

MoA core Magnolia API

Marius Kleppe Larnøy

January 12, 2022

Here we outline the core MoA Magnolia API, including padding operations. For now all modules are **signature** modules, but we will introduce axioms when the core structure is in place.

Rather than going really abstract top-level when designing the API, this specification will be developed more with a concrete implementation in mind.

1 Core concepts/signatures

1.1 signature/concept Array

The generic Array signature. Contains the basic types we need, and the core functions.

Types:

- `A`: generic (A)rray type
- `E`: generic (E)lement type
- `I`: generic (I)ndex type
- `Int`: some form of integer
- `Shape`: type to represent the collection of an arrays dimensional lengths

Unary Operations

- `dim(a:A):Int`
- `shape(a:A):Shape`
- `total(s:Shape):Int`

(Helper predicates? not sure about these)

- `validIndex(i:I,a:A)`
- `partialIndex(i:I,a:A)` (i.e. `total(i) < total(shape(a))`)
- `totalIndex(i:I,a:A)` (i.e. `total(i) == total(shape(a))`)

Binary Operations

- `cat(a:A,b:A):A guard shape(a) == shape(b)`
- `psi(i:I,a:A):A guard partialIndex(i,a)`
- `psi(i:I,a:A):E guard totalIndex(i,a)`
- `take(i:I,a:A):A guard validIndex(i,a)`
- `drop(i:I,a:A):A guard validIndex(i,a)`

2 concept Padding

TODO

3 concept MoA

Here we import the generic Array concept, and can define separate array types. This also allows us to define more operations requiring finer typing.

Imports/renamings

- `Array[A => LinearArray, I => LinearIndex]`
- `Array[A => MultiArray, I => MultiIndex]`

Operations

- `iota(i:Int):LinearArray`
- `ravel(a:MultiArray):LinearArray`
- `reshape(s:Shape,a:MultiArray):MultiArray`
`guard total(s)==total(shape(a))`

4 General notes, thoughts

- Question: overload functions like `total` on argument type, or define separate functions for different types. E.g. `total` of a shape necessary, or just design shape signature with a function that yields the total
- Question: specify Util concepts like `Int` and abstract structures like `Semi-group/Monoid`. Not part of core `moa`, but useful
- Question: Would it be any benefit to further split up the concepts? e.g. some functions in one concept, padding separated maybe? not sure

- Follow-up: Might be best to define operations closer to their intended arguments. The upside to doing it like this is that we keep the signatures lean and clean, and don't have to import redundant signatures for defining operations. Also avoid circular dependencies
- Follow-up2: separate DNF and ONF level operations?
- some sort of Map concept? more of a underlying concept, how does the operations actually apply to arrays
- Note: Separating the types, while bringing some much needed distinctions between elements, brings with it its own set of issues. Neccessarily it leads to many of the functions requiring overloading, bringing redundancy into the code. Might need some design rethinking to avoid it, or maybe it is a neccessary evil.