Trace-based capture and replay for detecting Go concurrency bugs

Study project

Erik Kassubek

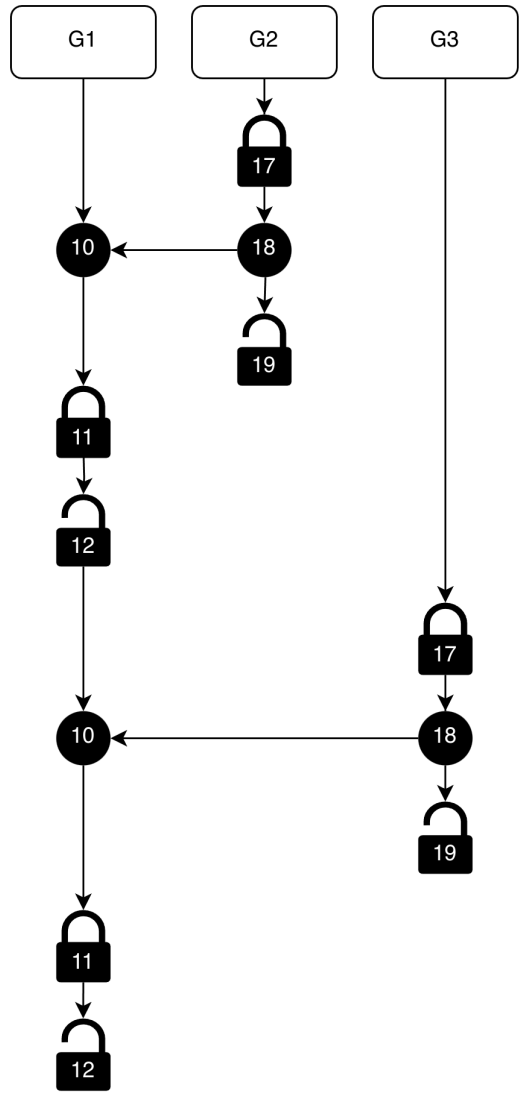
Department of Computer Science
Albert Ludwig University of Freiburg

04.06.2025

Advisor: Prof. Dr. Sulzmann
Examiner: Prof. Dr. Thiemann

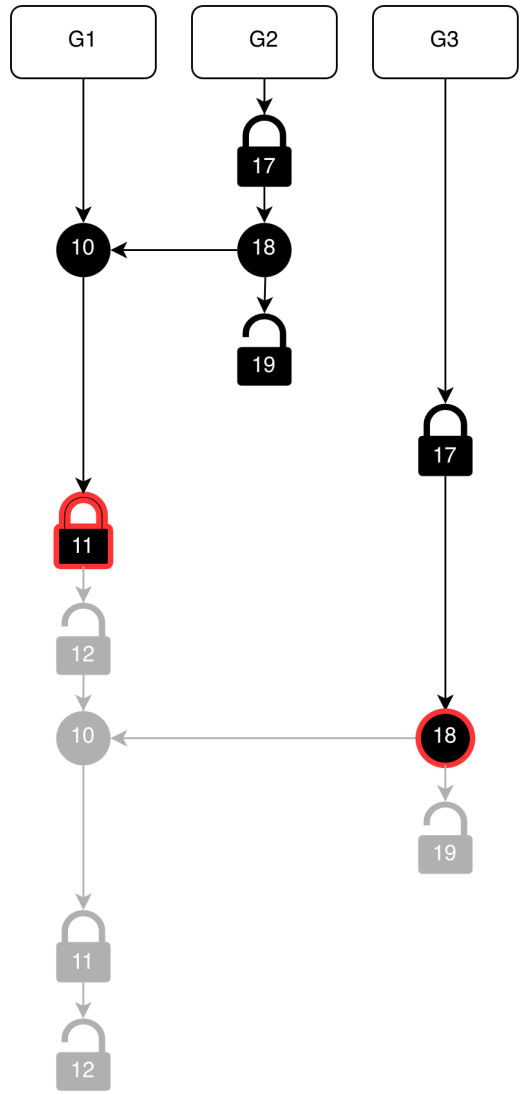
Concurrency Bugs

```go
// mixed deadlock
// simplified from Kubernetes
var (
  m = sync.Mutex{}
  c = make(chan bool)
)

func R1() {
  for i := 0; i < 2; i++ {
    <-c
    m.Lock()
    m.Unlock()
  }
}

func R2() {
  m.Lock()
  c <- true
  m.Unlock()
}

func main() {
  go R1()  // G1
  go R2()  // G2
  R2()     // G3
}
```

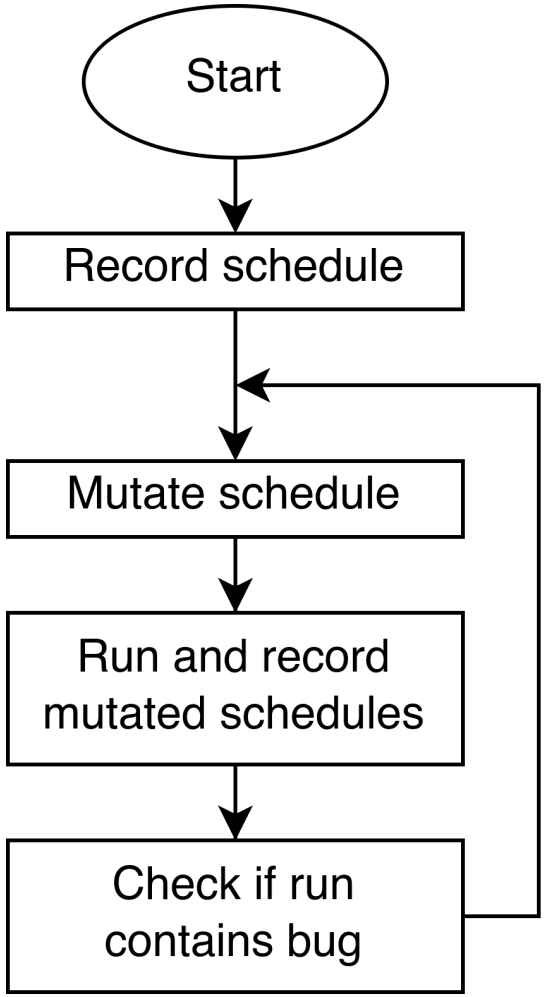


Concurrency Bugs

```go
// mixed deadlock
// simplified from Kubernetes
var (
  m = sync.Mutex{}
  c = make(chan bool)
)

func R1() {
  for i := 0; i < 2; i++ {
    <-c
    m.Lock()
    m.Unlock()
  }
}

func R2() {
  m.Lock()
  c <- true
  m.Unlock()
}

func main() {
  go R1()  // G1
  go R2()  // G2
  R2()     // G3
}
```



Fuzzing


Start
Record schedule
Mutate schedule
Run and record
mutated schedules
Check if run
contains bug

Project

- Implement/improve execution tracing of go programs
- Implement replay mechanics for go programs
- Implement simple fuzzing approaches

advocate	redraw gfuzz mut	1 hour ago
doc	simplify trace format for select	2 days ago
doc_proj	simplify trace format for select	2 days ago
examples	add achievements to Project	4 days ago
go-patch	temp fix for stuck trace write	13 hours ago
.gitignore	remove eval folder, update go-patch	last month
LICENSE	first commit	2 years ago
PROJECT.md	add pre	4 days ago
README.md	improve simple leak detection	3 weeks ago

