Kevin Lopez

CIS352

05/09/21

Bonus project

Learning about typed racket is interesting as it completely allows for easier debugging. If an input does not follow the typing required by each function, then it throws the error clearly on the screen. Even though this does require an extra amount of work, it is useful in seeing where each part of the input breaks. It is very rigid and provides no room for mistakes which is important in programming in general instead of the newer languages. It was like my experience with Haskell in a previous class but after reading more, I realized that Racket prioritizes the program while Haskell prioritizes the type checking itself.

Typed Racket is different from regular Racket in some ways. Besides the obvious type checking, It is considered faster in some cases due to the automatic checking. There are also more libraries provided and it tells you the types you are missing. For the type annotation form, you would use the ‘ : ‘ syntax. On top of that form, there is also the define form that is used for the definition of variables in internal and top-level contexts. For let you need to provide a type annotation for every variable bound. For example, a let form would be implemented by ‘ (let ([x : Number 2]) (add1 x)) ’. An arrow → indicates when there exists a function type. Data structures within Typed Racket involved different types of variants, and they are represented by using union types. Using the examples provided in the documentation, I saw you can implement a Tree data structure with it. Using this, programs can run faster and with more efficient debugging with clear error codes. You do this by altering the structure definitions and the top-level definitions according to their types.