Free monoids take a price HIT

Gergő Érdi
http://unsafePerform.IO/

Haskell.SG, January 2020.

1. Recap: your grandma's free monoids

Monoids

```
record Monoid A : Type where
  field
    set : isSet A

    _◇_ : A → A → A
    ϵ   : A

    unit-l : ∀ x     → ϵ ◇ x ≡ x
    unit-r : ∀ x     → x ◇ ϵ ≡ x
    assoc  : ∀ x y z → (x ◇ y) ◇ z ≡ x ◇ (y ◇ z)

open Monoid {{...}}
```

Syntax of monoids

```
data MonoidSyntax A : Type where
  Element : A → MonoidSyntax A
  _:◇:_   : MonoidSyntax A → MonoidSyntax A → MonoidSyntax
  :ϵ:     : MonoidSyntax A
```

Syntax of monoids

```
data MonoidSyntax A : Type where
  Element : A → MonoidSyntax A
  _:◇:_   : MonoidSyntax A → MonoidSyntax A → MonoidSyntax
  :ϵ:     : MonoidSyntax A
```

Is MonoidSyntax a monoid?

Syntax of monoids

```
data MonoidSyntax A : Type where
  Element : A → MonoidSyntax A
  _:◇:_   : MonoidSyntax A → MonoidSyntax A → MonoidSyntax
  :ϵ:     : MonoidSyntax A
```

Is MonoidSyntax a monoid?

Regardless of the carrier type A, this is **not a lawful monoid**; for example:

```
xs ◇ (ys ◇ zs) = xs :◇: (ys :◇: zs)
(xs ◇ ys) ◇ zs = (xs :◇: ys) :◇: zs
```

Syntax of monoids

```
data MonoidSyntax A : Type where
  Element : A → MonoidSyntax A
  _:◇:_   : MonoidSyntax A → MonoidSyntax A → MonoidSyntax
  :ϵ:     : MonoidSyntax A
```

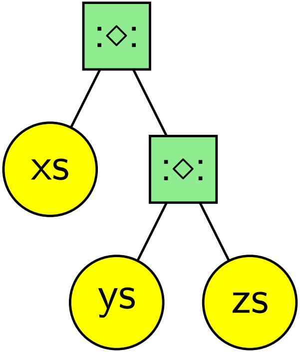
Is MonoidSyntax a monoid?

Regardless of the carrier type A, this is **not a lawful monoid**; for example:

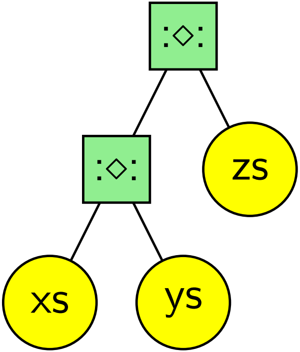
```
xs ◇ (ys ◇ zs) = xs :◇: (ys :◇: zs)
(xs ◇ ys) ◇ zs = (xs :◇: ys) :◇: zs
```

There is too fine a structure!

Is `MonoidSyntax` a monoid?



$xs \diamond (ys \diamond zs)$



$(xs \diamond ys) \diamond zs$

Monoid homomorphisms

```
record isHom (M : Monoid A) (N : Monoid B) (ϕ : A → B) : Ty
  open Monoid M renaming (_◇_ to _◇₁_; ϵ to ϵ₁)
  open Monoid N renaming (_◇_ to _◇₂_; ϵ to ϵ₂)
  field
    map-unit : ϕ ϵ₁ ≡ ϵ₂
    map-op   : ∀ x y → ϕ (x ◇₁ y) ≡ ϕ x ◇₂ ϕ y

Extends : (A → B) → (A → T) → (T → B) → Type
Extends f inj ϕ = ϕ ∘ inj ≡ f

Hom-Extends : (M₀ : Monoid T) (M : Monoid B) →
  (A → B) → (A → T) → (T → B) → Type
Hom-Extends M₀ M f inj ϕ = isHom M₀ M ϕ × Extends f inj ϕ
```

Free monoids

```
Unique : (A : Type) (P : A → Type) → Type
Unique A P = Σ[ x ∈ A ] Σ[ _ ∈ P x ]
  ∀ (y : A) → P y → y ≡ x

record IsFreeMonoidOver (A : Type) (M₀ : Monoid T) : Type₁
  field
    inj : A → T
    free : {{M : Monoid B}} (f : A → B) →
      Unique (T → B) (Hom-Extends M₀ M f inj)

IsFreeMonoid :
  {F : Type → Type} (FM : ∀ {A} → isSet A → Monoid (F A)) -
  Type₁
IsFreeMonoid {F} FM = ∀ {A} (AIsSet : isSet A) →
  IsFreeMonoidOver A (FM AIsSet)
```

List A is a free monoid

_++_ is associative simply because there is no place to hide for a tree structure in a chain of _∷_'s.

```
listMonoid : isSet A → Monoid (List A)
listMonoid {A = A} AIsSet = record
  { set = isOfHLevelList 0 AIsSet
  ; _◇_ = _++_
  ; ϵ = []
  ; unit-l = λ xs → refl
  ; unit-r = ++-unit-r
  ; assoc = ++-assoc
  }

listIsFree : IsFreeMonoid listMonoid
```

The price of free

We had to **think** to come up with the representation [a] for the free monoid, it didn't **follow mechanically** from the definition of monoids.