Project

- Implement/improve execution tracing of go programs
- Implement replay mechanics for go programs
- Implement simple fuzzing approaches

advocate	redraw gfuzz mut	1 hour ago
doc	simplify trace format for select	2 days ago
doc_proj	simplify trace format for select	2 days ago
examples	add achievements to Project	4 days ago
go-patch	temp fix for stuck trace write	13 hours ago
.gitignore	remove eval folder, update go-patch	last month
LICENSE	first commit	2 years ago
PROJECT.md	add pre	4 days ago
README.md	improve simple leak detection	3 weeks ago

Project

- Implement/improve execution tracing of go programs
- Implement replay mechanics for go programs
- Implement simple fuzzing approaches

advocate	redraw gfuzz mut	1 hour ago
doc	simplify trace format for select	2 days ago
doc_proj	simplify trace format for select	2 days ago
examples	add achievements to Project	4 days ago
go-patch	temp fix for stuck trace write	13 hours ago
.gitignore	remove eval folder, update go-patch	last month
LICENSE	first commit	2 years ago
PROJECT.md	add pre	4 days ago
README.md	improve simple leak detection	3 weeks ago

Tracing

Tracing

- Record concurrency operations
  - fork, routine end
  - mutex
  - channel, select
  - once, wait group, conditional variables
  - atomic operations
- Implemented in runtime
- Routine local recording

Tracing

Timestamp

- When was operation executed
- Required to reconstruct global trace
- Implemented as atomic.Int32

Tracing

Elements

```go
func (m *Mutex) Lock() {  // simplified
  advocateIndex := runtime.AdvocateMutexPre(m.id, runtime.OperationMutexLock)

  m.mu.Lock()

  runtime.AdvocateMutexPost(advocateIndex, true)
}
```

```go
type g struct { // simplified
  ...  // routine data, e.g. stack parameter

  advocateRoutineInfo *AdvocateRoutine
}
```

Replay

Replay

- Execute program schedule based on trace
- Wait and release mechanism
- Keep track of next operation to be executed
- Operations wait until it is there turn

Replay

Key

```go
func BuildReplayKey(routine int, file string, line int) string {
  return intToString(routine) + ":" + file + ":" + intToString(line)
}
```

```go
c := make(chan int, 2)

for i := range 2 {
  c <- 1
}

a := func() {
  <-c
}

go a()
go a()
```

- Note: Inlining has been deactivated

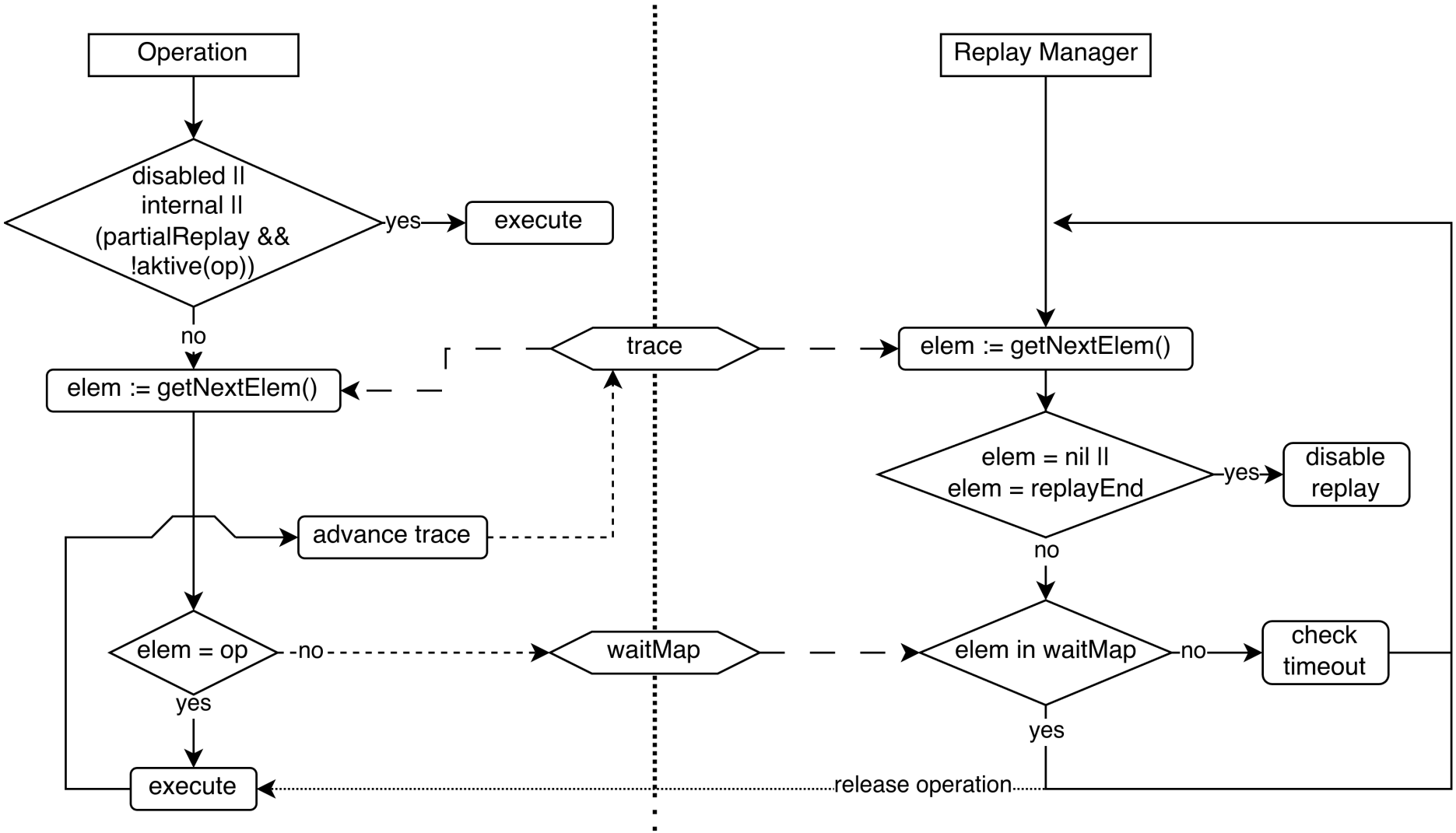
Replay

In Operation

```
1  func (m *Mutex) Lock() {  // simplified
2    active, chWait, chAck := runtime.WaitForReplay(2, true)
3
4    if active {
5      defer func() { chAck <- struct{}{ }}()
6      replayElem := <-chWait
7    }
8
9    ...
10
```

