6CCS3CFL – Strand 1, Coursework 04

Horia Tudor Pavel Simon

Question 1

A:

Note: *Both assembled .j files for fib.while and factorial.while are submitted along with the code and this report. Also Jasmin needs to reside under jvm/jasmin-2.4/jasmin jar in order for the files to be assembled.*

If I am using the *compile_run* function, input is not recognized by either cmd/PowerShell or Windows Subsystem for Linux. So I have used *compile_all* function and then run the programs manuall by calling java <file-name>.<file-name>

Files can be compiled by running:

```
scala code_cw4.scala <file-name>.while
```

running this command will compile the code in this order:

1. lex the input ->
2. generate serialized file with tokens ->
3. deserialize tokens from the token file ->
4. Parse the token list ->
5. Compile the program to .j file ->
6. Assemble it to .class file

And then run by:

```
java <file-name>.<file-name>
```

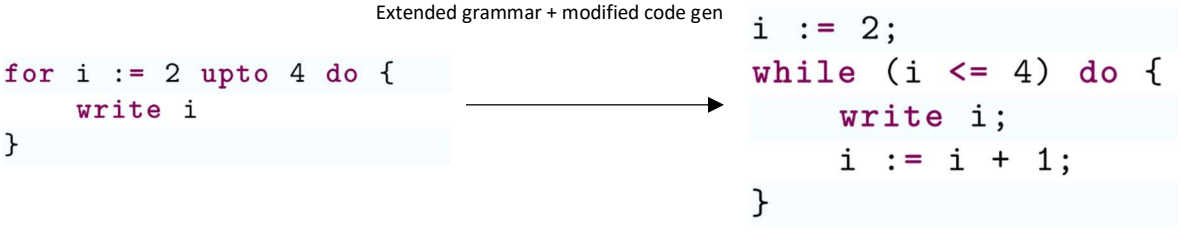
I implemented my own write(String)V before I saw the code that you gave us in the cw4.pdf. I kept mine although they are the same.

Question 2

A:

I chose to add the for-loop to the while language by extending the grammar to include a **for** construct. Then, in the code generation part, I optimized the for-loop to a while loop with initialization before execution; just like the translation you gave us in the coursework description.

Extended grammar + modified code gen



Question 3

A:

The assembler instructions for the program below are in the "test_for.j" file.

My decision is determined by the language features here and I wish to leave it like this for now. The lack of scope in our language allows for constructs like:

```
for i := 1 upto 10 do {
    for i := 1 upto 10 do {
      write i
    }
}
```

to work without reporting any errors. The WHILE language compiler will enter the first loop and then the second loop. In the second loop, it will iterate from 1 to 10 and print the numbers. After this executes, it goes back to the start of the first loop (by the goto instruction) and checks if i <= 10, but i would already be 11. So, it will exit the outer loop.

The output of this program is:

```
1
2
3
4
5
6
7
8
9
10
```

We can nest as many loops as we want, the output will always be the innermost loop execution:

```
for i := 1 upto 10 do {
      for i := 1 upto 10 do {
            for i := 1 upto 10 do {
                  for i := 1 upto 10 do {
                        for i := 1 upto 10 do {
                              write i
                        }
                  }
            }
      }
}
```

In this case the output is the same as above.