Procesadores de lenguajes. Práctica 10.

ANÁLISIS SEMÁNTICO COMPLETO

Índice

1.1 Código a utilizar

- Con respecto al código de la práctica anterior, se han añadido:
  - Tinto.parser.TintoParser.java
  - Tinto.parserjj.TintoParser.jj
  - Implementaciones del analizador del cuerpo de las funciones, la primera manualmente y la segunda por medio de JavaCC.

1.2 Verificaciones semánticas

- El objetivo del análisis semántico es construir el Árbol de Sintaxis Abstracta en paralelo al proceso de verificación sintáctica. Para ello es necesario incluir una serie de funciones auxiliares que permiten verificar que la información semántica obtenida en el análisis es correcta. Estas funciones se utilizan posteriormente dentro de las acciones semánticas que construyen la estructura de datos. A continuación, se enumeran las funciones de verificación semántica incluidas en el análisis del cuerpo de las funciones. Estas funciones de verificación se han incluido tanto en el analizador realizado a mano (tinto.parser.TintoParser) como en el analizador descrito por medio de JavaCC (TintoParser.jj).
  - boolean verifyReachableCode(Statement stm, BlockStatement block): verificación semántica que comprueba que la sentencia a añadir a un bloque es alcanzable.
  - boolean verifyFinishedFunction(Token tk, BlockStatement body, SymbolTable symtab): verificación semántica que comprueba que el código de una función alcanza siempre un return.
  - boolean verifyDuplicatedVariable(Token tk, SymbolTable symtab): verificación semántica que comprueba que una variable no esté duplicada.

1.2 Verificaciones semánticas

- boolean verifyConditionType(Token tk, Expression expr): verificación semántica que comprueba que una condición sea de tipo booleana.
- boolean verifyReturnType(Token tk, Expression expr, SymbolTable symtab): verificación semántica que comprueba que una instrucción return devuelve un tipo de dato correcto.
- boolean verifyKnownVariable(Token tk, SymbolTable symtab): verificación semántica que comprueba la existencia de la variable de una instrucción de asignación.
- boolean verifyAssignTypes(Token tk, Variable var, Expression expr): verificación semántica que comprueba los tipos de datos en una instrucción de asignación son correctos.
- boolean verifyKnownLibrary(Token tk, SymbolTable symtab): verificación semántica que comprueba la existencia de una biblioteca en la tabla de símbolos.
- boolean verifyKnownFunction(Token tk, CallParameters param, SymbolTable symtab): verificación semántica que comprueba la existencia de una función en la biblioteca activa.
- boolean verifyKnownFunction(Token tk, CallParameters param, LibraryDeclaration library): verificación semántica que comprueba la existencia de una función pública en una cierta biblioteca.

1.2 Verificaciones semánticas

- boolean verifyBooleanTypes(Token tk, Expression exp1, Expression exp2): verificación semántica que comprueba que los dos operandos de una expresión lógica (AND, OR) sean booleanos.
- boolean verifyRelationTypes(Token tk, int relop, Expression exp1, Expression exp2): verificación semántica que comprueba que los tipos de los dos operandos de una relación sean correctos.
- boolean verifyBooleanType(Token tk, Expression expr): verificación semántica que comprueba que una expresión sea de tipo booleana para poder apilcar el operador NOT.
- boolean verifyNumericType(Token tk, Expression exp): verificación semántica que comprueba que dos expresiones sean de tipo numérico para poder aplicar los operadores '+','-','*' y'/'.
- boolean verifyIntegerTypes(Token tk, Expression exp1, Expression exp2): verificación semántica que comprueba que dos expresiones sean de tipo entero para poder aplicar el operador '%'.
- boolean verifyIntegerValue(Token tk): verificación semántica que comprueba que el valor de un literal entero sea correcto.

1.3 Acciones semánticas

- Las acciones semánticas son funciones dedicadas a contruir las estructuras del Árbol de Sintaxis Abstracta. Estas funciones se insertan dentro del proceso de verificación sintáctica de manera que el AST se vaya construyendo en paralelo al proceso de análisis sintáctico. A continuación se enumeran las acciones semánticas que se han incluido en los analizadores tinto.parser.TintoParser y TintoParser.jj.
  - int[] actionAddArgumentType(int type, int[]args): Añade un código de tipo a una lista de códigos de tipo. Esta acción se utiliza para construir la lista de tipos que permita identificar a la función cuyo cuerpo se va a comenzar a analizar.
  - void actionAddStatement(BlockStatement block, Statement stm): Añade una instrucción a un bloque de instrucciones
  - void actionSetFunctionBody(Token tk, BlockStatement block, SymbolTable symtab): Asigna el cuerpo de una función.
  - Variable actionAddDeclaration(SymbolTable symtab,int type,Token tid): Incluye la declaración de variable local dentro de la función analizada.