Replay

Replay



Replay

Select

```
select {
case c <- 1:
case <- d:
case <- e:
case f <- 1:
}
```

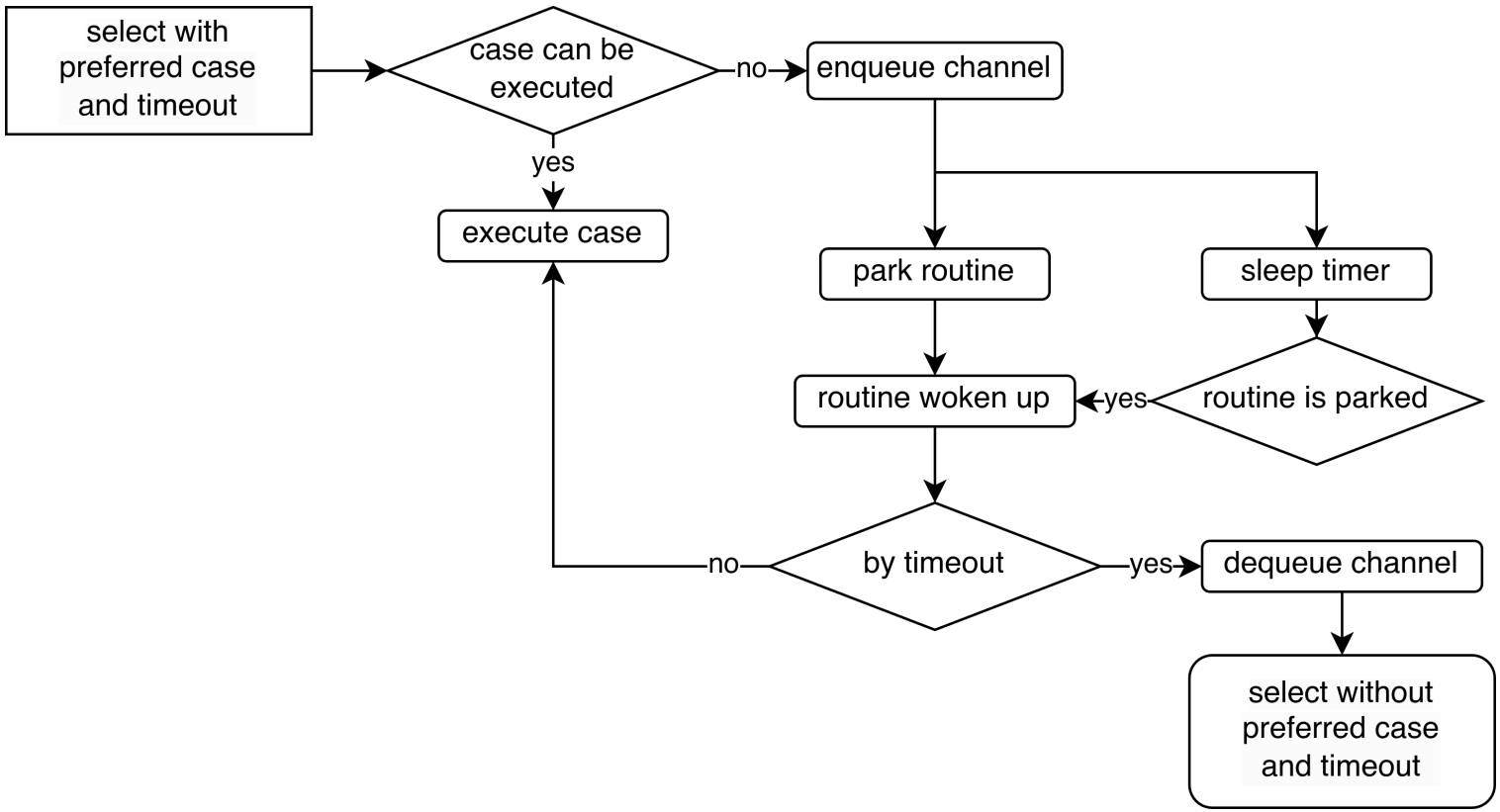
```
scases = [
  {c, send},
  {f, send},
  {e, recv},
  {d, recv}
]

selectedIndex = 2
```

```
// simplified
for _, casi := range pollorder {
  cas = &scases[casi]  // *scase
  c = cas.c            // *hchan

  if casi != selectedIndex {
    continue
  }

  if casi >= nsends {
    ...                // enqueue as recv
  } else {
    ...                // enqueue as send
  }

  ...
}
```

Replay

Select with timeout

- First try only preferred case
- On timeout, if no partner is available, try all
- Required for some fuzzing approaches



Fuzzing

Fuzzing

GFuzz I

- Based on work by Lio et al. [1]
- Reorder concurrent communication with selects

[1] Z. Liu et al., "Who goes first? detecting go concurrency bugs via message reordering"

Fuzzing

GFuzz: Example

```go
func main() {
  c := make(chan int, 2)
  d := make(chan int, 2)
  e := make(chan int, 1)

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

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

  select {
  case <-c:
  case <-d:
    close(e)
  }

  select {
  case <-c:
  case <-d:
    e <- 1
  }
}
```

Fuzzing

GFuzz II

- For each select, get executed case
- Mutate preferred cases
- Replay select with timeout
- No replay of operation order

Fuzzing

GoPie

- Based on work by Jiang et al. [2]
- Improvement of GFuzz
- Mutate order of channel and mutex
- Implement original and improved version

[2] Z. Jiang et al., "Effective Concurrency Testing for Go via Directional Primitive-Constrained Interleaving Exploration"

Fuzzing

GoPie: Mutation and Replay

- Mutate order of concurrency operations
- Construct relations between operations
- Rule based mutation on subset (scheduling chains) of operations
  - Abridge
  - Flip
  - Substitute
  - Augment
- Partial replay of subset

Fuzzing

GoPie+: Improvements I

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

  go func() {
    c <- true
  }

  go func() {
    c <- false
  }

  if res := <- c; res {
    // code that contains mutated section
    ...
  }
}
```

Fuzzing

GoPie+: Improvements II

- full replay before mutated operations to guaranty that the program reaches the modified code block
- full coverage of all concurrency primitives
- consider scheduling chains resulting from mutated program runs
- mutate partially executed operations
- exclude superfluous mutations

- Apply GoPie to GoBench [3]
- Number of found bugs:

Paper	GoPie	GoPie+
66	66	69

[3] T. Yuan et al., "GoBench: A Benchmark Suite of Real-World Go Concurrency Bugs"

Summary

- Recording
- Replay
- Fuzzing
  - GFuzz
  - GoPie / GoPie+

Sources

[1] Z. Liu, S. Xia, Y. Liang, L. Song, and H. Hu, "Who goes first? detecting go concurrency bugs via message reordering," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22, Lausanne, Switzerland: Association for Computing Machinery, 2022, pp. 888–902, ISBN: 9781450392051. DOI: 10.1145/3503222.3507753. [Online]. Available: https://doi.org/10.1145/3503222.3507753.

[2] Z. Jiang, M. Wen, Y. Yang, C. Peng, P. Yang, and H. Jin, "Effective concurrency testing for go via directional primitive-constrained interleaving exploration," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '23, Echternach, Luxembourg: IEEE Press, 2024, pp. 1364–1376, ISBN: 9798350329964. DOI: 10.1109/ASE56229.2023.00086. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00086.

[3] T. Yuan, G. Li, J. Lu, C. Liu, L. Li, and J. Xue, "Gobench: A benchmark suite of real-world go concurrency bugs," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 187–199. DOI: 10.1109/CGO51591.2021.9370317.

Appendix

Atomic Old

