Report – Richard Arthurs - 2465714A

Stage 1 (syntactic analysis):

Extension A (for):

For this first stage I was tasked with implementing new grammar for the 'for' procedure which I later would be integrating fully. I added a simple FOR statement to the com (commands) section of the grammar. The command would assign a value to ID then iterate until it reaches the e2 value each time doing the sequential command.

Extension B (switch):

For this first stage I was tasked with implementing new grammar for the 'switch' procedure which I later would be integrating fully. I added a simple SWITCH statement to the com (commands) section of the grammar. This expression would check it the case was true or not then if true pass the corresponding command and if not revert to the default.

Stage 2 (contextual analysis):

Extension A (for):

Now the grammar had been extended to allow the 'for' command to be integrated it was time to create the missing methods in FunCheckerVisitor class. The visitFor would initially validate the type of the control variable. Visit each e1 and e2 expression and get create a type of variable for each. Then visit the sequential command and finally check that the defined type1 and type2 variables were integers (INT).

Extension B (switch):

Now the grammar had been extended to allow the 'switch' command to be integrated it was time to create the missing methods in FunCheckerVisitor class. Once again, I visited the ctx to retrieve its type and validate whether it was a valid type i.e. not BOOL or INT and if not return a report error. I then initialised lists for the guards and an empty list for the guards which have been checked. I then would loop through this list of guards, visiting each guard and checking the type and for duplicates. If the guard was of type BOOL I would just check the type and then add this to the list of checked guards and return to the start of the loop. But if the guard was an INT I would have to check the digit order was ascending and that said digits were themselves valid. If the

digits were not in the correct format, I would report an error stating the issues. Here once more checking for duplicate/overlapping guards. If the guard passed my validations, it would also be added to the checked guard list. Then at the end of the loop I would move onto the next expression visiting the ctx. Finally, visiting the default expression and returning null.

Stage 3 (code generation):

Extension A (for):

Now the contextual methods were implemented I had to design the code templates for my 'for' command. My visitFor method would initially visit the ctx.e1 and retrieve the identifier for the ctx. Then assign this id to a local address, then assign this address to varaddr for later. Then I would use the SVM implementation to manipulate my obj such that it would be checked, incremented then stored then finally jump back to the top. Finally exiting from the procedure with an exit address and jump address for the obj.

Extension B (switch):

Now the contextual methods were implemented I had to design the code templates for my 'for' command. My visitSwitch method initialised three lists one for the guards another for the sequential commands and finally one for the addresses. This method would visit each guard get the address and expression then JUMP to the next guard visit the sequential command get the address and JUMP back to the top, get the next guards address and send this to the obj with the last guards' address. This loop would continue until all guards were processed. Then once this action was completed get an exit address and iterate through the address list assigning this exit address to each one.

For testing purposes some adaptations were required in the SVM, these were as follows. I added the codes for LOADR COMPG and COPY with their corresponding mnemonics. Each one has been implemented a case which completes the desired actions.