Лабораторная работа № 2.1. Синтаксические деревья

6 марта 2024 г.

Егор Поршенко, ИУ9-61Б

Цель работы

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

Индивидуальный вариант

Любая неанонимная функция, возвращающая ровно одно значение типа int, должна выводить своё имя и возвращаемое значение.

Реализация

Демонстрационная программа:

```
package main

func foo(a int, b int) int {
    func() int {
        return 2
    }()
    return a + b
}
```

Программа, осуществляющая преобразование синтаксического дерева:

```
package main

import (
    "fmt"
    "go/ast"
    "go/format"
```

```
    "go/parser"
    "go/token"
    "os"
)

func changeIntFunctions(file *ast.File, fileset *token.FileSet) {
    ast.Inspect(file, func(node ast.Node) bool {
        funcDecl, ok := node.(*ast.FuncDecl)
        if !ok {
            return true
        }
        if len(funcDecl.Type.Results.List) == 1 { // return only value?
            returnType := funcDecl.Type.Results.List[0].Type

            if ident, ok := returnType.(*ast.Ident); ok && ident.Name == "int" { // return i
                printFuncName := &ast.ExprStmt{
                    X: &ast.CallExpr{
                        Fun: &ast.SelectorExpr{
                            X:   &ast.Ident{Name: "fmt"},
                            Sel: &ast.Ident{Name: "Println"},
                        },
                        Args: []ast.Expr{&ast.BasicLit{Kind: token.STRING, Value: "\"" + fun
"\""}},
                    },
                }

                funcDecl.Body.List = append([]ast.Stmt{printFuncName}, funcDecl.Body.List...

                // find return statement
                for _, stmt := range funcDecl.Body.List {
                    if retStmt, ok := stmt.(*ast.ReturnStmt); ok {
                        printReturn := &ast.ExprStmt{ // and print return statement
                            X: &ast.CallExpr{
                                Fun: &ast.SelectorExpr{
                                    X:   &ast.Ident{Name: "fmt"},
                                    Sel: &ast.Ident{Name: "Println"},
                                },
                                Args: retStmt.Results,
                            },
                        }

                        index := indexOf(funcDecl.Body.List, stmt) // insert before return
                        funcDecl.Body.List = append(funcDecl.Body.List[:index], append([]ast
, funcDecl.Body.List[index:]...)...)
                        break
                    }
```

```
                }
            }
        }

        return true
    })
}

func indexOf(list []ast.Stmt, stmt ast.Stmt) int {
    for i, s := range list {
        if s == stmt {
            return i
        }
    }
    return -1
}

func main() {
    if len(os.Args) != 2 {
        fmt.Printf("usage: astprint <filename.go>\n")
        return
    }

    fset := token.NewFileSet()

    if file, err := parser.ParseFile(
        fset,
        os.Args[1],
        nil,
        parser.ParseComments,
    ); err == nil {
        changeIntFunctions(file, fset)
        if format.Node(os.Stdout, fset, file) != nil {
            fmt.Printf("Formatter error: %v\n", err)
        }
        ast.Fprint(os.Stdout, fset, file, nil)
    } else {
        fmt.Printf("Error: %v", err)
    }
}
```

Тестирование

Результат трансформации демонстрационной программы:

```go
package main

func foo(a int, b int) int {
    fmt.Println("foo")
    func() int {
        return 2
    }()
    fmt.Println(a + b)
    return a + b
}
```

