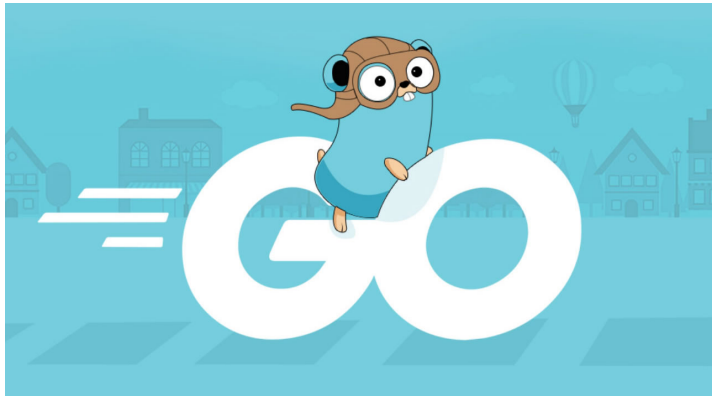
Go2Pins: a framework for the LTL verification of Go programs

A. Kirszenberg, A. Martin, H. Moreau, and E. Renault

July 12, 2021

The Go programming language



- Designed at Google in 2009
  - Used by Docker, Google, Facebook, Soundcloud, ...
  - 308,480 repositories on GitHub (as of June, 25th)
- Inspired by Hoare [1985] Communicating Sequential Processes (CSP)
- Communication using channels

A simple example

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

Fibonacci computation in Go

A simple example

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

Fibonacci computation in Go

How to ensure that this program loops infinitely?

A simple example

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

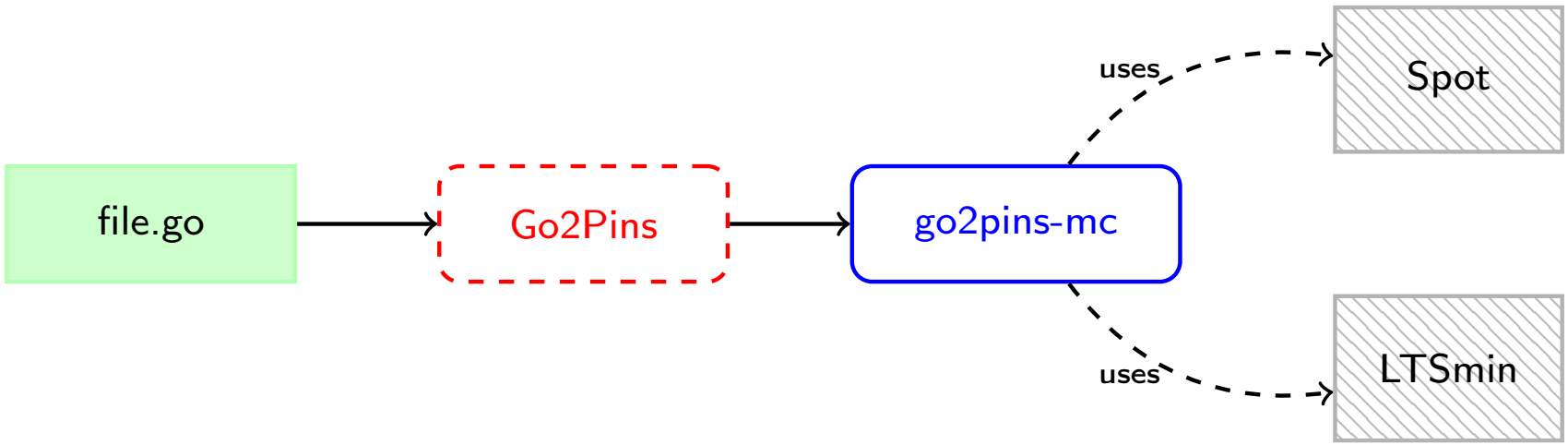
Fibonacci computation in Go

How to ensure that this program loops infinitely?

Check the LTL property
G "a < 10"

