Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Верификация и анализ программ

Отчет по лабораторной работе №1

Построение графа потока управления программы

Выполнил студент гр. 23541/3 Смирнов С.В.

(подпись)

Преподаватель Ицыксон В.М.

(подпись)

“ ” 2019 г.

Санкт – Петербург

2018

## Цель работы

Построить поведенческую модель программы по структурной модели. Программа должна осуществлять построение поведенческой модели опираясь на абстрактно-синтаксическое дерево(AST), написанной на языке Golang. В качестве результата необходимо выдавать текстовый файл с описанием поведенческой модели (CFG- граф потока управления).

## Теоретическая информация

Абстрактное синтаксическое дерево(AST) - помеченное ориентированное дерево, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.

Синтаксические деревья используются в парсерах для промежуточного представления программы между деревом разбора и структурой данных, которая за этим используется в качестве внутреннего представления в компиляторе или интерпретаторе компьютерной программы для оптимизации и генерации кода.

Граф потока управления(CFG) - множество всех возможных путей исполнения программы, представленное в виде графа.

В графе потока управления каждый узел графа соответствует базовому блоку - прямолинейному участку кода, не содержащему в себе ни операций передачи управления, ни точек, на которые управление передается из других частей программы. Имеется лишь два исключения:

* точка, на которую выполняется переход, является первой инструкцией в базовом блоке
* базовый блок завершается инструкцией перехода

## Ход выполнения работы

Общий план выполнения работы:

1. Получение AST из исходного кода на Golang
2. Построение CFG из AST
3. Отображение графа в наглядном виде

### Получение AST из исходного кода на Golang

Для построения AST из исходного кода на Golang использовалась библиотека go/ast[1]. Для построения CFG необходимо рекурсивно обойти полученное AST и в зависимости от типа текущей вершины обработать ее включение в дерево CFG.

### Построение CFG из AST

За рекурсивный обход AST и построение CFG отвечает структура builder и основной метод stmt.

type builder struct {

cfg \*CFG

current \*Block

}

func (b \*builder) stmt(\_s ast.Stmt) {

var label \*lblock

switch s := \_s.(type) {

case \*ast.BadStmt,

\*ast.SendStmt,

\*ast.IncDecStmt,

\*ast.GoStmt,

\*ast.DeferStmt,

\*ast.EmptyStmt,

\*ast.AssignStmt,

\*ast.ExprStmt:

b.add(s)

…

case \*ast.IfStmt:

if s.Init != nil {

b.stmt(s.Init)

}

then := b.newBlock("if.then")

done := b.newBlock("if.done")

\_else := done

if s.Else != nil {

\_else = b.newBlock("if.else")

}

b.add(s.Cond)

b.ifelse(then, \_else)

b.current = then

b.stmt(s.Body)

b.jump(done)

if s.Else != nil {

b.current = \_else

b.stmt(s.Else)

b.jump(done)

}

b.current = done

case \*ast.ForStmt:

b.forStmt(s, label)

…

В данном классе хранится текущая формируемая нода с символьным пояснением типа, индексом и дополнительно информацией, а также граф CFG в текущем состоянии. Метод stmt на вход принимает один оператор AST и вызывает вспомогательные методы обработки операторов.

### Отображение графа

Для отображения графа было решено использовать утилиту Graphviz[2]. Наиболее удобным форматом представления графа, считываемым данной утилитой является dot. Для формирования файла в данном формате из графа CFG был написан метод Format:

func (g \*CFG) Format() string {

var buf bytes.Buffer

for \_, b := range g.Blocks {

if len(b.Succs) > 0 {

for \_, succ := range b.Succs {

fmt.Fprintf(&buf, "\"%s[%d]\"", b.nodeType, b.index)

fmt.Fprintf(&buf, "->")

fmt.Fprintf(&buf, "\"%s[%d]\";\n", succ.nodeType, succ.index)

}

} else if len(b.Succs) == 0 && len(b.nodeType) > 0 {

fmt.Fprintf(&buf, "\"%s[%d]\"", b.nodeType, b.index)

fmt.Fprintf(&buf, "->")

fmt.Fprintf(&buf, "\"%s\";\n", "end")

}

}

return buf.String()

}

### Результат работы

Для проверки работоспособности использовался следующий входной файл(input.txt) с исходным кодом программы:

package main

func main() {

n := 4

if ( n < 3){

println("2")

}else {

n = 3

}

for i := 1; i < n; i++ {

if i > 1 {

println(i)

}

}

}

В результате работы программы формируется следующий выходной файл(assist.txt):

digraph CFG1{

"entry[0]"->"if.then[1]";

"entry[0]"->"if.else[3]";

