RRG Final Project

John Wang

October 13, 2022

Contents

1 Intro

The goal of this project is to make a JSON standard for representing structures in JSON.

The syntax should be concise, yet fully featured of all RRG features.

Once this syntax has been established, it could be built upon for other projects such as an implementation of the linking algorithm. JSON presents a language agnostic language that will allow it to have a variety of applications.

As a proof of concept, a visualizer was implemented. It takes in a file using the JSON format established, and outputs an SVG similar to the typical RRG diagrams.

The JSON standard serves as unified standard that could potentially allow for collaboration while working on RRG. For instance, there someone could implement the linking algorithm to transform semantic forms into a syntactic form. Then they could output the syntactic form as the given JSON standard, and use the visualizer to get a free image.

2 Structure

The JSON structure consists of an array of objects, and each object represents a word in the Sentence.

Each object has a field `phon`, which represents the surface phonological structure of the word. They can optionally contain `top` or `bot`, which represent the constituent projections and the operator projections. `top` takes a json object, and `bot` takes an array of json objects.

2.1 Top

The field `pos` determines the part of speech it should connect to. The field `kind` determines the constituent projection if it is part of Core, Clause, Core periphery, or Clause periphery or Sentence. These are represented by `Pred`, `Nuc`, `Core`, `Clause`, `CoreP`, `ClauseP` or `Sentence`

2.2 Bot

Bot contains an array of json objects that contain the fields: The field `op` determines which operator it should use. The field `kind` which can take the same values as the ones in `top`

Bot contains an array of such objects, because they can have multiple operator projections.

3 Examples

Simple example

This is on page 7 of the RRG book.

```
[
    {
    "phon" : "what",
    "top" : {
        "pos": "NP",
        "kind": "Clause"
        }
    },
    {
    "phon" : "did"
},
    {
```

```
    "phon" : "Robin",
    "top" : {
        "pos" : "NP",
        "kind" : "Core"
    }

},
    {
    "phon" : "show",
    "top" : {
        "pos" : "V",
        "kind": "Pred"
    }
},
    {
    "phon" : "to Pat",
    "top" : {
        "pos" : "PP",
        "kind": "Core"
    }
},
    {
    "phon" : "in the library",
    "top" :
    {
        "pos" : "PP",
        "kind": "CoreP"
    }
},
    {
    "phon" : "yesterday",
    "top" :
    {
        "pos" : "ADV",
        "kind": "CoreP"
    }
}

]
```

From page 14 of the book.


```
[
    {
        "phon" : "Will",
        "bot" : [

            {
                "op" : "IF",
                "kind": "Clause"
            },
            {
                "op" : "TNS",
                "kind": "Clause"
            }
        ]

    },
    {
    "phon" : "they",
    "top" :
            {
                "pos" : "NP",
                "kind": "Core"
            }
    },
    {
        "phon" : "have to",
        "bot" : [

            {
                "op" : "MOD",
                "kind": "Core"
            }
        ]

    },
    {
        "phon" : "be",
        "bot" : [
```

```
            {
                "op" : "ASP",
                "kind": "Nuc"
            }
        ]

    },
    {
        "phon" : "leaving",
        "top" :
                {
                    "pos" : "V",
                    "kind": "Pred"
                },
        "bot" : [

            {
                "op" : "ASP",
                "kind": "Nuc"
            }
        ]

    }

]
```

4 SVG generation

4.1 Usage

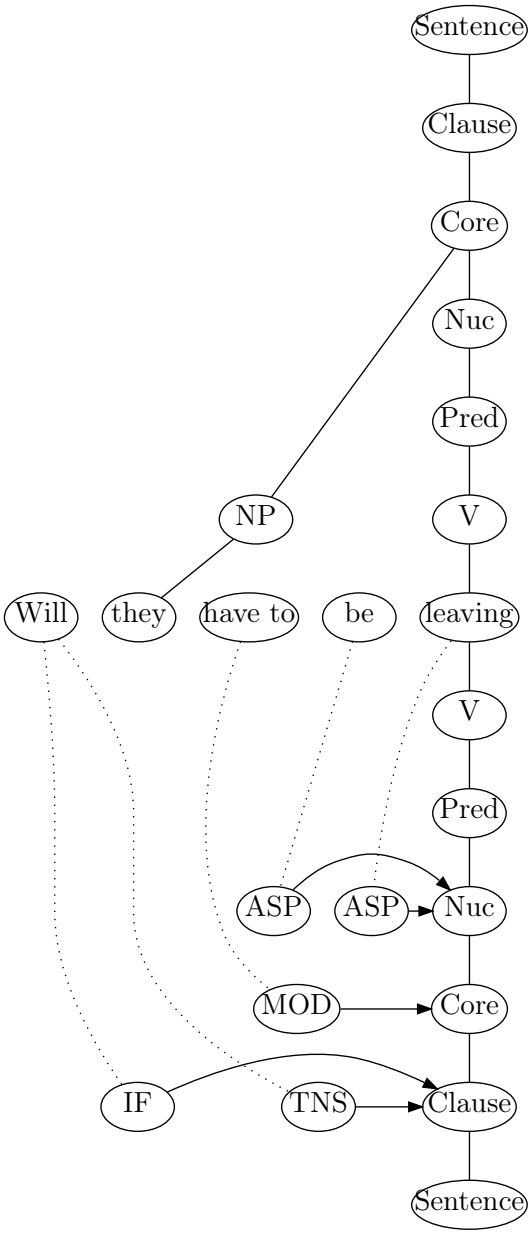
The program in written in the `rust` programming language. To run you will need to install `cargo` which provides a rust compiler.

