Hochschule Luzern
Departement für Informatik

PCP Projekt - Programmiersprachanalyse

Analyse der Programmiersprache GO

Studierende: Frederico Fischer, Oliver Werlen
Dozenten: Prof. Dr. Ruedi Arnold, Marcel Baumann
Abgabedatum: 26. Mai 2021

1 Einleitung

2 Vision, Geschichte & Verbreitung

3 Sprachkonstrukte

3.1 Goroutines, Channels & Select

3.1.1 Goroutines

Eine Goroutine ist ein leichtgewichtiger Thread, welcher von der Go runtime gemanaged wird. Goroutines nutzen dabei den selben Adressraum, daher muss der Zugriff auf geteilte Ressourcen synchronisiert werden. In Go gibt es Primitives, welche die Synchronisierung übernehmen. Jedoch werden diese nur selten genutzt, da in den meisten Fällen mit Channels gearbeitet wird.

3.1.2 Channels

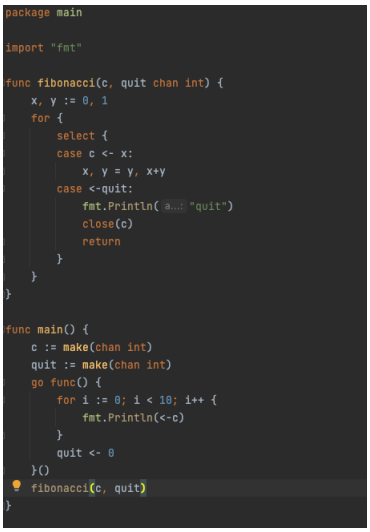
Ein Channel erlaubt einen einfachen Datenfluss. Per Default ist dabei das Senden und Empfangen blockierend, bis die andere Seite bereit ist. Goroutines lassen sich somit sehr leicht synchronisieren, ohne den Einsatz von Locks oder Variablen. Der Channel kann explizit vom Sender geschlossen werden. Damit wird dem Empfänger signalisiert, dass keine Werte mehr empfangen werden können. Das Schliessen des Channels sollte dabei exklusiv vom Sender ausgeführt werden. Senden auf einen geschlossenen Channel verursacht dabei "panic".

3.1.3 Select

Bei einem Select wird bei mehrfacher Auswahl gewartet, bis eine Operation laufen kann.

3.1.4 Beispiel

Im Beispiel unten werden alle oben genannten Sprachkonstrukte genutzt. Bei func() handelt es sich um eine Goroutine, welche 10 Werte von dem Channel c liesst und anschliessend einen Wert auf den Quit-Channel schreibt. Das Select diffenziert dabei zwischen den beiden Channels. Je nach Case wird entweder eine neue Fibonacci-Zahl berechnet oder der Channel geschlossen.



```go
package main

import "fmt"

func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
        case c <- x:
            x, y = y, x+y
        case <-quit:
            fmt.Println( a...: "quit")
            close(c)
            return
        }
    }
}

func main() {
    c := make(chan int)
    quit := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-c)
        }
        quit <- 0
    }()
    fibonacci(c, quit)
}
```

Abbildung 1: Beispiel Goroutine, Channel und Select,
Quelle: Autor

3.2 Maps & Slices

Erklärung und Codebeispiel

3.3 Structural & Nominal Typing

Erklärung und Codebeispiel

3.4 The Go Memory Model

Erklärung und Codebeispiel

3.5 Package Management

Erklärung und Codebeispiel

3.6 Defer

Erklärung und Codebeispiel

4 Fazit

4.1 Team-Fazit

4.2 Fazit Frederico Fischer

4.3 Fazit Oliver Werlen