Constraint Explanation :

I have used the Constraints explained on page number 308,309 in book Principles of Program Analysis by Nielson, Nielson and Hankin.

Constraints. Annotated types can contain arbitrary annotations and simple types can only contain annotation variables. To make up for this deficiency we shall introduce constraints on the annotation variables. A *constraint* is an inclusion of the form

$$\beta \supseteq \varphi$$

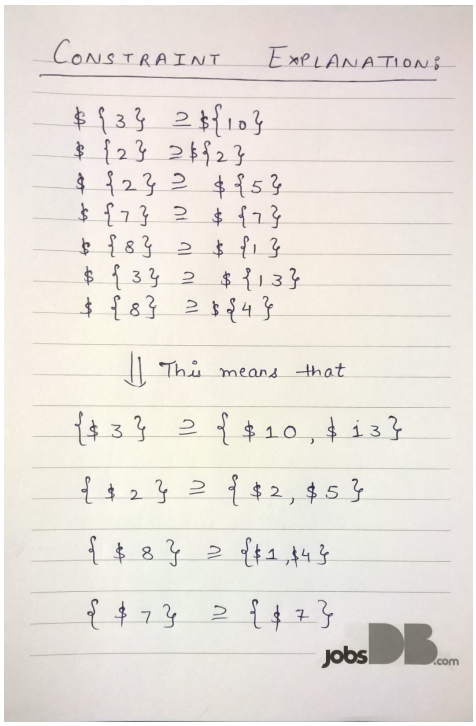
where β is an annotation variable and φ is a simple annotation. A constraint set C is a finite set of such constraints.

We shall write $\theta\ C$ for the set of inclusions obtained by applying the substitution θ to all the individual constraints of C: if $\beta \supseteq \varphi$ is in C then $\theta\ \beta \supseteq \theta\ \varphi$ is in $\theta\ C$. If C is a constraint set and θ is a simple substitution then also $\theta\ C$ is a constraint set.

When the program is executed, it will assign an annotation type to each function expression, function type and cons expression. When the types are unified and the substitutions are applied on them in the program, the annotations and substitutions are also applied at the same time.

For example, the constraint set generated from program t11.fun is :
{$3} >= {$10}
{$2} >= {$2}
{$2} >= {$5}
{$7} >= {$7}
{$8} >= {$1}
{$3} >= {$13}
{$8} >= {$4}

This can be explained as shown in the figure on the right.



{ $3 } >= { $10, $13}
{ $2 } >= { $2, $5}
{ $8 } >= { $1, $4}
{ $7 } >= { $7}

Failure Handling :

Whenever the program encounters below mentioned cases, it throws an exception, prints the error and terminates.
1) Undefined variable
2) types that cannot be unified like int, bool

3) Binary expression with different types
4) Type of condition is not BoolType
5) If and else blocks have different types
6) ConsEx.e2 is of invalid type i.e. the expression e2 is not of Cons or Nil type
7) Expression that does not map to the defined expressions in ast folder
8) Unification failure case.
    Example:
    8.1) TVar occurs in FunType
    8.2) TVar occurs in ListType

How to run the program :

1) From command line:
 java -cp path/to/antlr:path/to/Main Main <input_file>
  in which:
    path/to/antlr: the path to ANTLR library, e.g ./lib/antlr-4.5-complete.jar
    path/to/Main: the path to folder where Main class is built
    Main: name of Main class of this project
Example:
java -cp funla/lib/antlr-4.5-complete.jar:bin/ Main abs.fun

2) From Eclipse:
For the 1st part -> <filename> or <filename> false
For the 2nd part -> <filename> true

Use “true” as the 2nd parameter to the command line in order for the program to run the advanced part i.e. for the cons, nil and case expressions.

Test Case Organization:

Test cases related to the first part are inside the folder "basic" and test cases related to Advanced part are inside the folder “Advanced".
The output obtained after running this test cases is stored in corresponding txt files.