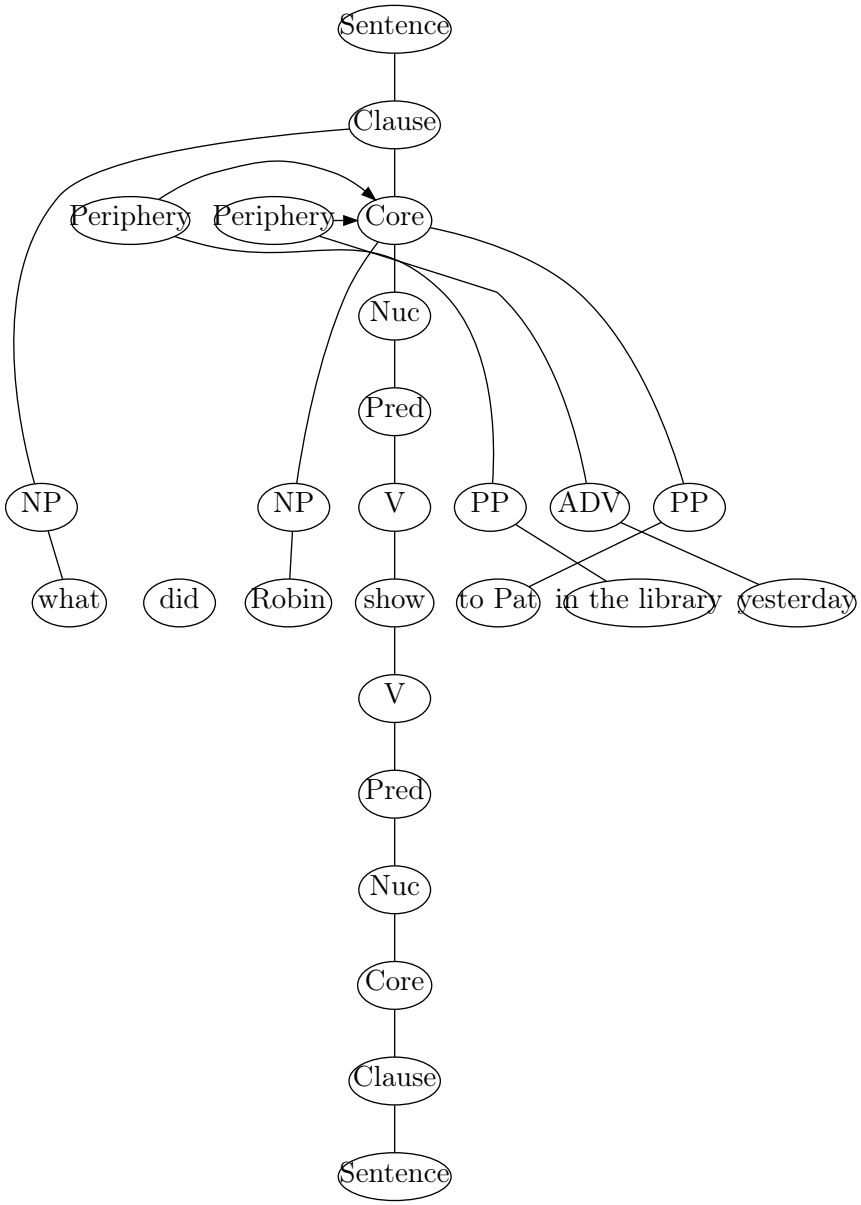
The program relies on compiling `dot` programs, which means you likely need `graphviz` installed on your computer.

To run the program, type

```
cargo run name_of_json_file.json name_of_output_file.svg
```

The above JSON examples are in the files `test1.json` and `test2.json` If everything goes well, you should have these images.


Sentence
Clause
Core
Nuc
Pred
V
NP
Will
they
have to
be
leaving
V
Pred
ASP
ASP
Nuc
MOD
Core
IF
TNS
Clause
Sentence



The program has not been extensively tested due to time constraints. However, the main features of operator projection and constituent project have been implemented. Peripheries have also been implemented, however for some reason likes to place the peripheries in non ideal spots. Operator projection has been also implemented, however, there is no hierarchy among

the operators, so it will not place `IF` below `TNS`, and instead will connect directly to the clause. Also, noun phrase structure has not been tested, however, due to its similarity with verb phrase structures, it should not be too hard to get working. Lastly, the program makes much more curved lines as opposed to the usual straight lines in typical RRG diagrams, however, this should not affect the semantics of diagram, only the appearance.