4.1 TypeCDiag

The attribute grammar of TypeCDiag can be found in src:CCO/Diag/Ag/Typing.ag. The generated haskell file can be found in src:/CCO/Diag/Ag.ag

4.1.1 AG Attributes

In this section

Syn tyCons – The TyCons used to distinguish between several types.

Inh Recurs – This inherrited attribute is used to distinguish between several ways of recursion.

syn ty – The ty is using the Ty datatype to store all the information about the type given by the input. The Ty will be later explained

syn tyErrs – The tyErrs uses a list of the Datatype TyErrs as type.

**4.1.2 Type checking**

To describe the implementation of the type system, we will first explain how we store the information from the diag to a new datatype. The new datatype is called Ty. This datatype consist of four types.

* TyCons – to get the type
* And three Maybe Ident to store the source, target and platform.

The implementation of the type system is done by

This section describes

attr Diag Diag\_

syn tycons :: {TyCons}

inh recurs :: {Recursive}

syn ty :: {Ty}

syn tyErrs use {++} {[]} :: {[TyErr]}

* Ag Attributes
* Type checking

This section describes the implementation of the TypeCDiag.

To describe the implementation of the typeCDiag we will first explain what kind of data type were used to distinguish between the types.

TyCons

* Prog
* Comp
* PlatF
* Runnable
* Framework
* Executed
* Compiled
* Not\_Executed

Recursive

* Left\_recursive
* Right\_recursive
* Not\_recursive

Something about the attribute of Diag

Need to write something about the ty

Program -> Ty Prog L

Interpreter -> Ty Interp (Just @l) Nothing (Just @m)

Compiler -> Ty Comp (Just @l1) (Just @l2) (Just @m)

Platform -> Ty PlatF Nothing Nothing (Just @m)

Execute -> con=Executed and uses the second type d2

And add to both of the recurs types Not\_recursive

Compile translate @d1.ty @d2.ty. The compile uses the translate function to obtain the correct Type and information. So have the following rules:

* When compiling a program; the translate function will give as output the Program, but the source language of the Program is replaced by the target language of the compiler
* When compiling an interpreter; the translate function will give as output the Interpreter, but the platform language is replaced by the target language of the compiler.
* When compiling an compiler; the translate function will give as output the compiler, but the platform language is replaced by the target language of the compiler.

We use several functions in the execute and compile to check whether a type is ill typed or not. Each of these function will return an error message in a List or returns an empty List. In the case of the error message we have determined that the type is ill-typed. All of these functions will be concatenated together, so in the end; when the type is ill-typed, it produces a pretty printed type error message. We used several function to produce a more detailed error message.

To check for errors we use the attribute tyErrs. (Execute)

* CheckRunnable
* checkFramework
* checkIfMatches
* checkExeOrCompile

To check for errors we use the attribute tyErrs (Compile)

* checkRunnable
* checkComp
* checkIfMatches
* checkExeInCompile
* checkExeOrCompile
* checkLandRrecurs
* checkLeftRightRecurs
* checkLeftRightRecurs

checkRunnable - This function is used in the Compile and Execute to check if the first type is of type Runnable. If this is not the case it is ill-typed and produces an error.

checkComp - This function is used in the Compile to check if the second type is of type Comp. If this is not the case it is ill-typed and produces an error.

checkIfMatches – In this function we pattern match on several cases to check if the language of the first type compatible is with the language of the second type. If this is not the case it ill-typed and produces an error.

checkExeInCompile – This function checks in the Compile if one the types is of type Executed. If this is the case, the type is ill-typed and produces an error.

checkExeOrCompile – This function is used in the Execute to check if the second type is of type Compiled or Executed. If this is the case it produces an error.

checkLandRrecurs – The checkLandRecurs checks if within a compile there is not a left recursive and right compilation at the same time

checkLeftRightRecurs - This function is used to check if a recursion is going one way. We do this by checking by pattern matching on the matchR function. This function produces an True when it does not have a recursion that is going both sides. If the function produces an false, it is ill-typed and will produce an error message.

LeftRecursion – Check if left type is recursive, we do this by pattern matching on tyCons and if it of type Compiled, it will return an Left\_recursive

Diag

**Program**

**Lhs.ty** – This type is defined by Prog and we defined as source language the l.

Lhs.tyCons - As tyCons we use Prog

**Interpreter**

**Lhs.ty –** This ty is defined by Interp and we used the @l as source and @m as platform.

ihs.tyCons – Interp

**Compiler**

**ihs.ty –** The ty is using the Comp as tyCons. The source is defined as @l1, the target as @l2 and platform as @m

**ihs.ty –** The ty is using the PlatF as tyCons. The platform is defined as @m.

**Execute. -** The tyCons is defined as Executed and we use as ty the ty of the second argument. Also we say that both the types are