Ivan Stoilchev

345959

Abstract

Have you ever thought of becoming engineer? If the answer is yes, you can go through this document and see what it is to be in the shoe of a software engineer. This document describes the creation process of a propositional logic program.

ALE 1   
 Design Documentation

FHICT English Stream

# Assignment 1: Parse + Tree

Normally when you start new course a student is used to starting with the easy stuff and then going to assignments that are more difficult. What really surprised me is that in this course, we started from the hardest assignments and then we went to the easier assignments. Because of that, week 1 took me more time then what I was expecting. But there was also a positive side to this approach. After I did the first two weeks the rest weren’t that hard.   
In week 1 the hardest thing for me was the creation of the tree.

How did I approached the Assignment? First I thought about how should I split the formula and create a binary tree from it. I have decided to remove all extra signs and to leave only letters, the signs that are showing what calculation should be done and “,” for splitting every note. In my opinion this make the program more user friendly because if the user miss one bracket in the formula the tree will still be created. After that, I read the formula sign by sign and putting every sign in new Node. Every Node has Id, parent id, two child ids, sign and two bools – one that checks if the node contains a letter like A,B,C etc. and the other is the value of the node that I am getting from the truth table from week 2. I am using the “,” sign for knowing what are the two children of a Node, so this sign is critical for the formula. All of the operators have two children, only “not” has one.

After the creation of the tree, the tree is shown in the console.

# Assignment 2: Truth table + Hash code

For me this week was the hardest one. I already had the tree from week one. The truth table I have split in two parts. Creation of all the possible values for the variables a, b, c etc. and finding the answer for each combination of 0 and 1. The first part wasn’t that hard. I am just creating a two dimensional array that is the truth table rows and columns. After that I am filing it with just simple math. You can find the method in class Tree, Method: CreateTable(). The second part was a lot harder than the first one. After I had all the things: Truth table and the binary tree I wasn’t sure how to combine them. After a few hours of thinking how the logic should be I have come with a recursion method in my mind. First I am assigning the values from the truth table to the letters in the binary tree. Each time I assign them after that I am calling the recursion method called BM() in the Tree class. Basically what the method is doing is going through each node and checking its children. If those children are letters I am getting the value from them and doing the calculation depending on what the sign of the node is. If the children are not letters I am just calling the same method for the child that is not a letter.

After that the truth table is shown in the ListView1 in the gui.

# Assignment 3: Simplify

After doing assignment 2 simplifying wasn’t that hard. What I am doing is getting all the rows with answer 1 from the truth table. After that I am comparing each one of them with the others and if there is only one different value and all the others are the same I am just replacing it with “\*”. If there are more than 1 differences but they are all for columns that have \* in them I am just getting the one with \*. After that I am comparing the answers with the rows from the truth table that have an answer 0 and if some of them coincided I am just resetting them to their previous condition.

I am not sure if this is a future but I am showing all the possible simplifications. For example if I manage to simplify 2 times, the program is showing all the simplifications from the second time but also the simplifications that were not used after the first simplification. For example if in the end we have 1\*1, 0\*0, \*1\*. The \*1\* is simplification created after the second loop but because we couldn’t simplified 1\*1 and 0\*0 they are still shown in the end.

# Assignment 4: Normalize

Normalization was the easiest assignment in my opining. How I did it is by checking each row in the truth table and then each value in the rows. For each value depending on if its 1 or 0 am returning the letter with or without “not” in front of it and with “&” between this letter and the formula that I already have. Each time when I go to new row I am just adding “or”, old formula and the new formula from this row.

# Assignment 5: Nandify

For me understanding how Nandify was harder than making it work. First of all I took some time to see how I should covert each expression. After I found how this should be done I created a method that create an expression depending on the sign that it gets. For example ~(a) must become %(a,a). After that I used almost the same BM() method that I used in week 2 for going through the binary tree. The difference was that this time I wasn’t returning a value that is 0 or 1 but I was returning the letter for example A, B, C etc. And of course depending on the sign between two letters they were converted to the right expression.

# Software design

To be honest the software design is not the best. A lot can be improved. There is some code that can be replaced or even removed. Right now I have Tree, Nandify, Normalizer, Node and Simplifier classes. I have decided to split them to so many classes for two reasons. First to be more tidy and second if there is future work it will be easier if the main functionalities are split in different classes.

During the course I tried to make everything work the way it is supposed to work even if that means that the code will not be as optimized and clean as possible. That’s because in my opinion it is harder to make something work as it is supposed to work than clean some code after that.

# GUI

For the GUI I just tried to make it as user friendly as possible. I have split the different functions on different buttons. For the tables I am using ListView because in my opinion it looks better. And for the Normalization and Nandify I am writing the formulas in text fields. This way the user will be able to copy them and run them again through the calculator.

# Testing

To be honest I didn’t spend a lot of time on the testing. I just did few test for nand, normalization and simplification.

# Conclusions and future implementations

In concussion, I would say that ALE is the really important subject that improved my programmer skills. It definitely was the most important subject of this block. The assignment weren’t super easy, the subject was interesting, the teacher was awesome and I am pretty sure I have learned the most from this course. For the future implementation XOR could be included and ASCII expressions could be converted into regular DNS expressions.