Exercise Sheet 3: Symboltables

Organizational issues

Keep in mind that the chosen **representative** of your group hands in your solution as Moodle does **not yet support group solutions**. Always make sure to include your names in your solution.

Hint:
If you have questions/problems solving this task, use the reference manual available here:
http://www.se-rwth.de/publications/MontiCore-5-Language-Workbench-Edition-2017.pdf
to find further explanations.

Exercise 3.1: Create Languages (1.5 Points)

Unzip the provided project and make yourself familiar with the given language project.
In this task you are given two grammars "`Automata`" and "`ClassDiagram`".
Furthermore, there are 3 Automata example models and 2 ClassDiagram example models in "`src/test/resources`".
As you may have noticed the predefined "`AutomataTool`" does not compile yet, since you need to override the Language classes for the Grammars in order to create their symboltable from a model.

a) Override the classes "`AutomataLanguage`" and "`ClassDiagramsLanguage`" with the TOP-mechanism. Predefine the "`langName`" and "`fileEnding`" in the overwritten languages. Note that the `fileEnding`-String must only contain the name of the fileEnding without the leading dot(.). Return a new instance of the specific `ModelLoader` in the "`provideModelLoader`" methods.
b) Also override the "`calculateModelNamesForState`"/ "`calculateModelNamesForCDType`" methods so that they only return the qualifier of a given String. E.g. name = "a.b.C.D" will return a Set containing one String: "a.b.C". If there is no qualifier just return an empty Set. It can be assumed that the only used separator is a point ('.'). Other separators do not need to be checked.

Exercise 3.2: Define Symbols (2.5 Points)

For working with the symboltable, first some more symbols should be defined. In this exercise you will learn how to create new symbols and how to extend symbols with additional functionality.

a) Mark the production "State" in Automata-Grammar as a symbol. Also mark the production interface "CDType" in ClassDiagrams-Grammar as a symbol and a scope.
b) Define a symbolrule for the symbol "CDType" which adds three attributes of type boolean, named "isPrivate", "isPublic" and "isProtected".
c) Use the TOP-mechanism to override the class "ClassDiagramsSymbolTableCreator". Override the methods "initialize_CDClass" and "initialize_CDInterface" and set the attributes "isPrivate", "isPublic" and "isProtected" in the CDTypeSymbol according to the Modifier defined in "ASTCDClass"/ "ASTCDInterface" in there.

Exercise 3.3: Resolving Delegates (4 Points)

Now that we have the symbols "CDTypeSymbol" and "StateSymbol" generated, we want to make it possible to also find CDTypeSymbols when, we actually resolved for StateSymbols.
This can be done with the help of implementing a "ResolvingDelegate" and then adding it to the AutomataGlobalScope.

a) Create a class called "State2CDTypeResolvingDelegate" and make it implement the generated interface "IStateSymbolResolvingDelegate". The new class should be located at src/main/java/automata/_symboltable/.
b) The created ResolvingDelegate should get an instance of a "ClassDiagramsGlobalScope" in its constructor and save it in a global attribute.
c) Implement the method "resolveAdaptedStateSymbol" by resolving all CDTypeSymbols in the "ClassDiagramsGlobalScope". Create a new StateSymbol for each CDTypeSymbol which is public. For the creation of these StateSymbols the AutomataSymTabMill should be used. Give the StateSymbols the same name as the corresponding CDTypeSymbol. Return the list of newly created StateSymbols.
d) Comment in the Test "StateResolvingDelegateTest" a check if your ResolvingDelegate works.
e) Now that you have created the Resolving Delegate add it also to the symboltable creation in the "AutomataTool". Do this by updating the "createSymbolTable" method in the Tool so that you create an instance of the "State2CDTypeResolvingDelegate" and then add it to the globalScope of the Type "AutomataGlobalScope".

Exercise 3.4: Create a CoCo (2 Points)

Now we want to check if in an Automata all Transitions actually refer to a StateSymbol/CDTypeSymbol with their "from" and "to" Name. For this a CoCo should be written.

a) Create a CoCo called “`ReferencedStateExists`”. The new class should be located at `src/main/java/automata/coco/`. Make this CoCo class implement a suited generated CoCo interface.
b) Implement the check method by resolving for a StateSymbol in the EnclosingScope of a Transition. Resolve for the names “`from`” and “`to`” of a transition. If one of the StateSymbols for “`from`” or “to” cannot be found Log an error. The error message should have the errorCode = “0xAut01” and an errorMessage should look like this: " The transition 'A - ab > C' references the state ‘C' which does not exist."
c) Add the CoCo to the CoCoChecker in the AutomataTool and check if the Test AutomataToolTest works now, by removing the `@Ignore` annotation.

As a solution of this Exercise sheet, zip and hand in your project without the generated target. Make sure the generated files are reproducible by just running `mvn install`. **Not working projects will not gain you any points**.