Вывод тестового примера на `stdout` (если необходимо)

```
package main

func foo(a int, b int) int {
    fmt.Println("foo")
    func() int {
        return 2
    }()
    fmt.Println(a + b)
    return a + b
}
     0  *ast.File {
     1  .  Doc: nil
     2  .  Package: sample.go:1:1
     3  .  Name: *ast.Ident {
     4  .  .  NamePos: sample.go:1:9
     5  .  .  Name: "main"
     6  .  .  Obj: nil
     7  .  }
     8  .  Decls: []ast.Decl (len = 1) {
     9  .  .  0: *ast.FuncDecl {
    10  .  .  .  Doc: nil
    11  .  .  .  Recv: nil
    12  .  .  .  Name: *ast.Ident {
    13  .  .  .  .  NamePos: sample.go:3:6
    14  .  .  .  .  Name: "foo"
    15  .  .  .  .  Obj: *ast.Object {
    16  .  .  .  .  .  Kind: func
    17  .  .  .  .  .  Name: "foo"
    18  .  .  .  .  .  Decl: *(obj @ 9)
    19  .  .  .  .  .  Data: nil
    20  .  .  .  .  .  Type: nil
    21  .  .  .  .  }
    22  .  .  .  }
    23  .  .  .  Type: *ast.FuncType {
```

```
24  .  .  .  .  Func: sample.go:3:1
25  .  .  .  .  TypeParams: nil
26  .  .  .  .  Params: *ast.FieldList {
27  .  .  .  .  .  Opening: sample.go:3:9
28  .  .  .  .  .  List: []*ast.Field (len = 2) {
29  .  .  .  .  .  .  0: *ast.Field {
30  .  .  .  .  .  .  .  Doc: nil
31  .  .  .  .  .  .  .  Names: []*ast.Ident (len = 1) {
32  .  .  .  .  .  .  .  .  0: *ast.Ident {
33  .  .  .  .  .  .  .  .  .  NamePos: sample.go:3:10
34  .  .  .  .  .  .  .  .  .  Name: "a"
35  .  .  .  .  .  .  .  .  .  Obj: *ast.Object {
36  .  .  .  .  .  .  .  .  .  .  Kind: var
37  .  .  .  .  .  .  .  .  .  .  Name: "a"
38  .  .  .  .  .  .  .  .  .  .  Decl: *(obj @ 29)
39  .  .  .  .  .  .  .  .  .  .  Data: nil
40  .  .  .  .  .  .  .  .  .  .  Type: nil
41  .  .  .  .  .  .  .  .  .  }
42  .  .  .  .  .  .  .  .  }
43  .  .  .  .  .  .  .  }
44  .  .  .  .  .  .  .  Type: *ast.Ident {
45  .  .  .  .  .  .  .  .  NamePos: sample.go:3:12
46  .  .  .  .  .  .  .  .  Name: "int"
47  .  .  .  .  .  .  .  .  Obj: nil
48  .  .  .  .  .  .  .  }
49  .  .  .  .  .  .  .  Tag: nil
50  .  .  .  .  .  .  .  Comment: nil
51  .  .  .  .  .  .  }
52  .  .  .  .  .  .  1: *ast.Field {
53  .  .  .  .  .  .  .  Doc: nil
54  .  .  .  .  .  .  .  Names: []*ast.Ident (len = 1) {
55  .  .  .  .  .  .  .  .  0: *ast.Ident {
56  .  .  .  .  .  .  .  .  .  NamePos: sample.go:3:17
57  .  .  .  .  .  .  .  .  .  Name: "b"
58  .  .  .  .  .  .  .  .  .  Obj: *ast.Object {
59  .  .  .  .  .  .  .  .  .  .  Kind: var
60  .  .  .  .  .  .  .  .  .  .  Name: "b"
61  .  .  .  .  .  .  .  .  .  .  Decl: *(obj @ 52)
62  .  .  .  .  .  .  .  .  .  .  Data: nil
63  .  .  .  .  .  .  .  .  .  .  Type: nil
64  .  .  .  .  .  .  .  .  .  }
