Report

CS 3512 - Programming Languages

Programming Project 01

Prepared by
190088H-Harshani Bandara
Date 2022/11/01

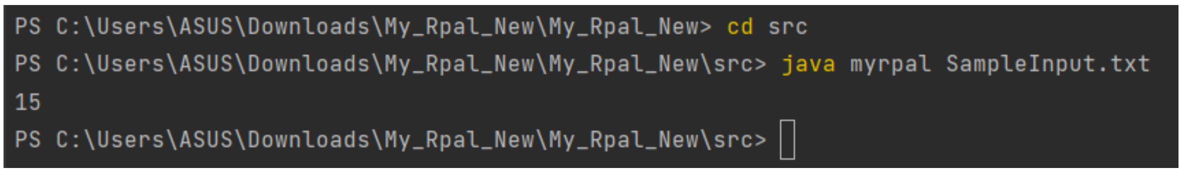
Table of content

Problem

The objective of this project is to implement an algorithm to convert Abstract Syntax Tree (AST) in to Standardize Tree (ST) and implement CSE machine. Your program should be able to read an input file which contains the AST of a program. The format of the AST is the same format that you can see by running “rpal.exe” with -ast flag. The output of your program should match the production of “rpal.exe“ for the relevant program. (Relevant program is the program that produces the AST which your program used as the input).

Run Programme

```
cd src
Java myrpal SampleInput.txt
```



```
PS C:\Users\ASUS\Downloads\My_Rpal_New\My_Rpal_New> cd src
PS C:\Users\ASUS\Downloads\My_Rpal_New\My_Rpal_New\src> java myrpal SampleInput.txt
15
PS C:\Users\ASUS\Downloads\My_Rpal_New\My_Rpal_New\src>
```

Solution

First thinks about the problem and do the following steps to get the final solution.

- Get the input file with the AST and read the file
- Create the ST to the given AST with a standardization tree of nodes
- Create the stack with all elements
- generate the CSE machine for created ST
- Finally evaluate the expression according to the rules of CSE

• Get the input file with the AST and read the file

In this project, first give the AST tree in the text file name as a command line argument. In this, we are given the AST as given below in the sampleInput.txt.

```
let
.function_form
..<ID:Sum>
..<ID:A>
..where
...gamma
....<ID:Psum>
....tau
.....<ID:A>
.....gamma
```

```
......<ID:Order>
......<ID:A>
...rec
....function_form
.....<ID:Psum>
.....,
......<ID:T>
......<ID:N>
.....->
......eq
.......<ID:N>
.......<INT:0>
......<INT:0>
......+
.......gamma
........<ID:Psum>
........tau
.........<ID:T>
.........-
..........<ID:N>
..........<INT:1>
.......gamma
........<ID:T>
........<ID:N>
.gamma
..<ID:Print>
..gamma
...<ID:Sum>
...tau
....<INT:1>
....<INT:2>
....<INT:3>
....<INT:4>
....<INT:5>
```

If there is no file name that throws the exception.
If there exists a file given in the argument, First Read the above text file line by line with trim the spaces and generate the list with each line.
There is a function to identify all the escape sequences that are given following.

- BS
- FF
- NL
- CR,
- TAB
- Double quote
- Single quote

In each line, we need to calculate the number of dots in the beginning to identify the depth of the root.
According to the lines that are read, create a tree using the CreateTree class.

- Create the ST to the given AST with a standardization tree of nodes

When we create the tree of ST or AST, should implement the tree node. For that task implement a node class with the parent, children, value and label.
There are functions to copy node, add child, get child, check whether are there any child, clear and add a child to the node.

```
public class Node {
    /**
     * Represents a node in the ast and st.
     *main two types they are child and parent
     * Label represents the type of Node.
     * EleValue represents the value of node
     */
    private final ArrayList<Node> children;
    private Node parent;
    private String label;
    private String value;
    /**
    *node with one argument called label
     */
    public Node(String label) {
        this.label = label;
        this.children = new ArrayList<>();
    }

    /**
     * node with two argument both lable and value
     */
    public Node(String label, String value) {
        this.label = label;
        this.value = value;
        this.children = new ArrayList<>();
    }
    Node copy() {
        Node copied = new Node(label, value);
        for (int i = 0; i < getNumChild(); i++) {
            copied.addChild(getChild(i).copy());
        }
        return copied;
    }
    //get parent node
    Node getParent() {return parent;    }
    public String getLabel() {
        return label;
    }
    public String getValue() {
        return value;
    }
    public int getNumChild() {
        return children.size();
    }
    boolean hasChildren(int n) {
        return children.size() == n;
    }
    public boolean isLabel(String label) {
        return getLabel().equals(label);
    }
    public Node getChild(int i) {
        return children.get(i);
    }
    public void forEachChild(Consumer<? super Node> action) {
        children.forEach(action);
    }
    void setLabel(String label) {this.label = label;this.value = null;    }
    void clearChildren() {children.forEach(child -> child.parent = null);children.clear();}
    void addChild(Node child) {children.add(child);child.parent = this;}
}
```

The creatine AST from the text file, is converted to the ST using ASTtoST class. First implement the all converters for AST node labels that are given below.