Introducing Go2Pins

A framework for the LTL verification of Go programs

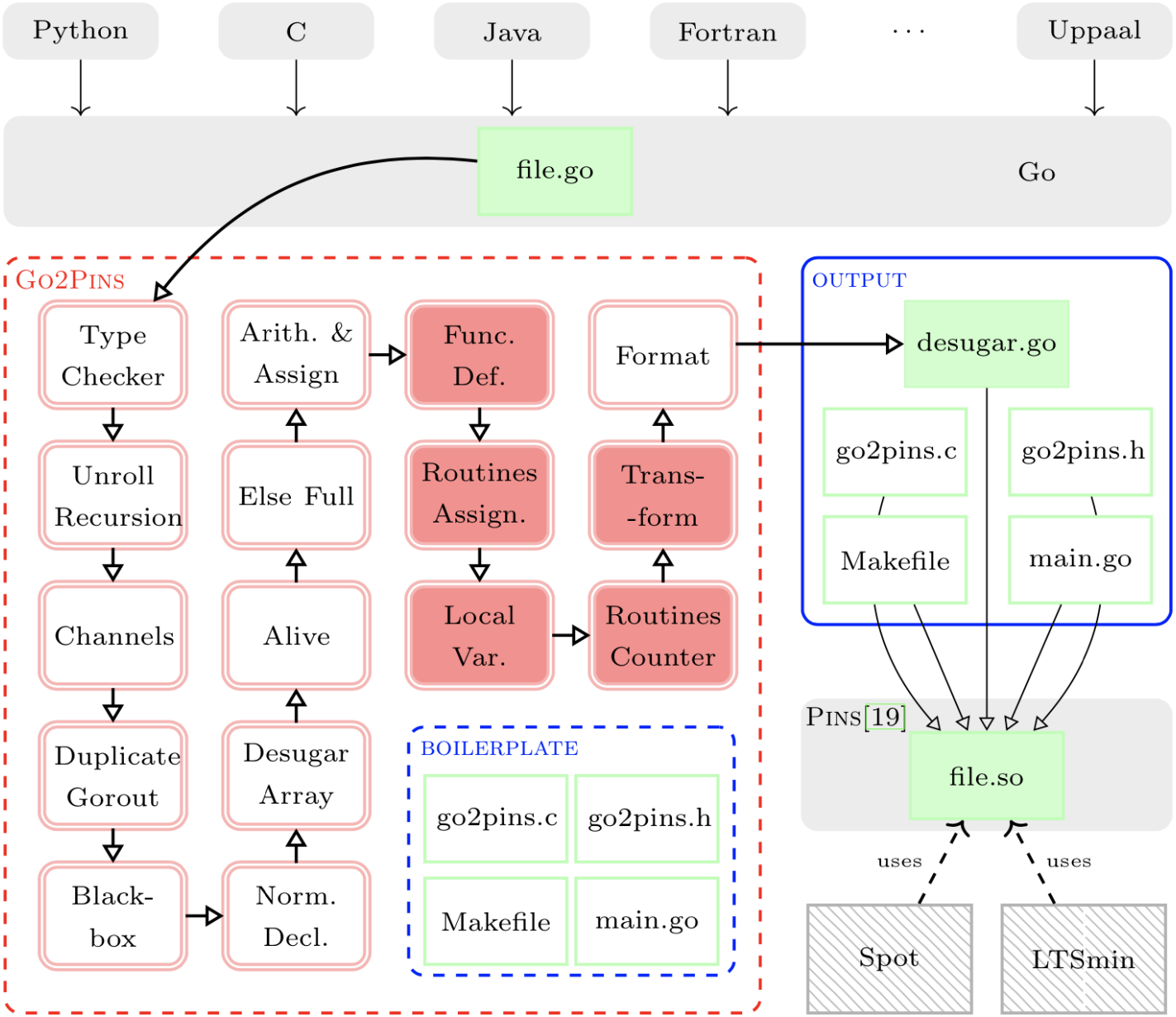


Go2Pins: overview

- A transpiler that translates the input Go program into an output Go program
  - Benefits from both the reflexivity and the standard library
  - Work in steps, each step desugaring a specific feature from the AST

- The output respects the PINS interface [Kant et al., 2015].
  - One function for retrieving the initial state of the system,
  - One for computing the successors of a state,
  - A state is represented as a vector of integer
  - *Any program that exposes this interface is compatible with any (explicit or symbolic) model checking solution that supports it*

Go2Pins: architecture



Core translation: step 1

Build a (finite) state vector for the program

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

a
n
n0
n1
n2
i
is_alive_a
is_alive_n1
is_alive_n0
is_alive_n2
res0
PC_1
PC_2
fibo caller
fibo callerLabel
main caller
main callerLabel