```
1  // doc.go
2  func AddInt32(addr *int32, delta int32) (new int32)
3
4  // type.go
5  func (x *Int32) Add(delta int32) (new int32) {
6    return AddInt32(&x.v, delta)
7  }
8
9  // asm.s
10 TEXT ·AddInt32(SB),NOSPLIT,$0
11   JMP internal/runtime/atomic·Xadd(SB)
```

Appendix

Atomic New

```go
// advocate_atomic.go
func AddInt32(addr *int32, delta int32) (new int32) { // simplified
  runtime.AdvocateAtomic(addr, runtime.OperationAtomicAdd, 2)
  return AddInt32Advocate(addr, delta)
}

// type.go
func (x *Int32) Add(delta int32) (new int32) {
  return AddInt32AdvocateType(&x.v, delta)
}

// advocate_atomic_type.go
func AddInt32AdvocateType(addr *int32, delta int32) (new int32) { // simplified
  runtime.AdvocateAtomic(addr, runtime.OperationAtomicAdd, 3)
  return AddInt32Advocate(addr, delta)
}

// doc.go
func AddInt32Advocate(addr *int32, delta int32) (new int32)

// asm.s
TEXT ·AddInt32Advocate(SB),NOSPLIT,$0
  JMP internal/runtime/atomic·Xadd(SB)
```

Tracing: Routine Local Tracing

```go
type g struct { // simplified
  ...  // routine data, e.g. stack parameter

  advocateRoutineInfo *AdvocateRoutine
}
```

```go
type AdvocateRoutine struct {
  id          uint64
  Trace       []traceElement
  G           *g
  replayID    int
  maxObjectId uint64
}
```

Appendix

Tracing: Elements

```go
type Mutex struct {
  _ noCopy

  mu isync.Mutex

  id uint64 // id for the recording
}

func (m *Mutex) Lock() {  // simplified
  if m.id == 0 {
    m.id = runtime.GetAdvocateObjectID()
  }

  advocateIndex := runtime.AdvocateMutexPre(m.id, runtime.OperationMutexLock)

  m.mu.Lock()

  runtime.AdvocateMutexPost(advocateIndex, true)
}
```

Algorithm 1: ReplayManager

```
1  while true do
2      if waitForAck ≠ nil then
3          waitAck();
4          nextElem();
5      nElem ← nextElemInTrace();
6      if nElem == nil || nElem == ReplayEnd then
7          return
8      if elem, ok := waitMap[nElem.key]; ok then
9          elem.chWait.send(nextElem);
10         if elem.waitForAck then
11             waitAck();
12         nextElem();
13     if len(waitMap) > 0 && timeSinceLastRelease > timeoutTime then
14         timeout();
```

Appendix: Replay: WaitForReplay

Algorithm 2: WaitForReplay(op)

```
1  if replayDisabled || ignored then
2  |   return false, nil, nil;
3  chWait ← make(chan ReplayElem, 1);
4  chAck ← make(chan struct, 1);
5  elem ← nextElemInTrace();
6  if match(op, elem) then
7  |   chWait.send(elem);
8  |   if op.WaitForAck then
9  |   |   waitForAck ← {op, chAck};
10 |   else
11 |   |   nextElem();
12 else
13 |   waitMap[op.key] ← {op, chWait, chAck};
14 return true, chWait, chAck;
```

Replay

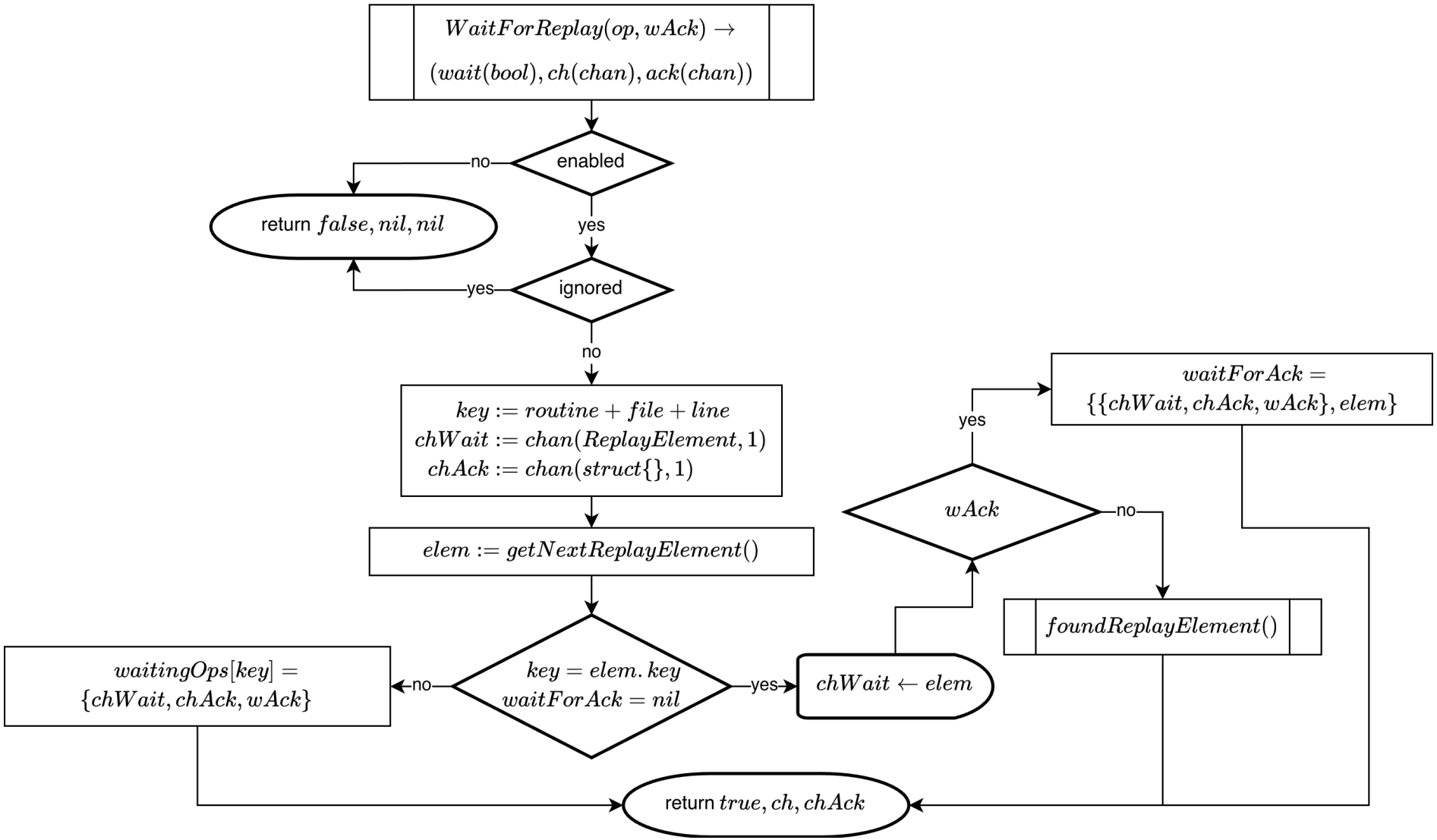
Key

