TEC | Tecnológico de Costa Rica

Proyecto 2

La fase de análisis contextual de un compilador

Instituto Tecnológico de Costa Rica

Compiladores e Intérpretes

Profesor: Ignacio Trejos Zelaya

Estudiantes

Tania María Sánchez Irola	2018138723
Luis Diego Alemán Zúñiga	2018135386
Oswaldo Ramírez Fernández	2020044182

Tabla de contenido

Comprobación de tipos para las variantes de los comandos repeat

1. Se comprueba el tipo de la expresión para asegurarse que es booleana.
2. se visita el command ast
3. si existe, se visita el ast del command leave

- El código para verificar para las distintas formas de repeat es esencialmente el mismo.

```
         TypeDenoter eType = (TypeDenoter)ast.eAST.visit(this,
null);
        if (! eType.equals(StdEnvironment.booleanType))
            reporter.reportError("Boolean expression expected
here", "", ast.eAST.position);
        ast.cAST.visit(this, null);
        if(ast.lAST != null){
            ast.lAST.visit(this, null);
        }
        return null;
```

Solución para manejo de alcance, del tipo y de la protección de la variable de control del comando for sus variantes condicionadas

1. El tipo se maneja de manera implícita como Integer
2. Se maneja la variable de control con su propio AST.
3. Además se agrega el ast a visitSimpleVname para ser reconocido como una variable(que no se comporta como tal) especial de tipo integer

```
public Object visitForCommand(ForCommand ast, Object o) {
. . .
idTable.openScope();       idTable.enter(ast.fdAST.iAST.spelling,
ast.fdAST);
. . .
idTable.
idTable.closeScope();

  public Object visitSimpleVname(SimpleVname ast, Object o) {
. . .
else if(binding instanceof ForVarDeclaration){
          ast.type = StdEnvironment.integerType;
          ast.variable = false;
. . .}
```

Soluciones del comando Choose

Al ser un grupo de 3 no se realizó el comando Choose.

Describir el procesamiento de la declaración de variable inicializada

1. Se agrega el identificador a la tabla
2. Si el identificador existe se genera un error
3. En la función visitSimpleVname se asigna el tipo y se considera como variable.

```
       idTable.enter(ast.I.spelling, ast);
        if(ast.duplicated){
            reporter.reportError ("identifier \"%\" already declared",
                                  ast.I.spelling, ast.position);
        }

----**********

public Object visitSimpleVname(SimpleVname ast, Object o) {
...
else if(binding instanceof InitializedVarDeclaration){
          ast.type = ((InitializedVarDeclaration) binding).E.type;
          ast.variable = true;
      }
```

Descripción de su validación de la unicidad de los nombres de parámetros en las declaraciones de funciones o procedimientos.

1. Para los procedimientos y funciones, el código preexistente ya solucionaba la comprobación de nombres.
2. para los ProcFuncs en el recursive, se realizó un procedimiento similar, pero se manejó por separado de la visita a los sub árboles.

```
public void declareProcFunc(ProcFunc ast){
     if(ast.fAST != null){
. . .
   FuncDeclaration func = ast.fAST;
   idTable.enter (func.I.spelling, func);
   if (func.duplicated)
             reporter.reportError ("identifier \"%\" already declared",
                                   func.I.spelling, func.position);
. . .
   else{
      idTable.enter (proc.I.spelling, proc);
      if (proc.duplicated)
             reporter.reportError ("identifier \"%\" already declared",
                                    proc.I.spelling, proc.position);
. . .
```

Procesamiento de la declaración compuesta private

Para realizar esta declaración, primeramente decidimos investigar bien el cómo funcionaba la tabla de investigación, luego de una reunión con el profesor donde nos aclaró ciertas dudas que teníamos al inicio, logramos comprender un poco mejor el funcionamiento, decidimos que debíamos realizar las siguientes tareas:

