Project Four

Collin Drake

Collin.Drake1@Marist.edu

May 7, 2024

Lab 9

Crafting a Compiler Exercises:

Exercise: 5.5

Transform the following grammar into LL(1) form using the techniques presented in Section 5.5:

1 DeclList -> DeclList ; Decl
2 -> Decl
3 Decl -> IdList : Type
4 IdList -> IdList , id
5 -> id
6 Type -> ScalarType
7 -> array (ScalarTypeList) of Type
8 ScalarType -> id
9 -> Bound . . Bound
10 Bound -> Sign intconstant
11 -> id
12 Sign -> +
13 -> -
14 -> λ
15 ScalarTypelist -> ScalarTypeList , ScalarType
16 -> ScalarType

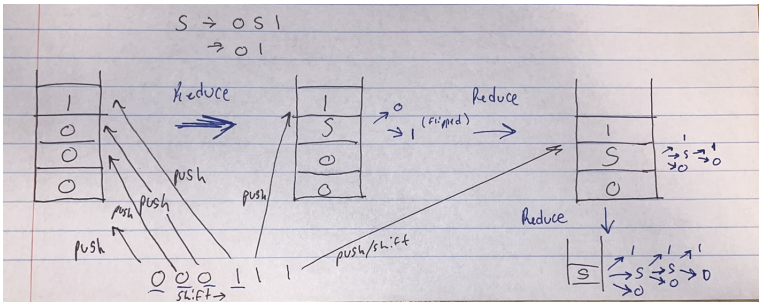
1. DeclList => Decl DeclList *
2. DeclList * => DeclList
3. => ε
4. Decl => IdList Type
5. IdList => id IdList *
6. IdList * => IdList
7. => ε
8. Type => ScalarType
9. => array (ScalarTypeList) of Type
10. ScalarType => Bound *
11. Bound => Sign intConstant
12. => Bound *
13. Bound * => id Bound **
14. Bound ** => id
15. => ε
16. Sign => +
17. => -
18. => λ
19. ScalarTypeList => ScalarType ScalarTypeList *
20. ScalarTypeList * => ScalarTypeList
21. => ε

Dragon Exercises:

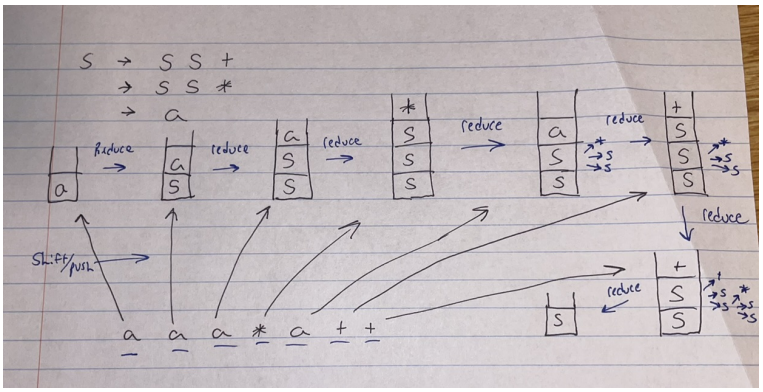
Exercise: 4.5.3

Give bottom-up parses for the following input strings and grammars:

a) The input 000111 according to the grammar of Exercise 4.5.1.



b) The input aaa * a + + according to the grammar of Exercise 4.5.2.



Project 4

Code Generation

In the Compilation process, Code Generation is the fourth and final step. In this step, we accept the Abstract Syntax Tree and Symbol Table produced by the Semantic Analyzer and make the 6502 Op Codes needed to execute our programs in a specific Operating System.

To test my Code Generator I used the test cases below:

Test Case 1

```
    {
      int a
      a = 1
      {
        int a
        a = 2
        print(a)
      }
      string b
      b = "drake"
      print("collin")
      print(a)
      print(b)
    }$
```

Test Case 2

```
    {
        int a
        boolean b
        a = 2+3+4
        b = (2 == 2)
        print(a)
        print(b)
        print(2+2)
        print((2 == 3))
    }$
```

Test Case 3

```
    {
        int a
        a = 5
        if(true == false){
            a = 7
            print(a)
        }
    }$
```

Test Case 4

```
{
    int a
    a = 5
    if(2==2){
        a = 7
        print(a)
        if(2==3){
            print(a)
        }
        string b
        b = "you win"
        print(b)
    }
    print(true)
}$
```

Test Case 5

```
{
    int x
    x = 9
    print(x)
    while (x == 9){
        x = 1+x
        print(x)
    }
}$
```

Appendix

BranchTableVariable.java

```
/*
    Branch table variable file
    Creates objects for the code generation branch table
*/

//The branch table variable class!
public class branchTableVariable {
    String temp;    //jump to variable
    int distance;    //how many bytes to jump

    //Static table variable constructor -- initializes all variables
    public branchTableVariable(String temp, int distance){
        this.temp = temp;
        this.distance = distance;
    }
}
```

StaticTableVariable.java

```
/*
    Static table variable file
    Creates objects for the code generation variable table
*/

//The static table variable object class!
public class staticTableVariable {
    String tempAddress;    //store the accumulator in a temp location/address TX
    String var; //what variable is this?
    int scope;   //keeps track of the scope the variable is located in
    int offset; //number of offset following code section of the array
    (location of variable stored)

    //Static table variable constructor -- initializes all variables
    public staticTableVariable(String temp, String var, int scope, int offset){
        this.tempAddress = temp;
        this.var = var;
        this.scope = scope;
        this.offset = offset;
    }
}
```

References

Links

Below are the resources I have used to create simple, readable, and beautiful code.

- 6502 op codes: labouseur.com
- output checking: Arnell Compiler