```go
func BuildReplayKey(routine int, file string, line int) string {
  return intToString(routine) + ":" + file + ":" + intToString(line)
}
```

```go
c := make(chan int, 2)

for i := range 2 {
  c <- 1
}

a := func() {
  <-c
}

go a()
go a()
```

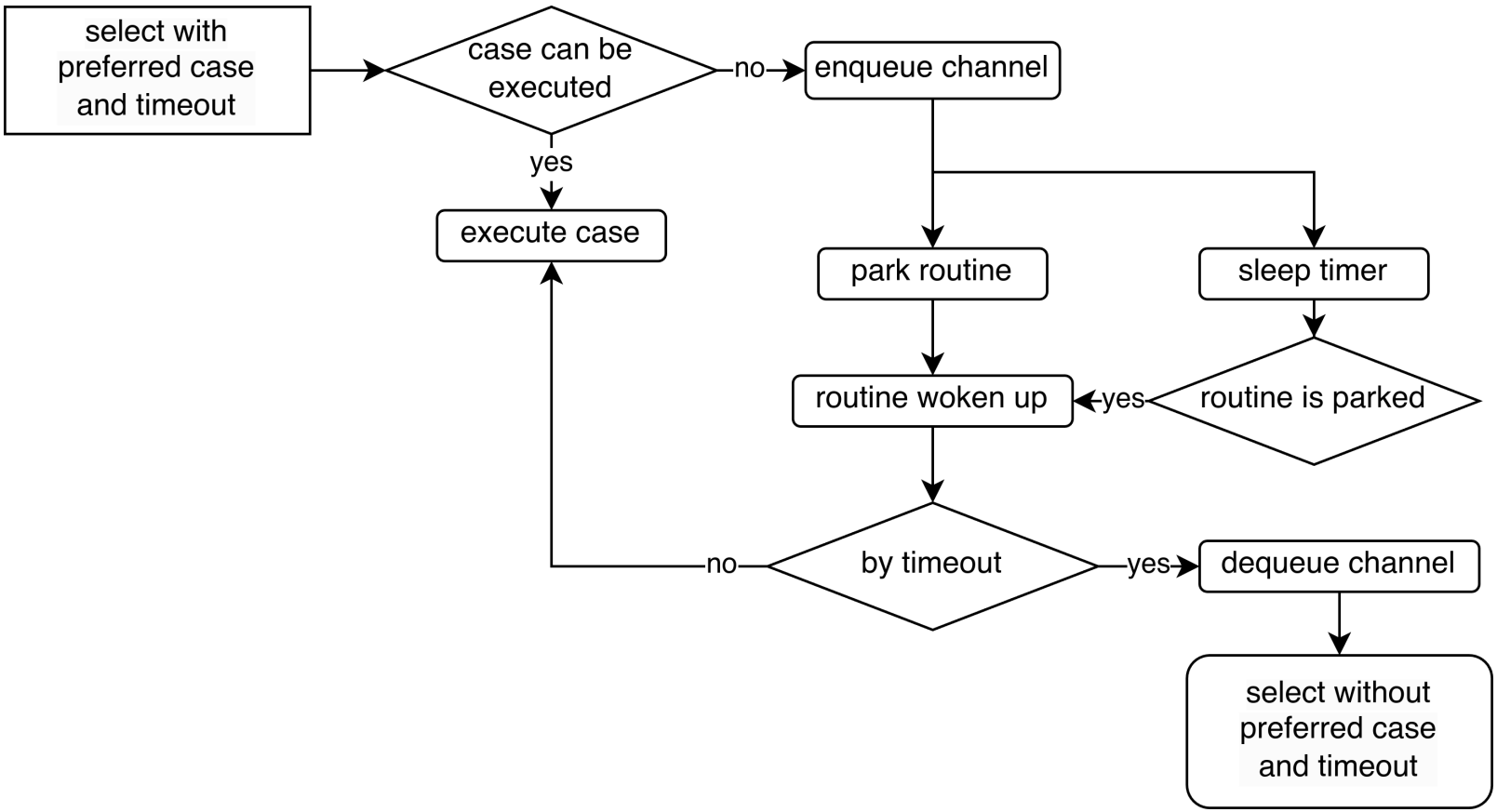
- Note: Inlining has been deactivated

Appendix: Replay: In Operation II



Replay

Select with timeout



Appendix

Select Timeout I

```go
func selectgo(...) (int, bool) {  // simplified
  fuzzingEnabled, fuzzingIndex, timeout := AdvocateFuzzingGetPreferredCase(2)

  if fuzzingEnabled {
    if ok, i, b, _ := selectWithPrefCase(..., timeout); ok {
      return i, b
    }
  }

  return originalSelect(...)
}
```

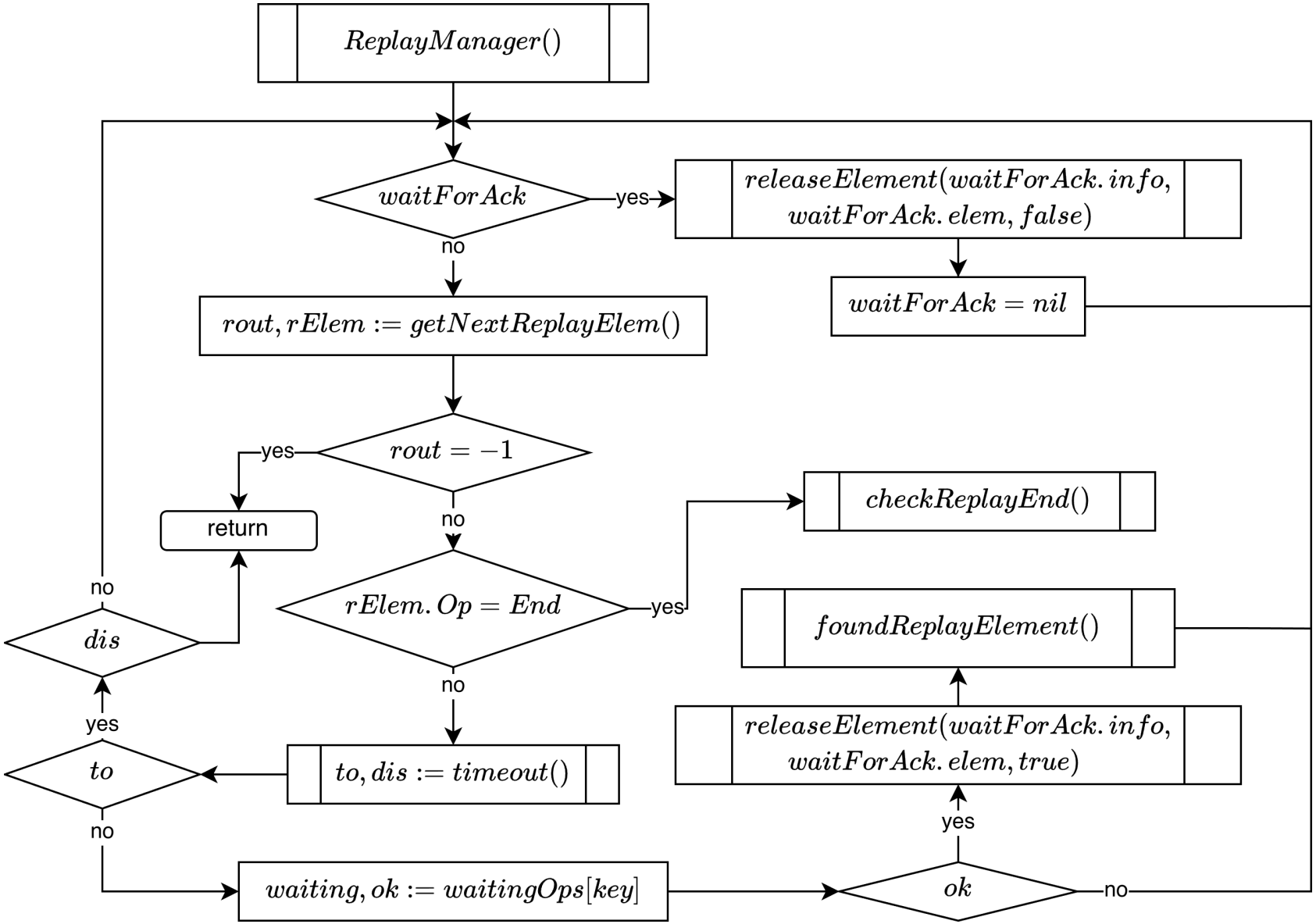
Appendix

Select Timeout II