Core translation: step 2

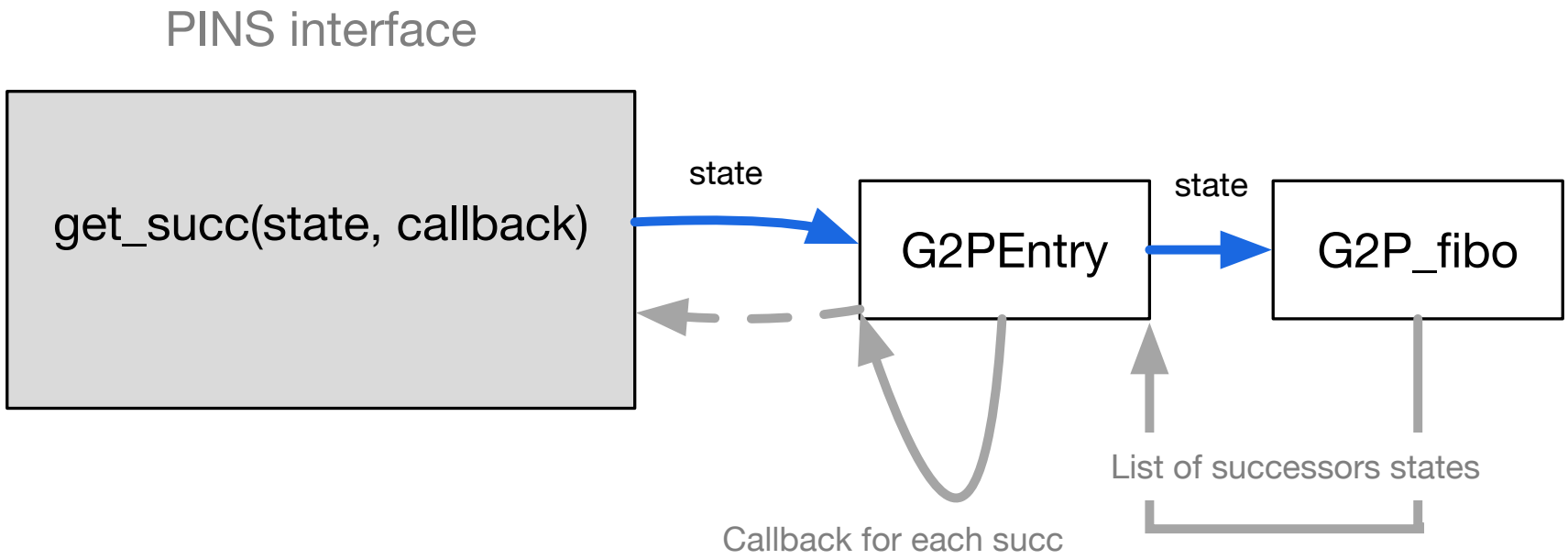
Extract *atomic* operations

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

```
1  type state [17]int
2  func G2PF_fibo(s state) state{
3    switch s.LabelCounter {
4      case 0: goto label0
5      //...
6      case 12: goto label12
7    }
8    label0: // n0 := 0
9      s.fibo.n0 = 0
10     s.LabelCounter = 1
11     s.fibo.isalive = 1
12     return s
13   //...
14   label12: // return n1
15   s.fibo.res0 = s.fibo.n1
16   s.fibo.FunctionCounter =
17       s.fibo.caller
18   s.fibo.LabelCounter =
19       s.fibo.callerLabel
20   return s
21 }
```

Core translation: step 3

Dispatch to the correct function and fit the PINS interface



Core translation: step 4

Build a package (library) that contains all previous transformations

Link it with the go2pins-mc binary that also provides facilities:

- to list all variables of the transformed program
- to display the state space
- to chose the backend (LTSmin or Spot) to use
- to specify the number of threads to use with this backend
- to check LTL properties
- ...

Handling Concurrency

```
1  var com chan int = make(chan int)
2
3  func producer() {
4          com <- 51
5  }
6
7  func cons() {
8          a := 42
9          a = <-com
10         a = a + 1
11 }
12
13 func main() {
14         go producer()
15         cons()
16 }
```

Producer/consumer in Go

Handling Concurrency

```
1  var com chan int = make(chan int)
2
3  func producer() {
4          com <- 51
5  }
6
7  func cons() {
8          a := 42
9          a = <-com
10         a = a + 1
11 }
12
13 func main() {
14         go producer()
15         cons()
16 }
```

Producer/consumer in Go

How to handle concurrency?

Handling Concurrency

```
1  var com chan int = make(chan int)
2
3  func producer() {
4          com <- 51
5  }
6
7  func cons() {
8          a := 42
9          a = <-com
10         a = a + 1
11 }
12
13 func main() {
14         go producer()
15         cons()
16 }
```

Producer/consumer in Go

How to handle concurrency?

G2P_entry contains a scheduler that computes all possibles synchronizations from potential successors

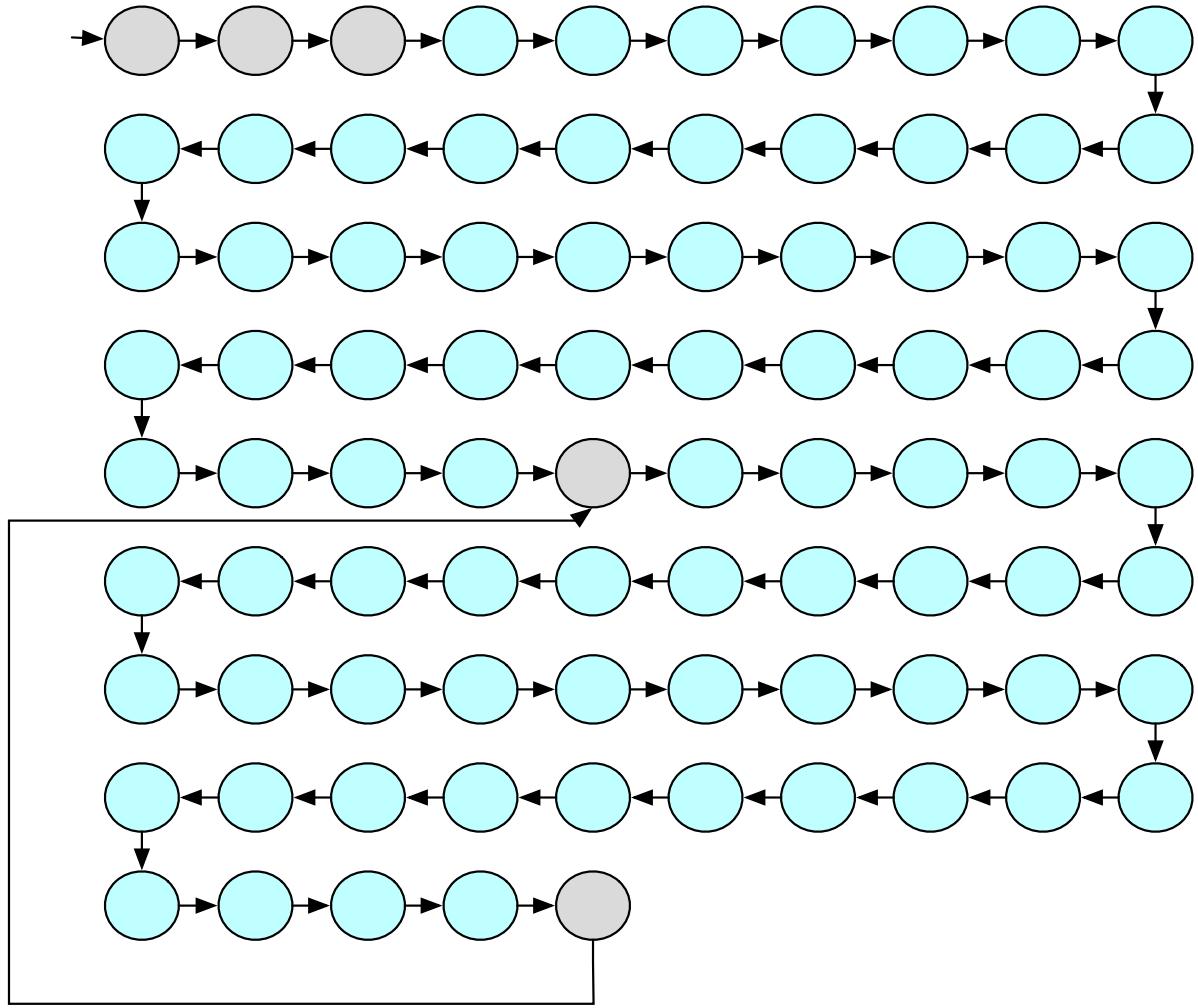
State vector is augmented with slots that are in charge of tracking potential synchronization

Abstraction with Black-Box Transitions (1/2)

How to reduce state space size?

