Go is a programming language prominent in Cloud Computing.

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function

```
var a string
func setA() { a = "hello" }

func main() {
  setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function

```
var a string
func setA() { a = "hello" }

func main() {
  setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function

```
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function

```
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
-

```
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello" }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()

  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()
  <- done
  print(a)
}
```

Go is built for concurrency.

- Spawning a new thread *(goroutine)* is as easy as calling a function
- Synchronization is done via channel communication

```
var done = make(chan bool, 10)
var a string
func setA() { a = "hello"; done <- true }

func main() {
  go setA()
  <- done
  print(a)
}
```

Since it is easy to make concurrency mistakes,
Go has a built-in data-race detector.

Since it is easy to make concurrency mistakes,
Go has a built-in data-race detector.

```
go run -race my_program.go
```

Repairing the Go data-race detector.

A story on applied research.

Daniel S. Fava

danielsf@ifi.uio.no

Department of informatics
University of Oslo, Norway

Our *story* has a theoretical basis in two active areas of research.

Our *story* has a theoretical basis in two active areas of research.

1. Memory model

Our *story* has a theoretical basis in two active areas of research.

1. Memory model
2. Data-race detection

Our *story* has a theoretical basis in two active areas of research.

1. Memory model

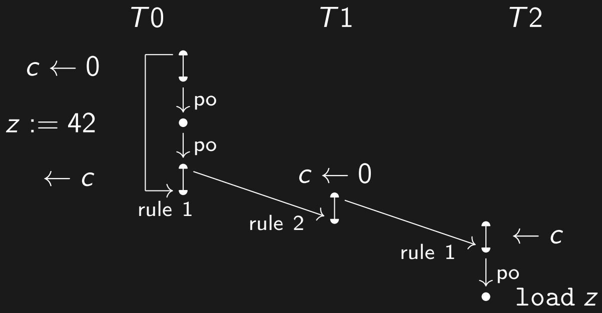
2. Data-race detection

What happens when a detector is at odds with the memory model

Our *story* also has real practical implications.

Our *story* also has real practical implications.

- Mismatch lead to the under-reporting of data-races
  - No warning about missing some synchronization
- A bug on a tool to find bugs (compound effect)
- Bug evaded Go maintainers for six years



T0
T1
T2
c ← 0
po
z := 42
po
← c
rule 1
c ← 0
rule 2
← c
rule 1
po
load z

We implemented a fix accepted by the Go community.
Here we share three main lessons learned.

We implemented a fix accepted by the Go community. Here we share three main lessons learned.

1. Mind the Gap

We implemented a fix accepted by the Go community. Here we share three main lessons learned.

1. Mind the Gap
2. Models don't have to be right, they have to be useful

We implemented a fix accepted by the Go community. Here we share three main lessons learned.

1. Mind the Gap
2. Models don't have to be right, they have to be useful
3. Bad news is good news

1. Mind the Gap.

1. Mind the Gap.

Go memory model

1. Mind the Gap.

Go memory model

Go data-race detector

1. Mind the Gap.

Go memory model

- succinct document
- written in English
- technical vocabulary

Go data-race detector

1. Mind the Gap.

Go memory model

- succinct document
- written in English
- technical vocabulary

Go data-race detector

- thousands of lines of code
- different projects & repos
- different languages
  (Go, C/C++, assembly)

1. Mind the Gap.

Go memory mod

- succinct d
- written in
- technical

Formal model

Small-step operational semantics

[Fava et al., 2018a, Fava and Steffen, 2020]

detector

s of lines of code

rojects & repos

anguages

++, assembly)

1. Mind the Gap.

Go memory mod

- succinct d
- written in
- technical

Formal model

Small-step operational semantics

[Fava et al., 2018a, Fava and Steffen, 2020]

- succinct

detector

- s of lines of code
- projects & repos
- anguages
++, assembly)

1. Mind the Gap.

Go memory mod

- succinct d
- written in
- technical

Formal model
Small-step operational semantics
[Fava et al., 2018a, Fava and Steffen, 2020]

- succinct
- executable

detector

s of lines of code
rojects & repos
anguages
++, assembly)

1. Mind the Gap.

Go memory mod

- succinct d
- written in
- technical

Formal model

Small-step operational semantics

[Fava et al., 2018a, Fava and Steffen, 2020]

- succinct
- executable
- formal:

detector

- s of lines of code
- projects & repos
- anguages
  ++, assembly)

1. Mind the Gap.

Go memory mod

- succinct d
- written in
- technical

Formal model

Small-step operational semantics

[Fava et al., 2018a, Fava and Steffen, 2020]