Let

```
     Standardize the LET node

      let               gamma
     /  \              /   \
    =    P        =>      lambda    E
    /\                /  \
 X  E              X   P
```

Where

```
Standardize the WHERE node

 where          gamma
   / \               /    \
 P  =    =>   lambda  E
  /\                 /  \
 X E               X    P
```

Function_form

```
Standardize the FCN_FORM node

    FCN_FORM              =
     /  |      \                /    \
    p  V+   E      =>    P   +lambda
```

```
                          /  \
                        V     .E
```

And

```
Standardize the AND node
      AND          =
          |           /  \
        =++  =>   ,  TAU
        / \           |      |
      X   E      X++  E++
```

Rec

```
  Standardize the REC node

      REC             =
       |             /  \
      =       =>   X  gamma
     / \              /     \
   X  E             yStar  lambda
                              /  \
                            X    E
```

Lambda

```
Standardize the Multi-func param (lambda) node

   lambda               ++lambda
     /   \                    /  \
  V++   E    =>          ++V   .E
```

Within

```
Standardize the WITHIN node

       WITHIN           =
         /    \            /  \
       =     =      =>    X2 gamma
       / \   / \                  /   \
     X1 E1 X2 E2      lambda  E1
                   /  \
                X1  E2
```

At

```
Standardize the @ node
       @              gamma
      / |  \           /        \
    E1 N E2  =>  gamma  E2
            /  \
           N   E1
```

- Operation handling

  OperationHandler class consist of three interface to identify boolean, arithmetic and array type operations. All the types of implementations of tree are implemented in this class.

- Create the stack with all elements

To store the elements create the following stack class

```
public class Stack<T extends EleValueOrTuple> implements Iterable<T> {
    protected final java.util.Stack<T> stack;

    Stack() {
        stack = new java.util.Stack<>();
    }
    void push(T element) {
        stack.push(element);
    }
    T pop() {
        return stack.pop();
    }
    boolean isEmpty() {
        return stack.isEmpty();
    }
    int size() {
        return stack.size();
    }
    @Override
    public String toString() {
        return stack.toString();
    }

    @Override
    public Iterator<T> iterator() {
        return stack.iterator();
    }
}
```

