Programming Languages Project

D. M. G. C. Dassanayake

170097P

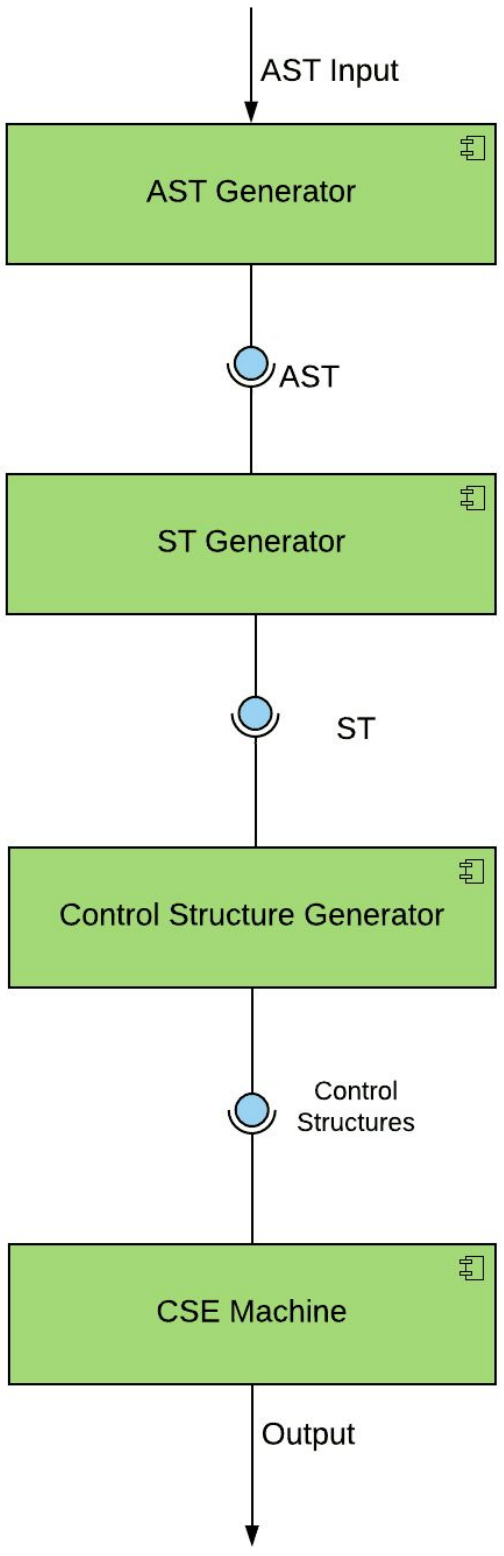
Contents

Overview

The project can be broadly segmented into three components namely,

1. AST Generator
2. ST Generator
3. Control Structure Generator
4. CSE Machine

Each component accepts input and passes the processed output to the next component along the pipeline.


AST Input
AST Generator
AST
ST Generator
ST
Control Structure Generator
Control
Structures
CSE Machine
Output

Components

1. AST Generator Component

The ASTGenerator reads the given AST from the input and processes it to create an AST.

a. ASTGenerator Class

A tree structure is used to store the entire AST.
The following methods are implemented within the class in order to achieve this functionality.

ArrayList<String> readInput()	Reads the input from the file and returns nodes as a string arraylist
Void printAST(Node node)	Prints the AST when the root node is given
Node createNode(String name, Node parent, int depth)	Creates a node in the AST
int findDepth(String name)	Find the depth from the root to a node given its name
Node addNode(Node lastNode, String name)	Creates a node using createNode() and sets its parent
SyntaxTree createAST()	Creates and returns the complete syntax tree

2. ST Generator Component

The ST Generator accepts the AST as the input and standardizes all necessary Nodes.
The following types of nodes are standardized.

- let
- where
- function_form
- Lambda
- within
- and
- rec

a. STGenerator Class

STGenerator acts as the driver class in the STGenerator Component. It will conduct a preorder traversal and call standardize method of Standardizer class to standardize each node.