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```

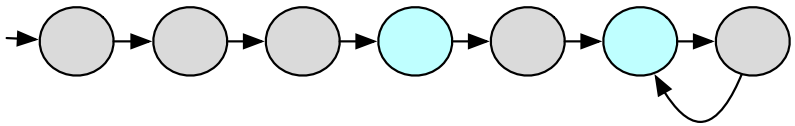
85 states
85 transitions



Abstraction with Black-Box Transitions (2/2)

- **Automatically** detect relevant functions based on observed atomic propositions
- Extended to handle global variables (details in the paper)

```
1  func fibo(n int) int {
2    n0 := 0
3    n1 := 1
4    for i := 0; i < n; i++ {
5      n2 := n0 + n1
6      n0 = n1
7      n1 = n2
8    }
9    return n1
10 }
11
12 func main() {
13     a := 1
14     for ; a < 10; {
15           a = fibo(5)
16     }
17 }
```



7 states
7 transitions

Some notes on Go2Pins

- **Currently restricted to only integer variables ...**
- ... BUT blackbox transitions can help to support complex types
- ... For instance, we can call any function from the go runtime

- **Can handle recursion ...**
- ... BUT only up to a (user specifiable) fixed depth

- **For now only a fixed number of go-routines are supported**

Related work

- Static analysis of operations on channels
  - [Liu et al., 2016] developed a tool that statically detects patterns of bugs and fix them according to some strategies
  - Other approaches focus on extracting channels operations [Liu et al., 2016; Lange et al., 2018, 2017; Ng and Yoshida, 2016; Dilley and Lange, 2020]
- [Dekker et al., 2014; Giunti, 2020] convert formal program into Go
- JPF [Visser et al., 2018] requires providing the source code of the standard library and relies on a Virtual Machine.
- SPIN atomics, d steps and c code instructions
- Approaches based on LLVM [Zaks and Joshi, 2008,?]

Benchmark: setup

Models:

- 41 files (in C) coming from the RERS challenges
- tranlated using c4go in Go
- resulting in 1 909 345 LOC

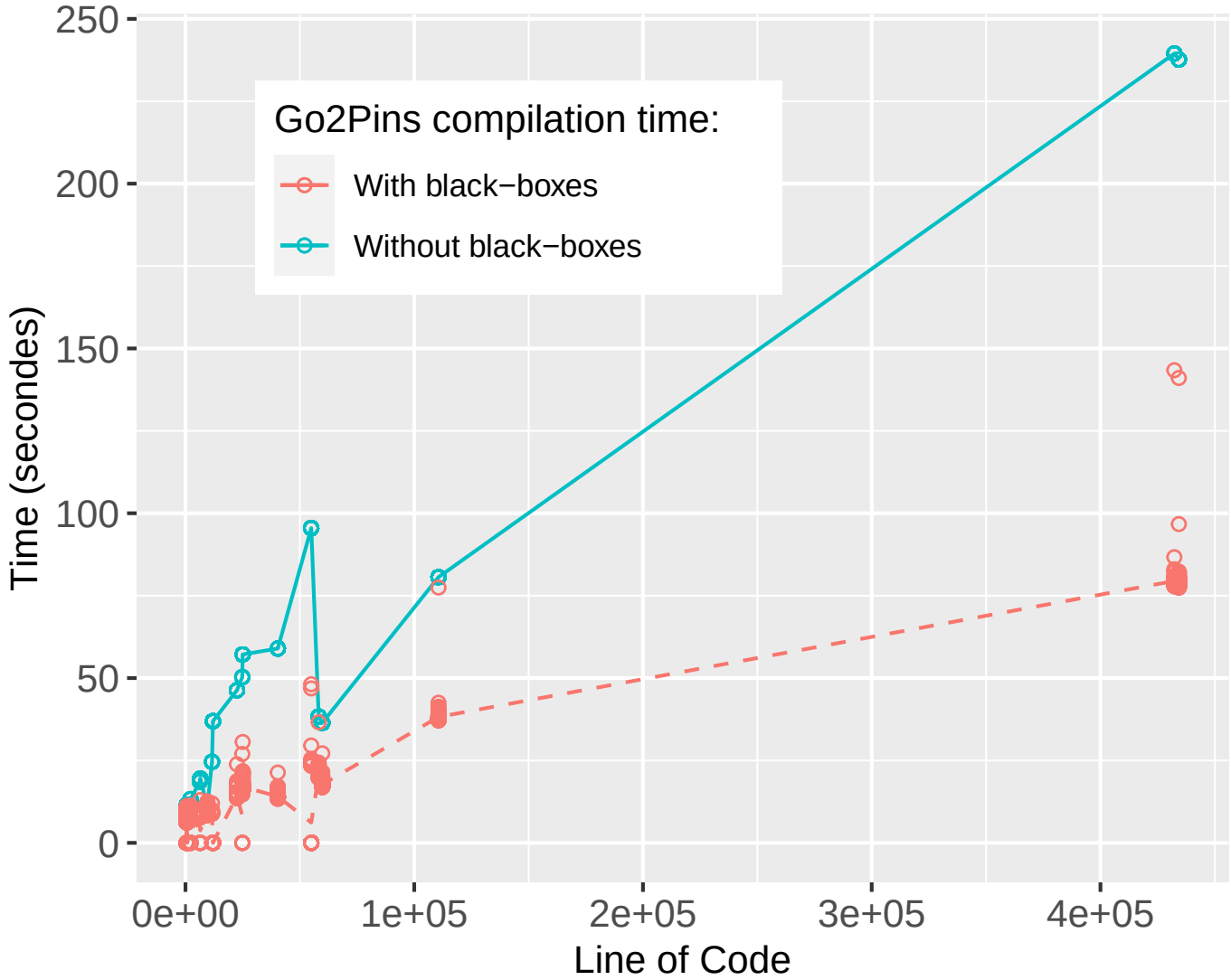
Formulae:

