Notes and Patterns in Concurrent Channel-Based Programming

Bruno Bonatto

December 15, 2022

Introduction

Concurrent programming is a very difficult problem, fortunately many incredibly intelligent people have spent many hours developing tools and theories to help us develop correct and secure concurrent programs. This is a collection of notes on some modern approaches to concurrent programming.

Motivational Example

```
func Generate(ch chan<- int) {
for i := 2; ; i++ {ch <- i}
}
func Filter(in <-chan int, out chan<- int, prime int) {
for {
i := <-in
if i%prime == 0 {out <- i}
}
}
func main() {
ch := make(chan int)
go Generate(ch)
for i := 0; ; i++ {
prime := <-ch
ch1 := make(chan int)
go Filter(ch, ch1, prime)
ch = ch1
```

Communicating Sequential Processes

Lorem

Pi Calculus

A formal system for computation, analogous to the lambda calculus, focused on concurrency and message passing.

Core Language

Only two kinds of entities: processes and channels.

- $\bar{x}y$ sends the value y along the port/channel x.
- $x(z)$ receives a value from x and assigns it to the variable z.
- $e_1 \cdot e_2$ composes e_1 and e_2 sequencially.
- $e_1 | e_2$ composes e_1 and e_2 parallelly

Example

$$\bar{x}y \cdot e_1 \mid x(z) \cdot e_2 \;\Rightarrow\; e_1 \mid \{z \mapsto y\}e_2$$

This "program" sends the value y through the channel x from one process to another, the sending process then continues to execute the program e_1; while the receiving process executes the program e_2 but with all instances of the variable z replaced by the value y.

Core Language

Fresh channels can be introduced with ν.
The expression $(\nu x)e$ creates a fresh channel x in the scope e.
For example:

$$(\nu x)(\bar{x} \cdot e_1 \mid x(z) \cdot e_2)$$

Localizes the channel x, ensuring that no other processes can interfere with communication on x.

Common Concurrency Errors

Channel safety

Once a channel is closed, receive actions always succeed, but all send and close actions raise a runtime error.

Global deadlocks

The Go runtime contains a *global* deadlock detector that signals a runtime error when *all* goroutines in a program are stuck. However it is often the case that, when certain libraries are imported (such as the commonly used `net` package), the global deadlock is silently *disabled*.

Partial deadlocks

Sometimes called *liveness* failures, partial deadlocks occur when a program's communication cannot progress despite some of its goroutines not being stuck.

Partial Deadlock Example

```
func prod(ch chan int) {
for i:=0; i < 5; i++ {ch <- i}
close(ch)
}
func cons(ch1, ch2 chan int) {
for {
select {
        case x:=<-ch1: print(x)
case x:=<-ch2: print(x)
}
}
}
func main() {
ch1, ch2 := make(chan int), make(chan int)
go prod(ch1)
go prod(ch2)
```

References

- Hoare (1978)
- Pierce, Turner (2000)
- Lange, Ng, Toninho, Yoshida (2017)
- Lange, Ng, Toninho, Yoshida (2018)
- Castro, Hu, Jongmas, Ng, Yoshida (2019)