1.3 Acciones semánticas

- Vvoid actionAddAssignement(BlockStatement block, Token tid, Variable var, Expression exp): Añade una instrucción de asignación a un bloque de instrucciones.
- Statement actionGetStatementFromBlock(BlockStatement block): Obtiene la instrucción asociada a una declaración de variables. Si no hay inicializaciones devuelve null. Si sólo hay una inicialización devuelve esa instrucción de asignación. Si hay más de una devuelve el bloque de asignaciones.
- Statement actionIfStatement(Token tk, Expression cond, Statement thenStm, Statement elseStm): Crea una instrucción if.
- Statement actionWhileStatement(Token tk, Expression cond, Statement body): Crea una instrucción while.
- Statement actionReturnStatement(Token tk, Expression exp, SymbolTable symtab): Crea una instrucción return.
- Statement actionAssignStatement(Token tk, Expression exp, SymbolTable symtab): Crea una instrucción de asignación.
- Statement actionCallStatement(Token tk, CallParameters param, SymbolTable symtab): Crea una instrucción de llamada a una función de la misma biblioteca.

1.3 Acciones semánticas

- VStatement actionCallStatement(Token tid1, Token tid2, CallParameters param, SymbolTable symtab): Crea una instrucción de llamada a una función pública de una biblioteca importada.
- Expression actionOrExpression(Token tk,Expression exp1, Expression exp2): Crea una expresión binaria que define un OR entre dos expresiones.
- Expression actionAndExpression(Token tk,Expression exp1, Expression exp2): Crea una expresión binaria que define un AND entre dos expresiones.
- Expression actionRelExpression(Token tk, int op, Expression exp1, Expression exp2): Crea una expresión binaria que define una relación entre dos expresiones.
- Expression actionUnaryExpression(Token tk, int op, Expression exp): Crea una expresión unaria sobre otra expresión.
- Expression actionSumExpression(Token tk,int op, Expression exp1, Expression exp2): Crea una expresión binaria en forma de suma o resta.
- Expression actionProdExpression(Token tk,int op, Expression exp1, Expression exp2): Crea una expresión binaria en forma de producto, división o módulo.

1.3 Acciones semánticas

- VLiteralExpression actionIntegerLiteral(Token tk): Crea un literal de tipo entero.
- Expression actionReferenceExpression(Token tid1,Token tid2,CallParameters param, SymbolTable symtab): Crea una expresión de referencia a una variable o a una función.
- Expression actionVariableExpression(Token tid, SymbolTable symtab): Crea una expresión de referencia a una variable.
- Expression actionCallExpression(Token tk, CallParameters param, SymbolTable symtab): Crea una expresión de llamada a una función de la misma biblioteca.
- Expression actionCallExpression(Token tid1, Token tid2, CallParameters param, SymbolTable symtab): Crea una expresión de llamada a una función pública de una biblioteca importada.

1.4 Gramática atribuida de tinto

- El análisis semántico requiere modificar la gramática a analizar incorporando atributos a los símbolos de la gramática. Estos atributos pueden ser heredados (si representan información que se le comunica al analizador del símbolo) o sintetizados (si representan información que construye el símbolo en paralelo a su análisis sintáctico). Las acciones y verificaciones semánticas se introducen en las reglas sintácticas para construir el valor de los atributos sintetizados.

- El análisis semántico del cuerpo de las funciones parte de una situación en la que la información de cabecera de todas las bibliotecas implicadas ya ha sido analizada e incluida en la tabla de símbolos. Es decir, este análisis semántico parte de un objeto SymbolTable que contiene la lista de objetos LibraryDeclaration que describen las bibliotecas a utilizar. Cada objeto LibraryDeclaration contiene la lista de objetos Function que describen las funciones incluidas en la biblioteca, pero inicialmente estos objetos Function tienen su campo body a nulo, es decir, no contienen aun la descripción de las instrucciones de cada función.

