**Haskell project report**

**The basic Predicates :-**

**a) The explanation of the notation:**

**-Variable : is a data type that has certain atttributes defined as the alphabete from A to Z, and another two variable :MyVariable and Fall of certain specifications that would be explained later in the report**

**-Constant: is a data type that defines that something is a constant in the code, and has a constructor (Const)**

**-Term: is a data type to define a term , it contains two options , either a Constant or a Variable , so it has two constructors: Constant: that takes a Constand data type , and Variable: that takes a variable data type**

**-Predicate: is a data type to defina a predicate, it has a constructor (Pred) , that has a String to define the name of the predicate and a list of terms**

**-Goals: is a data type to define the goals in a certain Rule, it has a construcor Nil to define empty goals (for the facts) or Constructor (Goal) that takes a list of predicates**

**-Rule: is a data type that define the Rules, it has a contructor (Rulee) that takes a predicate which is the head of the rule and goals which is the body of the rule , if the goals are Nil then it is a fact, else it is a rule**

**-Answer: is a data type that has two aternatives either Yes or No**

**-Solution: is a data type to define a solution, it has a constructor Sol , it takes an answer and a lost of pairs, if the answer is No the list would be empty , if Yes, then the list would contain pairs of Variables, and their substitutions of type constant**

**all the types are deriving show and equal to be able to print them and to compare them with each other**

***b)UnifyWithHead :***

takes 2 inputs : 2 predicates and returns a solution which indicates which variables are bound and their substituted values.

It simply checks if the names of the 2 predicates are equal and then checks if their terms could be equated using the function equate.

Ex :

Main> unifyWithHead (Pred "Parent" [Variable X,Variable Y]) (Pred "Parent" [Constant (Const("myriame")) ,Constant (Const("ayman"))])

Sol Yes [(X,Constant (Const "myriame")),(Y,Constant (Const "ayman"))]

Main> unifyWithHead (Pred "Parent" [Variable X,Variable Y]) (Pred "male" [Constant (Const("myriame")) ,Constant (Const("ayman"))])

Sol No []

**c)applySolSubInBody: it takes a Solution and applies it in the body of the rule , it substitutes each variable in the solution with its value defined in the solution:**

**it has two helpers: equatePred: which takes a solution and the list of variables of each predicate , it applies equateItem to every list and for every pair of solution**

**equateItem takes a pair, it searches the list of Terms that was in the predicate and it matches the variables, if the variable in the pair was found, it substitutes the variable with its constant value**

**applySolSubInBody applies equatePred for every predicate in the Goals**

**Main> applySolSubInBody (Sol Yes [(X,Constant (Const 3))]) (Goal [Pred "" [Variable X, Constant (Const 5), Variable X, Variable Y]])**

**Goal [Pred "" [Constant (Const 3),Constant (Const 5),Constant (Const 3),Variable Y]]**

**Main> equatePred (Sol Yes [(X,Constant(Const 3))]) [Variable X, Variable Y]**

**[Constant (Const 3),Variable**

**Main> equateItem (X,Constant(Const 3)) [Variable X, Variable Y]**

**[Constant (Const 3),Variable Y]**

**d) *allSolutions :***

the input to this function will be a query and the knowledge base "rules" and will return a list of type solution representing all the possible solutions of the following query.

Main> allSolutions (Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D]) [Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil, Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]]) ]

[Sol Yes [(A,Constant (Const "slim")),(B,Constant (Const "mina"))]]

**Helpers for part b):-**

*1- Equate :*

Takes 2 inputs ( 1 input and return another function that takes the other input)

Inputs : 2 lists of terms and unify each term in the first list with the corresponding term in the second list, and returns a list of couples ( the term from the first list and the corresponding term)

If the 2 corresponding terms are 2 constants it checks if the 2 terms are equal or not, if not equal or if the 2 lists are not of the same length the function ends the returned list with (Fall, Constant x) to say that the 2 lists can not be equal.