1. Los elementos de la tabla de identificación se comunican entre sí mediante una lista enlazada o pila, según como se quiera ver
2. Teniendo esa estructura como guía, primeramente debíamos guardar el “nodo” que estaba justo antes de empezar la declaración del private, a este lo llamamos nodo **salida**.
3. Luego de lo anterior, manejamos los elementos que hay dentro de forma usual, excepto cuando empezamos la segunda declaración (después del in), aquí guardamos el primer “nodo” o identificador que se declara dentro, a este lo llamamos nodo **apuntador** y además, guardamos el elemento que estaba justo antes de entrar al in, a este lo llamamos nodo **último**.
4. Al salir del private, hacemos que el nodo *anterior* al **apuntador**, sea el nodo **salida**, esto en caso de que el nodo apuntador no sea null, en caso contrario, nos vamos devolviendo a partir del último elemento, y tomamos el que esté inmediatamente después del nodo **último** y hacemos que su nodo anterior sea la **salida**.

   Al hacer esto, cuando se recorra la lista, se van a ignorar todos los nodos que estaban dentro de la primera declaración del private (antes del in), por lo que efectivamente se exporta la segunda declaración únicamente

Con lo anterior ya definido, decidimos implementarlo, añadiendo a la tabla de identificación, dos nuevas variables:

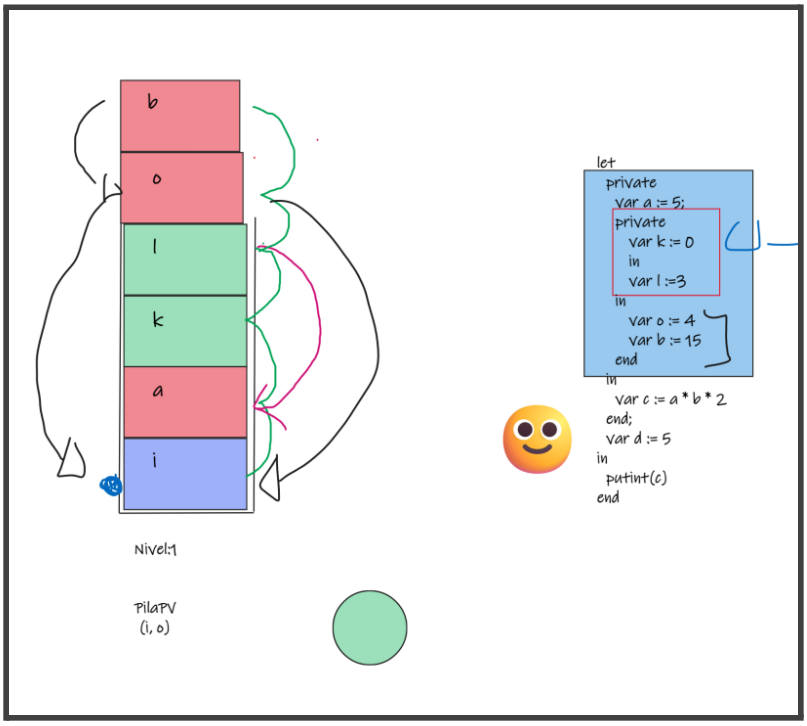
- **privatestack**: es una pila donde guardamos elementos del tipo (**salida**, **apuntador, último**), para así manejar correctamente privates anidados
- **waitingdeclaration**: una variable booleana de control, que indica cuando entramos dentro de la segunda declaración del private, para así definir el apuntador.

1. Realizamos un openPrivate():
   - Añadimos un elemento a la pila del privatestack, con un la salida = latest, y apuntador = null)
2. Visitamos la primera declaración
3. Hacemos un waitDeclaration()
   - Ponemos waitingdeclaration en true
4. Visitamos la segunda declaración y definimos el nodo apuntador, y el último.
5. Hacemos el closePrivate()
   - Hacemos un pop al privatestack, y hacemos que el nodo anterior al apuntador, sea el nodo de salida.

```
idTable.openPrivate();
ast.dAST.visit(this,null);
idTable.waitDeclaration();
ast.dAST2.visit(this, null);
idTable.closePrivate();
return null;
```

