Prueba de Desarrollo II

Miguel Estevez

2017-0200

1. Condicional Controlado de Dijkstra

(a) Proponga y justifique los atributos necesarios que expandan la gramática dada para producir una gramática de atributos que traduzca un Condicional Controlado de Dijkstra a JAVA ó C#.

```
code = vector<string>
variables = set<variable>
```

code sera el código generado hasta esa regla. Asi se podra tener la parte del codigo que corresponde tanto a cada bexpr y su código correspondiente si es verdadera. Lo cual serviría para poder generar de manera correcta las clases para grd_if_body.

variables seria todas la variables que han sido usadas. Lo cual ayudaría para saber que variables son usada en cada grd_if_body para que asi puedan ser atributos de la clase que se le creara.

(b) Especifique las acciones semánticas necesarias para implementar la gramática de atributos sugerida en el punto anterior y construya dicha gramática.

```
grd_if-> IF grd_if_body END_IF
    {
        generarCodigoCorrida(
    }
    ;
grd_if_body -> bexpr "->" code "||" grd_if_body
    {
        set<variable> variables = union($1.variables,$3.variables);
        generarClase($1.code,$3.code,variables);
    }
    ;
grd_if_body -> bexpr "->" code
    {
        set<variable> variables = union($1.variables,$3.variables);
        generarClase($1.code,$3.code, variables);
    }
    ;
```

generarClase(string bexp, string code, vector variables);

- **bexp** es el código generado para la expresión booleana que va en el if.
- **code** es el código que se ejecutara si la expresión booleana es verdadera.
- **variables** son la variables que se necesitan para ejecutar el código y la expresión booleana;

```
struct variable {
    string lexema;
    type tipo;
}
map<string, vector<variable>> ifBodyVariables;
stack<pair<string,vector<variable>>>StackIfClasses;
int contador = 0;
string generadorNombre(){
    return "If"+to_string(++contador);
}
string generarClase(string exprCode, string bodyCode, vector<variable>variables){
    string nombreClase = generadorNombre();
    /** generacion de codigo en java **/
    addStatement("public class "+nombreClase+" extends Thread {");
    for(variable var: varieables){
        addStatement("private "+getType(var.tipo)+" "+var.lexema+";");
```

```
    }
    addStatement("public "+nombreClase+"(");
    for(int i = 0;i < variables.length;i++){
        variable var = variables[i];
        string temp = getType(var.tipo)+" "+var.lexema;
        if(i < variables.length -1){
            temp += ",";
        }
        else{
            temp +=') {'
        }
        addStatement(temp);
    }
    for(int i = 0;i < variables.length;i++){
        addStatement("this."+variables[i].lexema+"="+variables.lexema+";");
    }
    addStatement("}");
    addStatement("public void run() {");
    addStatement("if("+exprCode+") {");
    addStatement(bodyCode);
    //Cierre del if
    addStatement("}");
    //Cierre de la funcion run()
    addStatement("}");
    //Cierre de la clase
    addStatement("}");
    StackIfClasses.push(make_pair(nombreClase,variables));
    return nombreClase;
}
string generarCodigoCorrida(){
    //aqui se generaria el codigo de corrida
    //se crearan un objecto de cada clase creada
    //luego se pondran dentro de un arreglo Thread[]
    //Se correran cada Thread
}
```

(c) Justifique el porqué la gramática de atributos dada en el punto anterior, resuelve el problema de traducción planteado.

Esta gramática resuelve el problema planteado ya que trabaja a cada una de los if de maneras separadas con la creación de clases que heredan de Thread. Ademas esto hace que se puede correr de paralelamente porque heredan de Thread y la implementacion de la función run() seria el código equivalente en java de uno de los estatutos del if. Hay que tener una forma de poder guardar todo el código generado que va dentro de cada if statement. A la vez del que va en la expresión.

