Mark Gordon (H00228757)

Foundations – Part 3

This report will discuss: improvements made to part 2, problems which still exist from part 2 and how I have implemented the functions in part 3 along with any issues that exist with them.

Existing errors from previous parts

Unfortunately, the program will occasionally give the wrong result for some parts of part 2 while checking membership. This is either by giving 'has no value' when it should return 0, vice versa, or displaying the input message wrongly for example "Let x55 be x55" instead of "Let x55 be x33 \u2208 x55". This is detailed more in Part 2 Report.

Removal of the toString() method

In part 2 of the coursework, I had included a '*toString'* method which was clearly disallowed. So, to begin this section of the coursework I worked to remove this. This was simply done by calling '*.getValue().get(0);*' instead of '*.toString'*. '*getValue'* will return the ArrayList which holds the values and the .get(0) will simply return the first element which is the desired value. Unfortunately, Strings are still used in the program when writing to the file and for comparisons. However, I feel like this is still an improvement on part 2.

Fixing membership tests

Unfortunately, for Part 2 I left in an issue where membership tests would always result in false when checking membership of a variable/set/pair instead of just a number. This was because I was accidentally checking if the set contained the *ValueNode* itself instead of its value, this has now been fixed by simply checking for the value instead.

Function application

This works by first getting the value being applied in the application. It will then simply loop through the values of the set it is being applied to until a value of identical value is found

and then set the value to the output associated with this input value and break. If no match is found then it does not have a value.

Is Function

Checking if an input is a function works by first ensuring it is a set. If it is, then it will loop for each value in the set ensuring it is an ordered pair and adding the input value to the *TreeSet.* The number of values in the *TreeSet* is then compared to the original size of the set. If they are equal then it is a valid function as all inputs are unique. If not, then the inputs in the pairs were not unique and thus one was removed by the *TreeSet* meaning it was not a valid function.

Function Space

This is the most complicated function. It begins by looping through the 2 sets in the function and gets their variable values and the values contained within the set.

Next, we loop the pairs contained in the first set. It will take the current pair and loop through the pairs in the other set, creating new pairs with all the possibilities of their input/output values before adding the resulting pair to a set, and adding the set's variable and print out value to a map with their variable added to a list of variables that will be added to the final set. An example of this can be seen with the sets {0,1} and {4,6} producing the sets {(0,4)}, {(0,6)}, {(1,4)} and {(1,6)}.

Next what is done, is that the pair values produced in the previous part are now all looped through with their input values compared. If their input values are the same such as (0,4) and (0,6), they can't be combined to create a new function set {(0,4), (0,6)} as then an input of 0 would bring 2 results. If, however, the input values differ such as in the case of (0,4) and (1,6), then they can be combined with the new set {(0,4), (1,6)} added to the final set. This is repeated until all new sets are found and added.

After all new sets for the final set have been produced, their print out values are added to a *TreeSet* so that duplicates can be removed and put in order. After this, they are then added to the final set which is then added to the map and written to file.

Unfortunately, there are some issues with this function. In its current state it will only handle a max of 2 child values such as {1,2} and {4,6}. Handling sets like {1,2,3} and {4,5,6} will produce no value. On top of this issue, although it produces the correct values in the set for {1,2} and {4,5}, it will mix the order up in the set as it is ordered by ascii value, so for example {} will come last in the set despite being first in the test results.

Mark Gordon (H00228757)

Domain

This function will loop for the values in a set. It will ensure all values are ordered pairs and get their inputs. It will add all input values into a *TreeSet* to ensure order and remove duplicates. The print out value will be stored as a key in a map that connects to the variable value. This is so that when the *TreeSet* orders and removes duplicates, the variable value can be found again to be added to the final set.

Union

The union function will loop through both sets, getting their values and storing them into a *TreeSet* to ensure order and remove duplicates which may exist in both sets. Like the Domain function, it will store the print out value as a key in a map which connects to the variable value so that it can be reobtained later when adding to the final set.

Inverse

The inverse function will loop through the set, getting the ordered pairs and then getting their input and output values. After this, a new pair will be created with the input and output values switched before being added into the final set.

Diagonalization

For diagonalization, I will show an example of my understanding of it and how I have implemented the function. Here is the example:

The operation's value is a function F such that dom(F) ⊆ V1 and for every i such that i ∈ V1 it holds that:

– If V2(i)(i) has no value, then F(i) = V4.

– If V2(i)(i) has a value and V3(V2(i)(i)) has no value, then F(i) has no value.

– If V3(V2(i)(i)) has a value, then F(i) = V3(V2(i)(i))

Mark Gordon (H00228757)

In this example I will use the values:

V1 = {0, 1}

V2 = {(0, {(0, 1), (1, 1)}), (1, {(0, 0), (1, 0)})}

V3 = {(0, 1), (1, 0)}

V4 = 2

My working is this:

First, we work out V2(i)(i). I have interpreted this as looping through each value in the V1 set and doing function application with it. So, to obtain the first value:

V2(0)(0) = {(0, 1), (1, 1)}) (0) = 1

As this has a value, we know it's not the first rule. So, we test if V3(1) has a value, which it does, 0.

So, it's not the second rule and is in fact the third rule, the result is 0. We then create a pair using the original V1 value as the input, and the resulting value as the output. This creates (0,0) which is then added to the final set.

This process is then repeated with the next value in the V1 set which is 1.

V2(1)(1) = {(0, 0), (1, 0)} (0) = 0

V3(0) = 1

So, we obtain the pair (1,1) which is added to the final set giving a final answer of {(0, 0), (1, 1)}.

The programming of this function begins by getting the 4 values diagonalization is being applied to and then loops threw the V1 set values, trying to apply V2(i)(i) where i is the value in the set. If a null value is returned for reach value, then the final answer is given as a set of pairs containing (i, V4). If it has a value and V3(V2(i)(i)) has no value for each value in the set, then the result is that it has no value, and if it does have a value, then simply this value is returned.