- Implement operations

In the operation handler class,

- generate the CSE machine for created ST

The created ST send to the CSE machine and generates CSE according to the given tree using the 13 rules.
Using the evaluateTree function, evaluate the CSE and give the output of the code.

```
import java.util.ArrayList;

/**
 * CSE machine to evaluate the traversed tree
 */
public class CSEMachine {
    private final Stack<EleValue> eleValues;
    private final Stack<EleValueOrTuple> eleValueOrTuples;
    private final OperationHandler operationHandler;
    private final ArrayList<Environment> environments;
    private final ArrayList<Stack<EleValue>> CS;

    public CSEMachine(ArrayList<Stack<EleValue>> controlStructures) {
        this.CS = controlStructures;
        this.eleValueOrTuples = new Stack<>();
        this.operationHandler = new OperationHandler();

        eleValues = new Stack<>();
        eleValues.push(new EleValue("environment", "0"));
        eleValues.push(new EleValue("delta", "0"));
        eleValueOrTuples.push(new EleValue("environment", "0"));
        environments = new ArrayList<>();
        environments.add(new Environment());
    }

    @Override
    public String toString() {
        return eleValues + "\n" + eleValueOrTuples + "\n" + currentEnvironment() + "\n";
    }

    /**
     * Get current environment index by peeking the stack and fining the closest
environment element.

     */
    private int currentEnvironmentIndex() {
        int closestEnvironment = 0;
        for (EleValueOrTuple element : eleValues) {
            if (element instanceof EleValue && element.isLabel("environment")) {
                String closestEnvironmentStr = ((EleValue) element).getValue();
                closestEnvironment = Integer.parseInt(closestEnvironmentStr);
            }
        }
        return closestEnvironment;
    }
```

```
/**
 * Get current environment
 */
private Environment currentEnvironment() {
    return environments.get(currentEnvironmentIndex());
}

/**
 * Start processing the control stack to evaluate result.
 */
public void evaluateTree() {
    while (!eleValues.isEmpty()) {
        EleValue currentElement = eleValues.pop();

        if (currentElement.isLabel("gamma")) {
            EleValueOrTuple firstElem = eleValueOrTuples.pop();
            EleValueOrTuple secondElem = eleValueOrTuples.pop();
            if (firstElem.isLabel("yStar")) {
                Rule12(secondElem);
            } else if (firstElem.isLabel("eta")) {
                eleValueOrTuples.push(secondElem);
                Rule13(currentElement, firstElem);
            } else if (firstElem.isLabel("lambda")) {
                EleValue firstValue = (EleValue) firstElem;
                if (firstValue.getValue().contains(",")) {
                    Rule11(firstElem, secondElem);
                } else {
                    Rule4(firstElem, secondElem);
                }
            } else if (firstElem.isLabel("tau")) {
                Rule10(firstElem, secondElem);
            } else {
                Rule3(firstElem, secondElem);
            }
        } else if (currentElement.isLabel("delta")) {
            int controlIndex = Integer.parseInt(currentElement.getValue());
            extractDelta(controlIndex);
        } else if (currentElement.isLabel("id")) {
            Rule1(currentElement);
        } else if (currentElement.isLabel("lambda")) {
            Rule2(currentElement);
        } else if (currentElement.isLabel("environment")) {
            Rule5(currentElement);
        } else if (currentElement.isLabel("beta")) {
            Rule8();
        } else if (currentElement.isLabel("tau")) {
            Rule9(currentElement);
        } else if (!Rule6_7(currentElement)) {
            eleValueOrTuples.push(currentElement);
        }
    }
}
```

```
/**
 * Extract elements from the control structure specified.
 */
private void extractDelta(int controlIndex) {
    Stack<EleValue> control = CS.get(controlIndex);
    for (EleValue controlElem : control) {
        this.eleValues.push(controlElem);
    }
}

/**
 * rule 1
 */
private void Rule1(EleValue name) {
    String id = name.getValue();
    EleValueOrTuple value = currentEnvironment().lookup(id);
    if (value == null) {
        value = new EleValue(id);
    }
    eleValueOrTuples.push(value);
}

/**
 * rule 2
 */
private void Rule2(EleValue lambda) {
    String[] kAndX = lambda.getValue().split(" ");
    String c = Integer.toString(currentEnvironmentIndex());
    String[] newValues = {kAndX[0], kAndX[1], c};
    EleValueOrTuple newLambda = new EleValue("lambda", String.join(" ", newValues));
    eleValueOrTuples.push(newLambda);
}

/**
 * rule13
 */
private void Rule3(EleValueOrTuple rator, EleValueOrTuple rand) {
    EleValueOrTuple result = operationHandler.apply(rator, rand);
    eleValueOrTuples.push(result);
}

/**
 * rule4
 */
private void Rule4(EleValueOrTuple lambda, EleValueOrTuple rand) {
    if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
        String[] kAndXAndC = ((EleValue) lambda).getValue().split(" ");
        String k = kAndXAndC[0];
        String x = kAndXAndC[1];
        String c = kAndXAndC[2];
        Environment envC = environments.get(Integer.parseInt(c));

        Environment newEnvironment = new Environment(envC, x, rand);
        String newEnvIndex = Integer.toString(environments.size());
```

```
            environments.add(newEnvironment);
            eleValues.push(new EleValue("environment", newEnvIndex));
            eleValues.push(new EleValue("delta", k));
            eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
            return;
        }
        throw new ExceptionHandlerOfCSE("Expected lambda element but found: " +
lambda);
    }


    /**
     * rule 5
     */
    private void Rule5(EleValue env) {
        EleValueOrTuple value = eleValueOrTuples.pop();
        EleValueOrTuple envS = eleValueOrTuples.pop();
        if (envS instanceof EleValue && envS.isLabel("environment")) {
            if (env.equals(envS)) {
                eleValueOrTuples.push(value);
