LPro-P2 - 103652

Overview

This is the report referring to Phase 2 of the Programming Languages course for Instituto Superior Técnico regarding static type checker implementation on a Compiler, further extended with Recursive Type Definition. Due to a problem with gitlab, I am not able to make my repository public, and as such must submit a github repository instead. I am terribly sorry for the inconvenience but, from what I read regarding my issue, there did not appear to be a simple fix.

Additional features added for Phase 2

For this phase of the project, I've added static type checking, custom type declarations, including recursive types, a Product type named Struct, a Sum type named Union, a match operation over Union Structures and String concatenation.

Implementation of Static Type Checker

There are 4 separate parts to the implementation of the Type Checker.

Creation of ASTTypeDef and changes to ASTLet

Firstly, a new node ASTTypeDef was created to permit the declaration of ASTTId values associated with ASTType values. Then, ASTLet was altered to permit the static type declaration of an identifier variable. For matters of similarity in behavior, these two nodes were joined together to form ASTTypeDef. This was originally done with the intent of permitting intermittent calls, but ultimately those were disallowed so this wound up simply as a code clean up step.

Creation of ASTType nodes

Secondly, it was necessary to create a new Super Class named ASTType, with consequent creation of all possible types for each Node. These are:

- ASTTArrow -> type for function definitions
- ASTTBool -> type for boolean values
- ASTTId -> type for identifiers, that may then be associated to a value / type
- ASTTInt -> type for integer values
- ASTTList<Type> -> Static Type for Cons and LazyCons
- ASTTRef -> Type for Box values
- ASTTString -> Type for String Values
- ASTTStruct -> Type for Product type values
- ASTTUnion -> Type for Sum Type values
- ASTTUnit -> Type for a form of Nil type values

Creation of isSubtypeOf and Simplify

Third, all of these classes require the creation of 1 method: isSubtypeOf(ASTType otherType, Environment<ASTType> e)

thisType.IsSubtypeOf(ASTType otherType, Environment<ASTType> e)

IsSubtypeOf is a fairly direct method at its base implementation. Given an ASTType otherType, it applies the known Subtyping rules to determine whether thisType is a subtype of otherType. Complex types (such as ASTTArrow, ASTTUnion and ASTTStruct) will result in recursive calls to isSubtypeOf().

thisType.Simplify(Environment<ASTType>)

Simplify is a method used for flattening Id types. Consider the following example:

```
Type A = int;
Type B = A;
Let x:B = 2;;
```

For acceleration of future processes any “Id Chains” are flattened, resulting in the operation expressions being interpreted as:

```
Type A = int;
Type B = int;
Let x::int = 2;;
```

This method works through its own recursive call over ASTTId, ASTTStruct, ASTTUnion and ASTTArrow.

Creation of typeChecker Method together with Environment<ASTType>

Lastly, for each ASTNode, a typeChecker method was created that calculates (before runtime) what the type of any given expression will be. It does so through use of isSubtypeOf and Recursive calls to typeChecker for expression simplification. This is obtained by creating an Environment<ASTType> where we call exp.typecheck(Environment) before calling the exp.eval(). Especially relevant is ASTLetAndType, which confirms expressions on the right of a let definition match the expected type declared on the left.

End of Static Type Checking

All of these features, when put together, result in static type Checking being implemented.

Implementation of Recursive Types

The implementation of Recursive Type required the change of the two methods created above: isSubtypeOf and Simplify

Changes to Simplify()

Due to the nature of Simplify, recursive types resulted in two things: Interpreter Errors from nonexistent ASTTypes in the Environment, and infinite loops.

To solve this, Simplify was fully removed from the program and compensated by the following changes to isSubtypeOf()

Changes to isSubtypeOf()

isSubtypeof() had a change that forced the simplification of any ASTTId() types it found. Following the logic of StepOrValue() but for TypeChecking, if the left ASTType (thisType) is an ASTTId, its associated ASTType is found and isSubtypeOf recursively called. If not, and the right ASTType (otherType) is an ASTTId, its associated ASTType is found and isSubtypeOf recursively called. If neither is an ASTTId, then the same rules created before for definition of subtyping are checked, returning true or false according to the result of the operation.

Project Structure

There are 3 folders within the project:

- The Project folder contains all the code respective to the project as well as the scripts makeit.sh, which compiles the program, breakit.sh which decompiles the project and X++.sh which, if called without arguments, initiates the compiled program; and if called with a directory as an argument, it runs all tests within that folder, expecting a TypeCheck Error if the test starts with /* Fails */ and expecting success otherwise
- The Project/starterkit/tests folder contains the entirety of the tests applicable to the project. This includes both the (modified for the project) tests given by the professor and the custom made tests
- The SASyLF+starter folder contains the SASyLF proof concerning the proof for type safety upon addition of product types (pairs).