1.4 Gramática atribuida de tinto

- El objetivo del analizador semántico del cuerpo de las funciones es analizar un fichero ".tinto" y buscar en primer lugar a que biblioteca incluida en la tabla de símbolos representa. Esta biblioteca se asigna como biblioteca activa. A partir de aquí, el analizador recorre el fichero ".tinto" analizando las funciones. Cada vez que encuentra una declaración de función, el analizador busca cual es el objeto Function asociado a esa función. Este objeto ya ha sido creado en la etapa de análisis de cabecera, aunque su campo body se debe encontrar a nulo. El objeto Function correspondiente se asigna como función activa lo que significa que el cuerpo que se va a analizar corresponde a esa función. Una vez asignada la función activa, el analizador recorre el cuerpo de la función construyendo un objeto BlockStatement que contiene la descripción de las instrucciones del cuerpo de la función. Finalmente este objeto BlockStatement se asigna al campo body de la función activa.

1.4 Gramática atribuida de tinto

- Teniendo en cuenta el proceso anterior, la mayoría de los símbolos de la gramática van a utilizar el objeto SymbolTable como atributo heredado, ya que contiene toda la información recopilada hasta el momento. A continuación se describen los atributos añadidos a cada símbolo. Las reglas correspondientes se han modificado para incorporar las llamadas a las acciones semánticas que construyen el valor de estos atributos. En algún caso, cuando el valor de los atributos es fácil de calcular se ha incluido el código directamente en la regla en vez de una llamada a una acción semántica.
  - CompilationUnit: Utiliza la tabla de símbolos como atributo heredado.
  - ImportClauseList: No requiere atributos heredados ni sintetizados ya que toda la información que aporta ya ha sido incluida en la fase del análisis de cabecera.
  - ImportClause: No requiere atributos heredados ni sintetizados.
  - TintoDecl: Utiliza la tabla de símbolos como atributo heredado.

1.4 Gramática atribuida de tinto

- LibraryDecl: Utiliza la tabla de símbolos como atributo heredado.
- FunctionList: Utiliza la tabla de símbolos como atributo heredado.
- FunctionDecl: Utiliza la tabla de símbolos como atributo heredado. La regla de este símbolo recopila la información sobre la función a analizar, la asigna como función activa y analiza su cuerpo.
- NativeDecl: Utiliza la tabla de símbolos como atributo heredado. Las bibliotecas nativas no van a ser tratadas ya que sus funciones solo están declaradas y no hay que procesar el cuerpo de sus funciones.
- NativeFunctionList: Utiliza la tabla de símbolos como atributo heredado.
- NativeFunctionDecl: Utiliza la tabla de símbolos como atributo heredado pero no realiza ninguna modificación.
- Access: No requiere atributos heredados ni sintetizados ya que el tipo de acceso de la función (público o privado) ya ha sido reconocido en la fase del análisis de cabecera.
- FunctionType: No requiere atributos heredados ni sintetizados ya que el tipo de datos de la función ya ha sido reconocido en la fase del análisis de cabecera.

1.4 Gramática atribuida de tinto

- Type: Utiliza un atributo sintetizado que representa el código del tipo de dato reconocido.
- ArgumentDecl: Utiliza un atributo sintetizado que contiene la lista de códigos de tipo de los argumentos de la función a analizar. Esta información se utiliza para buscar la función dentro de la bibioteca activa.
- ArgumentList: Utiliza como atributo heredado una lista de códigos de tipo y le añade los nuevos códigos que reconozca. Devuelve como atributo sintetizado la nueva lista con los tipos añadidos.
- Argument: Utiliza como atributo heredado una lista de códigos de tipo y le añade el tipo de dato del argumento analizado, devolviendo la lista como atributo sintetizado.
- MoreArguments: Utiliza como atributo heredado una lista de códigos de tipo y le añade los nuevos tipos de los argumentos que encuentre.
- FunctionBody: Recibe como atributo heredado la tabla de símbolos y reconoce el cuerpo de la función. Para ello crea un objeto BlockStatement vacío y lo envía como atributo heredado al símbolo StatementList para que le añada las instrucciones del cuerpo de la función. Por último ejecuta la acción semántica actionSetFunctionBody() para asignar el cuerpo a la función activa.

