Лабораторная работа № 2.1. Синтаксические деревья

Скоробогатов С.Ю.

20 августа 2013

1 Цель работы

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

2 Исходные данные

В качестве исходного языка и языка реализации программы преобразования синтаксических деревьев выберем язык Go. Пакеты `"go/token"`, `"go/ast"` и `"go/parser"` из стандартной библиотеки этого языка содержат готовый «front-end» компилятора языка Go, а пакет `"go/format"` восстанавливает исходный текст программы по её синтаксическому дереву. Документацию по этим пакетам можно посмотреть по адресу http://golang.org/pkg/go/.

Построение синтаксического дерева по исходному тексту программы выполняется функцией `parser.ParseFile`, возвращающей указатель типа `*ast.File` на корень дерева.

Синтаксические деревья в памяти представляются значениями структур из пакета `"go/ast"`. Изучать синтаксические деревья удобно по их листингам, порождаемым функцией `ast.Fprint`. Небольшая программа `astprint`, которая, ко всему прочему, демонстрирует вызов парсера для построения синтаксического дерева программы, представлена на листинге.

```
package main

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

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

    // Создаём хранилище данных об исходных файлах
    fset := token.NewFileSet()

    // Вызываем парсер
    if file, err := parser.ParseFile(
        fset,                // данные об исходниках
        os.Args[1],          // имя файла с исходником программы
        nil,                 // пусть парсер сам загрузит исходник
        parser.ParseComments, // приказываем сохранять комментарии
    ); err == nil {
        // Если парсер отработал без ошибок, печатаем дерево
        ast.Fprint(os.Stdout, fset, file, nil)
    } else {
        // В противном случае, выводим сообщение об ошибке
        fmt.Printf("Error: %v", err)
    }
}
```

Напомним, что для компиляции программы `astprint` нужно выполнить команду

```
go build astprint.go
```

Обход синтаксического дерева в глубину реализован в функции `ast.Inspect`, которая вызывает переданную ей в качестве параметра функцию для каждого посещённого узла дерева. С помощью этой функции удобно осуществлять поиск узлов определённого типа в дереве. Например, представленная на листинге ниже функция `insertHello` выполняет поиск всех операторов `if` в дереве и вставляет в начало положительной ветки каждого найденного оператора печать строки `"hello"`.

Скачать тестовый пример: demo.go.

```
func insertHello(file *ast.File) {
    // Вызываем обход дерева, начиная от корня
    ast.Inspect(file, func(node ast.Node) bool {
        // Для каждого узла дерева
        if ifStmt, ok := node.(*ast.IfStmt); ok {
            // Если этот узел имеет тип *ast.IfStmt,
            // добавляем в начало массива операторов
            // положительной ветки if'а новый оператор
            ifStmt.Body.List = append(
                []ast.Stmt{
                    // Новый оператор — выражение
                    &ast.ExprStmt{
                        // Выражение — вызов функции
                        X: &ast.CallExpr{
                            // Функция — "fmt.Printf"
                            Fun: &ast.SelectorExpr{
                                X:   ast.NewIdent("fmt"),
                                Sel: ast.NewIdent("Printf"),
                            },
                            // Её параметр — строка "hello"
                            Args: []ast.Expr{
                                &ast.BasicLit{
                                    Kind:  token.STRING,
                                    Value: "\"hello\"",
                                },
                            },
                        },
                    },
                },
                ifStmt.Body.List...,
            )
        }
        // Возвращая true, мы разрешаем выполнять обход
        // дочерних узлов
        return true
    })
}
```

Восстановление исходного текста программы из синтаксического дерева осуществляется функцией `format.Node`. Эта функция не обращает внимания на координаты узлов дерева, выполняя полное переформатирование текста программы, поэтому при преобразовании дерева координаты новых узлов прописывать не нужно.

3 Задание

Выполнение лабораторной работы состоит из нескольких этапов:

1. подготовка исходного текста демонстрационной программы, которая в дальнейшем будет выступать в роли объекта преобразования (демонстрационная программа должна размещаться в одном файле и содержать функцию `main`);
2. компиляция и запуск программы `astprint` для изучения структуры синтаксического дерева демонстрационной программы;
3. разработка программы, осуществляющей преобразование синтаксического дерева и порождение по нему новой программы;
4. тестирование работоспособности разработанной программы на исходном тексте демонстрационной программы.

Преобразование синтаксического дерева должно вносить в преобразуемую программу дополнительные возможности, указанные в индивидуальном варианте.

4 Индивидуальный вариант

Если в исходной программе при создании отображения с помощью функции `make` не указан размер отображения, следует этот размер сделать равным 16.

5 Отчёт по лабораторной работе № 2.1

Отчёт выполняется в разметке Markdown по следующему шаблону:

````
% Лабораторная работа № 2.1. Синтаксические деревья
% 21 февраля 2023 г.
% Вася Пупкин, ИУ9-63Б
# Цель работы
‹переписываете цель работы из задания›
# Индивидуальный вариант
‹переписываете индивидуальный вариант›
# Реализация
Демонстрационная программа:
```go
xxxxx
```
Программа, осуществляющая преобразование синтаксического дерева:
```go
xxxxx
```
# Тестирование
Результат трансформации демонстрационной программы:
```go
xxxx
```
Вывод тестового примера на `stdout` (если необходимо)
```
xxxx
```
# Вывод
‹пишете, чему научились›
````

В отчёте приведён лишь необходимый минимум.

Шаблон отчёта

Ваш отчёт будет конвертирован в PDF при помощи pandoc следующей командой:

```
pandoc \
  --pdf-engine=xelatex \
  -V 'mainfont:Liberation Serif' \
  -V 'monofont:Liberation Mono' \
  "$SOURCE" -o "$PDF"
```

Язык реализации: Markdown

Код решения

Из файла

Отправить