The methods in the STGenerator class are,

void traverse(Node node, SyntaxTree ast)	Traverses the ast in a preorder traversal
void createST(SyntaxTree ast)	The helper function calling traverse()

b. Standardizer class

The Standardizer class identifies the type of each node and uses specific methods to standardize all required nodes.

The methods in Standardizer are,

static void standardize (Node node, SyntaxTree ast)	Identifies the type of node and calls specific standardizer methods
static void standardizeLet (Node node, SyntaxTree ast)	Standardizes 'let' and alters the Syntax Tree
static void standardizeWhere (Node node,	Standardizes 'where' and alters the Syntax

SyntaxTree ast)	Tree
static void standardizeFcnForm (Node node, SyntaxTree ast)	Standardizes ‘function_form’ and alters the Syntax Tree
static void standardizeTuple (Node node, SyntaxTree ast)	Standardizes ‘tau’ and alters the Syntax Tree(obsolete due to CSE rules 9,10)
static void standardizeMultiParameter (Node node, SyntaxTree ast)	Standardizes ‘,’ and alters the Syntax Tree(obsolete due to CSE rule 11)
static void standardizeWithin (Node node, SyntaxTree ast)	Standardizes ‘within’ and alters the Syntax Tree
static void standardizeUnary (Node node, SyntaxTree ast)	Standardizes unary operators and alters the Syntax Tree(obsolete due to CSE rules 6)
static void standardizeBinary (Node node, SyntaxTree ast)	Standardizes binary operators and alters the Syntax Tree(obsolete due to CSE rule 7)
static void standardizeAt (Node node, SyntaxTree ast)	Standardizes @ and alters the Syntax Tree
static void standardizeSimultaneous (Node node, SyntaxTree ast)	Standardizes simultaneous nodes and alters the Syntax Tree
static void standardizeConditional (Node node, SyntaxTree ast)	Standardizes ‘->’ and alters the Syntax Tree(obsolete due to CSE rule 8)
static void standardizeRec (Node node, SyntaxTree ast)	Standardizes ‘rec’ and alters the Syntax Tree

3. Control Structure Generator.

In order to feed the ST to the CSE machine, the control structures are generated using the control structure generator.

a. ControlStructGenerator Class

ControlStructGenerator traverses the ST in a preorder traversal and creates the control structures necessary to be fed into the CSE machine. It uses an arraylist of arraylists to store the set of control structures.

The methods in the class are,

ArrayList<ArrayList<ControlElement>> preOrderTraverseHelper (SyntaxTree st)	Initializes the preorder traversal with the root node
void preOrderTraverse(Node node, int currentcs)	Recursively iterates through all nodes in a preorder manner and creates the control structures

4. CSE Machine

CSE Machine takes the processed control Structure and processes the controls and produces the last result. The control and the environment stacks are implemented with Java arrays while the stack is implemented using a stack implementation.

a. CSEMachine class

CSEMachine class intakes the control structure and manages the control, stack and the environment according to the CSE rules to produce the final output.

void runCSE()	Runs the CSE Machine according to rules and generate output
ControlElement getCurrent(String key)	Returns the element currently on the top of the control
ArrayList<ControlElement> getTuple(String key)	Returns a tuple of bounded elements of an element of a given name
String tuplePrinter(ControlElement element)	Implements the printer functionality
boolean integerValidator(String integer)	Returns true if given string can be parsed to int, false otherwise
boolean booleanValidator (String integer)	Returns true if given string can be parsed to int, false otherwise

b. Environment class

The class that creates the blueprint for the environment stack elements.

5. Common Classes and Data Structures

a. Node Class- Tree node class
b. SyntaxTree Class- Tree class used to create AST and ST
c. ControlElement Class- Element class which is used to create objects that are stored in control and the stack.
d. NodeType enum- Type of the Node