ \mathbb{N} \to \mathsf{Set}$$
$$\mathsf{Bin\text{-}[}\ n\ \mathsf{]} = \Sigma\ \mathsf{Bin'}\ (\lambda\ b \to \mathsf{Erased}\ (\mathsf{to\text{-}}\mathbb{N}\ b \equiv n))$$

The type is propositional:

$$\{@0\ n : \mathbb{N}\} \to \mathsf{Is\text{-}prop}\ \mathsf{Bin\text{-}[}\ n\ \mathsf{]}$$

$$\mathsf{Is\text{-}prop} : \mathsf{Set}\ a \to \mathsf{Set}\ a$$
$$\mathsf{Is\text{-}prop}\ A = (x\ y : A) \to x \equiv y$$

Non-indexed binary numbers

Binary natural numbers:

$$\mathsf{Bin} : \mathsf{Set}$$
$$\mathsf{Bin} = \Sigma\ (\mathsf{Erased}\ \mathbb{N})\ (\lambda\ n \to \mathsf{Bin\text{-}[}\ \mathsf{erased}\ n\ \mathsf{]})$$

Returns the erased index:

$$@0\ \lfloor\_\rfloor : \mathsf{Bin} \to \mathbb{N}$$
$$\lfloor\ ([\ n\ ]\ ,\ \_)\ \rfloor = n$$

[]-cong

A key lemma:

$$
\begin{array}{l}
\textsf{[]-cong} : \\
\quad \{@0\ A : \textsf{Set}\ a\}\ \{@0\ x\ y : A\} \to \\
\quad \textsf{Erased}\ (x \equiv y) \to [\ x\ ] \equiv [\ y\ ]
\end{array}
$$

[]-cong

A key lemma:

$$
\begin{array}{l}
[]\text{-cong} : \\
\quad \{@0\ A : \mathsf{Set}\ a\}\ \{@0\ x\ y : A\} \rightarrow \\
\quad \mathsf{Erased}\ (x \equiv y) \rightarrow [\ x\ ] \equiv [\ y\ ]
\end{array}
$$

With the K rule and propositional equality:

$$[]\text{-cong}\ [\ \mathsf{refl}\ ] = \mathsf{refl}$$

[]-cong

A key lemma:

```
[]-cong :
  {@0 A : Set a} {@0 x y : A} →
  Erased (x ≡ y) → [ x ] ≡ [ y ]
```

With the K rule and propositional equality:

```
[]-cong [ refl ] = refl
```

With Cubical Agda and paths:

```
[]-cong [ eq ] = λ i → [ eq i ]
```

[]-cong

A key lemma:

$$
\begin{array}{l}
[]\text{-cong} : \\
\quad \{@0\ A : \mathsf{Set}\ a\}\ \{@0\ x\ y :\ A\} \rightarrow \\
\quad \mathsf{Erased}\ (x \equiv y) \rightarrow [\ x\ ] \equiv [\ y\ ]
\end{array}
$$

With the K rule and propositional equality:

$$[]\text{-cong}\ [\ \mathsf{refl}\ ] = \mathsf{refl}$$

With Cubical Agda and paths:

$$[]\text{-cong}\ [\ eq\ ] = \lambda\ i \rightarrow [\ eq\ i\ ]$$

In both cases []-cong is an equivalence that maps [refl x] to refl [x].

Non-indexed binary numbers

Recall:

$$\mathsf{Bin} : \mathsf{Set}$$
$$\mathsf{Bin} = \Sigma\ (\mathsf{Erased}\ \mathbb{N})\ (\lambda\ n \to \mathsf{Bin}\text{-}[\ \mathsf{erased}\ n\ ])$$

$$@0\ \lfloor\_\rfloor : \mathsf{Bin} \to \mathbb{N}$$
$$\lfloor\ ([\ n\ ]\ ,\ \_)\ \rfloor = n$$

Equality follows from equality for the erased indices:

$$\begin{aligned}&\mathsf{Erased}\ (\lfloor\ x\ \rfloor \equiv \lfloor\ y\ \rfloor) && \simeq \\ &\mathsf{proj}_1\ x \equiv \mathsf{proj}_1\ y && \simeq \\ &x \equiv y\end{aligned}$$

Addition

```
abstract

  plus : Bin' → Bin' → Bin'
  plus = … -- Add with carry.

_⊕_ : Bin → Bin → Bin
([ m ] , m' , p) ⊕ ([ n ] , n' , q) =
  ([ m + n ] , plus m' n' , [ … ])
```

Conversion to/from unary natural numbers?

Goal:

- $\mathsf{Bin} \simeq \mathbb{N}$ (in a non-erased context).
- With the forward direction pointwise equal to $\lfloor\_\rfloor$ (in an erased context).