- succinct
- executable
- formal: use logic to state and prove properties

detector

s of lines of code

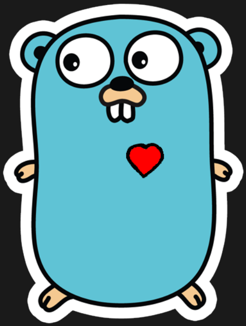
projects & repos

anguages

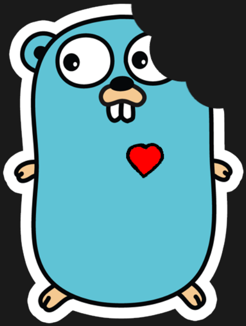
++, assembly)

2. Models don't have to be right, they have to be useful.

2. Models don't have to be right, they have to be useful.



2. Models don't have to be right, they have to be useful.



without interfaces

2. Models don't have to be right, they have to be useful.

without interfaces
or packages

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines

Concurrency

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines

Concurrency

Channels

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines

Concurrency

Channels

Synchronization

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines	Channels
Concurrency	Synchronization

2. Models don't have to be right, they have to be useful.

without interfaces
or packages
or pointers
or arrays, maps...

Goroutines	Channels	Shared memory
Concurrency	Synchronization	

2. Models don't have to be right, they have to be useful.

2. Models don't have to be right, they have to be useful.

Our operational-semantics was an approximation of the Go memory model

2. Models don't have to be right, they have to be useful.

Our operational-semantics was an approximation of the Go memory model

The model was sufficiently accurate to be useful

3. Bad news is good news.

3. Bad news is good news.

- ✗ High effort in formalizing and proving properties of SW.
- ✗ Industry is busy delivering.

3. Bad news is good news.

- ✗ High effort in formalizing and proving properties of SW.
- ✗ Industry is busy delivering.

- ✓ Academia can develop valuable artifacts not common in industry.

3. Bad news is good news.

- ✗ High effort in formalizing and proving properties of SW.
- ✗ Industry is busy delivering.

- ✓ Academia can develop valuable artifacts not common in industry.
- ✓ There is room for collaboration.

Three main open questions remain..

1. By taking channels seriously,
   can we improve data-race detection?

1. By taking channels seriously, can we improve data-race detection?

- Most data-race detectors are based on locks (including Go's)
  `acquire` and `release`

1. By taking channels seriously, can we improve data-race detection?

- Most data-race detectors are based on locks (including Go's)
  `acquire` and `release`
- Synchronization via channels is *different* from locks
  `send` is not a combination of `acquires` and `releases`

1. By taking channels seriously, can we improve data-race detection?

- Most data-race detectors are based on locks (including Go's)
  `acquire` and `release`
- Synchronization via channels is *different* from locks
  `send` is not a combination of `acquires` and `releases`
- Partial answer to the above question is Yes
  The proposed fix is faster,

1. By taking channels seriously, can we improve data-race detection?

- Most data-race detectors are based on locks (including Go's)

  `acquire` and `release`
- Synchronization via channels is *different* from locks

  `send` is not a combination of `acquires` and `releases`
- Partial answer to the above question is Yes

  The proposed fix is faster, and consumes less memory

1. By taking channels seriously, can we improve data-race detection?

- Most data-race detectors are based on locks (including Go's)
  `acquire` and `release`
- Synchronization via channels is *different* from locks
  `send` is not a combination of `acquires` and `releases`
- Partial answer to the above question is Yes
  The proposed fix is faster, and consumes less memory

"Channels at the head of the queue!"

2. Can we improve our formalization
   to better match Go's memory model?

2. Can we improve our formalization to better match Go's memory model?

- We have ideas on further relaxing the proposed memory model

2. Can we improve our formalization to better match Go's memory model?

- We have ideas on further relaxing the proposed memory model
- Can we relax the model "all the way"?

2. Can we improve our formalization to better match Go's memory model?

- We have ideas on further relaxing the proposed memory model
- Can we relax the model "all the way"?

  Without adding *out-of-thin-air* behavior into the model

3. How to automate bug finding?

3. How to automate bug finding?

- Connect formal model to Go compiler/runtime

3. How to automate bug finding?

- Connect formal model to Go compiler/runtime
- Find bugs in compiler/runtime

3. How to automate bug finding?

- Connect formal model to Go compiler/runtime
- Find bugs in compiler/runtime
- Verify correctness of compiled code
  across different compilation targets (HW)

In summary,

In summary,

Specification

Implementation

In summary,

Specification

Formal model

Implementation

In summary,

Specification

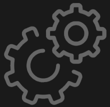
Formal model

Implementation

In summary,


Specification


Formal model

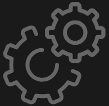

Implementation

- Abstraction

In summary,

Specification

Formal model



Implementation

- Abstraction
- Value-added

Questions?

Questions?

