Lab 5 - Relations

1. **Introduction**: My lab calculates the relations between sets. Inputs are loaded from a text file. The comparisons my program makes are reflexive, symmetric, transitive, antisymmetric, and equivalence. My program uses a library to make comparisons, but all the rest of the code is completely my own.
2. **Process**: My program makes extensive use of libraries for calculating sets. I tried to compensate for the ease of using a library by challenging myself to write my own code as idiomatic, or rusty, as I could.

The first step of my program is loading a vector from a file. Each line on the file has two arrays, a first array S, and a second array of tuples, L. I used custom types as writing the full type name out in strict typed rust is a hassle and makes for ugly code. Below are my custom types.

type LPair<'a> = Vec<(&'a str, &'a str)>; // {(a, b), ...}

type SSet<'a> = Vec<&'a str>; // {1, 2, 3}

type FullSet<'a> = (SSet<'a>, LPair<'a>); // (S, L)

For loading the file I wanted to have my code be fully functional. Rust has a feature where its function calls are often chained together instead of being nested like in other programs. This allows long chains of code to stay somewhat readable, as seen below.

fn sets\_from\_line<'a>(line: &'a str) -> Option<FullSet<'a>> {

    Regex::new(r"(?x)

        \[

            (?P<S> .\*  ) # Capture first set, S

        \] , \[\[        # Match delimitating commas and brackets

            (?P<L> .\*)   # Capture second set, L

        \]\] ")

    .unwrap()

    .captures(line)? // Apply regex to passed arg, line

    .iter()

    .map(|x| x.unwrap().as\_str()) // convert S and L to strings

    .collect::<Vec<&'a str>>()

    .get(0..3) // Grab all matches

    .map(|slice| // Match 0 dropped, because it's the entered string.

        (slice[1] // S

            .split(",")

            .collect::<Vec<&'a str>>()

        ,

        (slice[2] // L

            .split("],[") // Break into pairs

            .map(|x| x // Turn pairs into tuples

                .split(",")

                .collect::<Vec<&'a str>>())

            .map(|x| (x[0], x[1]))

            .collect::<Vec<(&'a str, &'a str)>>()

        ))

    )

}

The second step of my code is to take the vectors returned by the function above and passes them into the five relations functions. For this I created another function that takes the vector of tuples, L, and returns an output ready to be printed to console containing the results of each relation. I use an array of function pointers to make adding more set relations easy and elegant.

fn format\_results(set: LPair) -> String {

    let operations: Vec<(RelationOperation<&str>, &str)> = vec

        (Relation::is\_reflexive,     "reflexive"),

        (Relation::is\_symmetric,     "symmetric"),

        (Relation::is\_transitive,    "transitive"),

        (Relation::is\_antisymmetric, "antisymmetric"),

        (Relation::is\_equivalence,   "equivalence")

    ];

    operations.iter()

        .map(|(op, name)| format!("{:14}: {}", // Print to string,

            name, // The name of the function.

            op(&Relation::from\_iter(set.clone())))) // Function result (true/false)

        .collect::<Vec<String>>()

        .join("\n") // Each function goes on a new line.

}

My final step is to print the results. This and some light formatting is done in my main.

fn main() {

    std::fs::read\_to\_string("sets.txt")

        .unwrap()

        .replace(" ", "") // remove spaces

        .lines() // for each line

        .map(|line: &str| {

                println!("Line: {}", line); // Print relation set before relations table

                sets\_from\_line(line).unwrap() // Load sets from line

            })

        .map(|set| format\_results(set.1)) // set.1 is the LPair

        .for\_each(|x| println!("{}", x)) // Print each result

        ;

}

1. **Testing**: The program only takes input from a static file, so testing was easy, just to make sure the file is formatted correctly. However, incorrectly formatted files will cause problems. All brackets must have matching pairs and be in the proper location, but when formatted properly all Unicode and Ascii characters, besides commas and square brackets can be handled.

![A screenshot of a computer screen

Description automatically generated

The last line is my own, the second to last if Jon’s, and all others are pulled directly from the assignment document. All the inputs work as expected and all special characters I tried worked, see line 3.

1. **Results**: I checked my results with several classmates, and we all got identical results for both the provided inputs and all further inputs we added. Here are my results:

A screen shot of a computer program

Description automatically generated

1. **Conclusion**: I really pushed myself to learn rust better on this lab. Using libraries made it easier for me to focus in this way. However, the downside to using libraries and writing writing susinct rust is that my program is very short. Most of the time I look over my program and tidy up redundant or overly complex code to make it shorter, but I didn’t have to do that much on this lab. My code is already short and the only code I’d change is the regex code, which can be replaced with a spit. Originally I wanted to use regex to find every element in the L set but I could not get this to work.
2. **References and Acknowledgements:** All of the code in main.rs is my own, but none of the code outside it is. Lib.rs and relations.rs are completely written by Jmaarleveld. The relations Crate they made does all the relations calculations in my assignment. Note that the crate wouldn’t compile so I downloaded the source code from GitHub and pasted it into my own assignment.

<https://github.com/jmaarleveld/rust-relations>

1. **Extra credit**: None