Robinson Unification Algorithm in F#
Learning version

This is a learning version of the Robinson unification algorithm. A final different version will become part of a library for doing AST transformations.

I wrote the code in this un-factored manor to make it easy to understand without deciphering it from some other reference. While I know from personal experience that the unification algorithm is not easy to learn, it can be even harder without someone guiding you. Hopefully this version is clear enough to learn from on its own.

Unification Algorithm uses

1. AST transformations
2. Type Inference
3. Term rewriting
4. Theorem proving
5. Prolog
6. Natural language processing
7. Pattern matching
8. Combinatorial test case generation
9. Extract sub structures from structured data such as an XML document
10. Symbolic computation i.e. calculus

There are several forms of the unification algorithm; the one here is associated with syntactic unification.

For a quick introduction to the unification algorithm see Wikipedia. You can skip the parts that mention the semantic or higher order forms.

See: Wikipedia: Unification

```
module UnificationLearning

// Unification algorithm
//
// Portions Copyright (C) 2012 by Tomas Petricek
// Copyright (C) 2012 by Eric Taucher
// License: Creative Commons BY-SA Version 3.0
//
// This code is derived from a StackOverflow answer by Tomas Petricek
// See: http://stackoverflow.com/a/9525471/1243762
//
// In short, this is a reference and learning version, not a production version.
//
// This is an implementation of the Robinson algorithm without using unify to update unify's structures.
// If you look carefully at most functional implementations of the Robinson unification algorithm you will see that they
// are recursive and that they also use the unify function to update the internal structures.
// It would be like using regular expressions inside of a regular expression engine.
// You have to know regular expressions in order to understand the regular expression engine.
// Works if you already know it, but for a first introduction, it is quite confusing.
//
// Since this is my first use of a functional language, and I needed to understand the unification algorithm in detail,
// I did this implementation, which is actually the last version in a series of axiomatic progressions;
// thus the names with 900.
//
// Because this was done as an axiomatic progression, it has full traceability for finding bugs.
// Because of the way the sub functions are implemented it allows one to understand what is going on
// without having to compare the internal data structures before and after a function call.
// Because of the way the sub functions are implemented, I can give meaningful names to each step,
// instead of trying to factor out the steps in my head and keep track of the level of recursion.
```

```
type Expr900 =
    | Value900 of int
    | Variable900 of string
    | Structure900 of string * Expr900 list

type Substution900 =
   | ValueSubstution900 of string * int
   | VariableSubtution900 of string * string
   | StructureSubtution900 of string * string * Expr900 list

// Define occurs check function
let rec occursIn var xs =
    match xs with
    | (Value900 y)::xs1                  -> occursIn var xs1
    | (Variable900 y)::xs1 when var = y  -> true
    | (Variable900 y)::xs1 when var <> y -> occursIn var xs1
    | (Structure900(s,l))::xs1           ->
        let result = occursIn var l
        match result with
        | true                           -> true
        | false                          -> occursIn var xs1
    | []                                 -> false
    | _                                  -> failwith "How did we get here?"
```

```
let unify900 xs ys =
    // Define unify function
    let rec unify xs ys mgu =
        // Define replace function
        let replace sub x y mgu =
            // Define replacePattern function
            let rec replacePattern sub xs =

                // Define replacePatternUtil function
                let rec replacePatternUtil sub xs acc =
                    match sub with
                    | ValueSubstution900(var,value) ->
                        match xs with
                        | (Structure900(f,l))::xs1                -> replacePatternUtil sub xs1 ((Structure900(f,(replacePattern sub l)))::acc)
                        | (Variable900 x)::xs1      when x = var  -> replacePatternUtil sub xs1 ((Value900 value)::acc)
                        | (Variable900 x)::xs1                    -> replacePatternUtil sub xs1 ((Variable900 x)::acc)
                        | (Value900 x)::xs1                       -> replacePatternUtil sub xs1 ((Value900 x)::acc)
                        | []                                      -> (List.rev acc)
                        | _                                       -> failwith "How did we get here?"
                    | VariableSubtution900(var1,var2) ->
                        match xs with
                        | (Structure900(f,l))::xs1                -> replacePatternUtil sub xs1 ((Structure900(f,(replacePattern sub l)))::acc)
                        | (Variable900 x)::xs1      when x = var1 -> replacePatternUtil sub xs1 ((Variable900 var2)::acc)
                        | (Variable900 x)::xs1                    -> replacePatternUtil sub xs1 ((Variable900 x)::acc)
                        | (Value900 x)::xs1                       -> replacePatternUtil sub xs1 ((Value900 x)::acc)
                        | []                                      -> (List.rev acc)
                        | _                                       -> failwith "How did we get here?"
                    | StructureSubtution900(var,f,l) ->
                        match xs with
                        | (Structure900(f,l))::xs1                -> replacePatternUtil sub xs1 ((Structure900(f,(replacePattern sub l)))::acc)
                        | (Variable900 x)::xs1      when x = var  -> replacePatternUtil sub xs1 ((Structure900(f,l))::acc)
                        | (Variable900 x)::xs1                    -> replacePatternUtil sub xs1 ((Variable900 x)::acc)
