Manual Técnico

Antennae Java

Josue Guillermo Orellana Cifuentes

201801366 Facultad de Ingeniería USAC

Lógica utilizada para el analizador de copias

El enunciado del proyecto describía los casos en donde una clase, una función/método y una variable eran consideradas copia. Con esto en mente se diseñó un análisis el cual comienza comparando el nombre de una clase del archivo dudoso con el original, de hallar una coincidencia cumple con el criterio de estar en una clase con el mismo nombre por lo cual pueden existir funciones copia. Como paso fundamental se compara la cantidad de funciones de ambas clases, de no ser la misma cantidad entonces se descarta que sea una clase copia. Se compara cada una de las funciones de la clase proveniente del archivo dudoso con las funciones de la clase del archivo original, con esto se efectúa el análisis de la similitud entre los parámetros y el tipo de retorno del método/función si esto coincide se considera como una clase copia. Cuando se obtiene una clase copia se analizan sus variables y se comparan con la coincidencia de la función en la clase del archivo original. Cuando se haya la similitud entre el tipo y la ubicación de la variable se considera como una variable copia. Este análisis se realiza con todos los elementos del archivo y con cada uno de los archivos generando consigo la tabla de copias que de descarga cuando existe por lo menos una detección de copia.

Gramática en la que se basó el analizador sintáctico

El analizador sintáctico se desarrolló con la implementación de la herramienta jison ajustando la siguiente gramática de tal forma que jison pudiera crear el analizador LALR(1).

```
P comentario -> // *cualquier cosa* \n
         /* *cualuier cosa* */
P Tipo Dato -> int
         I double
         | boolean
         | char
         | String
P_GeneralBody-> P_comentario
         | P_Imports
         | P Definicion clase
P Definicion clase -> class id { P contenidoClase }
P_Imports -> import id;
P_Declaracion_Var -> P_Tipo_Dato P_lista_ID P_Declaracion_Var'
P Declaracion Var' -> :
              | = expresion;
P Asignacion -> id
```

```
P_Asignacion' -> (P_Argumentos);
         = expresion;
P_Argumentos -> expresion P_Argumentos'
P_Argumentos' -> , expresion P_Argumentos'
P_if -> if ( P_condicion ) { P_sentecias } P_if'
P if'-> else P if"
    | epsilon
P_if" -> if ( P_condicion ) { P_sentecias } P_if'
       | { P_sentecias }
P Switch -> switch (expression) { P casos P default}
P_casos -> case expresion : sentencias P_casos'
P_casos' -> case expresion : sentencias P_casos'
P_default -> default : sentencias
         | espsilon
P_Do_While -> do { sentencias } while ( P_condicion )
P_While -> while( P_condicion ) { sentencias }
P_For -> for(P_for_var condicion; id P_for_change){ sentencias }
P_for_var -> P_Tipo_Dato id = expresion;
         | id = expresion;
P for change -> ++
         |--
P_Declaracion_funciones -> void P_Declaracion_funciones'
                 | P_Tipo_Dato id(P_Parametros) { sentencias}
P Declaracion funciones' -> id (P Parametros){ sentencias }
                 | main() { sentencias }
P_Parametros -> P_Tipo_Dato id P_Parametros'
P_Parametros' -> , P_Tipo_Dato id P_Parametros'
P Print -> System.out.P Print'(expresion);
P_Print' -> println
       print
expresion -> TE'
E'-> +FT'
```

```
| - FT'
T->FT'
T'-> *FT'
  | /FT'
F-> (expresion)
  |id P_llamada
  | entero
  | decimal
  | cadena
  | true
  | false
P_llamada -> (P_Argumentos)
P_contenidoClase -> P_comentario
            | P_Declaracion_funciones
            | P_Declaracion_Var
           | P_Asignacion
sentencias -> P_Declaracion_Var
         | P_Asignacion
         | P_if
         | P_Switch
         | P_While
         | P_Do_While
         | P_For
         | break;
         | return expresion;
         | continue;
         | P_Print
```