Type Inferencing in MPL

August 3, 2017

1 Solving Type Equations

As seen in the previous section, a type equation is defined as one of the following two things:

$$\begin{aligned} \textit{Type Equation} \;:=\; & \textit{Type Variable} \;=\; \textit{Type Expression}, \;\; \textit{Or} \\ & \forall u_1, \ldots, u_m \;\; \exists\, v_1, \ldots, v_n \;.\; \{\textit{Type Equation}\} \end{aligned}$$

Tha above definiton of type equation can be expressed by Haskell data type `TypeEqn` as below:

```
type UniVars    = [String]
type ExistVars  = [String]

data TypeEqn =    TSimp  (Type,Type)
                | TQuant (UniVars,ExistVars) [TypeEqn]
```

- The non recursive/simple branch of the definition is representend by the constructor `TSimp` and the recursive branch containing the quantifiers is represented by the constructor `TQuant`.
- The `TSimp` constructor takes a pair of type expressions as inputs represented by `Type` which is a data type defining the valid MPL types, both concurrent and sequential.
- First argument of the `TQuant` constructor is a pair containing the list of universally and existentially quantified variables represented by the type snyonyms `UnivVars` and `ExistVars` respectively. Both `UnivVars` and `ExistVars` are type synonymous to a list of strings.
- The second argument of the type constructor `TQuant` is a list of type equations. Thus, the data type `TypeEqn` is a recursive data type.

1.1 Solving

Solving type equations involves two steps:

- First step of the solution of type equation deals with the transformation of the type equations to a data structure called **package**. The package data structure and the transformation of type equations to the package data structure is described in Section .
- The generated packages are reduced using specified reduction rules. The reduction rules for a package are described in Section . The reduction rule are repeatedly applied to the package till it reaches a normal form. The normal form of the package has been described in Section .

1.2 Package

A **package** is a four tuple of a pair of list of free variables, a list of universally quantified variables, a list of existentially quantified variables and a list of substitutions. The package data structure is defined by the Haskell type `Package` as shown below:

```
type FreeVars = String
type Substitution = (String,Type)

type Package = (([FreeVars],[FreeVars]),[ExistVars],[UnivVars],[Substitution])
```

There are two main advantages of solving the type equations by converting them to packages.

- Packages keep intact the "hierarchical structure of the type inference tree for a construct" and use it when solving the type equations. This results in faster solution of the type equations.
- Packages keep the type errors localised during the type equation solving process leading to error messages with better location information.

For the type equation of a given construct, the normal form of the corresponding packages contains the typing information of the construct. The next few sections will be description of the steps required in solving type equations.

A **package** is a four tuple of a pair of list of free variables, a list of existentially quantified variables, a list of universally quantified variables and a list of substitutions.

Substitutions are entities of the form "**Var = Type expression**". **Var** stands for type variables. **Type Expressions** are the types built inductively from the atomic types using the type and protocol formation rules.

Free variables are type variables used in the body of the package or the substitution list (the fourth argument of the package) from which the universally and existentially quantified variables have been removed. For the ease of reducing the packages, the free variables have been represented as a pair. The first argument of the pair is a list of all the variables on the left side of the substitution in the substitution list. The second argument of the pair the concatenation free variables found in every type expression in the substitution list.

For a simple equation like $X = Y \times Z$ where X, Y and Z are free variables, the package structure is given below.

$$\Big(([X],[Y,Z]),[\,],[\,],\Big[X = Y \times Z\Big]\Big)$$

Consider the below equation with existentialy quantified variables X and Y.

$$\exists\, X,Y.\ \begin{array}{l} Z = (X,Y) \\ X = Int \end{array}$$

The package for the above existential equation is:

$$\Big(([Z],[\,]),[X,Y],[\,],\Big[\begin{array}{l} Z = (X \times Y), \\ X = Int \end{array}\Big]\Big)$$

The above generated package for the quantified equation can be reduced to its normal form using the existential variables to guide the reduction process.

$$\Big(([Z],[\,]),[Y],[\,],\Big[\quad Z = (Int \times Y)\quad\Big]\Big)$$

The **normal form** of a package is defined as the most simplified form of the package where reductions are no longer possible.