- 5 064 LTL formulae
- 35% are verified and 65% are violated
- 25% pure guarantee, 44% pure safety, 2% pure obligation, 12% pure persistence, 12% pure recurrence, and 5% pure reactivity

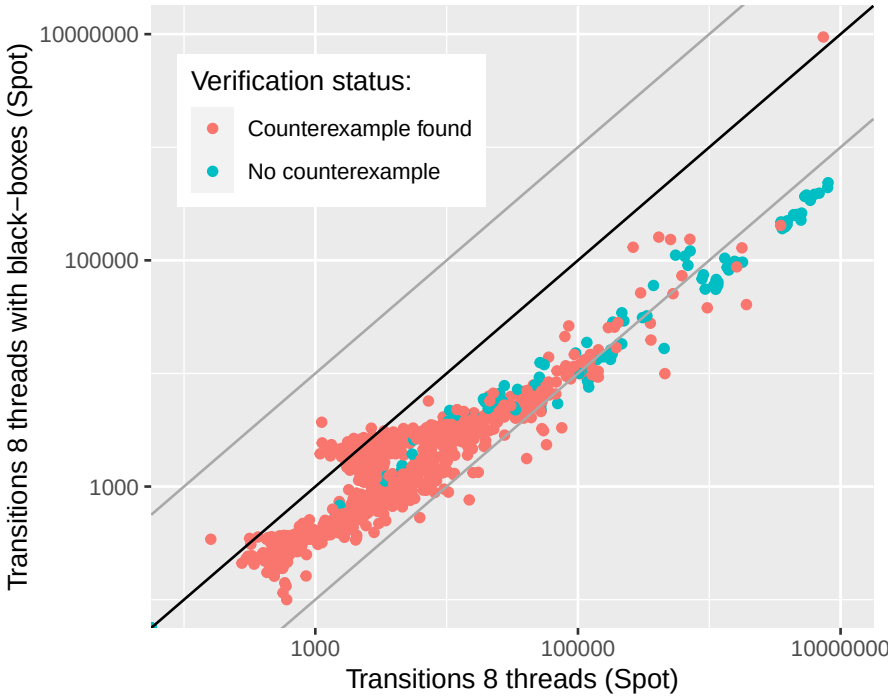
Backend: Spot and LTSmin

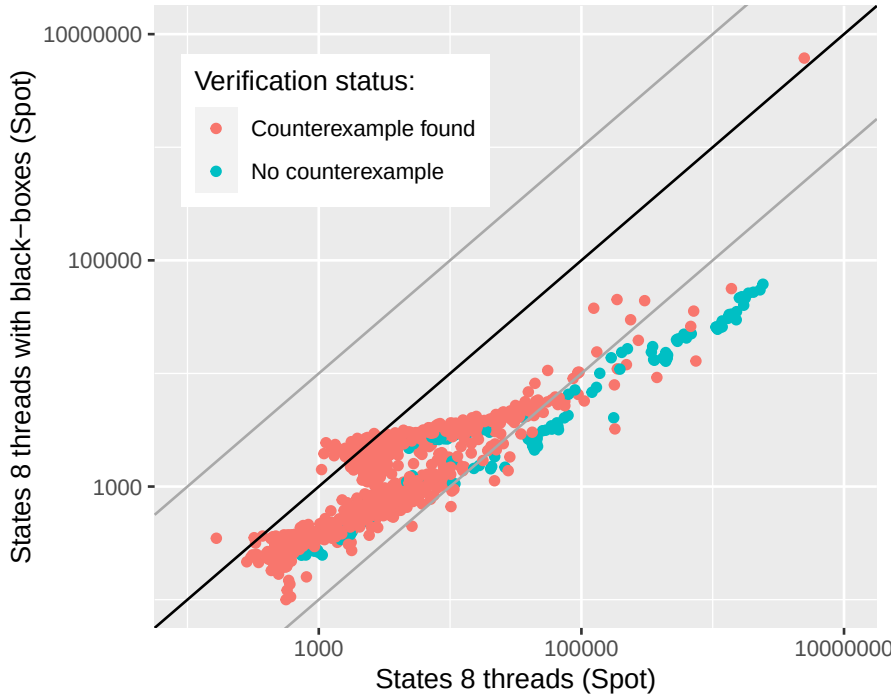
200 Go, 8 threads, 4 minutes timeout on a 24 cores Intel Xeon @2.66GHz

