Project #5 Report

Group Members: Risheed Malatombee, Kyle Hallman, Amaan Vania, Karmit Patel

Table of Contents

Introduction

Problem Statement

During the phase where our team had just formed, we were given a list of projects to work on, the one that stuck out to us was project number 5. The reason why our group chose this project is because all of us were familiar with CTL formulas, and translations. We also took on this task because we saw it as a challenge to solve this problem. The problem in this case was determining how to take a CTL formula and translate it into Existential Normal Form and Positive Normal Form respectively. Through many trials and combinations, along with much research done in the translations and syntax we were able to complete this task and demonstrate a solution.

Summary of Sections

In this report, we will be demonstrating our findings and work done for project 5, this includes our design which constitutes of our class diagram, our sequence diagram and our architecture used before implementation. The design phase of our project was to have a good understanding of how we wanted objects interacting with other classes and methods, and developing an approach that we were all on the same page with in order to begin implementation. For the implementation phase of this project, we understood that there was patterns being used in the original ctl package that we were familiar with, namely the visitor pattern, and with that, we were able to walk through the code and as we implemented the translation functions, we documented each method for reader understanding. Lastly for the testing phase, we mention which packages we used and the advantages of doing so, how this led to our robust testing, and the overall testing coverage to ensure no loose ends were left untested.

Design

Class Diagram

For a better image of our class and sequence diagrams designated by the captions for each image, please refer to our github repository containing our diagrams for the design section.