1.4 Gramática atribuida de tinto

- StatementList: Recibe como atributo heredado la tabla de símbolos y un objeto BlockStatement. La gramática de este símbolo recoge los ojetos Statement generados como atributos sintetizados por el símbolo Statement y los añade al bloque de instrucciones por medio de la acción actionAddStatement().
- Statement: Recibe como atributo heredado la tabla de símbolos y genera como atributo sintetizado un objeto Statement.
- Decl: Este símbolo reconoce una lista de declaraciones de variables. Recibe como atributo heredado la tabla de símbolos. Recoge el tipo de datos de la declaración, el identificador de la primera variable y su posible inicialización. Como acciones semánticas debe añadir la declaración de la variable (actionAddDeclaration()) y añadir la posible inicialización como una instrucción de asignación incuida en un bloque. Genera como atributo sintetizado el bloque con todas las instrucciones de asignación correspondientes a las inicializaciones de las variables declaradas.
- MoreDecl: Continua el reconocimiento de una lista de declaraciones de variables. Recibe como atributo heredado la tabla de símbolos, el tipo de datos de la declaración y el bloque de instrucciones utilizado para almacenar las asignaciones correspondientes a las inicializaciones de las variables declaradas.

1.4 Gramática atribuida de tinto

- Assignement: Reconoce la inicialización opcional de una variable en su declaración. Recibe como atributo heredado la tabla de símbolos y genera como atributo sintetizado el objeto Expression que describe la expresión de inicialización (o null si no hay inicialización).
- IfStm: Reconoce una instrucción if. Utiliza la tabla de símbolos como atributo heredado. La gramática recoge la información asociada a esta instrucción (condición, instrucción then e instrucción else) y lanza la acción semántica actionIfStatement() devolviendo el objeto IfStatement como atributo sintetizado.
- ElseStm: Reconoce la clausula else opcional de una instrucción if. Utiliza la tabla de símbolos como atributo heredado. Genera como atributo sintetizado el objeto Statement con la descripción de la instrucción else o una referencia nula si no aparece la clausula.
- WhileStm: Reconoce una instrucción while. Utiliza la tabla de símbolos como atributo heredado. La gramática recoge la información asociada a esta instrucción (condición y cuerpo del bucle) y lanza la acción semántica actionWhileStatement() devolviendo el objeto WhileStatement como atributo sintetizado.
- ReturnStm: Reconoce una instrucción return. Utiliza la tabla de símbolos como atributo heredado. Recoge la información asociada a esta instrucción (la expresión opcional) y lanza la acción semántica actionReturnStatement() devolviendo el objeto ReturnStatement como atributo sintetizado.

1.4 Gramática atribuida de tinto

- NoStm: Reconoce una instrucción sin contenido. Devuelve una referencia como atributo sintetizado.
- IdStm: Reconoce una instrucción que comienza por un identificador. Utiliza la tabla de símbolos como atributo heredado. La gramática recoge el identifiador inicial y la envía como atributo heredado al símbolo IdStmContinue, recogiendo el objeto Statement que genera este símbolo como atributo sintetizado y devolviendo a su vez como atributo sintetizado del símbolo.
- IdStmContinue: Reconoce una instrucción que comienza por un identificador. Utiliza como atributos heredados la tabla de símbolos y el identificador de comienzo de la instrucción. A partir de este identificador aparecen tres casos. El primero es que se trate de una instrucción de asignación. En ese caso se recoge el objeto Expression que describe la parte derecha de la asignación y se ejecuta la acción actionAssignStatement(). La segunda opción es que se trate de una llamada a una función de la misma biblioteca. En ese caso se recoge el objeto CallParameters que describe los argumentos de llamada y se ejecuta la acción actionCallStatement(). La última posibilidad es que se trate de la llamada a una función de una biblioteca externa. En ese caso se recoge el segundo identificador y el objeto CallParameters que describe los argumentos de llamada y se ejecuta la acción actionCallStatement() correspondiente. En los tres casos se devuelve el resultado de las acciones como atributo sintetizado del símbolo.

1.4 Gramática atribuida de tinto

- BlockStm: Reconoce un bloque de instrucciones. Recibe la tabla de símbolos como atributo heredado. En primer lugar debe crear un nuevo ámbito de declaración de variables (symtab.createScope()) y un objeto BlockStatement vacío donde almacenar las instrucciones reconocidas en el bloque. A continuación llama al símbolo StatementList usando el objeto BlockStatement como atributo heredado. Este símbolo se encargará de introducir el contenido del bloque. Por último se destruye el ámbito de declaración de variables (symtab.deleteScope()) y se devuelve el objeto BlockStatement como atributo sintetizado
- Expr: Reconoce una expresión. Se trata del símbolo encargado de reconocer cualquier tipo de expresión de Tinto. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. Su gramática obtiene un primer objeto Expression como atributo sintetizado del símbolo AndExpr y lo envía como atributo heredado al símbolo MoreOrExpr que se encarga de añadir las posibles expresiones adicionales conectadas por operadores OR. Devuelve como atributo sintetizado el resultado del símbolo MoreOrExpr.
- MoreOrExpr: Reconoce una operación OR sobre una expresión previamente analizada. Recibe como atributos heredados la tabla de símbolos y la expresión ya reconocida. En caso de reconocer una operación OR con una nueva expresión, ejecuta la acción actionOrExpression() para construir un objeto BinaryExpression describiendo la operación OR entre expresión heredada y la reconocida. En caso de que no aparezca ninguna operación OR (regla lambda), se limita a devolver la expresión heredada como atributo sintetizado.