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids?

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids? `Map a Nat`

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids? `Map a Nat`
- Abelian groups?

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids? `Map a Nat`
- Abelian groups? `Map a Int`

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids? `Map a Nat`
- Abelian groups? `Map a Int`
- Groups?

The price of free

We had to **think** to come up with the representation `[a]` for the free monoid, it didn't **follow mechanically** from the definition of monoids.

What is a good representation for free...

- commutative monoids? `Map a Nat`
- Abelian groups? `Map a Int`
- Groups?

"I don't want to be thinking, I want to be HoTT!"

2. Free monoids in HoTT

A HoTT & free monoid

In a HoTT setting, we can write a free monoid **without thinking** by taking the monoid syntax and enriching it with the monoid law-induced equalities as a **higher inductive type**:

```
data HITMon A : Type where
  ⟨_⟩       : A → HITMon A
  :ϵ:       : HITMon A
  _:◇:_     : HITMon A → HITMon A → HITMon A

  :unit-l: : ∀ x       → :ϵ: :◇: x ≡ x
  :unit-r: : ∀ x       → x :◇: :ϵ: ≡ x
  :assoc:  : ∀ x y z  → (x :◇: y) :◇: z ≡ x :◇: (y :◇: z)

  trunc    : isSet (HITMon A)
```

HITMon is trivially a monoid

```
freeMonoid : ∀ A → Monoid (HITMon A)
freeMonoid A = record
  { set = trunc
  ; _◇_ = _:◇:_
  ; ϵ = :ϵ:
  ; unit-l = :unit-l:
  ; unit-r = :unit-r:
  ; assoc = :assoc:
  }
```

HITMon is trivially a monoid

```
freeMonoid : ∀ A → Monoid (HITMon A)
freeMonoid A = record
  { set = trunc
  ; _◇_ = _:◇:_
  ; ϵ = :ϵ:
  ; unit-l = :unit-l:
  ; unit-r = :unit-r:
  ; assoc = :assoc:
  }
```

… and it's also free:

```
freeMonoidIsFree : IsFreeMonoid (λ {A} _ → freeMonoid A)
```

List vs HITMon

The two are isomorphic.

From List to HITMon we can just go right-associated:

```
module ListVsHITMon (AIsSet : isSet A) where
  listIsSet : isSet (List A)
  listIsSet = isOfHLevelList 0 AIsSet

  fromList : List A → HITMon A
  fromList [] = :ϵ:
  fromList (x ∷ xs) = ⟨ x ⟩ :◇: fromList xs
```

List vs HITMon (cont.)

For the other direction, we map fiat equalities to list equality proofs:

```
toList : HITMon A → List A
toList ⟨ x ⟩ = x ∷ []
toList :ϵ: = []
toList (x :◇: y) = toList x ++ toList y
toList (:unit-l: x i) = toList x
toList (:unit-r: x i) = ++-unit-r (toList x) i
toList (:assoc: x y z i) = ++-assoc
  (toList x) (toList y) (toList z)
  i
toList (trunc x y p q i j) = listIsSet
  (toList x) (toList y)
  (cong toList p)
  (cong toList q)
  i j
```

List vs HITMon (cont.)

These two functions form an isomorphism, which we can lift using univalence into a type equality:

```
toList-fromList : ∀ xs → toList (fromList xs) ≡ xs
fromList-toList : ∀ x → fromList (toList x) ≡ x
```

```
HITMon≃List : HITMon A ≃ List A
HITMon≃List = isoToEquiv
  (iso toList fromList toList-fromList fromList-toList)
```

```
HITMon≡List : HITMon A ≡ List A
HITMon≡List = ua HITMon≃List
```

The free monoid

All free monoids over the same base set are isomorphic (and thus by univalence, equal) so it makes sense to talk about **the** free monoid.

The free monoid

All free monoids over the same base set are isomorphic (and thus by univalence, equal) so it makes sense to talk about **the** free monoid.

The free monoid

All free monoids over the same base set are isomorphic (and thus by univalence, equal) so it makes sense to talk about **the** free monoid.

Sketch of the proof:

- Suppose we have `M` and `N` free monoids over some `A`, and take the homomorphisms ϕ : `Hom N M` (since `N` is free) and ψ : `Hom M N` with $\phi \circ \mathtt{inj}_N \equiv \mathtt{inj}_M$ and $\psi \circ \mathtt{inj}_M \equiv \mathtt{inj}_N$
- We have $\phi \circ \psi$: `Hom M M`, with $\phi \circ \psi \circ \mathtt{inj}_M \equiv \mathtt{inj}_M$
- Now since `M` is free, take ι : `Hom M M` with $\iota \circ \mathtt{inj}_M \equiv \mathtt{inj}_M$ uniquely. This gives $\phi \circ \psi \equiv \iota \equiv \mathtt{id}$ since they all satisfy this property. Likewise for $\psi \circ \phi$.
- So ϕ and ψ form an isomorphism between `M` and `N`. □