                        | (Value900 x)::xs1                       -> replacePatternUtil sub xs1 ((Value900 x)::acc)
                        | []                                      -> (List.rev acc)
                        | _                                       -> failwith "How did we get here?"
                // Call replacePatternUtil function
//                printfn "Before replacePatternUtil: substution: %A pattern: %A" sub xs
                let result = replacePatternUtil sub xs []
//                printfn "After  replacePatternUtil: substution: %A pattern: %A \n" sub result
                // return result
                result
```

```
    // Define updateMgu function
    let rec updateMgu sub mgu =
        // Define replaceMguUtil function
        let rec replaceMguUtil sub mgu acc =
            match mgu with
            | ((ValueSubstution900(mguVar,mguTerm))::xs1)
                -> replaceMguUtil sub xs1 ((ValueSubstution900(mguVar,mguTerm))::acc)
            | ((VariableSubtution900(mguVar1,mguVar2))::xs1)
                -> replaceMguUtil sub xs1 ((VariableSubtution900(mguVar1,mguVar2))::acc)
            | ((StructureSubtution900(mguVar,f,l))::xs1)
                -> replaceMguUtil sub xs1 ((StructureSubtution900(mguVar,f,(replacePattern sub l)))::acc)
            | []
                -> (List.rev acc)
        // Call replaceMguUtil function saving result
        let replacedMgu = replaceMguUtil sub mgu []
        // append the new substution to the updated mgu
        // While this is not a best practice,
        // its obvious that sub is being appended to the mgu
        replacedMgu@[sub]
    // Call unify function
    unify (replacePattern sub x) (replacePattern sub y) (updateMgu sub mgu)

// Define unifyStructure function
// Note: The mgu from the structure, i.e. structureMgu, is passed as the mgu for the remaining unfication.
// If not, then how would the final mgu be able to include the mgu from the substructure?
let unifyStructure s1 s2 xs ys mgu =
    let (structureResult,structureMgu) = unify s1 s2 mgu
    match structureResult with
    | true -> unify xs ys structureMgu
    | false -> (false, [])
```

```
    // Define unifyUtil function
    let unifyUtil xs ys mgu =
        match xs with
        | (Value900 x)::xs1 ->
            match ys with
            | (Value900 y)::ys1          when x = y                                    -> unify xs1 ys1 mgu
            | (Value900 y)::ys1          when x <> y                                   -> (false, [])
            | (Variable900 y)::ys1                                                     -> replace (ValueSubstution900(y,x)) xs1 ys1 mgu
            | (Structure900 _)::ys1                                                    -> (false, [])
            | []                                                                       -> (false, [])
            | _                                                                        -> failwith "How did we get here?"
        | (Variable900 x)::xs1 ->
            match ys with
            | (Value900 y)::ys1                                                        -> replace (ValueSubstution900(x,y)) xs1 ys1 mgu
            | (Variable900 y)::ys1       when x = y                                    -> unify xs1 ys1 mgu
            | (Variable900 y)::ys1       when x <> y                                   -> replace (VariableSubtution900(x,y)) xs1 ys1 mgu
            | (Structure900(s1,l1))::ys1 when occursIn x l1                            -> (false, [])
            | (Structure900(s1,l1))::ys1                                               -> replace (StructureSubtution900(x,s1,l1)) xs1 ys1 mgu
            | []                                                                       -> (false, [])
            | _                                                                        -> failwith "How did we get here?"
        | (Structure900(s1,l1))::xs1 ->
            match ys with
            | (Value900 y)::ys1                                                        -> (false, [])
            | (Variable900 y)::ys1       when occursIn y l1                            -> (false, [])
            | (Variable900 y)::ys1                                                     -> replace (StructureSubtution900(y,s1,l1)) xs1 ys1 mgu
            | (Structure900(s2,l2))::ys1 when (s1 = s2) && (l1.Length = l2.Length)     -> unifyStructure l1 l2 xs1 ys1 mgu
            | (Structure900(s2,l2))::ys1 when (s1 <> s2) || (l1.Length <> l2.Length)   -> (false, [])
            | []                                                                       -> (false, [])
            | _                                                                        -> failwith "How did we get here?"
        | [] -> (true,  mgu)
    // Call unifyUtil function
    unifyUtil xs ys mgu

// Call unify function
unify xs ys []
```

```
//#######################################

// Test x | f(x)

let unify901 =
    let expr001 = [Variable900("x")]
    let expr002 = [Structure900("f",[Variable900("x")])]
    printfn "Unify test: unify901 - \n\t expr 1: %A \n\t expr 2: %A" expr001 expr002
    let result = unify900 expr001 expr002
    let res, mgu = result
    printfn "\t result: %b  mgu: %A \n" res mgu

// OK, that works

//---------------------------------------

// Test a,f(b,b) | 1,f(1,b)

let unify902 =
    let expr001 = [Variable900("a");Structure900("f",[Variable900("b");Variable900("b")])]
    let expr002 = [Value900(1);Structure900("f",[Value900(1);Variable900("b")])]
    printfn "Unify test: unify902 - \n\t expr 1: %A \n\t expr 2: %A" expr001 expr002
    let result = unify900 expr001 expr002
    let res, mgu = result
    printfn "\t result: %b  mgu: %A \n" res mgu

// OK, that works

//---------------------------------------

// This is called from C# as
//
//    private static void UnificationTest()
//    {
//        UnificationLearning.tests();
//
//        Console.WriteLine("Press Enter to continue. ");
//        Console.ReadLine();
//    }

let tests() =
    unify901
    unify902
```