Some theory

Some equivalences

$$\text{Erased } \top \simeq \top$$

$$\text{Erased } \bot \simeq \bot$$

$$\text{Erased } ((x : A) \to P\ x) \simeq ((x : A) \to \text{Erased } (P\ x))$$

$$\text{Erased } ((x : A) \to P\ x) \simeq$$
$$((x : \text{Erased } A) \to \text{Erased } (P\ (\text{erased } x)))$$

$$\text{Erased } (\Sigma\ A\ P) \simeq$$
$$\Sigma\ (\text{Erased } A)\ (\lambda\ x \to \text{Erased } (P\ (\text{erased } x)))$$

Some preservation lemmas

For erased A : Set a and B : Set b:

$$
\begin{array}{llllllll}
@0 & (A \rightarrow & B) & \rightarrow & \text{Erased } A & \rightarrow & \text{Erased } B \\
@0 & A \Leftrightarrow & B & \rightarrow & \text{Erased } A & \Leftrightarrow & \text{Erased } B \\
@0 & A \twoheadrightarrow & B & \rightarrow & \text{Erased } A & \twoheadrightarrow & \text{Erased } B \\
@0 & A \leftrightarrow & B & \rightarrow & \text{Erased } A & \leftrightarrow & \text{Erased } B \\
@0 & A \simeq & B & \rightarrow & \text{Erased } A & \simeq & \text{Erased } B \\
@0 & A \rightarrowtail & B & \rightarrow & \text{Erased } A & \rightarrowtail & \text{Erased } B
\end{array}
$$

$$
@0\ \text{Embedding } A\ B \rightarrow
$$
$$
\quad \text{Embedding } (\text{Erased } A)\ (\text{Erased } B)
$$

H-levels

Erased commutes with Is-prop:

$$\text{Erased}\ (\text{Is-prop}\ A) \Leftrightarrow \text{Is-prop}\ (\text{Erased}\ A)$$

More generally:

$$\text{Erased}\ (\text{H-level}\ n\ A) \Leftrightarrow \text{H-level}\ n\ (\text{Erased}\ A)$$

Modality

Erased is a *left exact modality* in the sense of Rijke, Shulman and Spitters.

Back to the application

An equivalence

$$
\begin{array}{ll}
\text{Bin-[} n \text{]} & \simeq \\
\Sigma \ \text{Bin'} \ (\lambda \ b \to \text{Erased} \ (\text{to-}\mathbb{N} \ b \equiv n)) & \simeq \\
\Sigma \ \mathbb{N} \ (\lambda \ m \to \text{Erased} \ (m \equiv n))
\end{array}
$$

Note that n can be erased.

Another equivalence

The binary natural numbers are equivalent to the unary ones, both at compile-time and at run-time:

$$
\begin{aligned}
&\mathsf{Bin} && \simeq \\
&\Sigma\ (\mathsf{Erased}\ \mathbb{N})\ (\lambda\ n \to \mathsf{Bin\text{-}[}\ \mathsf{erased}\ n\ \mathsf{]}) && \simeq \\
&\Sigma\ (\mathsf{Erased}\ \mathbb{N})\ (\lambda\ n \to \Sigma\ \mathbb{N}\ (\lambda\ m \to \\
&\quad \mathsf{Erased}\ (m \equiv \mathsf{erased}\ n))) && \simeq \\
&\Sigma\ \mathbb{N}\ (\lambda\ m \to \Sigma\ (\mathsf{Erased}\ \mathbb{N})\ (\lambda\ n \to \\
&\quad \mathsf{Erased}\ (m \equiv \mathsf{erased}\ n))) && \simeq \\
&\Sigma\ \mathbb{N}\ (\lambda\ m \to \mathsf{Erased}\ (\Sigma\ \mathbb{N}\ (\lambda\ n \to m \equiv n))) && \simeq \\
&\mathbb{N} \times \mathsf{Erased}\ \top && \simeq \\
&\mathbb{N} \times \top && \simeq \\
&\mathbb{N}
\end{aligned}
$$

Another equivalence

The binary natural numbers are equivalent to the unary ones, both at compile-time and at run-time:

$$\mathsf{Bin} \simeq \mathbb{N}$$

In an erased context the forward direction is pointwise equal to $\lfloor \_ \rfloor$ (i.e. it returns the index).

