Лабораторная работа № 2.1. Синтаксические деревья

26 сентября 2024 г.

Кабанов Андрей Юрьевич, ИУ9-62Б

Цель работы

Целью данной работы является изучение представления синтаксических деревьев в памяти компилятора и приобретение навыков преобразования синтаксических деревьев.

Индивидуальный вариант

Для каждого оператора if подсчитать количество переходов по положительной ветке относительно общего количества вызова конкретного оператора if (во время выполнения программы).

Реализация

Демонстрационная программа:

```
package main

import "fmt"
func main() {
    var i int = 0
    if i++; (1 == 1) {
        i++
    } else if (1 == 0){
        i--
    } else {
        fmt.Printf("%d", i)
    }
    if true {
```

```
    }
}
```

Программа, осуществляющая преобразование синтаксического дерева:

```
package main

import (
    "fmt"
    "go/ast"
    "go/format"
    "go/parser"
    //"go/printer"
    "go/token"
    "os"
)

func insertIntVar(file *ast.File, name string, value int) {
    var before, after []ast.Decl

    if len(file.Decls) > 0 {
        hasImport := false
        if genDecl, ok := file.Decls[0].(*ast.GenDecl); ok {
            hasImport = genDecl.Tok == token.IMPORT
        }

        if hasImport {
            before, after = []ast.Decl{file.Decls[0]}, file.Decls[1:]
        } else {
            after = file.Decls
        }
    }

    file.Decls = append(before,
        &ast.GenDecl{
            Tok: token.VAR,
            Specs: []ast.Spec{
                &ast.ValueSpec{
                    Names: []*ast.Ident{ast.NewIdent(name)},
                    Type:  ast.NewIdent("int"),
                    Values: []ast.Expr{
                        &ast.BasicLit{
                            Kind:  token.INT,
                            Value: fmt.Sprintf("%d", value),
                        },
                    },
                },
```

```
            },
        },
    )
    file.Decls = append(file.Decls, after...)
}

func insertCounter(file *ast.File) int{
    globalIfCounter := 0
    ast.Inspect(file, func(node ast.Node) bool {
        ifCounter := 0
        localIfCounter := 0
        if blockStmt, ok := node.(*ast.BlockStmt); ok {
            loop:

            localIfCounter = 0
            for  i := 0; i< len( blockStmt.List); i++ {
                if ifStmt, ok := blockStmt.List[i].(*ast.IfStmt); ok{
                    localIfCounter++
                    if localIfCounter <= ifCounter {
                        continue
                    }
                    ifCounter++
                    first := blockStmt.List[0:i]
                    follow := blockStmt.List[i+1:]
                    newBlockStmt := []ast.Stmt{}
                    newBlockStmt = append(newBlockStmt, first...)
                    newBlockStmt = append(newBlockStmt,
                        &ast.IncDecStmt{
                            X:
                            ast.NewIdent(fmt.Sprintf("ifCounter%d",
                            globalIfCounter+ifCounter-1)),
                            Tok: token.INC,
                        },
                    )

                    ifStmt.Body.List = append(
                        []ast.Stmt{
                            &ast.IncDecStmt{
                                X:
                                ast.NewIdent(fmt.Sprintf("positiveCounter%d",
                                globalIfCounter+ifCounter-1)),
                                Tok: token.INC,
                            },
                        },
                        ifStmt.Body.List...,
                    )
```

```
newBlockStmt = append(newBlockStmt, ifStmt)
newBlockStmt = append(newBlockStmt, follow...)


newBlockStmt = append(newBlockStmt,
    &ast.ExprStmt{
        X: &ast.CallExpr{
            Fun: &ast.SelectorExpr{
                X: &ast.Ident{
                    Name: "fmt",
                    Obj: nil,
                },
                Sel: &ast.Ident{
                    Name: "Printf",
                    Obj: nil,
                },
            },
            Args: []ast.Expr{
                &ast.BasicLit{
                    Kind: token.STRING,
                    Value:
                    "\"if #%d all call is %d \\n
                     if #%d positive block
                     all call is %d\"",
                },
                &ast.BasicLit{
                    Kind: token.INT,
                    Value:
                    fmt.Sprintf("%d",
                    globalIfCounter+ifCounter-1),
                },
                &ast.Ident{
                    Name:
                    fmt.Sprintf("ifCounter%d",
                    globalIfCounter+ifCounter-1),
                },
                &ast.BasicLit{
                    Kind: token.INT,
                    Value:
                    fmt.Sprintf("%d",
                    globalIfCounter+ifCounter-1),
                },
                &ast.Ident{
                    Name:
                    fmt.Sprintf("positiveCounter%d",
                    globalIfCounter+ifCounter-1),
```

```
                                    },
                                },
                            },
                        },
                    )
                    blockStmt.List = newBlockStmt
                    goto loop
                }

            }
        }
        globalIfCounter += ifCounter
        return true
    })
    return globalIfCounter
}

func main() {
    if len(os.Args) != 2 {
        return
    }

    fset := token.NewFileSet()
    if file, err := parser.ParseFile(fset, os.Args[1],
    nil, parser.ParseComments); err == nil {

        ifCounter := insertCounter(file)
        for i := 0; i< ifCounter; i++ {
            insertIntVar(file,
            fmt.Sprintf("ifCounter%d", i), 0)
            insertIntVar(file,
            fmt.Sprintf("positiveCounter%d", i), 0)
        }
        if format.Node(os.Stdout, fset, file) != nil {
            fmt.Printf("Formatter error: %v\n", err)
        }
    } else {
        fmt.Printf("Errors in %s\n", os.Args[1])
    }
}
```

Тестирование

Результат трансформации демонстрационной программы:

```
package main

import "fmt"

var positiveCounter1 int = 0
var ifCounter1 int = 0
var positiveCounter0 int = 0
var ifCounter0 int = 0

func main() {
        var i int = 0
        ifCounter0++
        if i++; 1 == 1 {
                positiveCounter0++
                i++
        } else if 1 == 0 {
                i--
        } else {
                fmt.Printf("%d", i)
        }
        ifCounter1++
        if true {
                positiveCounter1++

        }
        fmt.Printf("if #%d all call is %d
        \n if #%d positive block all call is %d",
         0, ifCounter0, 0, positiveCounter0)
        fmt.Printf("if #%d all call is %d \n
        if #%d positive block all call is %d",
         1, ifCounter1, 1, positiveCounter1)
}
```

Вывод тестового примера на `stdout`:

```
if #0 all call is 1 if #0 positive block all call is 1
if #1 all call is 1 if #1 positive block all call is 1
```

Вывод

В ходе выполнения лабораторной работы были приобретены навыки работы с синтаксическими деревьями. В частности было изучено синтаксическое дерево для языка Golang.