Documentación de traductores

Objetivo general

* Diseñar e implementar un evaluador de programas sintáctica y semánticamente válidos de acuerdo al lenguaje GuardedUSB

¿Cómo correr el programa?

El usuario debe poseer python3 instalado en su computadora, así como la herramienta ply, ya que el lexer y el parser del intérprete requieren de funciones que esta ofrece para funcionar.

Si se cumplen estas condiciones, entonces el usuario deberá, desde la terminal de su coputadora, ejecutar el comando python3 GuardedUSB.py <nombre del archivo> donde el archivo contiene un programa escrito en GuardedUSB. Deberá poseer extensión .gusb, caso contrario el interpretador no lo leerá y notificará del error al usuario.

¿Cómo funciona el evaluador?

El evaluador de GuardedUSB funciona como un método para la clase Node llamado “evaluator”, que adicionalmente es heredado por todas las demás clases que son las utilizadas para construir el Árbol sintáctico abstracto (AST por sus siglas en inglés) correspondiente a un programa escrito en GuardedUSB. Para que el evaluador funcione correctamente, debe ser llamado desde la raíz del árbol y previamente debe haberse llamado al método checkStaticErrors sobre la misma, de manera de asegurarse que el programa input no tenga errores estáticos y que el AST sea decorado debidamente.

Es importante destacar, antes de describir más detalladamente el funcionamiento del evaluador, que para esta entrega fue modificada la estructura de la tabla de símbolos para almacenar los valores de cada variable declarada. Esto facilitó tanto la búsqueda de los valores asignados a las variables, como la detección de variables no inicializadas dentro de las operaciones.

De manera similar a checkStaticErrors, la función evaluator, no es más que un wrapper que simplemente inicializa la pila vacía de tablas de símbolos y luego llama a evaluatorAux sobre el mismo nodo que la llamó. EvaluatorAux se encarga de, dependiendo de la clase a la que pertenezca el nodo que que la invoca o la categoría del mismo (categoría es un atributo que permite identificar la entidad representada por el nodo), seleccionar y llamar el procedimiento adecuado para evaluar dicha instrucción. Por ejemplo, si encuentra un BlockNode, se debe llamar a evalBlockNode que empilará la tabla de símbolos del bloque (si la hubiera) antes de proceder a evaluar el contenido del mismo y al terminar esta tarea desempilará la tabla para evitar tener problemas de alcance de variables.

Otro ejemplo relevante son los evaluadores de expresiones booleanas y aritméticas: El parser y la forma en que el AST es construido permiten que con un simple recorrido postorden del subárbol que representa la expresión, se pueda obtener el resultado de la misma.

Todos estos procedimientos complementarios, llaman sobre sus nodos hijos a cualquier otro método que haga falta para evaluar las instrucciones o expresiones que estos representan. Esta serie de llamadas (que en la mayoría de los casos resulta en la recursión directa o indirecta de las funciones) permite realizar un recorrido similar al postorden del AST, aunque no se puede afirmar que sea exactamente este tipo de recorrido, ya que ciertos subárboles del AST (Correspondientes a las instrucciones do y for) pueden ser recorridos más de una vez antes de continuar con el resto del árbol.

Cabe acotar que el evaluador también se encarga de la notificación de errores dinámicos, por ejemplo la utilización de índices de arreglos fuera de rango. Este caso en particular es fácil de verificar, ya que la estructura utilizada para almacenar la información de los arreglos en la tabla de símbolos contiene información de los índices mínimos y máximos de cada arreglo

Explicación detallada de las funciones

**evaluator** Esta función es la que llamamos desde el archivo de la semantica, nos permite inicializar nuestra pila y hacer un llamado al evaluatorAux que hace una evaluación dependiendo de la instrucción

**evaluatorAux** Esta función nos permite dependiendo del tipo de instrucción, hacer las evaluaciones correspondientes

**evalDo** Nos permite evaluar el Do, donde para el iterador igual a False, vamos a evaluar cada guardia hasta que el iterator sea True

**evalGuard** Evaluamos las guardias, para ello también vamos a tener un parametro iterator. Evaluamos la expresión booleana a la guardia y si es True, evaluamos el bloque de código de instrucciones asociado. Si la longitud es 3, seguimos evaluando las guardias

**evalArrayExpForPrint** Nos permite evaluar las expresiones de tipo array en un print, con la diferencia de cuando el category es Ident, se hace un llamado a la función checkInitIdentForPrint

**evalArrayExp** Esta función nos permite evaluar las expresiones de tipo array

**evalBoolExp** Esta función nos permite evaluar las expresiones booleanas

**evalArithmeticExp** Esta función nos permite evaluar las expresiones ariméticas

**evalAsig** Evalúa la instrucción de la Asignación, en donde para el tipo de dato correspondiente, asignamos el valor a la variable

**evalConcat** Esta función evalua la concatenación teniendo a la expresión que le corresponde al hijo del nodo, evaluando el contenido del String ya que lo utilizaremos para efectos del print

**checkInitIdentForPrint** Chequea el Ident para un arreglo, pero además nos ayuda a construir para el print de arreglo para cada item, el print correspondiente con el indice en el cual forma parte

**evalStringContent** Esta función evalúa el contenido del String dado, si es de tipo aritmético o booleano los evalúa correspondientemente En cambio si es array, además de evaluarlo y obtener la lista de items, irá construyendo el print como corresponde (la relación de cada item con el indice que pertenece)

**evalPrint** Nos permite evaluar el contenido del String dependiendo del caso y del tipo

**evalPrintAux** Esta función nos permite diferenciar entre println y print. Además de "construir" el o los valores concatenados para imprimir

**evalFor** Función de evaluación de la instrucción del For. Obtenemos los indices correspondientes (inicio y fin) para crear un intervalo de iteracciones, para cada iteracción se evaluará el bloque que contiene el código del For

**evalRead** Es la evaluación del Read. Tenemos el caso de introducir un arreglo y un variable que se le pedirá al usuario que ingrese. Además al preguntar al usuario, se le muestra el tipo de variable o la longitud correspondiente al arreglo

**InitializeArray** Inicializa el Array con los valores que le corresponde

**arrayValue** si se hace consulta sobre un arreglo, se calcula el arreglo que fue consultado, en caso de que incluya asignaciones. Si no incluye asignaciones, solo retona el valor almacenado del arreglo. Si el arreglo no incluye asignaciones devuelve el valor del arreglo

**findArray** Esta función se utiliza para verificar sobre qué arreglo debemos llamar la función embebida y si tiene asignaciones, que ninguna se haga fuera del rango del arreglo

**checkInitIdent** Retorna el valor de una variable. Si no ha sido inicializada, devuelve error

**setVarValue** Dada una variable, guarda su valor en la tabla