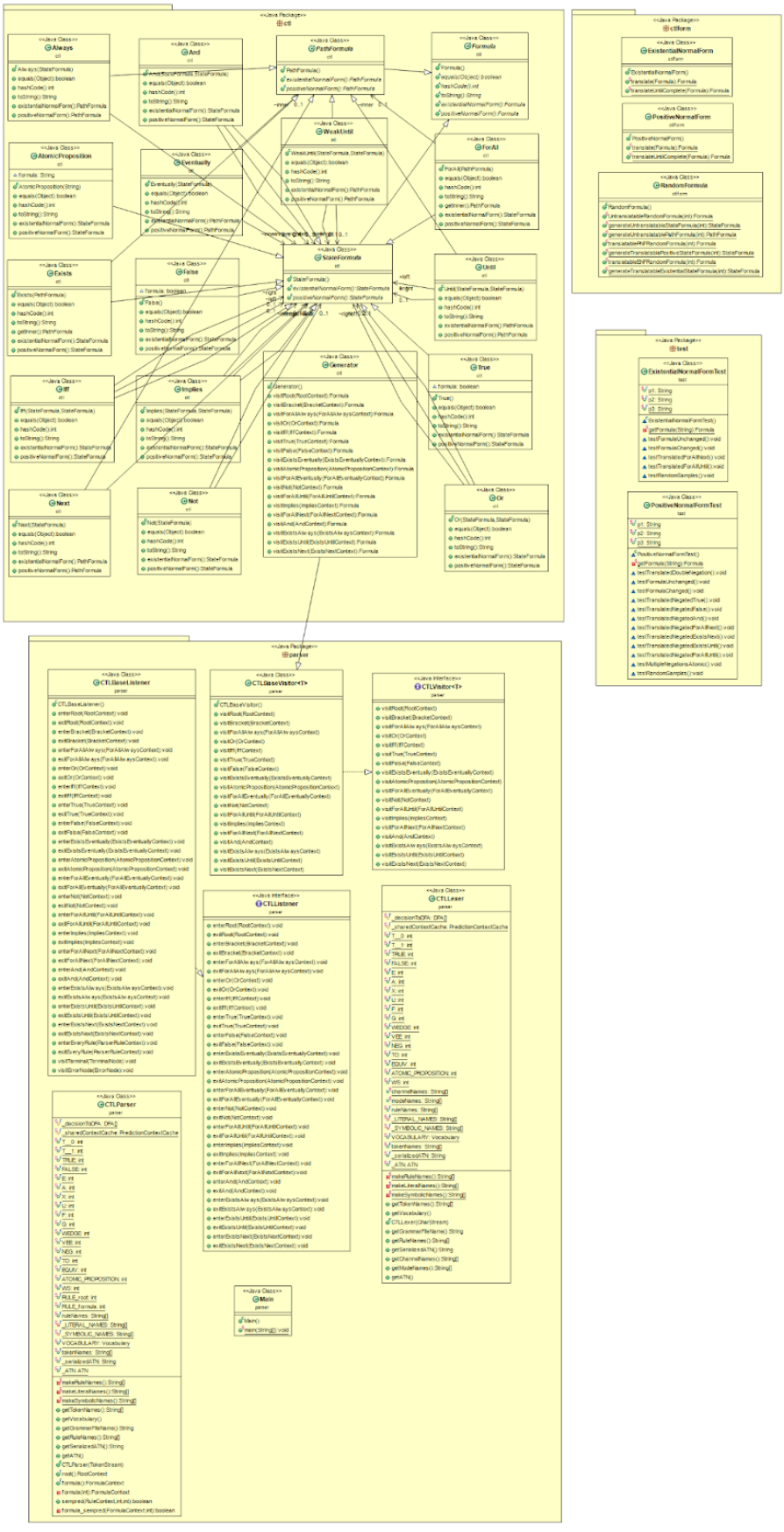


Figure 1: CTL and CTLForm package Class Diagram

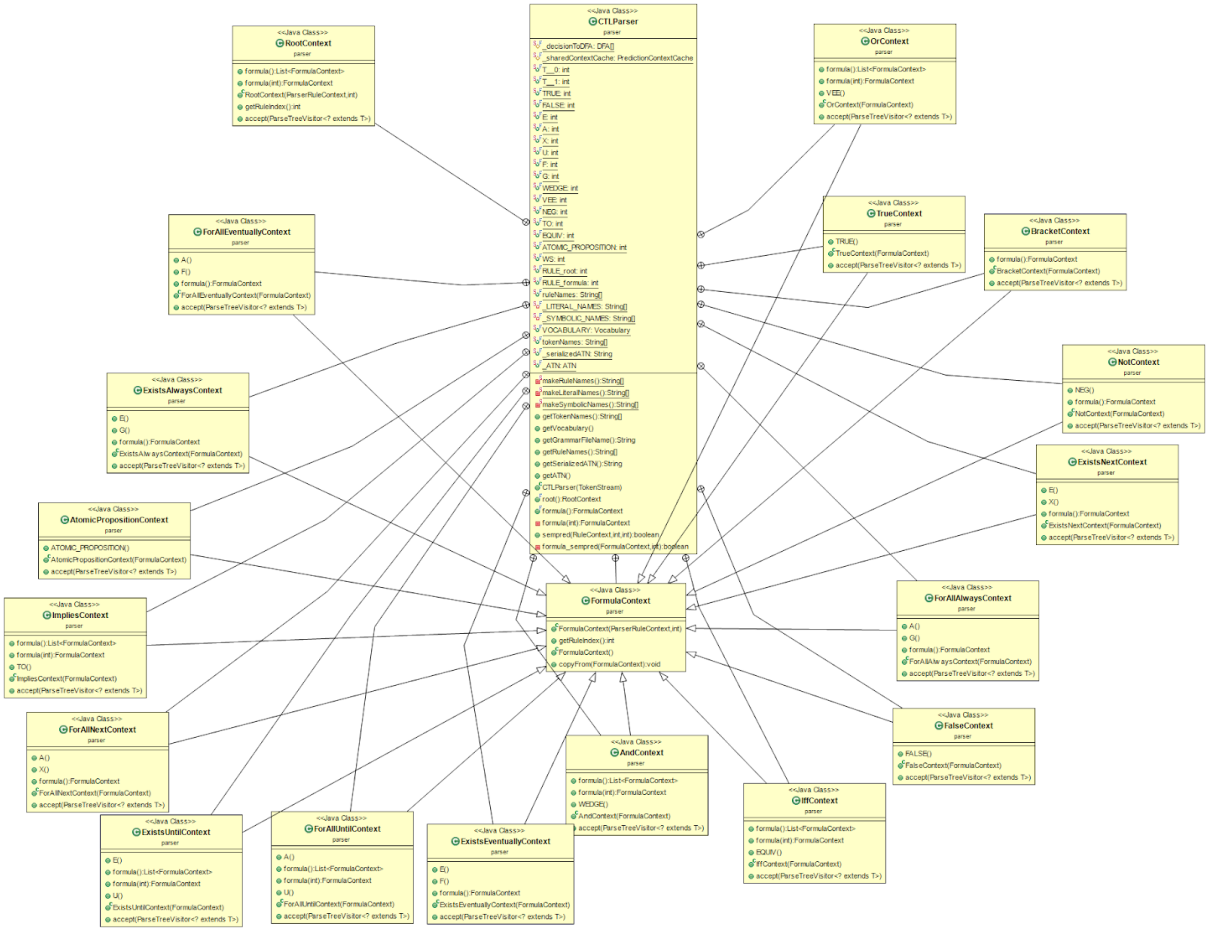