1.4 Gramática atribuida de tinto

- AndExpr: Reconoce una expresión que puede estar formada por operadores AND. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. En primer lugar almacena el objeto Expression obtenido como atributo sintetizado del símbolo RelExpr y lo envía como atributo heredado al símbolo MoreAndExpr que se encarga de añadir las posibles expresiones adicionales conectadas por operadores AND. Devuelve como atributo sintetizado el resultado del símbolo MoreAndExpr.
- MoreAndExpr: Reconoce una operación AND sobre una expresión previamente analizada. Recibe como atributos heredados la tabla de símbolos y la expresión ya reconocida. En caso de reconocer una operación AND ejecuta la acción actionAndExpression() que construye un objeto BinaryExpression describiendo la operación AND entre expresión heredada y la reconocida. En caso de que no aparezca ninguna operación AND (regla lambda), se limita a devolver la expresión heredada como atributo sintetizado.
- RelExpr: Reconoce una expresión que puede incluir un operador de relación (igual, distinto, mayor-que, etc.). Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. Almacena el objeto Expression obtenido como atributo sintetizado del símbolo SumExpr y lo envía como atributo heredado al símbolo MoreRelExpr que se encarga de añadir el operador de relación en caso de que aparezca. Devuelve como atributo sintetizado el resultado del símbolo MoreRelExpr.

1.4 Gramática atribuida de tinto

- MoreRelExpr: Reconoce una posible operación de relación entre una expresión previamente analizada y una nueva expresión. Recibe como atributos heredados la tabla de símbolos y la expresión ya reconocida. En caso de reconocer el operador de relación ejecuta la acción actionRelExpression() que construye un objeto BinaryExpression describiendo la operación de relación. En caso de que no aparezca ninguna operación de relación (regla lambda), se limita a devolver la expresión heredada como atributo sintetizado.
- RelOp: Reconoce un operador de relación. No necesita ningún atributo heredado. Devuelve como atributo sintetizado el código de la clase BinaryExpression que describe el operador de relación.
- SumExpr: Reconoce una expresión aritmética, es decir, una expresión que puede incluir operadores de suma. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. El primer paso consiste en analizar un posible operador unario (un cambio de signo, por ejemplo). A continuación almacena el objeto Expression obtenido como atributo sintetizado del símbolo ProdExpr y le aplica el operador unario por medio de la acción semántica actionUnaryExpression(). El siguiente paso consiste en reconocer el símbolo MoreSumExpr pasándole la expresión ya analizada como atributo heredado. Este símbolo se encarga de añadir posibles operaciones de suma o resta de expresiones. Finalmente, devuelve como atributo sintetizado el resultado del símbolo MoreSumExpr.

1.4 Gramática atribuida de tinto

- MoreSumExpr: Reconoce una posible operación de suma o resta entre una expresión previamente analizada y una nueva expresión. Recibe como atributos heredados la tabla de símbolos y la expresión ya reconocida. En caso de reconocer el operador de suma o resta, ejecuta la acción actionSumExpression() que construye un objeto BinaryExpression describiendo la operación aritmética. En caso de que no aparezca ninguna operación (regla lambda), se limita a devolver la expresión heredada como atributo sintetizado.
- UnOp: Reconoce un operador unario. No necesita ningún atributo heredado. Devuelve como atributo sintetizado el código de la clase UnaryExpression que describe el operador encontrado.
- SumOp: Reconoce un operador de suma o resta. No necesita ningún atributo heredado. Devuelve como atributo sintetizado el código de la clase BinaryExpression que describe el operador encontrado.
- ProdExpr: Reconoce una expresión que puede incluir operadores de multiplicación, división o módulo. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. Almacena el objeto Expression obtenido como atributo sintetizado del símbolo Factor y lo pasa como atributo heredado al símbolo MoreProdExpr. Este símbolo se encarga de añadir posibles operaciones de multiplicación o división entre expresiones. Finalmente, devuelve como atributo sintetizado el resultado del símbolo MoreProdExpr.