Ex :

Main> equate [Variable X,Variable Y] [Constant (Const("myriame")) ,Constant (Const("ayman"))]

[(X,Constant (Const "myriame")),(Y,Constant (Const "ayman"))]

Main> equate [Variable X] [Constant (Const("myriame")) ,Constant (Const("ayman"))]

[(X,Constant (Const "myriame")),(Fall,Constant (Const "ayman"))]

*2-checkLast :-*

takes 1 input : a list of coupled terms and checks if the last element in the list is (Fall with anything else ( which means that the lists couldn't be equated ) and the query will fail.

**Helpers for part c):-**

**EquateItem and equatePredicate and they are explained above with examples**

**Helpers for part d):-**

* ifVariable :: Term a -> Bool

**It takes a term a return true if it is variable and false if it is constant**

* countVariable :: Num a => [Term b] -> a

**It takes a list of terms and returns the number of variables in it**

* unifyWithHeadRule :: Eq a => Predicate a -> Rule a -> Goals a

**It takes a predicate and rule and if the predicate unifies with the head of the rule then it apply applySolSubInBody on the goals of the rule and the new goals**

**Example :-**

unifyWithHeadRule (Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D])

(Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]]))

Goal [Pred "f2" [Variable Z,Constant (Const "maha")]]

* found :: (Eq a, Eq b) => (b,a) -> [(b,a)] -> Bool

**it takes one term of a solution body (Variable,Term a) and complete solution body[(Variable,Term a)] and return true if the first input found in the second list and false if it is not found**

* foundVariable :: Eq a => (a,b) -> [(a,c)] -> Bool

**it takes the same input as found but it checks for the variable part only if it is existed it returns true otherwise returns false**

* soluAnd :: (Eq a, Eq b) => [(a,b)] -> [(a,b)] -> [(a,b)]

**takes two bodies of solutions**

**and checks as the following for every element in the first list and the second list ,Its aim to return the second list if every element in the first list found in the second list or if the elements in the first list does not found in the second list at all**

**foundVariable | found**

1. **0 continue cheking for the first list**
2. **1 Impossible case**
3. **0 empty list(un**

**-wanted case)**

**1 1 continue cheking for the first list**

* soluAndReal :: (Eq a, Eq b) => [(b,a)] -> [[(b,a)]] -> Bool

It takes a list of solution body and a list of lists of solution bodies and apply solAnd for the first input list on every list in the second input and if the first input satisfy any of the lists of the second list it returns true otherwise returns false .

* soluAndReal2 :: (Eq a, Eq b) => [[(b,a)]] -> [[(b,a)]] -> [[(b,a)]]

which takes two lists of solutions bodies and each body considered as a list and performs an operation like what &

it applies solAndReal for every list in the first input on the second input and it returns true then that list will be considered as a body of a successful solution and if not it is not successful solution

* isFact :: Eq a => Rule a -> Bool

It takes a Rule and checks if it is a fact or not ( Its Goals equals Nil )

* soluFact :: Eq a => Predicate a -> [Rule a] -> [Rule b] -> [Solution a]

it takes a predicate and list of rule and find the solution only if the predicate satisfies a fact in the knowledge base (list of rules )

* flat :: [[a]] -> [a]

It flattens a list of lists

* getGoalBody :: Goals a -> [Predicate a]

It takes a goal and returns its body (list of predicates ) and if it is empty it returns empty list

* solu1 :: Eq a => [Rule a] -> [Rule a] -> Predicate a -> [Solution a]

It is the building unit of all solutions which takes the knowledge base (list of rules ) two times (the first and the second input ) it plays with one and fix the other and takes a predicate to get its solution and searches all the knowledge base for that predicate if it found it as a a fact it unifyWithHead the fact with the predicate and add the solution to the resultant list of solutions.In case of rule ,it unifyWithHeadRule and take the goals and apply the same methods on each predicates on them to get a list of all possible solution rejected or accepted

Example :-

Main> solu1 [Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil,

Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])

