

MAlonzo

```
data Vec (A : Set) : Nat → Set where
```

```
  nil : Vec A 0
```

```
  cons : {n : Nat} → A → Vec A n → Vec A (n + 1)
```

```
vec-map : {A B : Set}{n : Nat} → (A → B) → Vec A n → Vec B n
```

```
vec-map f nil = nil
```

```
vec-map f (cons x xs) = cons (f x) (vec-map f xs)
```

MAlonzo генерирует:

```
data T1 a0 a1 a2 = C2 | C3 a0 a1 a2
```

```
d4 v0 v1 v2 v3 (C2) = unsafeCoerce C2
```

```
d4 v0 v1 v2 v3 (C3 v4 v5 v6) = unsafeCoerce (C3 (unsafeCoerce v4) ...)
```

```
ghci>:t d4
```

```
d4 :: a → a1 → a2 → a3 → (T1 t t1 t2) → b
```

```
data Vec a = Nil | Cons Nat a (Vec a)
```

```
vecMap :: Nat → (a1 → a2) → Vec a1 → Vec a2
```

```
vecMap n f v =
```

```
  case v of
```

```
    Nil → Nil
```

```
    Cons n0 x xs → Cons n0 (f x) (vecMap n0 f xs)
```

Agda next

```
f : {A : Set} → ({B : Set} → B) → A  
f = ...
```

вместе с `{-# COMPILED_EXPORT f fh #-}` породит

```
fh :: ∀a0. () → (∀b0. () → b0) → a0  
fh = ...
```

Что хочется

Идея

Нужно сохранить инварианты, поддерживаемые системой типов.

1. Экспортируем весь код в Haskell с помощью MAIOnzo.
2. Для выбранных имен пытаемся найти соответствующие типы на Haskell и генерируем обертки над скомпилированным кодом.

Реализованное решение

Указание на «экспорт» дается с помощью прагм

```
{-# EXPORT agda-name haskell-name #-}
```



```
data T (A : Set) : Set → Set where ...
```

вместе с `{-# EXPORT T Th #-}` породит

```
newtype Th (a0 :: *) (a1 :: *) = <hidden>
```



```
f : {A : Set} → ({B : Set} → B) → A
```

```
f = ...
```

вместе с `{-# EXPORT f fh #-}` породит

```
fh :: ∀(a0 :: *). (∀(a1 :: *). a1) → a0
```

```
fh = ...
```

Data

```
data Type (A : Set → Set) : Set → Set where ...
```

MAlonzo:

```
data T1 a0 ... ak = ...
```

Сгенерировано при `{-# EXPORT Type HType #-}`:

```
newtype HType (a0 :: * → *) (a1 :: *) =  
  HType (∀ b1 ... bk. T1 b1 ... bk)
```

Terms

```
f : {A : Set} → ({B : Set → Set} → B A) → A
f = ...
```

MAlonzo:

```
-- d1 :: a0 → (a1 → a2) → a3
d1 = ...
```

Сгенерировано при {-# EXPORT f fh #-}:

```
fh :: ∀(a0 :: *). (∀(a1 :: * → *). a1 a0) → a0
fh = λv0. unsafeCoerce (d1 (λ_. unsafeCoerce v0))
```

Q&A