Figure 2: Parser package Class Diagram

Sequence Diagram

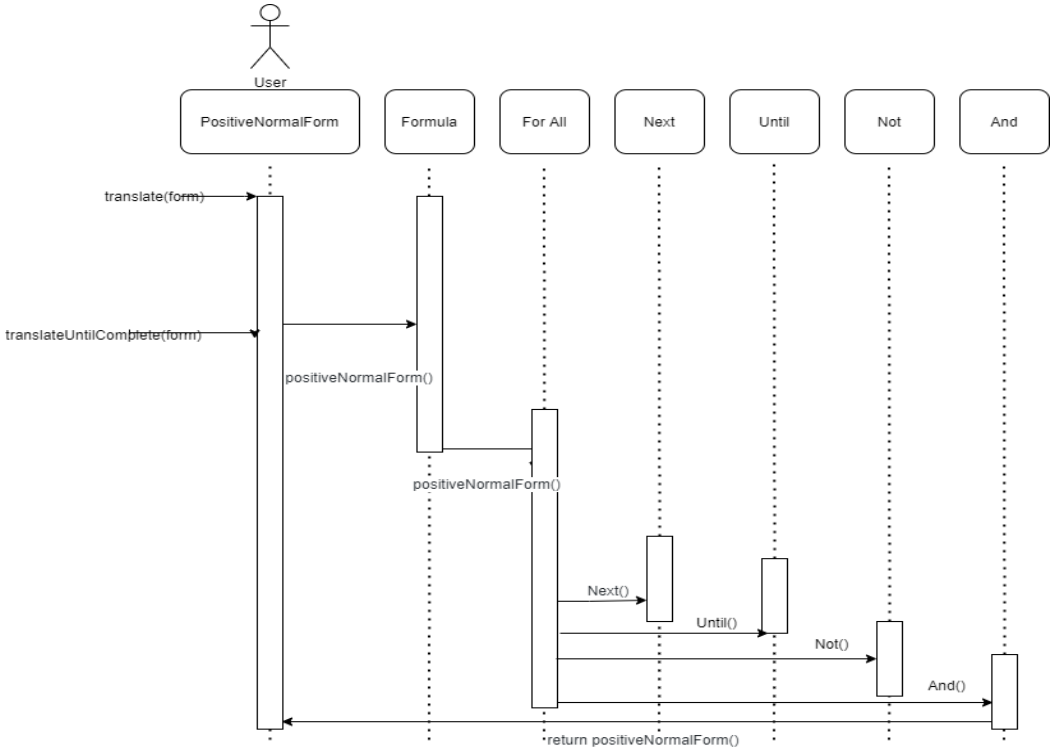
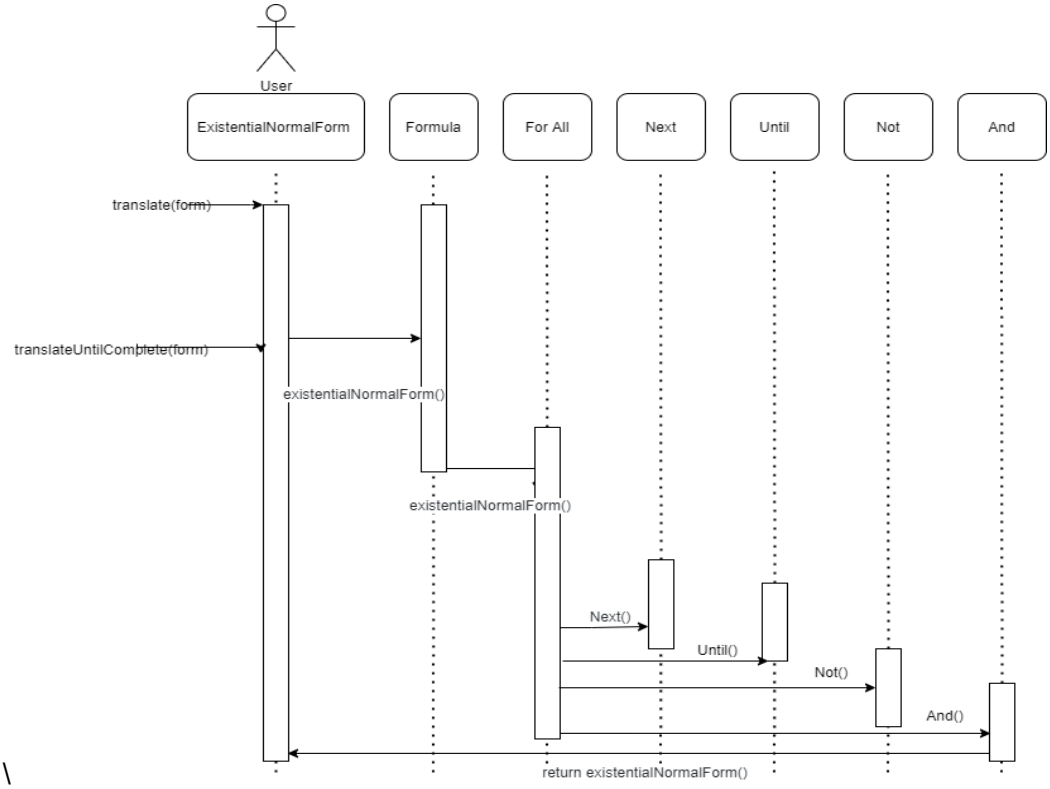


