Semntica de KDecaf

Carlos López

September 6, 2010

1 Antecedentes

El *stream* de caracteres de entrada es mapeado por el parser **KDecafParser** a instancias de nodos del AST; este se encarga de construr nodos como **Program** o un nodo **Declaration** que puede ser ya sea **VarDeclaration**, **StructDeclaration** o **MethodDeclaration**, nótese que **Declaration** es un nodo abstracto, es decir, el parser no podra construr un **Declaration** sin que este fuera una instancia de sus tipos específicos.

2 Estructuras

Existe un trait (similar a una interfaz en Java), que extiende de *Scope* => *SemanticResult* en Scala, eso significa que es una función que toma un argumento de tipo **Scope** y devuelve un valor de tipo **SemanticResult**.

Estos traits y clases se ecuentran definido en un archivo Semantics.scala:

```
package kmels.uvg.kdecaf.compiler.semantics

import kmels.uvg.kdecaf.compiler
import compiler.parsing.ast.{Node,InnerType}
import compiler.types.{aliases => typeAliases}
import typeAliases._

/**
 * Semantic rules
 *
 * @author Carlos Lopez
```

```
 * @version 1.0
 * @since 2.0
 */
trait SemanticRule{
  self: Node =>

  import typeAliases.Scope

  val typesShouldBeEqualIn: (InnerType,InnerType,String) =>
      SemanticResult = (node1,node2,message) =>
    if (node1.getUnderlyingType() == node2.getUnderlyingType())
      SemanticSuccess
    else
      SemanticError(message)

  def SemanticAction(f: Scope => SemanticResult) = new
      SemanticAction{
    def apply(attributes: Scope) = f(attributes)
  }

  def SemanticError(message: String) = new
      SemanticError((message,this.pos.line,this.pos.column))

  val semanticAction:SemanticAction
}

trait NoSemanticAction extends SemanticRule{
  self: Node =>
  val semanticAction = SemanticAction(
    attributes => SemanticSuccess
  )
}

trait SemanticResult

object SemanticSuccess extends SemanticResult

case class SemanticResults(val results:SemanticResult*) extends
    SemanticResult
```

```
abstract class SemanticAction extends (Scope => SemanticResult)

case class SemanticError(val errors:SemanticErrorMessage*) extends
    SemanticResult
```

Ntese que un resultado semntico puede ser cualquier instancia de los siguientes casos:

- Una lista de resultados semnticos: **SemanticResults**
- Un resultado semntico que tiene sentido: **SemanticSuccess**
- Un error semntico: **SemanticError**

Cada nodo implementa a **SemanticRule** por lo que se podría decir que cada nodo también es una regla semántica. Los resultados de las aplicaciones de las reglas semnticas son evaluados cuando se necesita, por ejemplo **Program** necesita de los resultados semnticos de cada una de sus declaraciones. De esta manera, los resultados semnticos se pueden propagar.

Todos los nodos tienen un tipo interno, esto lo restringe el trait InnerType, que forza a cada nodo a tener una función que no recibe argumentos y devuelve el tipo del nodo.

```
trait Node extends Positional with InnerType{
  override def toString = getClass.getName
}

trait InnerType {
  val getUnderlyingType: () => String
}
```

Algunos nodos no necesitan calcular su valor siempre que les sea requerido, esto se puede abstraer construyendo otros traits que extendien a **InnerType** y aplicandolos en nodos de la siguiente forma:

```
trait InnerInt extends InnerType{
  val getUnderlyingType = () => "Int"
}

case class ExpressionAdd(val exp1:Expression,val
    exp2:Expression)(implicit val m:Manifest[Int]) extends
    BinaryOperation[Int] with InnerInt
```

```
case class ExpressionSub(val exp1:Expression,val
    exp2:Expression)(implicit val m:Manifest[Int]) extends
    BinaryOperation[Int] with InnerInt
case class ExpressionMult(val exp1:Expression, val
    exp2:Expression)(implicit val m:Manifest[Int]) extends
    BinaryOperation[Int] with InnerInt
```

Ejemplo de Regla Semántica

Un nodo **Assignment** que representa a la producción *Location* '=' *Expression* verifica lo siguiente:

- El tipo de Location es el mismo que el de expression.
- Location exista como variable declarada.

```
case class Assignment(val location:Location,val
    expression:Expression) extends Statement with InnerTypeVoid{

  val semanticAction = SemanticAction(
    attributes => {
      val typeEqualityResult = typesShouldBeEqualIn(
        location,expression,
        "cannot assign expression of type
            "+expression.getUnderlyingType()+" to "+location.name+"
            of declared type "+location.getUnderlyingType()
      )

      SemanticResults(typeEqualityResult,location.semanticAction(attributes),expression.
    }
  )
}
```

Para más información sobre las reglas semánticas y nodos existentes, visitar [1]:

References

[1] Repositorio Construcción de Compiladores, kmels.net, http://uvg.kmels.net/code/cc3007.