A Continuación mostramos el pseudocódigo y algunos diagramas que realizamos para poder realizar la solución:

```
class idTable{
    level
    latest
    privateStack (salida, apuntador)
    watingDeclaration: false

    enter(){
        if(watingDeclaration){
        setPrivatePointe(entry)
        }
    }

    openPrivate(){
        privateStack.push(latest, null)
    }

    waitDeclaration(){ waitingDeclaration = true}

    setPrivatePointer(entry){
        privatestack.top().apuntador = entry
        watingDeclaration = false
    }


    closePrivate(){
        if(privateStack.size != 0 ):
        temp = privatestack.pop()
            temp.apuntador.latest = temp.salida
    }
}
```



Describir el procesamiento de la declaración compuesta recursive.

Interesa la no-repetición de los identificadores de función o de procedimiento (Proc-Funcs) declarados por esa declaración compuesta y que estos identificadores sean conocidos en los cuerpos de todas las declaraciones de funciones o procedimientos (Proc-Funcs) que aparecen en una misma declaración compuesta recursive.

Para el procesamiento de la declaración recursive, se requirió recorrer todos el árbol de procfuncs para declarar el identificador y los parámetros de cada uno de los procfunc. Una vez declarados, se procede a hacer las visitas correspondientes de los árboles internos de cada procfunc, de este modo se permite verse mutuamente y se evitan los errores al llamar prematuramente a un procfunc no declarado.

1. Se recorre los procfunc y se declaran en el idTable
2. Se recorren de nuevo los procfunc y esta vez se visitan los ast internos.

```
public Object visitRecursiveDeclaration(RecursiveDeclaration ast, Object o) {

        RecursiveDeclaration temp = ast;
        while(temp != null){
            declareProcFunc(temp.pfAST);
            temp = temp.pfcsAST;
        }
        temp = ast;
        while(temp != null){
            temp.pfAST.visit(this, null);
```

```
            temp = temp.pfcsAST;
        }

        return null;


public void declareProcFunc(ProcFunc ast){
        if(ast.fAST != null){
            FuncDeclaration func = ast.fAST;
            func.T = (TypeDenoter) func.T.visit(this, null);
            idTable.enter (func.I.spelling, func); // permits recursion
            if (func.duplicated)
              reporter.reportError ("identifier \"%\" already declared",
                            func.I.spelling, func.position);
            idTable.openScope();
            func.FPS.visit(this,null);
            idTable.closeScope();
        }
        else{
            ProcDeclaration proc = ast.pAST;

            idTable.enter (proc.I.spelling, proc);
            if (proc.duplicated)
              reporter.reportError ("identifier \"%\" already declared",
                                    proc.I.spelling, proc.position);
            idTable.openScope();
            proc.FPS.visit(this, null);// permits recursion
            idTable.closeScope();

        }
    }

    @Override
    public Object visitProcFunc(ProcFunc ast, Object o) {
        if(ast.fAST != null){
            FuncDeclaration func = ast.fAST;
            func.T = (TypeDenoter) func.T.visit(this, null);
            idTable.openScope();
            func.FPS.visit(this, null);
            TypeDenoter eType = (TypeDenoter) func.E.visit(this, null);
            idTable.closeScope();
            if (! func.T.equals(eType))
              reporter.reportError ("body of function \"%\" has wrong type",
                                    func.I.spelling, func.E.position);
        }
        else{
            ProcDeclaration proc = ast.pAST;
            idTable.openScope();
            proc.FPS.visit(this, null);
            proc.C.visit(this, null);
            idTable.closeScope();
        }
        return null;
    }
```

Nuevas rutinas de análisis contextual, así como cualquier modificación a las existentes.

Modificaciones a la identification table

- **openPrivate**(){} Se encarga de abrir un espacio privado,generando un elemento en pila.
- **waitDeclaration**(){} genera la espera para la siguiente declaración luego del “in” en un private, además de guardar el último elemento presente en la tabla de identificación(último elemento del espacio privado antes del “in”)
- **setPrivatePointer**(IdEntry){}asigna la primer declaración encontrada al tope de la pila del private.
- **closePrivate**(){} se encarga de cerrar el espacio privado, reasignar los punteros correspondientes y sacar el tope de la pila.