Figure 3: Positive Normal Form Sequence Diagram



\

Figure 4: Existential Normal Form Sequence Diagram

Implementation

Design Patterns

For the design patterns section, upon receiving the ctl package and other packages, we quickly noticed that the visitor pattern was implemented to reference different operators and their specific functionality and behaviour. That said, our approach to implementing the translation functions was to make use of the visitor class and the different operators and the inherited Formula class and subsequent classes as well. By doing so, we implemented the existentialnormalform() and positivenormalform() methods to use in the visitor pattern to be used in our new classes dedicated to translation to visit each operator. In the translation classes we made use of the Formula, StateFormula, and PathFormula classes to create objects that we were able to traverse operators to complete the translation function.

Code Walkthrough

As mentioned before, we started the implementation by exploring the visitor pattern implemented into the code already, from there for each operator we implemented new methods capable of doing each operations translation and then returning it upon call. Once this was done, we made classes dedicated to each translation type where we made use of each of the classes and their translate functions in order to perform the action. Each of those classes contained methods capable of performing each of the laws that were specified for existential and positive normal form formulas respectively, each of which made use of the existentialnormalform() and positivenormalform() methods that were implemented in each operation class.

Once done, we decided to ensure that our implementation was correct, hence we made a class to make random formulas to robustly test that our implementation left CTL formulas changed or unchanged if needed, which was then verified in the test package we developed. Furthermore in the implementation, we made additional testing dedicated to particular CTL formulas to ensure translations were done correctly.

Testing

Testing Packages

For the testing phase of the project, we decided to use JUnit package 5 for our testing as we were most familiar with and have used in the past. Furthermore, we decided to use this testing package since it incorporated the functions that we thought were relevant in testing our solution. Methods such as assert, assertEquals, assertTrue, assertFalse, and returning combinations of Formula object were used with the help of this package, and was used to properly document whether or not there were any bugs, and how to trace them back for review purposes.

Robust Testing

It was imperative in our testing phase that we covered every possible test that we could think of. Originally we started the testing phase by providing CTL formulas that we made

ourselves and then passing that information to the test and providing an expected output. The purpose of this was to ensure that our implementation correctly translated formulas from CTL to the appropriate forms. Furthermore, to make the testing absolutely robust when encountering any formula possible, we used the RandomFormula class to develop random formulas to test. We made tests to determine if the implementation would leave unchanged, or change formulas. Fortunately, our tests here came back 100% positive as our implementation knew when to do a translation and when not to do a translation from CTL to the respective forms. From here we could not do more from the random formulas as there was no way of confidently providing expected results for each of the tests, but we included as many formulas and their expected translation to ensure the work was done correctly.

Test Coverage

Test coverage wise, our coverage of the testing was generally 97%. We believe that the testing that was done was done properly, however, the fact that not all paths were explored to ensure that every combination possible translated is what we think is the reason why the test coverage was not 100%. In the future, had we had more time to work on this, we would have implemented some algorithm capable of partially providing expected results, to at the very least increase the test coverage.

Conclusion

Overall, although we believe that we did well, there's always room for improvement. However, with the design we formulated in regards to the class diagram and understanding the inheritance, calls, extensions and arguments passed, to the sequence diagram that tracked the movement and manipulation of objects, we believe that the design was thorough and easy to understand from a technical or non-technical point of view. On the idea of readability, we believe that our implementation best represents our design based on design pattern choice and flow of our implementation to ensure the functionality is as solid as possible. And finally, we ensured the implementation was solid with the use of robust testing, and testing any random or written formula possible to deliver the correct solution and appropriate coverage. In conclusion, as a team, we saw this project as both a lesson learned and a job well done. I believe that each of us have benefited from understanding the logic from a code perspective or the CTL formulas, and will definitely be an asset during the exam and going forward.