] [Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil,

Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])

] (Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D])

[Sol No [],Sol Yes [(Z,Variable W)]

* getSolutionBody :: Solution a -> [(Variable,Term a)]

It takes a solutions and returns its body if the answer is yes and returns empty list if the answer is No

* getSolutionBta37btSolutions :: [Solution a] -> [[(Variable,Term a)]]

it takes a list of the solution and returns a list of the bodies of that solution each body considered as a a list of (Variable ,Term a)

* filterLength :: Int -> [a] -> Bool

It takes an integer and a lsit and returns true if the length of the list equals that integer

* NegateFilterLength :: Int -> [a] -> Bool

**It negates** filterLength --🡪 not(filterLength)

* tryAllSolutions :: Eq a => Predicate a -> [Rule a] -> [[(Variable,Term a)]]

**It is the second building unit of allSolutions which takes the same input for all solutions but it performs solu1 function and tries to filter the solutions as the following**

**It divides the resultant solution list to two lists one with the length of the number of variables in the input predicate and the other list for the rest of the resultant solution list and performs soluAndReal2 on that two list and the result list is the correct solution**

**.But, the result is the body only of the solution without (Yes or No)**

* formSolution :: [(Variable,Term a)] -> Solution a

it takes a body of solution and form Yes solution with it

* formAllSolutions :: [[(Variable,Term a)]] -> [Solution a]

**it generalize what formSolution do but on a complete list of bodies to produce list of solutions**

* getVariableBody :: Term a -> Variable

**it Takes a term and returns the variable inside it**

* swapVariables :: [Term a] -> [(b,c)] -> [(Variable,c)]

**It takes a term and body of solution and replaces the variables in the body with variables in the term**

* allSolutionsHelper :: Eq a => Predicate a -> [Rule a] -> [Solution a]

**The third building unit for allSolutions ,**

**All what it does is calling formAllSolutions on the resultant solutions bodies from tryAllSolutions**

* sh2lbConstantM3Variable :: Term a -> Term b

it Takes a term but of constants and replace the first constant by a variable (A known variable called MyVariable)

* yarb25rCaseNotHandled :: Predicate a -> Predicate a

It takes a predicate as an input and its body will contain constants only so it replaces the first constant with (MyVariable) and returns the new predicate

* allSolutionHelper2MnGher25rCase :: Eq a => Predicate a -> [Rule a] -> [Solution a]

**It performs what allSolutions do but without handling the case when all the terms in the predicate are constants only**

* foundTrue :: [Bool] -> Bool

It takes a list of Booleans and returns true only if one of the elements of the list is true and returns false otherwise

* getFirstPredicateConstant :: Predicate a -> Term a

**which takes a predicate and return the first term in its body and it will be for sure constant in my implementation**

* filterVariableConstantSolution :: [(a,Term b)] -> [(a,Term b)]

It takes a body of solutions and returns only the elements in it which considered as variables equals constants

* getSolutionsHeadRule :: Eq a => Predicate a -> [Rule a] -> [Solution a]

it takes a predicate and the knowledge base and if found a rule Not fact satisfies the predicates it unifyWithHead the predicate with the head of the rule and returns that solution

* formHeadSolutions :: Eq a => Predicate a -> [Rule a] -> [Solution a]

It selects the correct solutions only from solutions produced from getSolutionHeadRule(the solutions which contaibs variable equals constant only )

The last three functions to handle the case which constants are found in the head of a rule not all variables

* allSolutionHelper2\_B\_25rCase :: Eq a => Predicate a -> [Rule a] -> [Solution b]

it performs what allSolution do but in case only that the input predicate contains constants only

* appendSolutions :: [Solution a] -> [Solution a] -> [Solution a]

It appends two Lists of Yes Solutions

* allSolutionsReal :: Eq a => Predicate a -> [Rule a] -> [Solution a]

**It performs what allSolutions do handling the case of all the terms in the input predicate are constants**

* allSolutions :: Eq a => Predicate a -> [Rule a] -> [Solution a]