Modificaciones al cheker

- Modificado el método **visitSimpleVname** para dar soporte a “ForVarDeclaration” y “InitializeVariableDeclaration”.
- Modificado el **visitArrayTypeDenoter**, se comprueba que en caso de existir límite superior, este sea mayor estricto que el límite inferior
- Agregado **visitRepeat(Repeat ast, Object o)** se encarga de hacer la revisión correspondiente al repeat y las variantes repeat while y repeat until, además de sus variantes con leave.
- Agregado **visitRepeatDo(Repeat ast, Object o)** se encarga de hacer la revisión correspondiente al repeat y las variantes repeat do while y repeat do until, además de sus variantes con leave.
- Agregado **visitForCommand(ForCommand ast, Object o)** se encarga de hacer la revisión correspondiente al For y las variantes for while y for until, además de sus variantes con leave.
- Agregado **visitForCommandDef(ForVarDeclaration ast, Object o)** se encarga de hacer la revisión correspondiente al ForVarDeclaration.
- Agregado **visitNothing(Nothing ast, Object o)** Visita el nothing, no tiene mayor objetivo que cumplir con la interfaz.
- Agregado **visitRecursiveDeclaration(RecursiveDeclaration ast, Object o)** se encarga de hacer la revisión correspondiente las declaraciones de recursive, y sus proc funcs.
- Agregado **declareProcFunc(ProcFunc ast)** se encarga de declarar los procFuncs y sus parámetros en la tabla de identificación, no realiza visitas.
- Agregado **visitProcFunc(ProcFunc ast, Object o),** realiza las visitas correspondientes a los sub-árboles de procFuncs.
- Agregado **visitPrivateDeclaration(PrivateDeclaration ast, Object o)**, se encarga de hacer la revisión correspondiente a la declaración privada, además de abrir y cerrar el espacio privado.
- Agregado **visitInitializedVarDeclaration(InitializedVarDeclaration ast, Object o)** Se encarga revisar **InitializedVarDeclaration**, y visitar sus subárboles.

Nuevos errores contextuales detectados, con los mensajes de error.

Error	Mensaje
Una variable/constante/proc/func ya fue declarado con ese mismo identificador	*“identifier \"%\" already declared"*
En private Dec1 y Dec2 son declaraciones generales	*“Can't be a single or compound declaration”*
En if las expresiones Exp y Expi deben ser de tipo booleano	*“Boolean expression expected here”*
En las variantes condicionadas del for_from_.._do_end, Exp3 debe ser de tipo booleano.	*“Boolean expression expected here”*
en repeat, EXP debe ser booleano	*“Boolean expression expected here”*
En for from Exp1..Exp2, EXP1 debe ser menor a EXP 2	*“Expression 2 must be bigger than Expression 1”*
En for from Exp1..Exp2, ambos deben ser integers	*“Integer expression expected here”*
Cuando un array es declarado con el límite inferior mayor que el superior	*“Lower bound is greater than upper bound”*

Pruebas que validan la fase de análisis contextual

Pruebas del compilador al procesar programas correctos

Objetivo del caso de prueba	Diseño del caso de prueba	Resultados esperados
verificar que detecta correctamente que es booleano en expression	repeat until 2=2 do nothing ; nothing end	compilación correcta, se compilo correctamente
detecta correctamente que las cotas del for sean válidas	for i from 0 .. 3 do putint(i); puteol() leave put('L') end	no da error
verificar que detecta correctamente que es booleano en expression	for i from 5 .. 6 while i = 5 do nothing ; nothing leave nothing end	no da error
verificar que detecta correctamente que es booleano en expression	for i from 5 .. 6 until i < 5 do putint(i); puteol() leave put ('W') end	no da error

