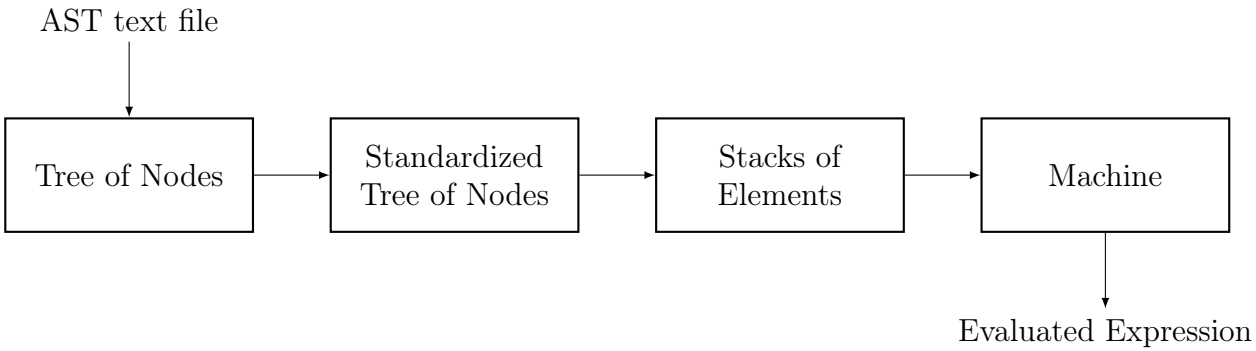
CS3152 Programming Languages Project Report

K. D. Sunera Avinash Chandrasiri (170081L)

June 19, 2020

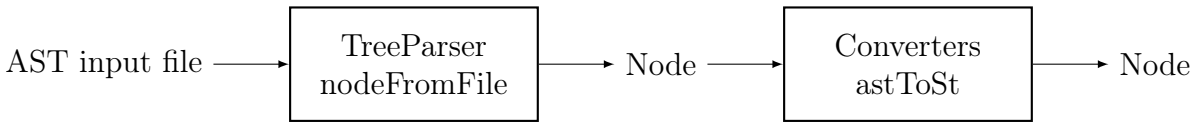
1 Overall Structure

The structure of the program is divided mainly into 2 packages; **tree package** and **cse package**. Tree package functions will read the AST structure from the file and standardize the tree. Tree package will mainly rely on **Node** objects. Cse package functions will traverse the standardized tree structure and generate the control structures and evaluate the result. Similar to tree package, cse package will mainly rely on stacks of **Element** objects.



2 Tree Package

This package will focus on reading the AST structure from the file and standardizing the tree. Following are the each class in this package and their objectives.



2.1 TreeParser class

Contains static utility functions to read the AST from file. *nodeFromFile()* will read a *Node* containing the AST structure from a given file.

Node nodeFromFile(String fileName)	Parse Node from the given file
Node nodeFromString(List<String> lines)	Parse Node from a list of string lines
List<String> readFile(String fileName)	Helper function to read a given file and output lines as an array of strings
String trimLeadingDots(String source)	Helper function to trim leading dots from a string
String unescapeJavaString(String st)	Helper function to unescape a string that contains standard escape sequences (\n, \t)

2.2 Node class

Node will represent a tree node in the AST/ST structure. Following are the attributes of the Node;

ArrayList<Node> children	References to child nodes.
Node parent	Reference to parent node (*null* if this node is a root node).
String label	Type of the node; let, where, lambda, id, str, int
String value	String representation of the value of the node(eg: string value of int node). This is not-null for leaf nodes.

The class contains methods such as **copy()** method and methods to manipulate the tree structure such as **clearChildren()** and **addChild()**.

2.3 Converters class

A utility class to standardize the abstract syntax tree.

void astToSt(Node node)	Converts the abstract syntax tree to a standardized tree recursively from the node. This uses below functions
void stForLet(Node rootNode)	Standardize the LET node
void stForWhere(Node rootNode)	Standardize the WHERE node
void stForFuncForm(Node rootNode)	Standardize the FCN_FORM node
void stForAnd(Node rootNode)	Standardize the AND node
void stForRec(Node rootNode)	Standardize the REC node
void stForLambda(Node rootNode)	Standardize the Multi parameter function (lambda) node
void stForWithin(Node rootNode)	Standardize the WITHIN node
void stForAt(Node rootNode)	Standardize the @ node

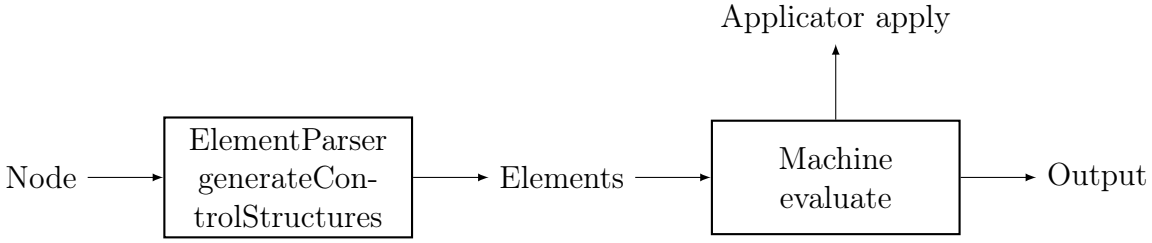
2.4 Other classes

AstException is a runtime exception which is thrown when standardizing the abstract syntax tree fails due to some reason.