It calls allSolutionsReal **plus handling the case when constants found in a head of a rule (not fact ) .**

The Knowledge Base

**-->[Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**-->allSolutions (Pred "male" [Variable A]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "female" [Variable A]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "parent" [Variable A, Variable B]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "parent" [Variable A, Constant (Const "reem")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "parent" [Variable A, Constant (Const "alex")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "father" [Variable A, Variable B]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Variable D, Variable F]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Constant (Const "amira"), Variable F]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Constant (Const "amira"), Constant (Const "alex")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Variable D, Constant (Const "slim")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Constant (Const "remon"), Constant (Const "azmy")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**--> allSolutions (Pred "daughterFather" [Constant (Const "reem"), Constant (Const "azmy")]) [Rulee (Pred "male" [Constant (Const "timmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "alex")]) Nil,**

**Rulee (Pred "male" [Constant (Const "slim")]) Nil,**

**Rulee (Pred "male" [Constant (Const "azmy")]) Nil,**

**Rulee (Pred "male" [Constant (Const "remon")]) Nil,**

**Rulee (Pred "female" [Constant (Const "amira")]) Nil,**

**Rulee (Pred "female" [Constant (Const "reem")]) Nil,**

**Rulee (Pred "female" [Constant (Const "wanda")]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "slim")),(Constant (Const "amira"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "wanda")),(Constant (Const "timmy"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "reem"))]) Nil,**

**Rulee (Pred "parent" [(Constant (Const "azmy")),(Constant (Const "remon"))]) Nil,**

**Rulee (Pred "father" [Variable X, Variable Y]) (Goal [Pred "male" [Variable X], Pred "parent" [Variable X, Variable Y]]),**

**Rulee (Pred "daughterFather" [Variable X, Variable Y]) (Goal [Pred "father" [Variable X, Variable Y], Pred "female" [Variable X]])]**

**-----> More Test Cases:**

**--> Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil,**

**Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])**

**allSolutions (Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D]) [Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil,**

**Rulee (Pred "f" [(Constant (Const "slim")), Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])**

**]**

**-->Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil**

**Rulee (Pred "f" [Variable U, Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])**

**allSolutions (Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D]) [Rulee (Pred "f2" [Variable W, Constant (Const "maha")]) Nil,**

**Rulee (Pred "f" [Variable U, Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])**

**]**

**-->allSolutions**

**(Pred "f" [Variable A, Constant (Const "maha"), Variable B, Variable C, Variable D])**

**[Rulee (Pred "f2" [Constant (Const "maha"), Constant (Const "maha")]) Nil,**

**Rulee (Pred "f" [Variable U, Variable X, (Constant (Const "mina")), Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant (Const "maha")]])]**

**-->allSolutions (Pred "f" [Variable A, Constant (Const "maha"),Variable B, Variable C, Variable D]) [Rulee**

**(Pred "f2" [Variable X, Constant (Const "maha")]) (Goal [Pred "f3" [Variable X]]), Rulee (Pred "f3" [Variable X]) (Goal (Pred "f4" [Variable X])),**

**Rulee (Pred "f4" [Constant (Const "maha")]) Nil,**

**Rulee (Pred "f" [Variable U, Variable X, Constant (Const "mina"),Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant(Const "maha")]])]**

**-->allSolutions (Pred "f" [Variable A, Constant (Const "maha"),Variable B, Variable C, Variable D]) [Rulee**

**(Pred "f2" [Variable X, Constant (Const "maha")]) (Goal [Pred "f3" [Variable X]]), Rulee (Pred "f3" [Variable X]) (Goal (Pred "f4" [Variable X])),**

**Rulee (Pred "f4" [Variable X]) Nil,**

**Rulee (Pred "f" [Variable U, Variable X, Constant (Const "mina"),Variable Y, Variable Z]) (Goal [Pred "f2" [Variable Z, Constant(Const "maha")]])]**