Benchmark: compilation time

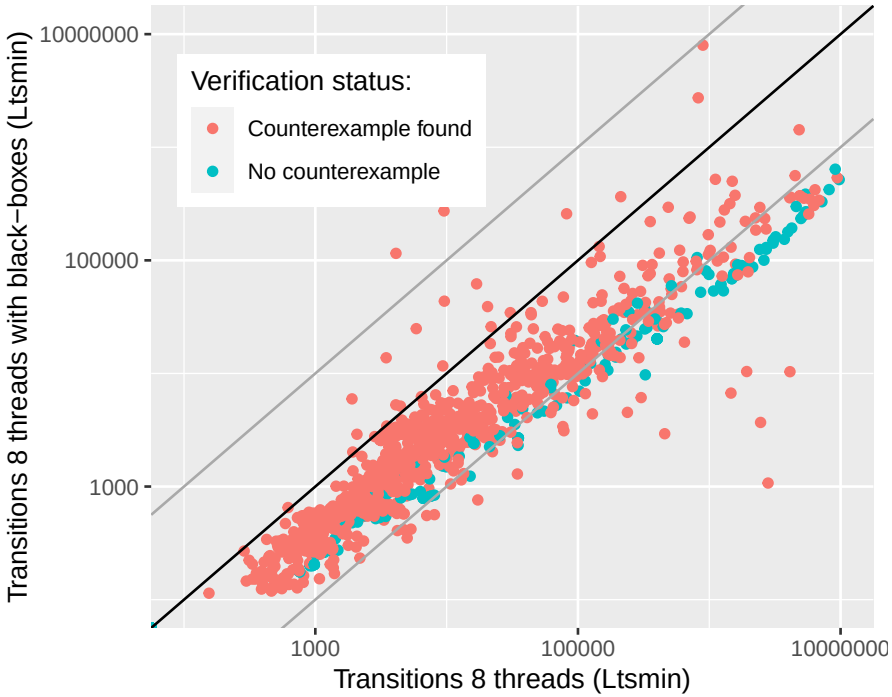


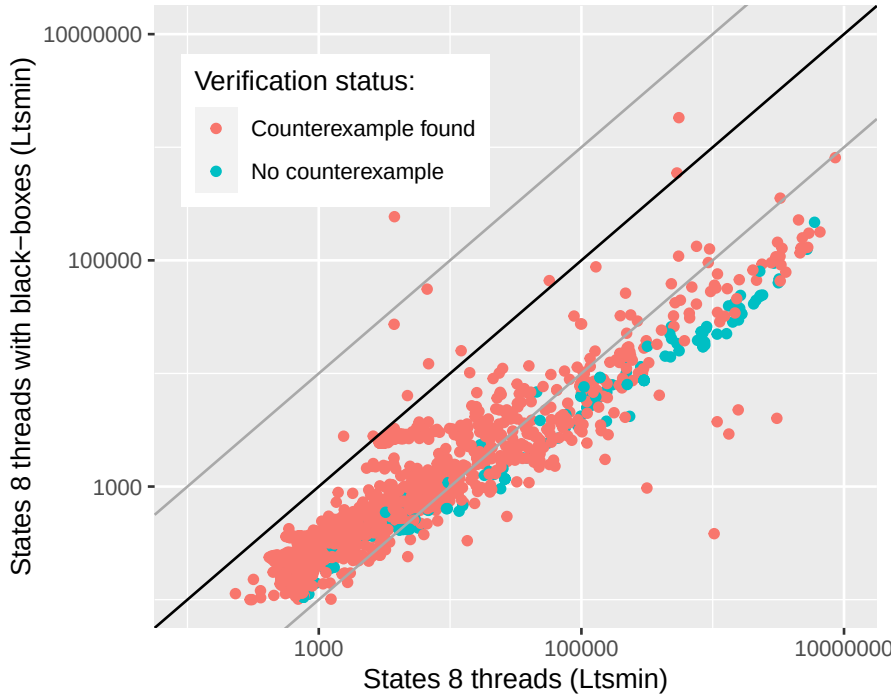
Benchmark: Spot



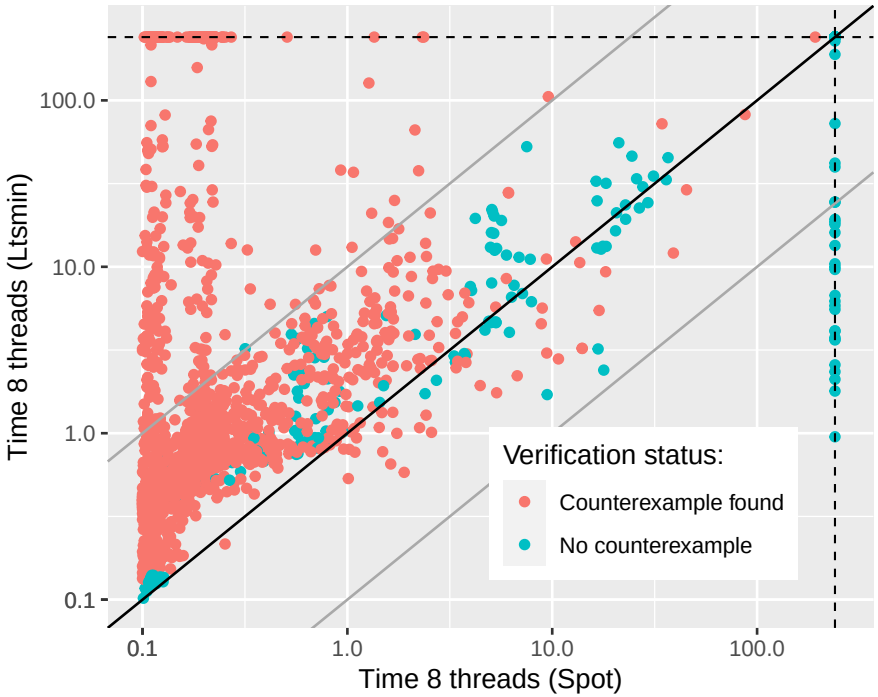


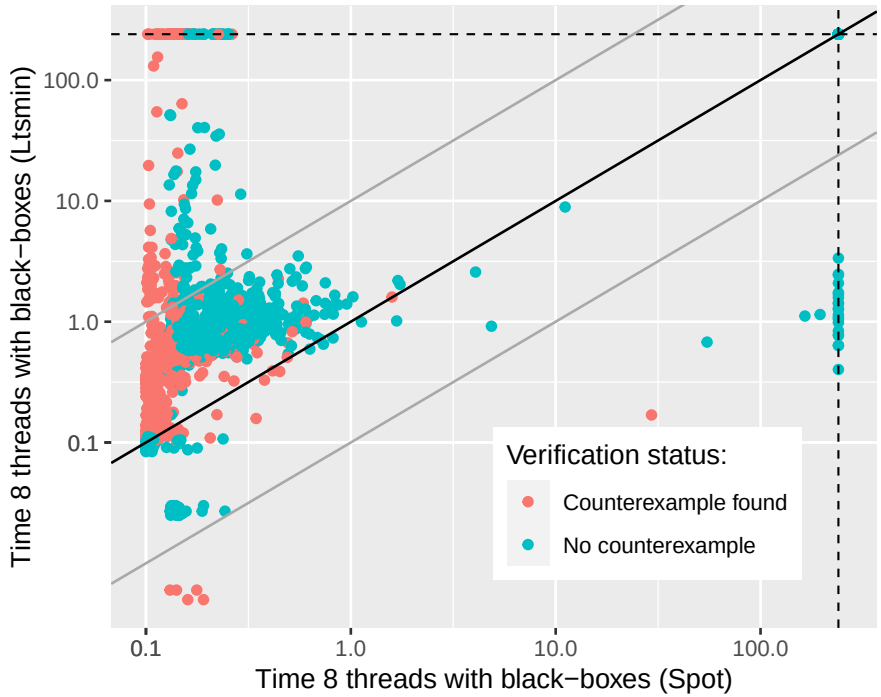
Benchmark: LTSmin





Benchmark: Spot vs. LTSmin

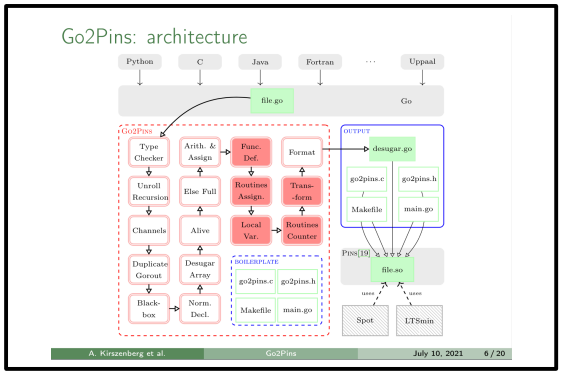


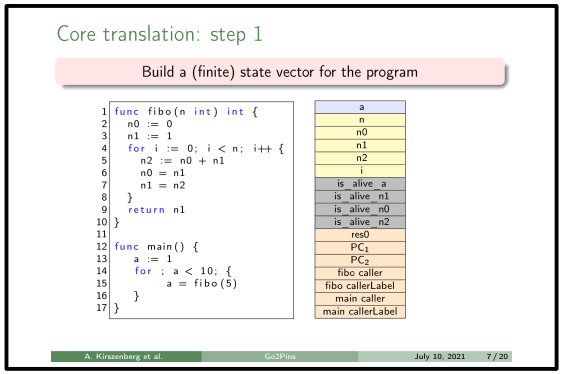