verificar que no de error NO pasar el ID por referencia.	for i from 1 .. 6
	until i < 5
	do
	putint(i);
	puteol()
	leave
	putint(i)
	end;
		no da error

verificar que el private está exportando correctamente	let
	private
	var a := 5;
	var b := 15
	in
	var c := 2
	end;
	in
	putint(c)
	end
		no da error

ver que en el recursive	let recursive	correcto

	proc Ping (x : Integer) ~ if x > 0 then Impr ('I') ; Pong (x - 1) else nothing end end and proc Pong (a : Integer) ~ if a > 0 then Impr ('0') ; Ping (a - 1) else nothing end end and proc Impr (c : Char) ~ put (c) end end in Ping (6) ! imprime IOIOIO end	

verificar que el private está exportando correctamente	let
	private
	var a := 5;
	private
	
	private
	
	var k := 0
	in
	var l :=3
	end
	in
	var o := l;
	var b := 15
	end
	in
	var k := 0;
	var c := a * b * 2
	end
	in
	putint(c)
	end
		correcto

verificación general del array	let var lista : array 1 .. 10 of array 1 .. 2 of Integer; var a := 4 in lista[a] [3] := 3 end	correcto

Pruebas del compilador para detectar errores.

Objetivo del caso de prueba	Diseño del caso de prueba	Resultados esperados
verificar que detecta correctamente que es booleano en expression	repeat until 2+2 do nothing ; nothing end	dio error de pedir booleano
verificar que detecta la validez de las colas	for i from 9 .. 3 do putint(i); puteol() leave put('L') end	da error de que exp 2 debe ser mayor a exp 1
verificar que los elementos de las colas del if sean integers	for i from 'a' .. 3 do putint(i); puteol() leave put('L') end	da error de que espera un integer
verificar que detecta correctamente que es booleano en expression	for i from 5 .. 6 while i + 5 do nothing ; nothing leave nothing end	da error de no ser boolean

ID no se puede utilizar en la cota del for	for i from 1 .. i
	until i < 5
	do
	putint(i);
	puteol()
	leave
	put('W')
	end;
		incorrecto, no se puede id en los expression

verificar que se maneje el if como constante correctamente	for i from 1 .. 6
	until i < 5
	do
	i := 3;
	putint(i);
	puteol()
	leave
	put('W')
	end
		el id no se puede modificar, lo manejamos como constante

verificar que el private está ocultando correctamente	let
	private
	var a := 5;
	private
	
	private
	var k := 0
	in
	var l :=3
	end
	in
	var o := 4;
	var b := 15
	end
		el a no es visible

	in var c := a * b * 2 end in putint(a) end	
identificar que se verifica identificadores duplicados	let var a := 2; var a := 'd' in nothing end	no se puede declarar una variable dos veces con el mismo identificador
identificar que se verifica el tipo de datos al hacer una asignación	let var a := 2 in a := 'd' end	no se puede asignar un tipo de datos distinto
identificar que se verifica identificadores duplicados	let recursive func f (): Integer ~ 1 and func g (b : Integer): Integer ~ 3 and func f (a : Integer): Integer ~ 2 end in putint(f(10)) end	es inválido repetir nombres en una declaración mutuamente recursiva
	for i from 1 .. 10 do getint (var i); putint (i) end	debe dar un error, variable protegida no debería poder pasarse por referencia
verificar que se detecta la validez de las cotas del array	let var lista : array 20 .. 10 of Integer in nothing end	Cota inferior es mayor que cota superior

Pruebas dirigidas a validar los analizadores léxico y sintáctico

Se usaron los mismos casos de prueba del proyecto anterior.

Discusión, análisis y conclusiones de los resultados obtenidos.

Es importante al hacer el análisis contextual, tener un claro entendimiento del funcionamiento del lenguaje en el que se está trabajando, esto ya que, si no se tiene un conocimiento claro, se puede dar que se olviden partes importantes, o por el contrario, se metan verificaciones que no deberían existir.