1.4 Gramática atribuida de tinto

- MoreProdExpr: Reconoce una posible operación de multiplicación o división entre una expresión previamente analizada y una nueva expresión. Recibe como atributos heredados la tabla de símbolos y la expresión ya reconocida. En caso de reconocer el operador de multiplicación o división, ejecuta la acción actionProdExpression() que construye un objeto BinaryExpression describiendo la operación aritmética. En caso de que no aparezca ninguna operación (regla lambda), se limita a devolver la expresión heredada como atributo sintetizado.
- ProdOp: Reconoce un operador de multiplicación, división o módulo. No necesita ningún atributo heredado. Devuelve como atributo sintetizado el código de la clase BinaryExpression que describe el operador encontrado.
- Factor: Reconoce un factor, es decir, una expresión que puede ser un literal, una referencia o una expresión genérica entre paréntesis. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. Si se trata de un literal devuelve como atributo sintetizado el valor devuelto por el símbolo Literal. Si se trata de una referencia devuelve como atributo sintetizado el valor devuelto por el símbolo Reference. Si se trata de una expresión genérica entre paréntesis devuelve como atributo sintetizado el valor devuelto por el símbolo Expr.

1.4 Gramática atribuida de tinto

- Factor: Reconoce un literal, es decir, un valor constante de un cierto tipo. Recibe la tabla de símbolos como atributo heredado y genera un objeto Expression como atributo sintetizado. En caso de que reconozca un literal de tipo entero, construye el atributo sintetizado por medio de la acción semántica actionIntegerLiteral(). Cuando se trata de un literal de tipo carácter o de tipo boolean se crea directamente el objeto CharLiteralExpression o BooleanLiteralExpression correspondiente.
- Reference: Reconoce una expresión que comienza por un identificador y que puede ser una referencia a una variable o la llamada a una función. Recibe como atributo heredado la tabla de símbolos y debe generar como atributo sintetizado el objeto Expression que describa la expresión. La gramática del símbolo consiste en reconocer el identificador inicial y enviarlo como atributo heredado al símbolo ReferenceContinue, generando como atributo sintetizado el valor devuelto por este símbolo.
- ReferenceContinue: Reconoce el resto de una expresión de tipo referencia. Si se trata de la referencia a una variable utiliza la acción semántica actionReferenceExpression() para generar un objeto VariableExpression. Si detecta un paréntesis abierto se trata de la llamada a una función de la biblioteca activa. En este caso recopila los argumentos de la llamada como atributo sintetizado del símbolo FunctionCall y genera un objeto CallExpression por medio de la acción semántica actionReferenceExpression(). El último caso consiste en detectar un ponto, lo que significa que la expresión es una llamada a una función pública de una iblioteca importada. En este caso se recopila el segundo identificador y los argumentos de llamada y se genera un objeto CallExpression por medio de la acción semántica actionReferenceExpression().

1.4 Gramática atribuida de tinto

- FunctionCall: Reconoce los argumentos de una llamada a una función. Recibe la tabla de símbolos como atributo heredado y devuelve como atributo sintetizado un objeto CallParameters. La gramática de este símbolo consiste en construir un objeto CallParameters vacío y pasarlo como atributo heredado al símbolo ExprList, que se encargará de añadir las expresiones a este objeto contenedor. Finalmente se devuelve el objeto como atributo sintetizado.
- ExprList: Reconoce una lista de expresiones separadas por coma. Recibe como atributos heredados la tabla de símbolos y un objeto CallParameters donde almacenar las expresiones reconocidas. La gramática consiste en reconocer la primera expresión y añadirla al objeto CallParameters y, a continuación, hacer una llamada al símbolo MoreExpr pasandole el objeto CallParameters como atributo heredado para que continue añadiendole las expresiones de la lista.
- MoreExpr: Reconoce el resto de una lista de expresiones separadas por coma. Recibe como atributos heredados la tabla de símbolos y un objeto CallParameters donde almacenar las expresiones reconocidas. La gramática consiste en reconocer una coma seguida de una expresión para añadirla al objeto CallParameters y, a continuación, hacer una llamada nueva llamada a MoreExpr para seguir reconociendo expresiones. Cuando el símbolo no encuentra una coma (regla lambda) no hace nada. En ese momento el objeto CallParameters ya contiene la lista de expresiones analizada.