                return;
            }
            throw new ExceptionHandlerOfCSE(String.format("Environment element
mismatch: %s and %s", env, envS));
        }
        throw new ExceptionHandlerOfCSE("Expected environment element but found: " +
envS);
    }

    /**
     * rule7
     */
    private boolean Rule6_7(EleValue element) {
        if (operationHandler.checkMathematicalOperation(element)) {
            EleValueOrTuple rator = eleValueOrTuples.pop();
            EleValueOrTuple rand = eleValueOrTuples.pop();
            EleValueOrTuple result = operationHandler.applyOperations(element, rator, rand);
            eleValueOrTuples.push(result);
        } else if (operationHandler.checkArrayOperation(element)) {
            EleValueOrTuple rand = eleValueOrTuples.pop();
            EleValueOrTuple result = operationHandler.apply(element, rand);
            eleValueOrTuples.push(result);
        } else {
            return false;
        }
        return true;
    }

    /**
     * rule8
     */
    private void Rule8() {
        EleValue deltaElse = eleValues.pop();
        EleValue deltaThen = eleValues.pop();
        EleValueOrTuple condition = eleValueOrTuples.pop();

        if (deltaElse.isLabel("delta") && deltaThen.isLabel("delta")) {
            if (condition.isLabel("true")) {
                eleValues.push(deltaThen);
                return;
            } else if (condition.isLabel("false")) {
                eleValues.push(deltaElse);
                return;
            }
            throw new RuntimeException("If condition must evaluate to a truth value.");
        }
        throw new ExceptionHandlerOfCSE("Expected delta elements.");
    }

    /**
     * rule 9
     */
    private void Rule9(EleValue tau) {
        int elements = Integer.parseInt(tau.getValue());
        EleValueOrTuple[] tupleElements = new EleValueOrTuple[elements];
        for (int i = 0; i < elements; i++) {
            tupleElements[i] = eleValueOrTuples.pop();
        }
        EleValueOrTuple tuple = new EleTuple(tupleElements);
        eleValueOrTuples.push(tuple);
    }

    /**
     * rule10
     */
    private void Rule10(EleValueOrTuple tuple, EleValueOrTuple index) {
        if (tuple instanceof EleTuple) {
            if (index instanceof EleValue && index.isLabel("int")) {
                int ind = Integer.parseInt(((EleValue) index).getValue());
                EleValueOrTuple value = ((EleTuple) tuple).getValue()[ind];
                eleValueOrTuples.push(value);
                return;
            }
            throw new ExceptionHandlerOfCSE("Expected integer index but found: " + index);
        }
        throw new ExceptionHandlerOfCSE("Expected tuple but found: " + tuple);
    }

    /**
     * rule 11
     */
    private void Rule11(EleValueOrTuple lambda, EleValueOrTuple rand) {
        if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
            if (rand instanceof EleTuple) {
                String[] kAndVAndC = ((EleValue) lambda).getValue().split(" ");
                String k = kAndVAndC[0];
                String[] v = kAndVAndC[1].split(",");
                String c = kAndVAndC[2];
                Environment envC = environments.get(Integer.parseInt(c));

                Environment newEnvironment = new Environment(envC);
                for (int i = 0; i < v.length; i++) {
                   newEnvironment.remember(v[i], ((EleTuple) rand).getValue()[i]);
                }
                String newEnvIndex = Integer.toString(environments.size());
                environments.add(newEnvironment);
                eleValues.push(new EleValue("environment", newEnvIndex));
                eleValues.push(new EleValue("delta", k));
                eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
                return;
             }
             throw new ExceptionHandlerOfCSE("Expected tuple but found: " + rand);
          }
          throw new ExceptionHandlerOfCSE("Expected lambda element but found: " +
    lambda);
       }

       /**
        * rule12
        */
       private void Rule12(EleValueOrTuple lambda) {
          if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
             String iAndVAndC = ((EleValue) lambda).getValue();
             EleValueOrTuple etaElement = new EleValue("eta", iAndVAndC);
             eleValueOrTuples.push(etaElement);
             return;
          }
          throw new ExceptionHandlerOfCSE("Expected lambda element but found: " +
    lambda);
       }

       /**
        * rule 13
        */
       private void Rule13(EleValue gamma, EleValueOrTuple eta) {
          if (eta instanceof EleValue && eta.isLabel("eta")) {
             String iAndVAndC = ((EleValue) eta).getValue();
             EleValue lambda = new EleValue("lambda", iAndVAndC);
             EleValue newGamma = new EleValue("gamma");

             eleValueOrTuples.push(eta);
             eleValueOrTuples.push(lambda);

             eleValues.push(gamma);
             eleValues.push(newGamma);
             return;
          }
          throw new ExceptionHandlerOfCSE("Expected eta element but found: " + eta);
       }
    }
```

Calling the executeTree we can get the final answer of the given AST.

- Exception handling

the First check is the argument given in the command line, if there are invalid things throws to the exception.
All the exceptions that are occurring on creating and evaluating the CSE machine are handled by exceptionHandlerOfCSE.
All the exceptions that are occurring on creating AST and ST are handled by exceptionHandlerOfAST.

References

[1] Wikipedia contributors, “Abstract syntax tree,” *Wikipedia, The Free Encyclopedia*, 10-Aug-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Abstract_syntax_tree&oldid=1103626323.

[2] “Abstract syntax tree (AST) in java,” *GeeksforGeeks*, 11-Aug-2021. [Online]. Available: https://www.geeksforgeeks.org/abstract-syntax-tree-ast-in-java/. [Accessed: 08-Dec-2022].

[3] “What is Syntax Tree?,” *Tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/what-is-syntax-tree. [Accessed: 08-Dec-2022].