```
1  func goparkWithTimeout(..., timeout int64) {  // simplified
2    mp := acquirem()
3    gp := mp.curg
4
5    // Setup timer if timeout is non-zero
6    if timeout > 0 {
7      go func() {
8        sleep(float64(timeout))
9        if readgstatus(gp) == _Gwaiting {
10         goready(gp, traceskip)
11       }
12     }()
13   }
14
15   // Run the original gopark logic
16   ...
```

Appendix

Replay: Replay Manager



Appendix

Replay: Release

```
1  // simplified
2  func releaseElement(elem replayChan, elemReplay ReplayElement, rel bool) bool {
3    if rel {
4      elem.chWait <- elemReplay
5      elem.released = true
6    }
7
8    if elem.waitAck {
9      select {
10     case <-elem.chAck:
11     case <-after(acknowledgementMaxWait):
12       ...  // store info that timeout
13     }
14   }
15
16   ...  // some other controls
17 }
```

Appendix

Replay: Timeout

- Program may get stuck
  - Something went wrong
  - E.g. error in trace rewrite, random element in program
- Only one element may be the problem
- Try to fix by
  - Release oldest waiting element
  - Skip one element in trace
- If it cannot be fixed
  - disable replay

Appendix

Fuzzing: Select with preferred case

```
1  func selectgo(...) (int, bool) {  // simplified
2    fuzzingEnabled, fuzzingIndex, timeout := AdvocateFuzzingGetPreferredCase(2)
3
4    if fuzzingEnabled {
5      if ok, i, b, _ := selectWithPrefCase(..., timeout); ok {
6        return i, b
7      }
8    }
9
10   return originalSelect(...)
11 }
```

Appendix

Fuzzing: Gopark with timeout