"if.then[1]"->"if.done[2]";

"if.done[2]"->"for.loop[6]";

"if.else[3]"->"if.done[2]";

"for.body[4]"->"if.then[8]";

"for.body[4]"->"if.done[9]";

"for.done[5]"->"end";

"for.loop[6]"->"for.body[4]";

"for.loop[6]"->"for.done[5]";

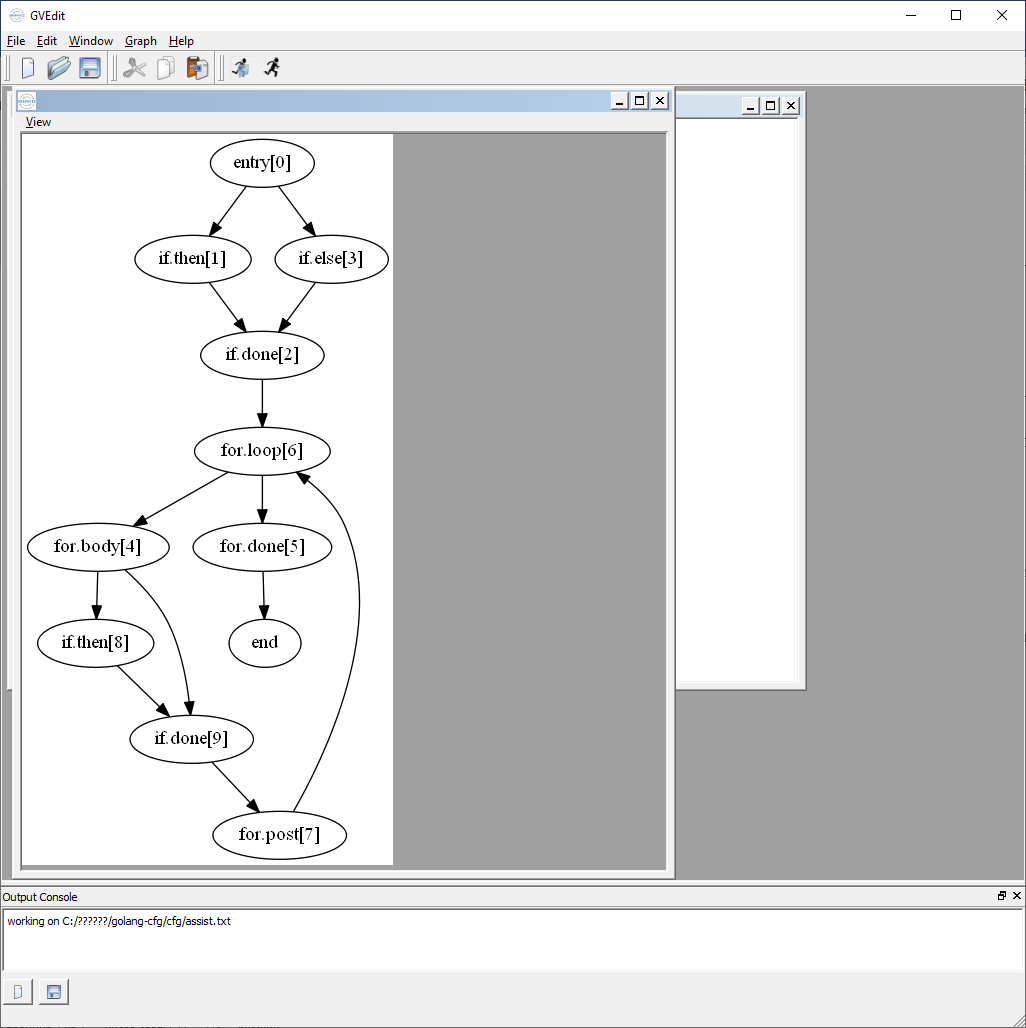
"for.post[7]"->"for.loop[6]";

"if.then[8]"->"if.done[9]";

"if.done[9]"->"for.post[7]";

}

При загрузке файла в Graphviz и построении отображения получаем следующий граф:



### Выводы

В ходе выполнения данной работы была разработана программа для построения CFG (Control Flow Graph) исходного кода программы на языке Golang. Получение модели программы производится в два этапа:

1. Построение AST из исходного кода
2. Построение CFG по полученному AST

Для построения AST использовалась библиотека go/ast. Полученная программа поддерживает только простейшие программы на Golang. Она не поддерживает:

* инструкции switch
* break/continue
* panic

## Источники

1. Библиотека для построения AST из исходного кода на Golang URL: <https://golang.org/pkg/go/ast/>
2. Утилита для отображения графов из различных форматов <https://www.graphviz.org/>