Stability

Stability

A type A is *stable* if Erased A implies A:

$$\text{Stable} : \text{Set}\ a \to \text{Set}\ a$$
$$\text{Stable}\ A = \text{Erased}\ A \to A$$

A type is *very stable* (or *modal*) if
[_] is an equivalence:

$$\text{Very-stable} : \text{Set}\ a \to \text{Set}\ a$$
$$\text{Very-stable}\ A = \text{Is-equivalence}\ ([\_]\ \{A = A\})$$

Double negation

Erased A implies $\neg\ \neg\ A$. Thus types that are stable for double negation are stable for Erased:

$$\{@0\ A : \mathsf{Set}\ a\} \to (\neg\ \neg\ A \to A) \to \mathsf{Stable}\ A$$

Types for which it is known whether or not they are inhabited are also stable:

$$\{@0\ A : \mathsf{Set}\ a\} \to A \uplus \neg\ A \to \mathsf{Stable}\ A$$

Stability of equality

Variants of Stable and Very-stable:

$$\begin{array}{l}\text{Stable-}{\equiv} : \text{Set } a \to \text{Set } a \\ \text{Stable-}{\equiv}\ A = (x\ y : A) \to \text{Stable } (x \equiv y)\end{array}$$

$$\begin{array}{l}\text{Very-stable-}{\equiv} : \text{Set } a \to \text{Set } a \\ \text{Very-stable-}{\equiv}\ A = \\ \quad (x\ y : A) \to \text{Very-stable } (x \equiv y)\end{array}$$

Decidable equality

Stable propositions are very stable:

$$\text{Stable } A \to \text{Is-prop } A \to \text{Very-stable } A$$

Thus types for which equality is decidable have very stable equality:

$$((x\ y : A) \to x \equiv y \uplus \neg\ x \equiv y) \to$$
$$\text{Very-stable-}{\equiv}\ A$$

Propositions

However, it is not the case that every very stable type is a proposition:

$$\neg\ (\{A : \mathsf{Set}\ a\} \to \mathsf{Very\text{-}stable}\ A \to \mathsf{Is\text{-}prop}\ A)$$

Erased Bool is not a proposition, but it is very stable:

$$\{@0\ A : \mathsf{Set}\ a\} \to \mathsf{Very\text{-}stable}\ (\mathsf{Erased}\ A)$$

Why is Bin-[n] propositional?

Lemma:

$$
\begin{array}{l}
\{@0\ x : A\} \to \\
\text{Very-stable-}\equiv\ A \to \\
\text{Is-prop}\ (\Sigma\ A\ (\lambda\ y \to \text{Erased}\ (y \equiv x)))
\end{array}
$$

Bin-[n] is propositional:

$$
\begin{array}{ll}
((x\ y : \mathbb{N}) \to x \equiv y \uplus \neg\ x \equiv y) & \to \\
\text{Very-stable-}\equiv\ \mathbb{N} & \to \\
\text{Very-stable-}\equiv\ \text{Bin'} & \to \\
\text{Is-prop}\ (\Sigma\ \text{Bin'}\ (\lambda\ b \to \text{Erased}\ (b \equiv \text{from-}\mathbb{N}\ n))) & \to \\
\text{Is-prop}\ (\Sigma\ \text{Bin'}\ (\lambda\ b \to \text{Erased}\ (\text{to-}\mathbb{N}\ b \equiv n))) & \to \\
\text{Is-prop Bin-[}\ n\ \text{]} &
\end{array}
$$

Closure properties

For Π:

$$(\forall\ x \to \text{Stable}\ (P\ x)) \to \text{Stable}\ ((x :\ A) \to P\ x)$$

$$(\forall\ x \to \text{Very-stable}\ (P\ x)) \to$$
$$\text{Very-stable}\ ((x :\ A) \to P\ x)$$

(The second property is proved using function extensionality.)

Closure properties

For Σ:

$$\text{Very-stable } A \to (\forall\ x \to \text{Stable } (P\ x)) \to \text{Stable } (\Sigma\ A\ P)$$

$$\text{Very-stable } A \to (\forall\ x \to \text{Very-stable } (P\ x)) \to \text{Very-stable } (\Sigma\ A\ P)$$

Closure properties

For equality:

Very-stable $A \to$ Very-stable-$\equiv$ A

Closure properties

For List:

$$\text{Stable-}\equiv\ A \to \text{Stable-}\equiv\ (\text{List}\ A)$$

$$\text{Very-stable-}\equiv\ A \to \text{Very-stable-}\equiv\ (\text{List}\ A)$$

Universes

Universes of very stable types are very stable (assuming univalence):

$$\text{Very-stable}\ (\Sigma\ (\text{Set}\ a)\ \text{Very-stable})$$

Discussion

- A surprising amount of theory for something as simple as Erased?
- Can []-cong be defined in plain Agda without K?
- Unclear whether erasure makes sense in Cubical Agda.