Attribute Grammar – *Identificator* – Ignacio Fernández Suárez (UO294177)

Attributes

Symbol	Attribute Name	Java Type	Inherited/Synthesized	Description
Variable	definition	VarDefinition	Synthesized	Nodo que apunta a la definición de la variable.
StructType	deftuple	Deftuple	Synthesized	Apunta a la deftuple a la que hace referencia
Procedure	invocation	Feature	Synthesized	Enlaza la invocación a la feature correspondiente
Field	deftuple	Deftuple	Synthesized	Asocia el campo a la deftuple correspondiente

Rules

Node	Predicates	Semantic Functions
program → classDef global? create feature* runInvocation	**Comprobamos si quedan features en la lista de constructores que no se definieron** constructor.isEmpty()	**Marcamos las features que están en 'create' y si se definieron** for(String name:createNode.idents) { Feature feature = features.get(name) feature.constructor = TRUE}
classDef → name:string		
runInvocation → procedure		
readStmt:stmt → expression*		
printStmt:stmt → expression* format:string		
assignStmt:stmt → assignment		
ifStmt:stmt → condition:expression ifStmts:stmt* elseStmts:stmt*		
fromStmt:stmt → declarations:assignment* condition:expression stmts:stmt*		
procedureStmt:stmt → procedure		
returnStmt:stmt → returnInvoc		

assignment → left:expression right:expression		
intLiteral:expression → value:string		
realLiteral:expression → value:string		
charLiteral:expression → value:string		
variable:expression → name:string	**Comprobamos que la variable haya sido previamente declarada.** varTable.getFromAny(variable.name) != NULL	**Enlazamos a la defínción de la variable** variable.definition = varTable.getFromAny(variable.name)
procedureExpression:expression → procedure		
arrayExpression:expression → array:expression index:expression		
structExpression:expression → struct:expression field:string		
minusExpression:expression → expression		
notExpression:expression → expression		
cast:expression → dataType expression		
arithmeticExpression:expression → left:expression operator:string right:expression		
comparisonExpression:expression → left:expression operator:string right:expression		
logicExpression:expression → left:expression operator:string right:expression		
procedure → name:string expression*	**Comprobamos que la invocación sea a una feature existente** features.get(procedure.name) != NULL	**Enlazamos la invocación a la feature** procedure.invocation = features.get(procedure.name)
integerType:dataType → ε		
doubleType:dataType → ε		
characterType:dataType → ε		
structType:dataType → name:string	**Comprobamos que exista dicha deftuple** deftuples.get(structType.name) != NULL	**Enlazamos a la deftuple correspondiente**

		structType.deftuple = deftuples.get(structType.name)
arrayType:dataType → size:string dataType		
voidType:dataType → ε		
errorType:dataType → ε		
create → idents:string*	**Comprobamos que no haya elementos repetidos en esta lista de constructores** for(String featureName: create.idents) { !names.add(featureName) }	**Añadimos el nombre de la feature a la lista de constructores** constructor.add(featureName)

feature → name:string params:varDefinition* dataType? localBlock? doBlock	**Comprobamos que no existan declaraciones de features ya declaradas**
	!features.containsKey(feature.name)
	
	Comprobamos que no existan parámetros duplicados dentro de la feature
	for(VarDefinition vd :feature.params){ names.add(vd.getName()) == TRUE }
	
	Comprobamos que no existan duplicados con locales y parámetros
	for (VarDefinition vd : feature.localBlock.varDefinitions){ !names.add(vd.name)}
		Al definirla la quitamos de la lista constructors
		constructor.remove(feature.name)
		
		Definimos el scope para los parámetros
		for (VarDefinition vd : feature.params) { vd.setScope(Scope.PARAMETER) }
		
		Definimos el scope para las variables locales
		for (VarDefinition vd : feature.localBlock.varDefinitions){ vd.setScope(Scope.LOCAL) }
		
		Añadimos los parámetros y variables locales en la tabla
		varTable.put(vd.name, vd)

returnInvoc → expression?		
localBlock → varDefinition*		
doBlock → stmt*		
global → globalTypes? varsTypes?		
globalTypes → deftuple*		
varsTypes → varDefinition*	**Comprobamos si hay variables duplicadas en el ámbito global**	**Definimos el scope para las variables globales**

	for(VarDefinition vd: varsTypes.varDefinitions) { varTable.getFromTop(vd.name) != NULL }	for(VarDefinition vd: varsTypes.varDefinitions) { vd.setScope(Scope.GLOBAL) } **Añadimos las variables globales en la tabla** varTable.put(vd.name, vd)
deftuple → name:string field*	**Comprobar que la deftuple no exista ya** deftuples.containsKey(deftuple.name) **Comprobamos que no haya campos duplicados** for(Field field : deftuple.fields){ !names.add(field.name)}	**Ascociamos el field a su deftuple padre** for(Field field : deftuple.fields) { field.setDeftuple(deftuple)}
field → name:string type:dataType		
varDefinition → name:string type:dataType		

Operators samples (cut & paste if needed):
⇒ ⇔ ≠ ∅ ∈ ∉ ∪ ∩ ⊂ ⊄ ∑ ∃ ∀