Lo primero que hay que tener en cuenta es que cada uno de los grd_if_body serán traducidos en JAVA como su propia clase en java. La cual tendrá que heredar de Threads. Cada una de estas clases sera provenidas por cada una de las variables que necesita para realizar su acción. Por ejemplo:

```
If Y Mod 2 = 0 -> Begin
                    Z := 2 * Y + 1;
                    Write(Z)
                  End
```

Se traduciría en java a:

```
    public class If1 extends Threads{
        private int Y;
        private int Z;
        public If1(int Y, int Z) {
            this.Y = Y;
            this.Z = Z;
        }
        public void run() {
            if(Y%2==0) {
                Z = 2 * Y + 1;
                System.out.println(Z);
            }
        }
    }
```

El hecho de que hereden de la clase Thread hará que se pueda tener un arreglo de **Thread[]** donde estarán cada un objecto de cada clase que se van a generar. Para que luego, se pueda iniciar cada thread con un for para evaluar cada una de las expresión de forma paralela. De la siguiente manera.

```
int Y;
int Z;
int W;
If1 var1 = new If1(Y,Z);
If2 var2 = new If2(Z, W);
If3 var3 = new If3(W, Y);
Thread array[3] = new Thread[]{var1,var2,var3};
for(Thread i : array){
    i.start();
}
```

2. Definición-uso usando la noción de contexto o alcance dinámico.

Se tendría que hacer una doble pasada para poder realizar bien este procedimiento. Ya que una variable que no fue declarada en un procedimiento puede que haya sido declarada en otro que llame a este. Entonces no se puede afirmar declaración uso hasta que no se pase de nuevo recorriendo el grafo de llamadas y revisando en las que no fueron declaradas una estructura de grafos donde una llamada a una función desde otra significaría, sea A la funciona que llama a la función B. Entonces seria A -> B.

La estructura de grafo seria un lista adyacente

```
map<string,vector<string> >listaAdj;
```

- El string seria el id de un procedures.
- El vector de string es los procedimientos que son llamados.

```
map<string,set<variable> > declaradas,noDeclaradas;
```

- El string seria el id del procedure.
- Si una variable ha sido declarada en un procedimientos esta se insertaría en el set de declaradas.
  En el caso contrario se insertaría en las que no fueron declaradas.

