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 packege will make use of the 4 main interface :

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

The funcion Struct will create a object of type struct using as input an array of object. This funcion create a structure of geometrical object starting from an array of object.The attribute of a structure are <body,box,name,dim,category>. this funcion 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 funcion is implemented with 4 methods.

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

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

The funcion 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 example are used to test the entirety of the package,meanwhile for testing the singolar funcion we created some easyer example in the notebook of the specific funcion

Start

Before starting to improve the code, the work started with understanding the package struct as a whole, right after we started to study the funcions that compose struct.jl dividing each funcion in a notebook and for each create some example to run the code and test it with the annotation @btime, @benchmark and @code_llvm (the last two annotation are not used in notebook with longer funcion for funcionability problem.

Optimization

The optimization started with the study of the book recommended in class "Julia High Performance - Second Edition - Optimizations, distributed computing, multithreading, and GPU programming with Julia 1.0 and beyond by Avik Sengupta" and with some topic found online for a better undestanding.

New funcion

What we used for improving the code was at first creating new funcion when it was needed, for example for the funcion <r(args..)> we splitted the funcion in two and then call different funcion(then since the rotation 3D was splittable again we create more funcion).

```
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
```

Type

What did the most for the performance improvment was a good typing and the use of the package StaticArray.jl and his type, an example of how much can one of this static array improve the code its written in the readme of the github page "The speed of *small* `SVector`s, `SMatrix`s and `SArray`s is often $> 10\times$ faster than `Base.Array`"

Annotation

In The code we used different annotation that help with the optimization of the code, the annotation more used were `@inline` and `@inbound` both annotation are descrived in the reccomended book but we also used online topic to better learn when to use it.

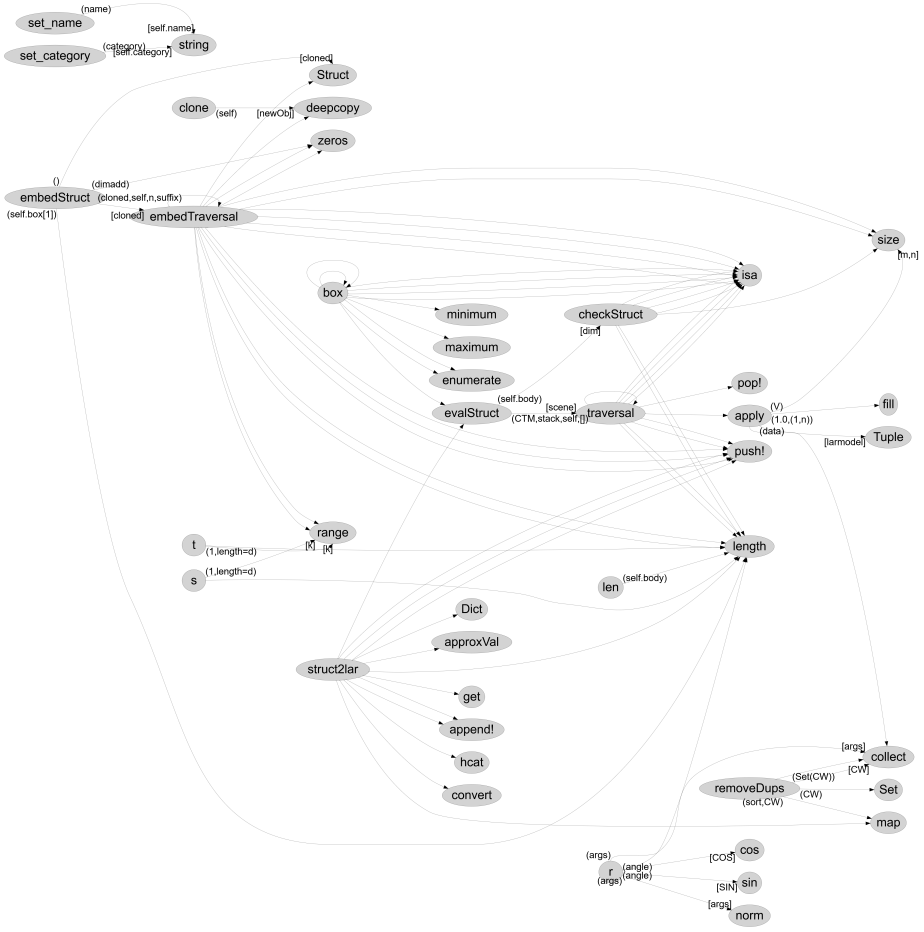

Figure 1: Dependency graph