Dynamic Analysis of Concurrent Go-Programs

Bachelorarbeit - Kolloquium

Erik Kassubek

Institut für Informatik
Albert-Ludwigs-Universität Freiburg

14.02.2023

Nebenläufigkeit in Go

- Go-Routine
  - leichtgewichtiger Thread
  - ermöglicht Nebenläufigkeit
- Mutexe (Locks)
  - Synchronisationsmechanismus
  - Löst das Problem des gegenseitigen Ausschluss
- Channel
  - Synchronisationsmechanismus
  - Ermöglichen Kommunikation zwischen Routinen

Mutex

```
func main() {
  var m sync.Mutex

  go func() {
    m.Lock()
    a()  // kritischer Abschnitt
    m.Unlock()
  }()

  m.Lock()
  b()  // kritischer Abschnitt
  m.Unlock()
}
```

Mutex

```
func main() {
  var m sync.Mutex

  go func() {
    m.Lock()
    a()  // kritischer Abschnitt
    m.Unlock()
  }()

  m.Lock()
  b()  // kritischer Abschnitt
  m.Unlock()
}
```

```
func main() {
  var m sync.RWMutex

  go func() {
    x.RLock()
    read()
    x.RUnlock()
  }()

  go func() {
    x.RLock()
    read()
    x.RUnlock()
  }()

  x.Lock()
  write()
  x.Unlock()
}
```

Mutex

```
func main() {
  m sync.Mutex
  n sync.Mutex

  go func() {
    m.Lock()
    n.Lock()
    n.Unlock()
    m.Unlock()
  }()

  n.Lock()
  m.Lock()
  m.Unlock()
  n.Unlock()
}
```

Mutex

```
func main() {
  m sync.Mutex
  n sync.Mutex

  go func() {
    m.Lock()
    n.Lock()
    n.Unlock()
    m.Unlock()
  }()

  n.Lock()
  m.Lock()
  m.Unlock()
  n.Unlock()
}
```

```
func main() {
  var m sync.Mutex

  m.Lock()
  m.Lock()
}
```

Channel - Unbuffered

```
func main() {
  c := make(chan int)

  go func() {
    c <- 1
  }()

  <- c
}
```

Channel - Unbuffered

```go
func main() {
  c := make(chan int)

  go func() {
    c <- 1
  }()

  <- c
}
```

```go
func main() {
  c := make(chan int)

  go func() {
    c <- 1
  }()

  go func() {
    <- c
  }()

  <- c
}
```

Channel - Buffered

```
func main() {
  c := make(chan int, 2)
  d := make(chan int)

  go func() {
    c <- 1
    c <- 1
    d <- 1
  }()

  <- d
  <- c
  <- c
}
```

Channel - Buffered

```go
func main() {
  c := make(chan int, 2)
  d := make(chan int)

  go func() {
    c <- 1
    c <- 1
    d <- 1
  }()

  <- d
  <- c
  <- c
}
```

```go
func main() {
  c := make(chan int, 1)
  d := make(chan int)

  go func() {
    c <- 1
    c <- 1
    d <- 1
  }()

  <- d
  <- c
  <- c
}
```

Channel - Close

- Schließt Channel $\Rightarrow$ keine weiter Kommunikation möglich
- Send auf geschlossenem Channel $\Rightarrow$ Laufzeitfehler

Channel - Close

- Schließt Channel ⇒ keine weiter Kommunikation möglich
- Send auf geschlossenem Channel ⇒ Laufzeitfehler

```
func main() {
  c := make(chan int)

  go func() {
    c <- 1
  }()
  go func() {
    <- c
  }()
  close(c)
}
```

Analyse

- Entwicklung und Implementierung eines Detektors zur
  - Erkennung von problematischen Situationen
  - Analyse von problematischen Situationen

Analyse

- Entwicklung und Implementierung eines Detektors zur
  - Erkennung von problematischen Situationen
  - Analyse von problematischen Situationen
- Dynamische Analyse
  - Instrumentierung
  - Programm (mehrfach) ausführen
  - Verhalten aufzeichnen $\Rightarrow$ Trace
  - Trace analysieren

Detektor - Instrumentierung

TODO: Instrumentierung

Detektor - Select

TODO: Instrumentierung Select

Detektor - Programm ausführen

TODO: Programm ausführen

Detektor - Trace

TODO: Trace

Detektor - Analyse - Mutexe

TODO: Analyse Mutexe

Detektor - Analyse - Channel

TODO: Analyse Channel