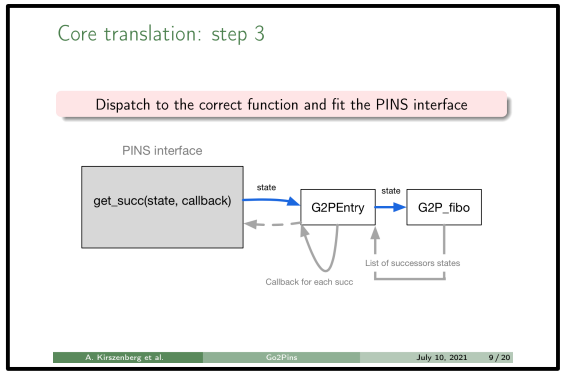
Future work

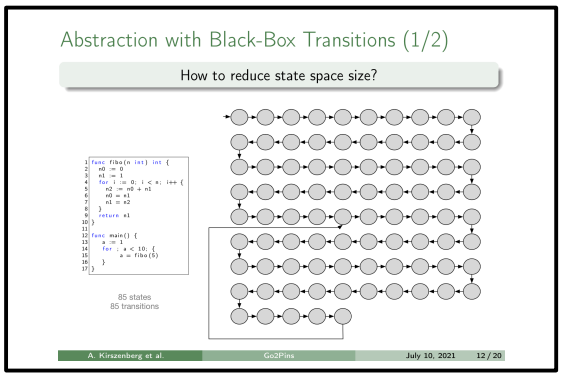
- Support more types
- Support Partial Order Reductions
- Study the relation between black-boxes and POR
- Finer blackboxes
- Check the fairness of the Go scheduler

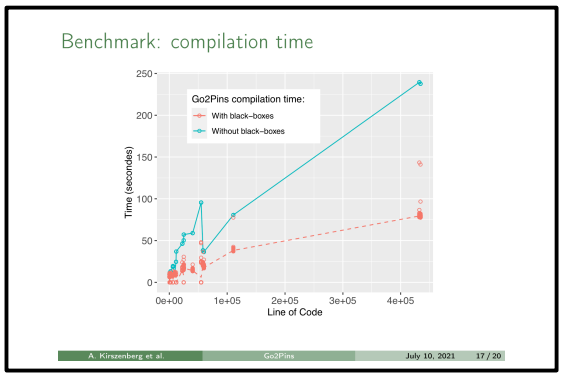
Questions?