```
1  func goparkWithTimeout(..., timeout int64) {  // simplified
2    mp := acquirem()
3    gp := mp.curg
4
5    if timeout > 0 {
6      go func() {
7        sleep(float64(timeout))
8        if readgstatus(gp) == _Gwaiting {
9          goready(gp, traceskip)
10       }
11     }()
12   }
13
14   ...
```

Appendix

GoPie: SC and Relations

- Mutate order of concurrency operations
- Scheduling chain:

$$SC = \{\langle o_1, r_1\rangle, \ldots, \langle o_n, r_n\rangle \mid r_i \neq r_{i+1}, 1 \leq i \leq n-1\}$$

- Relations:
  - Rule 1: $c' \in Rel_1(c)$ if c and c' neighbors in same routine
  - Rule 2: $c' \in Rel_2(c)$ if c and c' neighbors in different routines but on same primitive
  - Rule 3: $c' \in Rel_1(c), c'' \in Rel_2(c') \rightarrow c'' \in Rel_2(c)$
  - Rule 4: $c' \in Rel_2(c), c'' \in Rel_2(c') \rightarrow c'' \in Rel_2(c)$

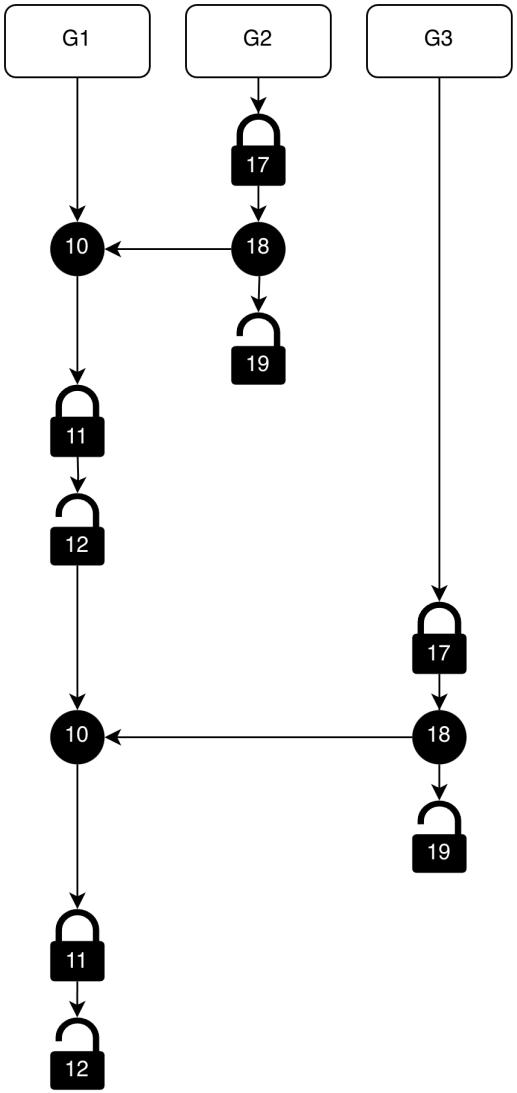
Appendix

GoPie: Mutation

- Mutate scheduling chains
  - Abridge
    - $\{o_i, o_{i+1}, \ldots, o_{j-1}, o_j\} \Rightarrow \{o_{i+1}, \ldots, o_{j-1}, o_j\}, \{o_i, o_{i+1}, \ldots, o_{j-1}\}$
  - Flip (in paper, not in implementation)
    - $\{\ldots, o_i, o_j, \ldots\} \Rightarrow \{\ldots, o_j, o_i, \ldots\}$
  - Substitute
    - $o_j \in Rel_1(o_i), \{\ldots, o_i, \ldots\} \Rightarrow \{\ldots, o_j, \ldots\}$
  - Augment
    - $o_j \in Rel_2(o_i), \{\ldots, o_i\} \Rightarrow \{\ldots, o_i, o_j\}$

Appendix

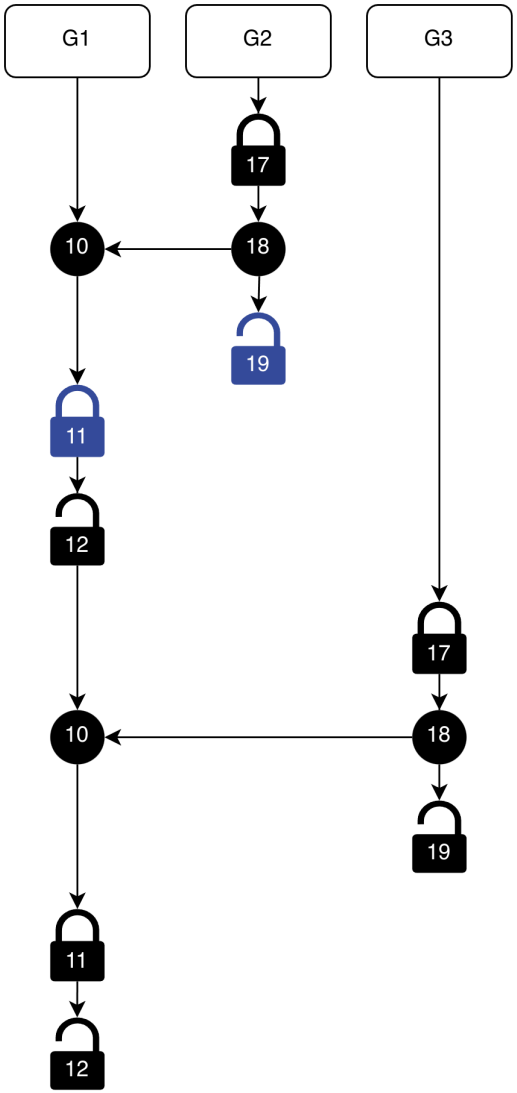
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain

Appendix

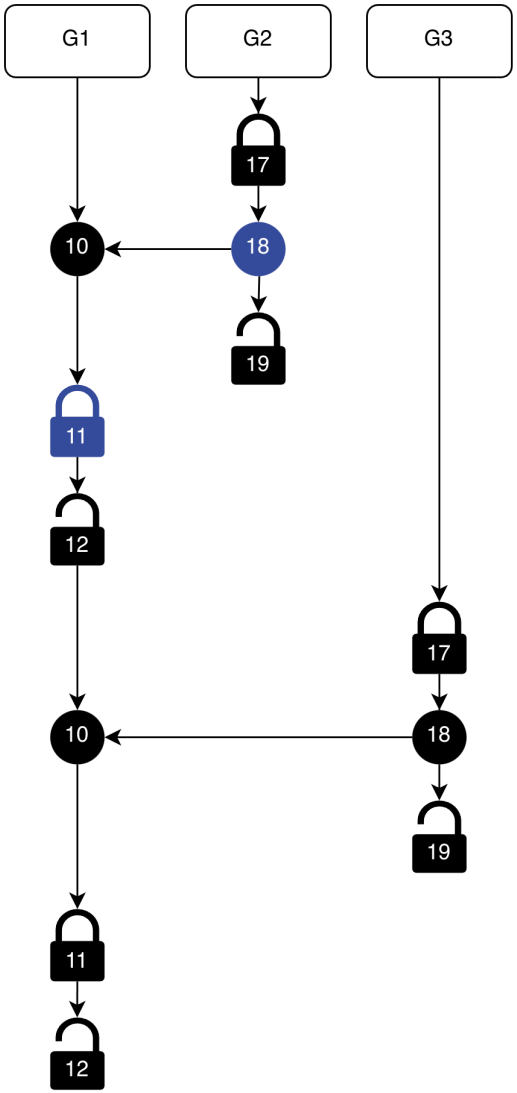
