

Initial Report : Struct

Andrea Iskander Belkhir id : 511089

Beyza Özdemir id : 576145

GitHub url : <https://github.com/andreaIskanderBelkhir/Struct.jl>

Introduction

struct is part of the julia library “LAR.jl”.this library perform geometric calculations on cellular complexes expressed through the Linear Algebraic Representation (LAR). Struct represent complex object and describe those object in they're coordinate system, in this way we can specify the edges. In the code Struct is used as a acyclic oriented graphs; an acyclic oriented graphs are a hierarchical structures formed by different component, and every component have they're different coordinate system

Main interface

A user wanting to use this package will make use of the 4 main interface :

- * Lar.Struct.
- * Lar.Apply.
- * Lar.Struct2lar.
- * Lar.EvalStruct.

The function Struct will create a object of type struct using as input an array of object. This function create a structure of geometrical object starting from an array of object.The attribute of a structure are <body,box,name,dim,category>. this function return a “Struct” type value and his coordinate system is based on the first object of the “struct” arguments.Also, the resulting geometrical value is often associated with a variable name.Every object in struct can be trasformed by a tensor within its own container The generation of containers may continue hierarchically by suitably applying **Struct**. this function is implemented with 4 methods.

The function apply use the larmodel in input with the affinateMatrix to return the larmodel as a tuple formed by points an array of cells.

The function struct2lar return the struct given by input as his lar representation.

The function evalStruct return the world coordinate of the struct in input

Example

To help with the project we used 3 example taken from the **LinearAlgebraicRepresentation.jl** package. the examples were chosen to have different complexity . Its possible to find the chosen example in the folder

/docs/examples. Those examples are used to test the entirety of the package, meanwhile for testing the singular function we created some easier examples in the notebook of the specific function.

Start

Before starting to improve the code, the work started with understanding the package structure as a whole, right after we started to study the functions that compose `struct.jl` dividing each function in a notebook and for each create some example to run the code and test it with the annotations `@btime`, `@benchmark` and `@code_llvm` (the last two annotations are not used in notebooks with longer functions for functionality problems). Before starting to optimize the code, we decided that it was better to improve the readability of the code using new functions when possible. For example, the function `<r(args...)>` can use 2 more functions: one for 2D rotation and another for the 3D.

```
function r(args...)
    n = length(args)
    if n == 1 # rotation in 2D
        mat=r2D(args)
    end

    if n == 3 # rotation in 3D
        mat=r3D(args)
    end
    return mat
end
```

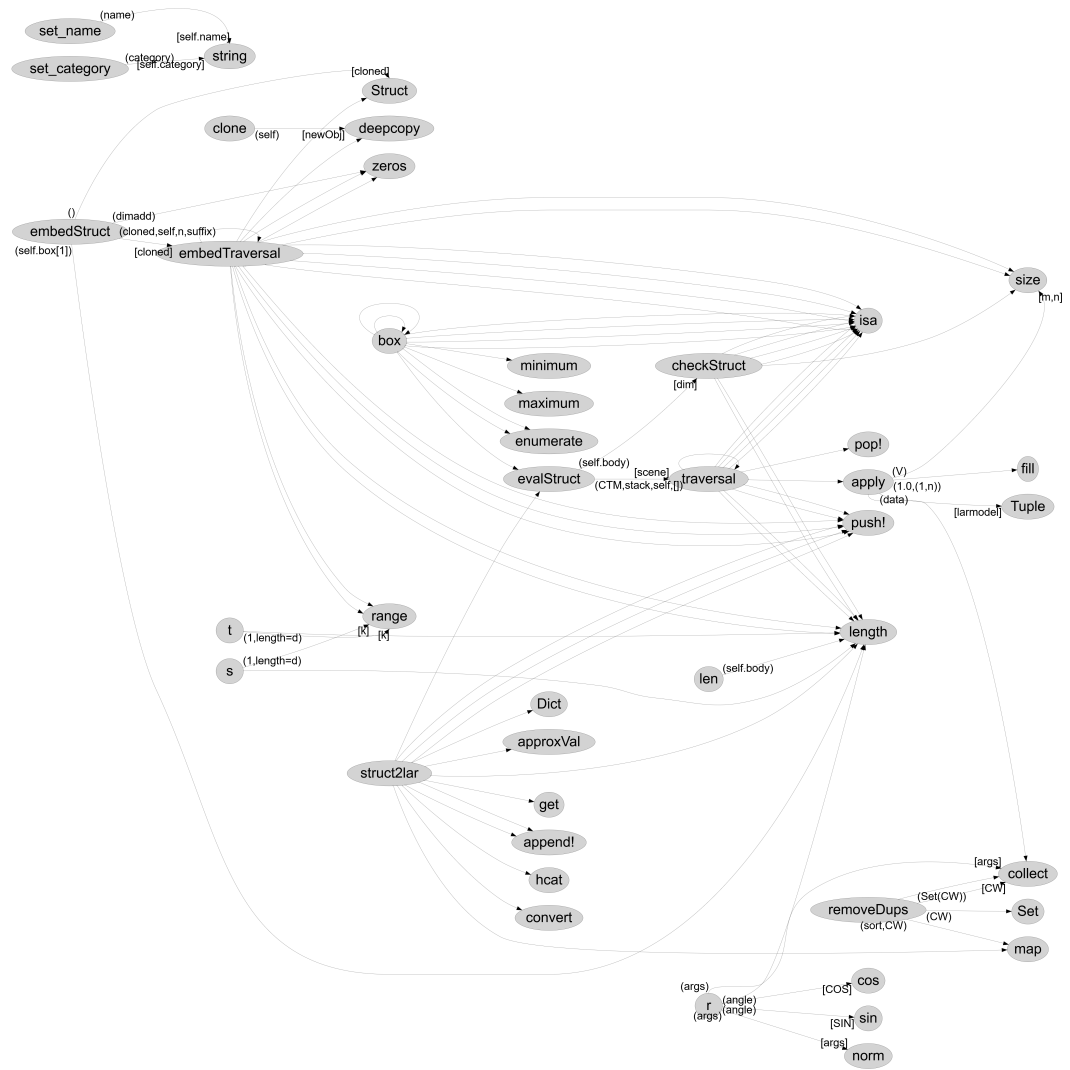


Figure 1: Dependency graph