Al realizar esta parte vimos lo importante de la fase léxica sintáctica, ya que nos permitió facilitar bastante el realizamiento de esta fase, y que el correcto funcionamiento de dicha fase es vital para el funcionamiento del compilador.

Además, algo que notamos especialmente con el private, es que es bueno graficar y hacer dibujos o similares, para poder visualizar de mejor manera conceptos que solo leídos o hablados no se llegan a explicar del todo claro, el poder visualizar bien y comprender mejor el cómo funciona, nos permitió llegar a una solución mucho más rápida, ya que pudimos planear y diseñar de antemano, y no haber entrado al código directamente a cambiar cosas esperando que funcionaran.

Experiencia de modificar fragmentos de un compilador / ambiente escrito por terceros.

La experiencia de modificar fragmentos de código de un compilador/ambiente escrito por terceras personas, puede ser frustrante. El hecho de que sea código ajeno y que para poder modificarlo sea completamente necesario el comprenderlo puede llegar a frustrar bastante en situaciones de proyecto. Si fuesen casos distintos donde uno lo hace por disfrute y con la intención de aprender por cuenta propia nos parece que podría llegar a ser muy interesante, retador y probablemente divertido.

Para poder realizar cambios en el código de otras personas es sumamente importante comprender cómo funciona el mismo (es por ello que se solicita bastante el uso de comentarios, pues a veces incluso uno mismo olvida cómo funciona), ya que si no se tiene este conocimiento es muy difícil saber exactamente dónde se deben realizar los cambios para que este funcione correctamente. Es por ello que muchas veces se invierte mucho del tiempo en lograr esa meta de comprensión.

Tareas realizadas por cada miembro del grupo de trabajo.

- ***Cambios sobre Triángulo:*** En grupo(Por medio de llamada en Telegram)
- ***Documentación:*** En grupo(Por medio de llamada en Telegram)

Indicar cómo debe compilarse el programa.

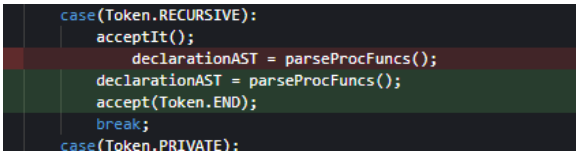
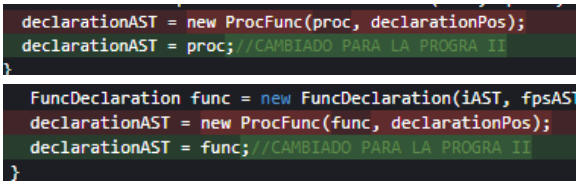
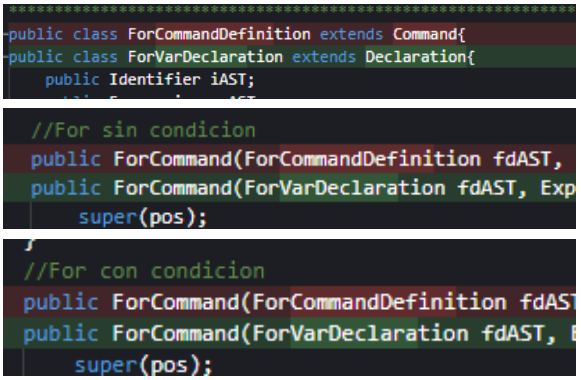
1. Utilizando el IDE NetBeans, abrir el proyecto fuente.
2. En la pestaña Projects, hacer click derecho sobre el proyecto
3. clickear la opción “Clean and Build”
4. Se generará un .jar en la carpeta dist

Indicar cómo debe ejecutarse el programa.

1. Tener el JRE instalado.
2. Doble click sobre el .jar del programa.
3. El programa se ejecutará.

Apéndices.

Cambios realizados al analizador lexico sintactico:

En parseCompoundDeclaration( ), en recursive, habiamos olvidado añadir el accept al TOKEN.END	
Arreglos en Proc, Func	
Arreglo en for, para manejar de manera correcta el ID	
Arreglos en Repeat Do