65  .  .  .  .  .  .  .  .  }
66  .  .  .  .  .  .  .  }
67  .  .  .  .  .  .  .  Type: *ast.Ident {
68  .  .  .  .  .  .  .  .  NamePos: sample.go:3:19
69  .  .  .  .  .  .  .  .  Name: "int"
```

```
 70  .  .  .  .  .  .  .  .  .  Obj: nil
 71  .  .  .  .  .  .  .  .  }
 72  .  .  .  .  .  .  .  .  Tag: nil
 73  .  .  .  .  .  .  .  .  Comment: nil
 74  .  .  .  .  .  .  .  }
 75  .  .  .  .  .  .  }
 76  .  .  .  .  .  .  Closing: sample.go:3:22
 77  .  .  .  .  }
 78  .  .  .  .  Results: *ast.FieldList {
 79  .  .  .  .  .  Opening: -
 80  .  .  .  .  .  List: []*ast.Field (len = 1) {
 81  .  .  .  .  .  .  0: *ast.Field {
 82  .  .  .  .  .  .  .  Doc: nil
 83  .  .  .  .  .  .  .  Names: nil
 84  .  .  .  .  .  .  .  Type: *ast.Ident {
 85  .  .  .  .  .  .  .  .  NamePos: sample.go:3:24
 86  .  .  .  .  .  .  .  .  Name: "int"
 87  .  .  .  .  .  .  .  .  Obj: nil
 88  .  .  .  .  .  .  .  }
 89  .  .  .  .  .  .  .  Tag: nil
 90  .  .  .  .  .  .  .  Comment: nil
 91  .  .  .  .  .  .  }
 92  .  .  .  .  .  }
 93  .  .  .  .  .  Closing: -
 94  .  .  .  .  }
 95  .  .  .  }
 96  .  .  .  Body: *ast.BlockStmt {
 97  .  .  .  .  Lbrace: sample.go:3:28
 98  .  .  .  .  List: []ast.Stmt (len = 4) {
 99  .  .  .  .  .  0: *ast.ExprStmt {
100  .  .  .  .  .  .  X: *ast.CallExpr {
101  .  .  .  .  .  .  .  Fun: *ast.SelectorExpr {
102  .  .  .  .  .  .  .  .  X: *ast.Ident {
103  .  .  .  .  .  .  .  .  .  NamePos: -
104  .  .  .  .  .  .  .  .  .  Name: "fmt"
105  .  .  .  .  .  .  .  .  .  Obj: nil
106  .  .  .  .  .  .  .  .  }
107  .  .  .  .  .  .  .  .  Sel: *ast.Ident {
108  .  .  .  .  .  .  .  .  .  NamePos: -
109  .  .  .  .  .  .  .  .  .  Name: "Println"
110  .  .  .  .  .  .  .  .  .  Obj: nil
111  .  .  .  .  .  .  .  .  }
112  .  .  .  .  .  .  .  }
113  .  .  .  .  .  .  .  Lparen: -
114  .  .  .  .  .  .  .  Args: []ast.Expr (len = 1) {
115  .  .  .  .  .  .  .  .  0: *ast.BasicLit {
```

```
116  .  .  .  .  .  .  .  .  .  .  ValuePos: -
117  .  .  .  .  .  .  .  .  .  .  Kind: STRING
118  .  .  .  .  .  .  .  .  .  .  Value: "\"foo\""
119  .  .  .  .  .  .  .  .  .  }
120  .  .  .  .  .  .  .  .  }
121  .  .  .  .  .  .  .  .  Ellipsis: -
122  .  .  .  .  .  .  .  .  Rparen: -
123  .  .  .  .  .  .  .  }
124  .  .  .  .  .  .  }
125  .  .  .  .  .  .  1: *ast.ExprStmt {
126  .  .  .  .  .  .  .  X: *ast.CallExpr {
127  .  .  .  .  .  .  .  .  Fun: *ast.FuncLit {
128  .  .  .  .  .  .  .  .  .  Type: *ast.FuncType {
129  .  .  .  .  .  .  .  .  .  .  Func: sample.go:4:2
130  .  .  .  .  .  .  .  .  .  .  TypeParams: nil
131  .  .  .  .  .  .  .  .  .  .  Params: *ast.FieldList {
