**Reo**

Fereidoon Mehri (95100275), Sobhan Mohammadpour (95106607)

PL Project

# How to Use:

Clone the repository, and add source files to the same folder; You need to use ` #lang reader "reo-reader.rkt"` for the first line. The next line specifies which network you want to run, and the output path. (the two networks compromise the two phases of the project, named “p1c” and “p2c” respectively.)

You can then add the two input nodes `a` and `b`, and the single output node `c`, each in their own respective lines. You can have multiple instances of the same network in a single file, separated by a single blank line.

Here is an example to start you off:

#lang reader "reo-reader.rkt"

p2c RESULT\_SEC2

a:("hello",null,"hello",null, null)

b:("world", null, "world",null, null)

c:("hello", "world","hello", "world", null)

a:("hello",null,"hello",null, null)

b:("world", null, "world",null, null)

c:(null,"hello", "world","hello", "world", null)

This example checks two instances of the second network type, and puts the result in a file named RESULT\_SEC2 in the current working directory.

# Dependencies:

Our project relies on the libraries and languages provided by <https://beautifulracket.com/>, which can be installed via the racket command line toolchain:

raco pkg install --auto beautiful-racket

Further instructions and description of this package can be found at <https://docs.racket-lang.org/br/> and https://docs.racket-lang.org/brag/. The more important dependency is `brag`, as we rely fully on it for lexing and prasing the language.

# Technical Specification of the Language:

One bonus benefit of using `brag` is that we get our formal grammar for free. You can check the `reo-parser.rkt` file, which I have catted here for your convenience ;) :

#lang brag

p1-program : /NEWLINE\* network DATA /NEWLINE+ instance (/NEWLINE /NEWLINE instance)\* /NEWLINE\*

@network : "p1c" | p2c

/instance : a /NEWLINE b /NEWLINE c

/a : /a (DATA /SEP)\* DATA?

/b : /b (DATA /SEP)\* DATA?

/c : /c (DATA /SEP)\* DATA?

See `brag`’s documentation on cutting and splicing for the `/`s and `@`s used, or simply ignore them. They have no effect on the grammar.

Data is a token compromised of at least one alphabetic character, a numeral, or of `-\_"`. Reserved keywords (like `a`) have priority in the lexer over DATA. Whitespace, the colon, and parentheses are ignored in the source code.

No comment syntax has been defined.

# Internals

All reo nodes and channels have been implemented in `reo-backend.rkt`, but some are not used in the two networks we have implemented in our language. However, it is now exceedingly easy to add new networks to the language; You only need to define the new network using the already implemented tools in `reo-backend.rkt`, like the functions `p1c` and `p2c`, and add the new network to the lexer’s list of reserved words (`reserved-terms`). Then you can add it to the parser’s network non-terminal. The lexer is entirely contained in `reo-reader.rkt`. Finally, adjust the `run-p` function in `reo-expander.rkt` to also handle your new network.

Further refinement of the language implementation is possible, so that adding new networks is easier. Pull requests are welcome.