Syntax of the Rauzy language

(I think we should use the formal definition)

Note that all characters used in the syntax are case sensitive

1. Formal definition

Both relations and Rauzy objects (we will use this term to refer to the objects we are describing in our language) are represented by Json objects. To distinguish them, a member “nature” will have a specific value (“relation” and “object” respectively).

For A a pattern, we define:
named(A)
NonEmptyString : A

Here is the grammar of a Rauzy object (the definition of a relation follows):

```
{
“nature” : “object”,
“extends” : string,
“objects” : { list(Named(Rauzy_object)) },
“relations” : { list(Named(relation)) },
“properties” : { list(Named(string)) },
“library” : string
}
```

The members “extends”, “objects”, “relations”, “properties” and “library” can be empty (e.g. “objects : {}), null (e.g. “relations” : null), or even not defined. In theses cases, it will be considered as empty.

Here is the grammar of a relation :

```
{“nature” : “relation”,
“extends” : string,
“from” : [ list(string) ],
“to” : [ list(string) ],
“directional” : true or false,
“properties” : { list(Named(string)) }
}
```

The members “extends” and “properties” can be empty (e.g. “objects : {}), null (e.g. “extends” : null), or even not defined. In theses cases , it will be considered as empty.

A library file will be as follows:

```
{
“nature” : “library”,
“relations” : { list(Named(relation)) },
“objects” : { list(Named(Rauzy_object)) }
}
```

2. Backus-Naur Form (BNF)

This form may have less precision than the other since it does not include all the conventions in json (you may have multiple spaces, have multiple definition of the same term etc). The formal definition is the reference.

```
<NonEmptyString> := ' “ ' ( <Alpha> | <Digit> )+ ' ” '
```

```
<String> ::= ' “ '( <Alpha> | <Digit> )* ' ” '
<Alpha> ::= [a_zA-Z]
<Digit> ::= [0-9]
<List(<A>)> ::= (<A> ',' <A>)*
<NonEmptyList(<A>)> ::= <A> (',' <A>)*
<Named(<A>)>::= <NonEmptyString> ':' <A>
```

We define recursively:

```
<Unordered(X)> ::=X
<Unordered(X, List)> ::= ( X ',' <Unordered(List)> ) | ( <Unordered(List)> ',' X )
<RauzyObject> ::= '{' <Unordered(
' “nature” '
':' ' “object” ' ,
(' “extends” ' ':' <String>)?,
(' “objects” '
':' '{' <List(<Named(RauzyObject)>)> '}' )?,
(' “relations” ' ':' '{' <List(<Named(Relation)>)> '}' )? ,
(' “properties” ' ':' '{' <List(<Named(String)>)> '}' )? ,
(' “library” '
':' <string>)? ) '}'
<RauzyRelation> ::= '{' <Unordered(
' “nature” ' ':' '“relation” ' ,
(' “extends” ' ':' <String>)? ,
' “from” ' ':' '[' <List(String)> ']' ,
' “to” ' ':' '[' <List(String)> ']' ,
'“directional” ' ':' 'True' | 'False' ,
(' “properties” ' ':' '{' <List(<Named(String)>)> '}' )? )
'}'
<Library> := '{' <Unordered(
' “nature” ' ':' '“library” ' ,
(' “relations” '
':' <String>)? ,
(' “objects” ' ':' '[' <List(<Named(<RauzyRelation>)>)> ']')? ,
(' “objects” ' ':' '[' <List(<Named(<RauzyObject>)>)> ']')? )
'}'
```