Thank you

References

- Go memory model (2014). The Go memory model.
  `https://golang.org/ref/mem`.
  Version of May 31, 2014, covering Go version 1.9.1
- Fava, D. (2020). Finding and fixing a mismatch between the Go memory model and data-race detector.
  Submitted for publication

References

- Fava, D. S. and Steffen, M. (2020). Ready, Set, Go!: Data-race detection and the Go language. *Science of Computer Programming*, 195:102473
- Fava, D., Steffen, M., and Stolz, V. (2018a). Operational semantics of a weak memory model with channel synchronization. *Journal of Logic and Algebraic Methods in Programming*. An extended version of the FM'18 publication with the same title

References

- GitHub (2020). [37355] runtime/race: running with -race misses races (mismatch with memory model).
  `https://github.com/golang/go/issues/37355`
- Gerrit (2020). [220419] runtime: swap the order of raceacquire() and racerelease().
  `https://go-review.googlesource.com/c/go/+/220419`
- Phabricator (2020). [d76322] tsan: Adding releaseacquire() to threadclock.
  `https://reviews.llvm.org/D76322`

We implemented a fix accepted by the Go community.

We implemented a fix accepted by the Go community.

2019

We implemented a fix accepted by the Go community.

2019 Nov

We implemented a fix accepted by the Go community.

2019 Nov Studying source

We implemented a fix accepted by the Go community.

2019	Nov	Studying source
2020	Jan	Experimentation

We implemented a fix accepted by the Go community.

2019	Nov	Studying source
2020	Jan	Experimentation
	Feb	

We implemented a fix accepted by the Go community.

2019	Nov	Studying source
2020	Jan	Experimentation
	Feb	Found bug by inspection

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	

We implemented a fix accepted by the Go community.

2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
	Apr	Tech report	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
	Apr	Tech report	
2018	Jan	Proofs	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
	Apr	Tech report	
2018	Jan	Proofs	
	Jul	Conference [Fava et al., 2018b]	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
	Apr	Tech report	
2018	Jan	Proofs	
	Jul	Conference [Fava et al., 2018b]	
2019	Feb	Journal [Fava et al., 2018a]	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

We implemented a fix accepted by the Go community.

2017	Jan	Research	
	Apr	Tech report	
2018	Jan	Proofs	
	Jul	Conference [Fava et al., 2018b]	
2019	Feb	Journal [Fava et al., 2018a]	
2020	Apr	Journal [Fava and Steffen, 2020]	
2019	Nov	Studying source	
2020	Jan	Experimentation	
	Feb	Found bug by inspection	First patch
	Mar	New primitive into TSan	Updated patch
	Apr	Fix to TSan primitive	
	May	Updated TSan lib in Go	Patch approved

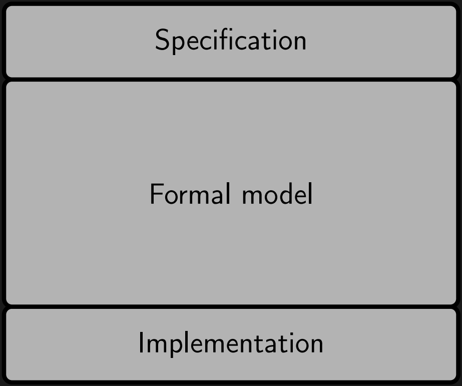
3. How to automate bug finding?

3. How to automate bug finding?

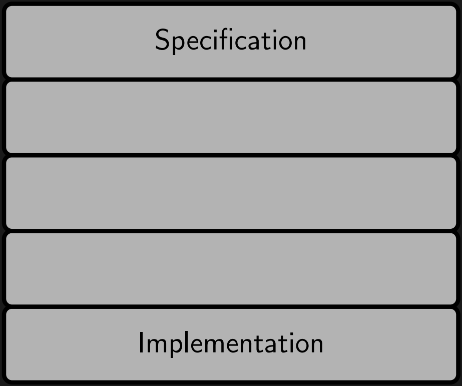
Specification

Implementation

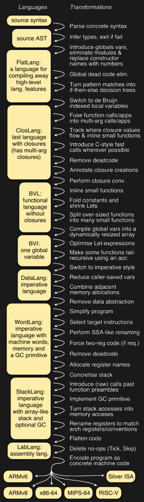
3. How to automate bug finding?



3. How to automate bug finding?



3. How to automate bug finding?



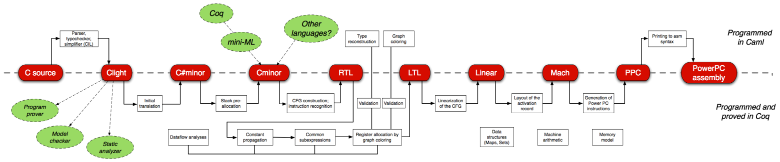
CakeML

verified compiler covering a *substantial* subset of Standard ML

3. How to automate bug finding?

CompCert

verified C compiler covering a *large* subset of C99



Go is also used to program the Internet of Things (IoT).

Go is also used to program the Internet of Things (IoT).

Go is also used to program the Internet of Things (IoT).

GOBOT

TinyGo

Go is also used to program the Internet of Things (IoT).

GOBOT

TinyGo

MAINFLUX