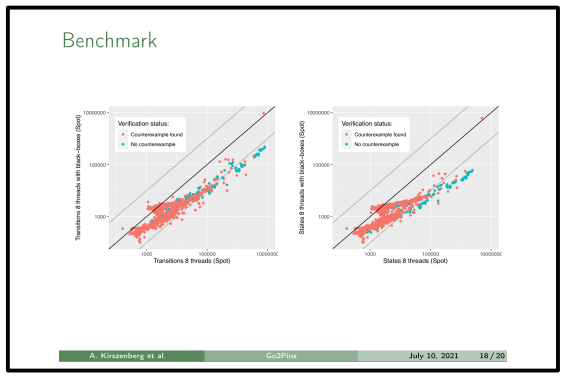

Go2Pins: architecture
Python
C
Java
Fortran
Uppaal
file.go
Go
Type Checker
Arith. & Assign
Func. Def.
Format
Unroll Recursion
Else Full
Routines Assign.
Trans-form
Channels
Alive
Local Var.
Routines Counter
Duplicate Gorout
Desugar Array
Black-box
Norm. Decl.
BOILERPLATE
go2pins.c
go2pins.h
Makefile
main.go
OUTPUT
desugar.go
Pins[19]
file.so
Spot
LTSmin
A. Kirszenberg et al.
Go2Pins
July 10, 2021
6 / 20


Core translation: step 1
Build a (finite) state vector for the program
func fibo(n int) int {
n0 := 0
n1 := 1
for i := 0; i < n; i++ {
n2 := n0 + n1
n0 = n1
n1 = n2
}
return n1
}
func main() {
a := 1
for ; a < 10; {
a = fibo(5)
}
}
a
n
n0
n1
n2
i
is_alive_a
is_alive_n1
is_alive_n0
is_alive_n2
res0
PC1
PC2
fibo caller
fibo callerLabel
main caller
main callerLabel
A. Kirszenberg et al.
Go2Pins
July 10, 2021
7 / 20


Core translation: step 3
Dispatch to the correct function and fit the PINS interface
PINS interface
get_succ(state, callback)
state
G2PEntry
state
G2P_fibo
List of successors states
Callback for each succ
A. Kirszenberg et al.
Go2Pins
July 10, 2021
9 / 20


Abstraction with Black-Box Transitions (1/2)
How to reduce state space size?
85 states
85 transitions
A. Kirszenberg et al.
Go2Pins
July 10, 2021
12 / 20


Benchmark: compilation time
Go2Pins compilation time:
With black-boxes
Without black-boxes
Time (seconds)
Line of Code
A. Kirszenberg et al.
Go2Pins
July 10, 2021
17 / 20


Benchmark
Verification status:
Counterexample found
No counterexample
Transitions 8 threads (Spot)
States 8 threads (Spot)
A. Kirszenberg et al.
Go2Pins
July 10, 2021
18 / 20

Bibliography I

Dekker, J., Vaandrager, F., and Smetsers, R. (2014). Generating a google go framework from an uppaal model. Master's thesis, Radboud University.

Dilley, N. and Lange, J. (2020). Bounded verification of message-passing concurrency in go using promela and spin. Electronic Proceedings in Theoretical Computer Science, 314:34–45.

Giunti, M. (2020). Gopi: Compiling linear and static channels in go. In Bliudze, S. and Bocchi, L., editors, Coordination Models and Languages, pages 137–152, Cham. Springer International Publishing.

Hoare, C. A. R. (1985). Communicating Sequential Processes. Prentice-Hall, Inc., USA.

Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., and van Dijk, T. (2015). Ltsmin: High-performance language-independent model checking. In Proceedings of the fifteen International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), pages 692–707.

Lange, J., Ng, N., Toninho, B., and Yoshida, N. (2017). Fencing off Go: Liveness and Safety for Channel-based Programming. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17), pages 748–761. ACM.

Lange, J., Ng, N., Toninho, B., and Yoshida, N. (2018). A Static Verification Framework for Message Passing in Go using Behavioural Types. In Proceedings of the 40th International Conference on Software Engineering (CSE'18), pages 1137–1148. ACM.

Bibliography II

Liu, Z., Zhu, S., Qin, B., Chen, H., and Song, L. (2016). Automatically detecting and fixing concurrency bugs in go software systems. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), volume 11, pages 2227–2240.

Ng, N. and Yoshida, N. (2016). Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis. In Proceedings of the 25th International Conference on Compiler Construction (CCC'16), pages 174–184. ACM.

Visser, W., Havelund, K., Brat, G., Park, S., and Lerda, F. (2018). Model Checking Programs. In Proceedings of the Automated Software Engineering (ASE'03), volume 10, pages 203–232. Springer.

Zaks, A. and Joshi, R. (2008). Verifying Multi-threaded C Programs with SPIN. In Proceedings of the 15 International SPIN Workshop on Model Checking of Software (SPIN'08), pages 94–107.