(a) Construya una gramática de atributos que resuelva el problema de definición-uso usando la noción de contexto dinámico y el fragmento relevante de la gramática libre de contexto de Mini-P.

```
program: title _SEMI block
        {
            checkNonDeclaredVariables();
            addICStatement(quadrupleOperator::HALT,0,0,0);
        }
        ;
procdef : ptitle _SEMI block
        {
            insertAllDeclared();
            insertAllNonDeclared();
        }
        ;
```

```
    stack<set<string>>llamadas;
    map<string,vector<string> >listaAdj;
    map<string,set<variable> > declaradas,noDeclaradas;
    stack<set<variable>>Usadas;
    variable{
        string name;
        type dataType;
        bool declarada;

        bool operator==(const variable& o){
            if(o.name == name && dataType == o.dataType && declarada == o.declarada)
                return true;
            return false;
    }
    void installOnTable(string varName, int scope, type varType) {
```

```
        transform(varName.begin(), varName.end(), varName.begin(), ::tolower);
        variable var;
        var.dataType = varType;
        var.name = varName;
        usadas.top().push_back(var);
        if(!checkTable(varName, scope)) {
            symbTableContent varContent;
            varContent.dataType = varType;
            symbTableStack[scope][varName] = varContent;
            REPORTS += "\ndeclaration of ... " + varName + " type " + typeLexeme[varType] + " scope " +
to_string(scope) + "\n";
        } else {
            REPORTS += "\nERROR Identifier redeclaration --> " + varName + " type " +
typeLexeme[varType] + " scope " +  to_string(scope) + " in line " + to_string(lineCounter) + ", was
already declared\n";
            semanticError = true;
        }
    }
   identifier getDeclaration(string varName, int scope){
        transform(varName.begin(), varName.end(), varName.begin(), ::tolower);
        variable var;
        var.declarada = false;
        var.name = varName;
        var.dataType = type::nonetype;
        int foundScope = scope;
        for(int i = scope; i >= 0; i--){
            if(symbTableStack[i].count(varName)) {
                var.dataType = symbTableStack[i][varName].dataType;
                var.tempNumber = symbTableStack[i][varName].tempNumber;
                foundScope = i;
                break;
            }
        }
        //Se quito para poner un error semantico si no es encontrado
        if(var.dataType != type::nonetype) {
            var.declarada = true;
            REPORTS += "\nuse of ... " + varName + " type " + typeLexeme[dvar.dataType] + " declared in
scope " + to_string(foundScope) + " used in scope " + to_string(scope)  + "\n";
        }
        usadas.top().push_back(var);
        return var;
    }
    void insertAllVariables(string funcion){
        set<variable>temp = Usadas.top();
        Usadas.pop();
        for(variable x : temp){
            if(x.declarada){
                declaradas[funcion].push_back(x);
            }
            else {
                noDeclaradas[funcion].push_back(x);
            }
        }
    }
    void llamadasFunciones(string function){
        set<string>temp = llamadas.top();
        llamadas.pop();
        for(String x: temp){
            listAdj[function].push_back(x);
        }
    }
    void checkNonDeclaredVariables(string funcion){
        set<string>visitados;
        queue<string> visitar;
        visitar.push(function);
        visitados.insert(ac);
        while(!visitar.empty()){
            string ac = visitar.front();
            visitar.pop();
            for(String temp: ListAdj[ac]){
                if(!visitados.count(temp)){
                    for(variables x: declaradas[ac]){
                        //si esa variable no fue declaradas en la funcion llamada, entonces se agrega
                        if(!declaradas[temp].count(x)){
```

```
                        declaradas[temp].insert(x);
                    }
                    //Si no esta declaradas en la funcion llamada, se remueve
                    if(noDeclaradas[temp].count(x)){
                        noDeclaradas[temp].remove(x);
                    }
                }
                visitados.insert(temp);
                visitar.push(temp);
            }
            else{
                //El caso que ya ha sido visitadas
                for(variables x: declaradas[ac]){
                    //Si no esta declaradas en la funcion llamada, se remueve
                    if(noDeclaradas[temp].count(x)){
                        noDeclaradas[temp].remove(x);
                    }
                }
            }
        }
    }
    //chequea que no halla ninguna variables que no haya sido declarada.
    for(auto it:noDeclaradas){
        if(it.second.size()>0){
            semanticError = true;
        }
    }
}
```

(b) Justifique, usando el ejemplo propuesto el porqué la solución dada en el punto resuelve el problema planteado

En este ejemplo, la variable tanto la variable x no ha sido declarada en el procedure B como la variable y en el procedure C. Entonces como se plantea, esto en la primera pasada no se tomara como un error de declaración uso, sino que se registrara como una variable que no ha sido registrada en esa función. También estarán las que si lo están. En el momento que se el block que la producción de program, ya todas las funciones habrán sido declaradas, en este caso A,B y C, junto a las variables que han sido usadas y declaradas en cada una. Se recorrerá un bfs comenzando por la función A donde va a recorrer las funciones a que esta llama y va a insertar las que fueron declaradas en esas función en las de sus hijos. Ademas ira chequeando si de las que tiene declaradas no son declaradas por uno de sus hijos. Por ejemplo la función B no tiene declarada la variable x pero A si, entonces se remueve del conjunto de variables no declaradas de B. Lo mismo pasa con C y su variable "y" que no ha sido declarada. Entonces cuando termine el bfs, se recorrerá el conjunto de variables no declaradas de cada uno y si encuentra que una función tiene alguna, ahí se reporta que hubo un error semántico.

(c) Resuelve la solución planteada el caso de llamadas recursivas? Justifique

Esta solución si tiene planteado en poder resolver la llamadas recursivas, como seria **A** llama a **B** y **B** llama a **A**. Ya que si un nodo(seria una función) ya halla sido visitado, no se va a poner en la cola para ser visitado otra vez pero si se va a resolver si tiene alguna variable no declarada y se removerá si ha sido declarada en la otra función.