CM4106 - Languages and Compilers

Coursework 2

Semantic Analysis and Code Generation

Statement of Compliance

Semantic Analyser

Fully working semantic analyser which checks for semantic errors and creates a symbol table

Code Generator

Fully working code generator which takes the annotated syntax tree and creates instructions for the abstract machine

Semantic Analyser

Symbol Table

The provided IdentificationTable.cs file has been expanded to fully implement the symbol table. This is simply a stack of dictionaries with the head being the current scope being operated on and the bottom of the stack being the Globlal Scope. Standard stack functions to peek, pop and push allow us to get the next node or to collapse a “branch”. As we move through the AST we can pop and push scopes off of and onto the table and we use the peep function to retrieve items in the stack.

Declarations and Variable Use

When a new variable is declared the symbol table is scanned for duplicate entries, if one is found then the duplicate flag is set to true, after which it is added into the current scope.

When it’s being used the entire scope is checked for a match to be returned otherwise we return null.

Type Checking

This is performed as the Semenatic Analyser moves through the tree. The symbol table contains our type definitions for each variable and any time one is used, we must consult to find the type before we do any type checking.

Program Scope

The program scope is inferred from the symbol table depth. Each dictionary is a higher level level of the scope as you move further up. When searching the scope for a variable we look for the nearest first. The program scope is enforced here by popping the current head off the stack, then in future, variable names will not find the referrer in the stack because the head with it in has been removed.

Code Generation

The code generator receives an Annotated Syntax Tree and employs the same technique of visiting each element in the AST as we used previously. This will then generate code for the TAM using the code written to handle the various Command and Declarations.

For example, the encoder file is used to create the language code template then the emitter creates the necessary components to be passed back to the instruction set.

In a general case of visiting a command or declaration a number of process' must be performed. First the memory needed to store the command is calculated and assigned, any and all terminals needed must be collected and pushed to the top of the stack. The sum of the current expression and the frame is then summed to update the amount of memory allocated. The expression and the frame is then popped off the stack and added to the current register.

In the case of runtime instructions such as JUMP, JUMPIF, LOAD etc. then the appropriate function is performed, in the case of a JUMP, a batchpatch is created to the memory address incase, in an if or while, an expression evaluates to falls, execution can jump back to that memory address and continue from there.

Tests

This program should compile successfully and when ran in the abstract machine interpreter should ask the user for a number below 11 and then print that number out until it reaches 1 and exit. The compiler compiled this program successfully.

```
 let
      const MAX ~ 10;
      var numberUnder11: Integer
in
      begin
            numberUnder11 := 6;
            if numberUnder11>0 then
                  if numberUnder11 < MAX then
                        while numberUnder11 > 0
                        do
                              begin
                                    putint(numberUnder11);
                                    put('.'); put(' ');
                                    numberUnder11 := numberUnder11-1
                              end
                  else
                        begin
put('e');put('r');put('r');put('o');put('r');put(' ');put('1') end
                        ! Number was too big
            else
                  begin
put('e');put('r');put('r');put('o');put('r');put(' ');put('2') end
                  ! Number was not positive
      end
```