1.3 Generating Package from an Equation

Lets call this function genPack function.

- **Input** - Type Equation
- **Output** - Reduced Package for the Type Equation

1.3.1 Steps of the function genPack

1. when the equations doesn't have any existentially or universally quantified variables, then put the equation in the body of the package (the fourth argument of the tuple) and free variables in the first argument. The second and the third arguments that contain the universally and existentially quantified variables are empty. For example - For an equation $X = Y \times Z$ which doesn't have any quantified variables, the package generated will be the following.

$$(([X],[Y \times Z]),[\,],[\,],[(X = (Y \times Z))]$$

2. When the input to the function is a type equation with quantified variables like below.

$$\forall\ P, Q\ \exists\ X, Y\ .\ Equations$$

   (a) **Generation of Inner Packages** - Apply the generate package function recursively to the *Equations* in the body of the outer quantified equation to come up with a list of packages.

   (b) **Reduction of Inner Packages to Normal Forms** - Reduce the packages to their normal form in the current context (with the given list of existential and universal variables). The normal form of a package means that the inner packages can be simplified further. This is explained in details in the section **"Reduction of Packages in a Type Equation."** The reduction process may consume some of the existential and universal variables of the outer equation. Lets call the remaining list of quantified variables as **remaining universal and existential variables**.

   (c) **Getting a Final Package** - Once the inner packages have been reduced to their normal form, they are combined together to come up with the final package for the current type equation. This is achieved in the below steps.

      i. To get the list of existential variables of the final package, the existential variables of the inner packages are concatenated together which in turn is concatenated with the remaining existential variable of the outer equation from the reduction step. Similarly, the list of the universal variables for the final package are obtained.

      ii. The substitution list of the inner packages are concatenated to get the substitution list of the final package.Finally the pair of free variables are obtained using the substitution list and the pair of lists of quantified variables.

1.4 Reducing a Package to its Normal Form in a Type Equation

- **Input** - The existential and universal quantifier variable of the outer type equation and a list of packages corresponding to the inner type equations.
- **Output** - A list of packages reduced to their normal form, remaining lists of existentially and universally quantified variables.Some of the afore mentioned variables are used in the reduction process.

1.4.1 Steps in the Reduction Algorithm

1. for all the existential variables do
   (a) Lookup for a substitution for the existentially quantified variable.
   (b) If the substitution exists in a particular package, remove this substitution from the substitution list of the package.
   (c) Remove the existential variable from the existential variable from the existential variable list of the outer type equation.
   (d) Linearize all the inner packages with this substitution say (a,expr).
   (e) Linearization means one of the two things.
      i. If there is an equation in the substitution list of a package, say (b,bexpr) and a is present in the body of bexpr, then replace a with aexpr inside the body of bexpr.
      ii. If there is an equation of the form (a,someexpr) then aexpr is unified with someexpr and the result of unification is added to the substitution list of the package and the package (a,someexpr) is removed.

      The step leads to removal of some/all of the existentially quantified variable along with modified packages in the package list.
2. For all the variables in the universal variable list do
   (a) Try finding a substitution for the universally quantified variable.
   (b) If there is no substitution for the variable then get rid of the variables from the universal quantifier list of the outer equation.
   (c) If there is a substitution for a variable say a, then check if there is a trivial substitution for a, i.e a substitution of the form a = a. A substitution with any other thing on the right hand side of the substitution is called a non trivial substitution.
   (d) If there exists a trivial substitution , then get rid of the substitution from the substitution list of the package and the universally quantified variable from the corresponding list of the outer equation.
   (e) If there is a non-trivial substitution, return an error.
3. The normal form package of the outer most equation shouldn't have any universal variables, may or may not have existential variables and will have some substitutions in the substitution

list. For a simple construct, the number will be 1. However, if we are type inferring constructs like mutually recursive functions, the number of substitutions will be equal to the number of constructs we are type inferring.