132  .  .  .  .  .  .  .  .  .  .  .  Opening: sample.go:4:6
133  .  .  .  .  .  .  .  .  .  .  .  List: nil
134  .  .  .  .  .  .  .  .  .  .  .  Closing: sample.go:4:7
135  .  .  .  .  .  .  .  .  .  .  }
136  .  .  .  .  .  .  .  .  .  .  Results: *ast.FieldList {
137  .  .  .  .  .  .  .  .  .  .  .  Opening: -
138  .  .  .  .  .  .  .  .  .  .  .  List: []*ast.Field (len = 1) {
139  .  .  .  .  .  .  .  .  .  .  .  .  0: *ast.Field {
140  .  .  .  .  .  .  .  .  .  .  .  .  .  Doc: nil
141  .  .  .  .  .  .  .  .  .  .  .  .  .  Names: nil
142  .  .  .  .  .  .  .  .  .  .  .  .  .  Type: *ast.Ident {
143  .  .  .  .  .  .  .  .  .  .  .  .  .  .  NamePos: sample.go:4:9
144  .  .  .  .  .  .  .  .  .  .  .  .  .  .  Name: "int"
145  .  .  .  .  .  .  .  .  .  .  .  .  .  .  Obj: nil
146  .  .  .  .  .  .  .  .  .  .  .  .  .  }
147  .  .  .  .  .  .  .  .  .  .  .  .  .  Tag: nil
148  .  .  .  .  .  .  .  .  .  .  .  .  .  Comment: nil
149  .  .  .  .  .  .  .  .  .  .  .  .  }
150  .  .  .  .  .  .  .  .  .  .  .  }
151  .  .  .  .  .  .  .  .  .  .  .  Closing: -
152  .  .  .  .  .  .  .  .  .  .  }
153  .  .  .  .  .  .  .  .  .  }
154  .  .  .  .  .  .  .  .  .  Body: *ast.BlockStmt {
155  .  .  .  .  .  .  .  .  .  .  Lbrace: sample.go:4:13
156  .  .  .  .  .  .  .  .  .  .  List: []ast.Stmt (len = 1) {
157  .  .  .  .  .  .  .  .  .  .  .  0: *ast.ReturnStmt {
158  .  .  .  .  .  .  .  .  .  .  .  .  Return: sample.go:5:3
159  .  .  .  .  .  .  .  .  .  .  .  .  Results: []ast.Expr (len = 1) {
160  .  .  .  .  .  .  .  .  .  .  .  .  .  0: *ast.BasicLit {
161  .  .  .  .  .  .  .  .  .  .  .  .  .  .  ValuePos: sample.go:5:10
```

```
162  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  Kind: INT
163  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  Value: "2"
164  .  .  .  .  .  .  .  .  .  .  .  .  .  .  }
165  .  .  .  .  .  .  .  .  .  .  .  .  .  }
166  .  .  .  .  .  .  .  .  .  .  .  .  }
167  .  .  .  .  .  .  .  .  .  .  .  }
168  .  .  .  .  .  .  .  .  .  .  .  Rbrace: sample.go:6:2
169  .  .  .  .  .  .  .  .  .  .  }
170  .  .  .  .  .  .  .  .  .  }
171  .  .  .  .  .  .  .  .  .  Lparen: sample.go:6:3
172  .  .  .  .  .  .  .  .  .  Args: nil
173  .  .  .  .  .  .  .  .  .  Ellipsis: -
174  .  .  .  .  .  .  .  .  .  Rparen: sample.go:6:4
175  .  .  .  .  .  .  .  .  }
176  .  .  .  .  .  .  }
177  .  .  .  .  .  .  2: *ast.ExprStmt {
178  .  .  .  .  .  .  .  X: *ast.CallExpr {
179  .  .  .  .  .  .  .  .  Fun: *ast.SelectorExpr {
180  .  .  .  .  .  .  .  .  .  X: *ast.Ident {
181  .  .  .  .  .  .  .  .  .  .  NamePos: -
182  .  .  .  .  .  .  .  .  .  .  Name: "fmt"
183  .  .  .  .  .  .  .  .  .  .  Obj: nil