NodeWithDepth is a Node class extended with depth attribute to help in reading the AST structure from the input file.

3 CSE Package

This package will traverse the standardized tree structure and generate the control structures and evaluate the result.



3.1 ElementParser class

This is a parser that will convert AST to Element stacks by pre-order traversal. This will create control structures; stacks of *Value*.

generateControlStructures(Node root)	Generates the control structure array by pre-order traversal
generateCsForLambda(node, controls, ...)	Split the control structure on *lambda* nodes and use a *delta* node to traverse in the sub tree
generateCsForIf(node, controls, ...)	Split *if* node to *then* and *else* delta nodes and traverse in sub-trees
generateCsForTau(node, controls, ...)	Add number of elements in tau node and traverse in each sub-tree

3.2 Element package

3.2.1 Element abstract class

Super element class which holds a **label** string. Label represents the type of the element such as lambda, delta.

3.2.2 Value subclass

This is the type of all elements except tuples. These have an additional **value** attribute of type **String** holding a string representation of the value stored inside the element. Value is null for types like *true*, *false*, *id* and not-null for types like *int*, *str*.

3.2.3 Tuple subclass

This is the element of tuple type which can hold multiple elements. The **value** attribute of this subclass is of **Element[]** type which allows to hold child elements. The label of this element is *tuple*.

3.3 Environment class

This is a class to represent the environment tree of the cse machine. Each environment has following attributes/methods.

Environment parent	Parent environment. Null if this is the primary environment.
HashMap<String, Element> memory	Memory to hold values of each reference. In the primary environment, this will hold all the in-built function names. In other environments, memory will be initialized as empty.
void remember(String key, Element value)	Remember an entry. This throws a runtime error if the key is already defined.
Element lookup(String id)	Get the value of a variable identified by the *key*. Returns null if this is refers to an in-built value. If undefined, this throws a runtime error.

3.4 Applicator package

3.4.1 Applicator class

The utility class to apply in-built operators and functions to parameters. All the function applications will will be handled by the **apply** methods. Following are the apply functions for unary functions and operators such as *neg*, *not* and binary operators such as *+*, *eq*. Also there are public helper functions to determine if an element is a binary operator or unary operator.

Element apply(Element operation, Element operand)
Element apply(Element operation, Element operand1, Element operand2)
boolean isBinaryOperation(Element op)
boolean isUnaryOperation(Element op)

Apart from that, there are other functions to perform the operator functionalities.

numericalOperator(operand1, operand2, operation)	Numeric operators helper
binaryBooleanOperator(operand1, operand2, operation)	Boolean operators helper
covertToString(element)	Element into string
booleanCondition(condition)	Boolean primitive into element
substringOperation(operand, operation)	String *stem, stern* helper
add(operand1, operand2)	Addition operator
subtract(operand1, operand2)	Subtraction operator
multiply(operand1, operand2)	Multiplication operator
power(operand1, operand2)	Power operator
divide(operand1, operand2)	Division operator
print(operand)	Print function
isString(operand)	Check if string
isInteger(operand)	Check if integer
isTruthValue(operand)	Check if true/false
isTuple(operand)	Check if tuple
order(operand)	Length of tuple
stern(operand)	String without first character
stem(operand)	First character
conc(operand)	String concatenation (partial)
conc(operator, operand2)	String concatenation
iToS(operand)	Integer to string
neg(operand)	Negative value
or(operand)	Or operator
and(operand)	And operator
eq(operand)	Equal operator
ne(operand)	Not equal operator
gr(operand)	Greater than operator
ls(operand)	Less than operator
ge(operand)	Greater than or equal operator
le(operand)	Less than or equal operator
aug(operand1, operand2)	Append element to tuple
extract(operation, operand)	Extract element from tuple

3.4.2 Operation Interfaces definitions

BinaryBooleanOperator	Helper lambda closure of $(boolean, boolean) \rightarrow boolean$
NumericalOperator	Helper lambda closure of $(int, int) \rightarrow int$
SubstringOperation	Helper lambda closure of $(String) \rightarrow String$

3.5 Machine class

Machine class evaluates the control stack. Following are the main attributes.

Stack<Value> control	Control
Stack<Element> stack	Stack
Applicator applicator	Application controller
ArrayList<Environment> environments	Environment array
ArrayList<Stack<Value> > controlStructures	All control structures
void evaluate()	Evaluate function to evaluate the stack control.

evaluate() will evaluate the control stack and apply CSE riles until control is empty.

3.6 Other classes

CseException is a runtime exception which is thrown when evaluating CSE machine fails due to some reason.

Stack Stack implementation to store Elements.