Scalar Actions in Lean's Mathlib

Based on a paper by Eric Wieser[1]

Steven Bronsveld Jelmer Firet

Radboud University Nijmegen

December 19, 2021

Overview of structures with multiplication

structure	0	1	$a \cdot b$	$a \cdot 0 = 0 \cdot a = 0$	$a \cdot 1 = 1 \cdot a = a$	$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$a \cdot b = b \cdot a$	$a + b$	$0 + a = a + 0 = a$	$(a + b) + c = a + (b + c)$	$a + b = b + a$	$(a + b) \cdot c = a \cdot c + b \cdot c$	$a \cdot (b + c) = a \cdot b + a \cdot c$
has_mul			✓										
mul_zero_class	✓		✓	✓									
monoid		✓	✓		✓	✓							
monoid_with_zero	✓	✓	✓	✓	✓	✓							
semiring	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
comm_semiring	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Overview of structures with scalar multiplication

structure	$r \bullet m$	$r \bullet 0_M = 0_M$	$0_R \bullet m = 0_M$	$1_R \bullet m = m$	$(r \cdot s) \bullet m = r \bullet (s \bullet m)$	$r \bullet (m + n) = r \bullet m + r \bullet n$	$(r + s) \bullet m = r \bullet m + s \bullet m$	$r \bullet m = r_M \cdot m = m \cdot r_M$	$r \bullet (m \cdot n) = (r \bullet m) \cdot (r \bullet n)$	$r \bullet 1_M = 1_M$
has_scalar	✓									
smul_with_zero	✓	✓	✓							
mul_action	✓			✓	✓					
mul_action_with_zero	✓	✓	✓	✓	✓					
distrib_mul_action	✓	✓		✓	✓	✓				
module	✓	✓	✓	✓	✓	✓	✓			
algebra	✓	✓	✓	✓	✓	✓	✓	✓		
mul_semiring_action	✓	✓		✓	✓	✓			✓	✓

has_scalar

Structure (algebra.group.defs)

```
class has_scalar (M : Type*) (α : Type*) :=
(smul : M → α → α)
infixr ` • `:73 := has_scalar.smul
```

Instance (group_theory.group_action.defs)

```
instance has_mul.to_has_scalar (α : Type*) [has_mul α]
: has_scalar α α := ⟨(*)⟩
```

has_scalar

Structure (algebra.group.defs)

```
class has_scalar (M : Type*) (α : Type*) :=
(smul : M → α → α)
infixr ` • `:73 := has_scalar.smul
```

Instance (group_theory.group_action.defs)

```
instance has_mul.to_has_scalar (α : Type*) [has_mul α]
: has_scalar α α := { smul := mul }
```

function.has_scalar

Instance

```
instance function.has_scalar (I α : Type*) [has_mul α]
: has_scalar α (I → α) := { smul := λ r v, (λ i, r * v i)}
```

What about `has_scalar M (I → κ → α)`?

Instance

```
instance function.has_scalar (I M α : Type*) [has_scalar M α]
: has_scalar M (I → α) := { smul := λ r v, (λ i, r • v i) }
```

What about `has_scalar M (Π i : I, f i)`?

Instance (group_theory.group_action.pi)

```
instance pi.has_scalar (...) [Π i, has_scalar M (f i)]
: has_scalar M (Π i : I, f i) := { smul := λ r v, (λ i, r • v i) }
```

function.has_scalar

Instance

```
instance function.has_scalar (I α : Type*) [has_mul α]
: has_scalar α (I → α) := { smul := λ r v, (λ i, r * v i)}
```

What about `has_scalar M (I → κ → α)`?

Instance

```
instance function.has_scalar (I M α : Type*) [has_scalar M α]
: has_scalar M (I → α) := { smul := λ r v, (λ i, r • v i) }
```

What about `has_scalar M (Π i : I, f i)`?

Instance (group_theory.group_action.pi)

```
instance pi.has_scalar (...) [Π i, has_scalar M (f i)]
: has_scalar M (Π i : I, f i) := { smul := λ r v, (λ i, r • v i) }
```

function.has_scalar

Instance

```
instance function.has_scalar (I α : Type*) [has_mul α]
: has_scalar α (I → α) := { smul := λ r v, (λ i, r * v i)}
```

What about `has_scalar M (I → κ → α)`?

Instance

```
instance function.has_scalar (I M α : Type*) [has_scalar M α]
: has_scalar M (I → α) := { smul := λ r v, (λ i, r • v i) }
```

What about `has_scalar M (Π i : I, f i)`?

Instance (group_theory.group_action.pi)

```
instance pi.has_scalar (...) [Π i, has_scalar M (f i)]
: has_scalar M (Π i : I, f i) := { smul := λ r v, (λ i, r • v i) }
```

function.has_scalar

Instance

```
instance function.has_scalar (I α : Type*) [has_mul α]
: has_scalar α (I → α) := { smul := λ r v, (λ i, r * v i)}
```

What about `has_scalar M (I → κ → α)`?

Instance

```
instance function.has_scalar (I M α : Type*) [has_scalar M α]
: has_scalar M (I → α) := { smul := λ r v, (λ i, r • v i) }
```

What about `has_scalar M (Π i : I, f i)`?

Instance (`group_theory.group_action.pi`)

```
instance pi.has_scalar (...) [Π i, has_scalar M (f i)]
: has_scalar M (Π i : I, f i) := { smul := λ r v, (λ i, r • v i) }
```

function.has_scalar

Instance

```
instance function.has_scalar (I α : Type*) [has_mul α]
: has_scalar α (I → α) := { smul := λ r v, (λ i, r * v i)}
```

What about `has_scalar M (I → κ → α)`?

Instance

```
instance function.has_scalar (I M α : Type*) [has_scalar M α]
: has_scalar M (I → α) := { smul := λ r v, (λ i, r • v i) }
```

What about `has_scalar M (Π i : I, f i)`?

Instance (group_theory.group_action.pi)

```
instance pi.has_scalar (...) [Π i, has_scalar M (f i)]
: has_scalar M (Π i : I, f i) := { smul := λ r v, (λ i, r • v i) }
```

Bibliography

Eric Wieser. "Scalar actions in Lean's mathlib". In: *arXiv:2108.10700 [cs]* (Aug. 10, 2021). arXiv: 2108.10700. URL: http://arxiv.org/abs/2108.10700 (visited on 12/09/2021).