184  .  .  .  .  .  .  .  .  .  }
185  .  .  .  .  .  .  .  .  .  Sel: *ast.Ident {
186  .  .  .  .  .  .  .  .  .  .  NamePos: -
187  .  .  .  .  .  .  .  .  .  .  Name: "Println"
188  .  .  .  .  .  .  .  .  .  .  Obj: nil
189  .  .  .  .  .  .  .  .  .  }
190  .  .  .  .  .  .  .  .  }
191  .  .  .  .  .  .  .  .  Lparen: -
192  .  .  .  .  .  .  .  .  Args: []ast.Expr (len = 1) {
193  .  .  .  .  .  .  .  .  .  0: *ast.BinaryExpr {
194  .  .  .  .  .  .  .  .  .  .  X: *ast.Ident {
195  .  .  .  .  .  .  .  .  .  .  .  NamePos: sample.go:7:9
196  .  .  .  .  .  .  .  .  .  .  .  Name: "a"
197  .  .  .  .  .  .  .  .  .  .  .  Obj: *(obj @ 35)
198  .  .  .  .  .  .  .  .  .  .  }
199  .  .  .  .  .  .  .  .  .  .  OpPos: sample.go:7:11
200  .  .  .  .  .  .  .  .  .  .  Op: +
201  .  .  .  .  .  .  .  .  .  .  Y: *ast.Ident {
202  .  .  .  .  .  .  .  .  .  .  .  NamePos: sample.go:7:13
203  .  .  .  .  .  .  .  .  .  .  .  Name: "b"
204  .  .  .  .  .  .  .  .  .  .  .  Obj: *(obj @ 58)
205  .  .  .  .  .  .  .  .  .  .  }
206  .  .  .  .  .  .  .  .  .  }
207  .  .  .  .  .  .  .  .  }
```

```
208  .  .  .  .  .  .  .  .  Ellipsis: -
209  .  .  .  .  .  .  .  .  Rparen: -
210  .  .  .  .  .  .  .  }
211  .  .  .  .  .  .  }
212  .  .  .  .  .  .  3: *ast.ReturnStmt {
213  .  .  .  .  .  .  .  Return: sample.go:7:2
214  .  .  .  .  .  .  .  Results: []ast.Expr (len = 1) {
215  .  .  .  .  .  .  .  .  0: *(obj @ 193)
216  .  .  .  .  .  .  .  }
217  .  .  .  .  .  .  }
218  .  .  .  .  .  }
219  .  .  .  .  .  Rbrace: sample.go:8:1
220  .  .  .  .  }
221  .  .  }
222  .  }
223  .  FileStart: sample.go:1:1
224  .  FileEnd: sample.go:8:3
225  .  Scope: *ast.Scope {
226  .  .  Outer: nil
227  .  .  Objects: map[string]*ast.Object (len = 1) {
228  .  .  .  "foo": *(obj @ 15)
229  .  .  }
230  .  }
231  .  Imports: nil
232  .  Unresolved: []*ast.Ident (len = 4) {
233  .  .  0: *(obj @ 44)
234  .  .  1: *(obj @ 67)
235  .  .  2: *(obj @ 84)
236  .  .  3: *(obj @ 142)
237  .  }
238  .  Comments: nil
239  .  GoVersion: ""
240  }
```

Вывод

В ходе выполнения лабораторной работы была достигнута цель по изучению представления синтаксических деревьев в памяти компилятора и приобретению навыков их преобразования. На примере демонстрационной программы на языке G o была реализована программа для трансформации синтаксических деревьев: программа модифицировала все неанонимн ые функции, возвращающие одно значение типа int, добавляя вывод имени функции и её возвращаемого значения.

`go/ast` может быть полезным при разработке компиляторов, инструментов синтаксического анализа и других прилож ений, работающих с исходным кодом.