Attribute Grammar de comprobación de tipos

Nodo	Predicados	Reglas Semánticas
program → *definitions*:definition*		
varDef:definition → *type*:type *name*:String		
funcDef:definition → *ident*:String *type*:type *localDefs*:varDef* *sentences*:sentence*		If return ∈ funcDef.sentences ¿ return.tipoRetorno = funcDef.type.tipoRetorno
structDef:definition → *name*:String *records*:recordDef*		
recordDef:definition → *name*:String *type*:type		
intType:type → λ		
realType:type → λ		
charType:type → λ		
arrayType:type → *length*:int *typeOf*:type		
structType:type → *name*:String		
functionType:type → *tipoRetorno*:type *parametros*:varDef*		
print:sentence → *expression*:expression *tipoPrint*:int	esPrimitivo(expression.tipo)	
assignment:sentence → *left*:expression *right*:expression	esPrimitivo(left.tipo) left.modificable mismoTipo(left.tipo, right.tipo)	
callProcedure:sentence → *ident*:String *arguments*:expression*	checkParamsForCallProc(callProcedure);	
ifElse:sentence → *condition*:expression *ifSentences*:sentence* *elseSentences*:sentence*	Condition.tipo == IntType	

read:sentence → *expression*:expression	esPrimitivo(expression.tipo) expression.tipo.modificable	
return:sentence → *expression*:expression	mismoTipo(expression.tipo, return.tipoRetorno)	
while:sentence → *condition*:expression *whileSentences*:sentence*	Condition.tipo == IntType	
arithmetic:expression → *left*:expression *operator*:String *right*:expression	esPrimitivo(left.tipo) esPrimitivo(right.tipo) mismoTipo(left.tipo, right.tipo)	Arithmetic.tipo = left.tipo Arithmetic.modificable = false;
callFunction:expression → *ident*:String *arguments*:expression*	checkParamsForCallFunction(callFunction);	callFunction.tipo = callFunction.definicion.tipo.tipoRetorno callFunction.modificable = false;
cast:expression → *castType*:type *expression*:expression	esCasteable(castType, expression.tipo)	Cast.tipo = castType Cast.modificable = false;
comparison:expression → *left*:expression *operator*:String *right*:expression	esPrimitivo(left.tipo) esPrimitivo(right.tipo) mismoTipo(left.tipo, right.tipo)	Comparison.tipo = IntType; Comparison.modificable = false;
fieldAccess:expression → *ident*:expression *fieldName*:String	Ident.tipo == StructType	fieldAccess.tipo = dot(fieldAccess.fieldName, ident.tipo.definicion) fieldAccess.modificable = true;
indexing:expression → *ident*:expression *index*:expression	Ident.tipo == ArrayType Index.tipo == IntType	Indexing.tipo = indexing.ident.tipo.typeOf
logic:expression → *left*:expression	Left.tipo == IntType	Comparison.tipo = IntType; Comparison.modificable = false;

operator:String *right*:expression	Right.tipo == IntType	
not:expression → *expression*:expression		
unaryMinus:expression → *expression*:expression		
variable:expression → *name*:String		Variable.tipo = variable.definicion.tipo Variable.modificable = true

charConstant:expression → *value*:String		charConstant.tipo = charType;
		
		charConstant.modificable = false;

intConstant:expression → *value*:String		intConstant.tipo = intType;
		
		intConstant.modificable = false;

realConstant:expression → *value*:String		realConstant.tipo = realType;
		
		realConstant.modificable = false;

Recordatorio de los operadores (para cortar y pegar): ⇒ ⇔ ≠ ∅ ∈ ∉ ∪ ∩ ⊂ ⊄ ∑ ∃ ∀

Atributos

Nodo/Categoría Sintáctica	Nombre del Atributo	Tipo Java	Heredado/Sintetizado	Descripción
Expresión	Tipo	Type	S	Tipo de la expresión
Expresión	Modificable	Boolean	S	Indica si se puede modificar la expresión
Return	tipoRetorno	Type	H	Se asigna el tipo de retorno al statement return desde el nodo padre de la definición de la función

```
// retorna true si tipoInicial es casteable a tipoFinal
private boolean esCasteable(Type tipoInicial, Type tipoFinal) {
    if (esPrimitivo(tipoInicial) && esPrimitivo(tipoFinal)) {
        if (!mismoTipo(tipoInicial, tipoFinal)) {
            return true;
        }
        return false;
    }
    return false;
}

// comprueba que un campo es parte de su struct y retorna su tipo
private Type dot(String fieldName, StructDef structDef) {
    for (Definition r : structDef.getRecords()) {
        RecordDef record = (RecordDef) r;
        if (record.getName().equals(fieldName))
            return record.getTipo();
    }
    return null;
}
```

```
// retorna true si tipoA es del mismo tipo que tipoB
private boolean mismoTipo(Type tipoA, Type tipoB) {
    if (tipoA.getClass().equals(tipoB.getClass())) {
        return true;
    }
    return false;
}

// retorna true si tipo pertenece a uno de los tipos primitivos(int,real,char)
private boolean esPrimitivo(Type tipo) {
    if (tipo instanceof IntType) {
        return true;
    }
    if (tipo instanceof RealType) {
        return true;
    }
    if (tipo instanceof CharType) {
        return true;
    }
    return false;
}
```

```java
// si los parametros de la funcion que corresponde con una
// llamada a funcion no coinciden
private void checkParamsForCallFunction(CallFunction cp){
    FunctionDef f = (FunctionDef) cp.getDefinicion();
    FunctionType ft = (FunctionType) f.getTipo();


    if(ft.getParamDefs().size() != cp.getArguments().size()) {
        errorManager.notify("Type Checking",
                "CALL FUNCTION: El número de parámetros no coincide", cp.getStart());
    } else {
        for(int i = 0; i <= cp.getArguments().size()-1; i++){

            VarDefinition vdef = (VarDefinition) ft.getParamDefs().get(i);

            if(cp.getArguments().get(i).getType().getClass() != vdef.getType().getClass()){
                errorManager.notify("Type Checking",
                        "CALL FUNCTION: El tipo del argumento es: " + cp.getArguments().get(i).getType().toString() +
                        " pero se esperaba: " + vdef.getType().toString(), cp.getArguments().get(i).getStart());
            }
        }
    }
}
```

```java
// si los parametros de la funcion que corresponde con un
// procedimiento no coinciden
private void checkParamsForCallProc(CallProcedure cp){

    FunctionDef f = (FunctionDef) cp.getDefinicion();
    FunctionType ft = (FunctionType) f.getTipo();


    if(ft.getParamDefs().size() != cp.getArgs().size()) {
        errorManager.notify("Type Checking",
                "CALL PROCEDURE: El número de parámetros no coincide", cp.getStart());
    } else {
        for(int i = 0; i <= cp.getArgs().size()-1; i++){

            VarDefinition vdef = (VarDefinition) ft.getParamDefs().get(i);

            if(cp.getArgs().get(i).getType().getClass() != vdef.getType().getClass()){
                errorManager.notify("Type Checking",
                        "CALL PROCEDURE: El tipo del argumento es: " + cp.getArgs().get(i).getType().toString() +
                        " pero se esperaba: " + vdef.getType().toString(), cp.getArgs().get(i).getStart());
            }
        }
    }


}
```