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$

Appendix

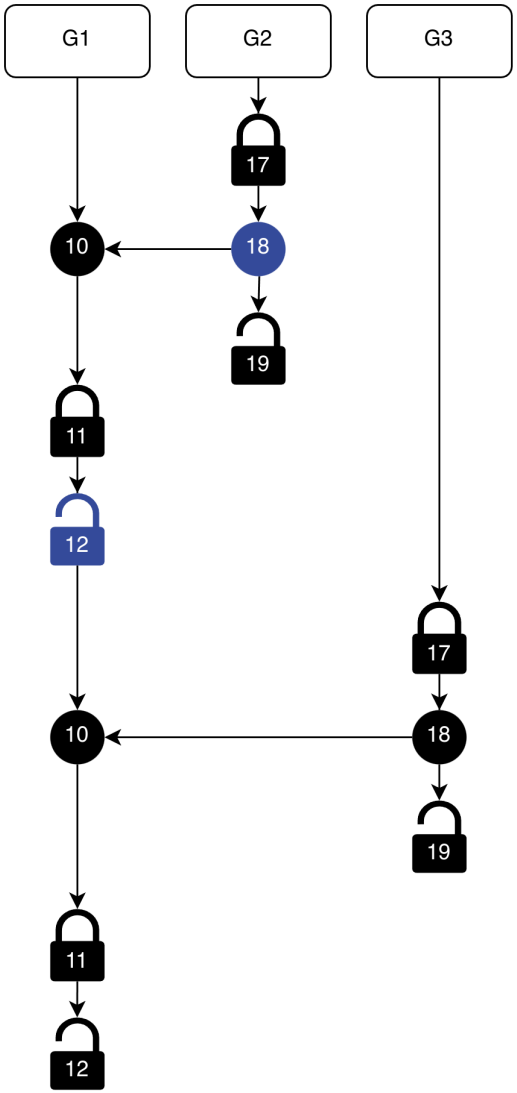
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$

Appendix

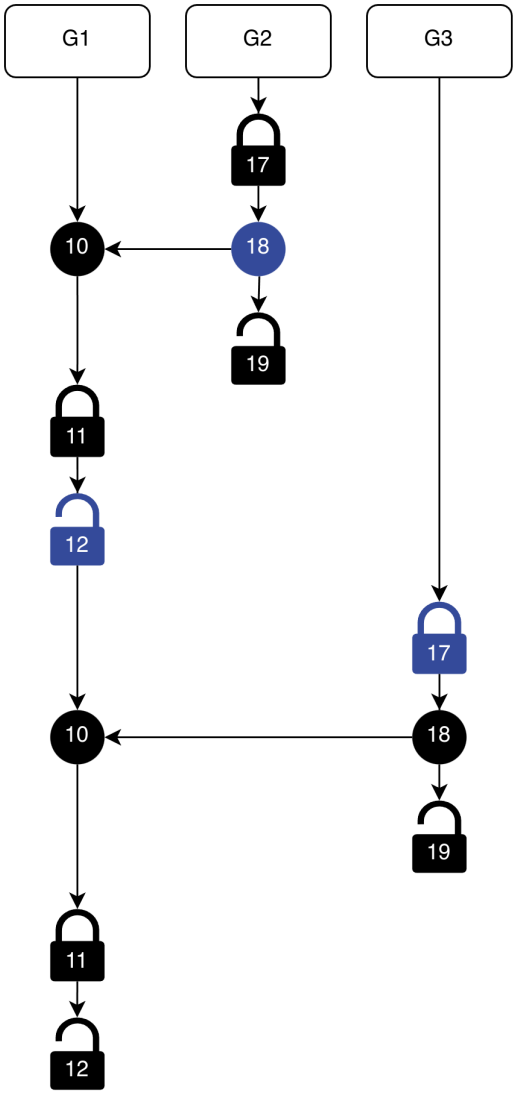
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$
Step 2	Substitute	$\{\langle G2, 18\rangle, \langle G1, 12\rangle\}$

Appendix

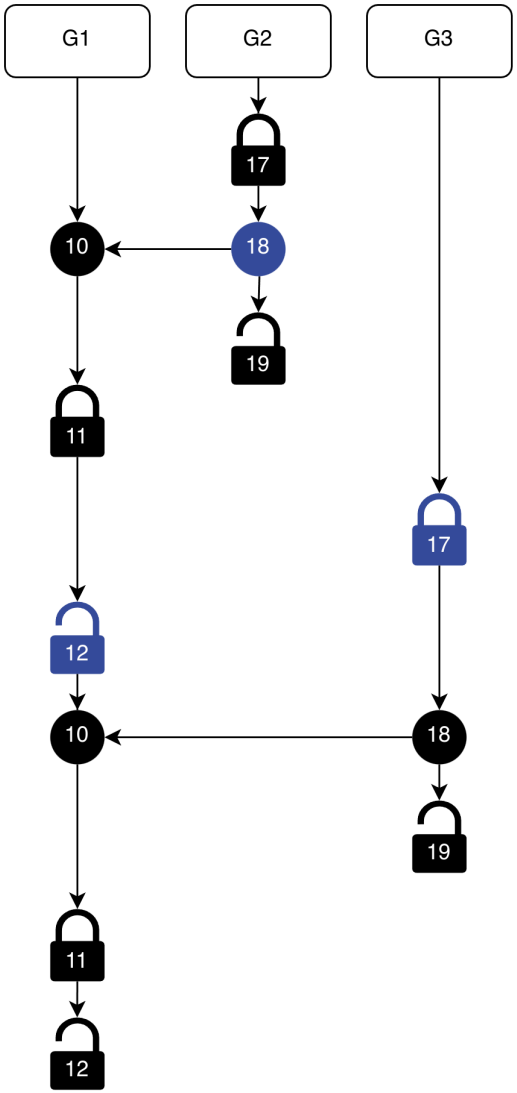
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$
Step 2	Substitute	$\{\langle G2, 18\rangle, \langle G1, 12\rangle\}$
Step 3	Augment	$\{\langle G2, 18\rangle, \langle G1, 12\rangle, \langle G3, 17\rangle\}$

Appendix

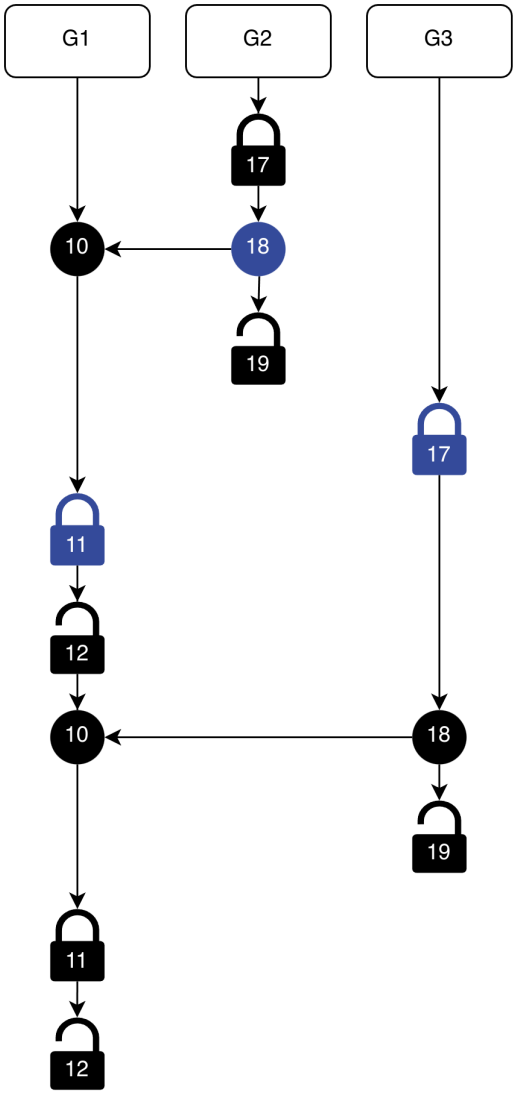
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$
Step 2	Substitute	$\{\langle G2, 18\rangle, \langle G1, 12\rangle\}$
Step 3	Augment	$\{\langle G2, 18\rangle, \langle G1, 12\rangle, \langle G3, 17\rangle\}$
Step 4	Flip	$\{\langle G2, 18\rangle, \langle G3, 17\rangle, \langle G1, 12\rangle\}$

Appendix

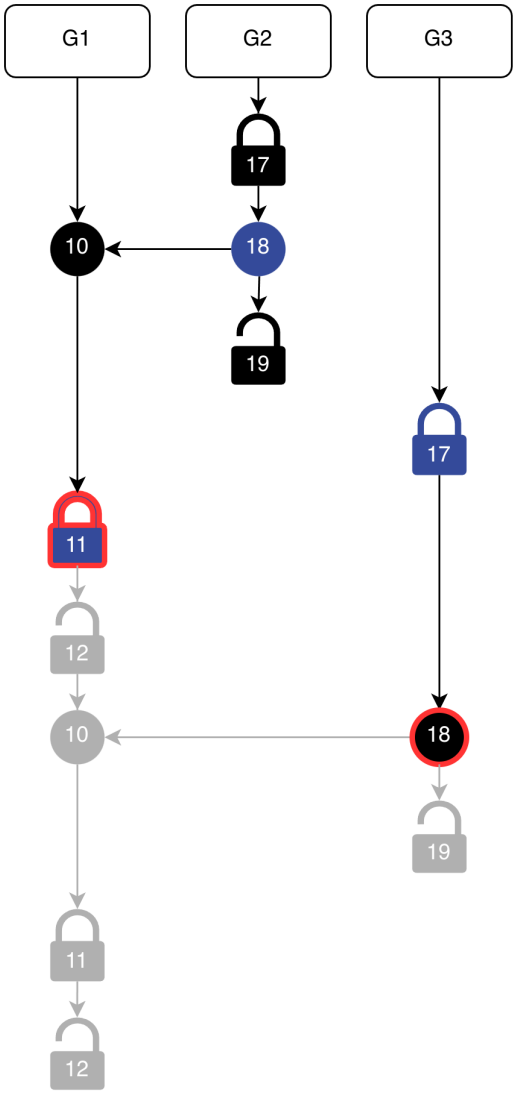
Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$
Step 2	Substitute	$\{\langle G2, 18\rangle, \langle G1, 12\rangle\}$
Step 3	Augment	$\{\langle G2, 18\rangle, \langle G1, 12\rangle, \langle G3, 17\rangle\}$
Step 4	Flip	$\{\langle G2, 18\rangle, \langle G3, 17\rangle, \langle G1, 12\rangle\}$
Step 5	Substitute	$\{\langle G2, 18\rangle, \langle G3, 17\rangle, \langle G1, 11\rangle\}$

Appendix

Fuzzing: GoPie: Example



Step	Mutation	Scheduling chain
Init	-	$\{\langle G2, 19\rangle, \langle G1, 11\rangle\}$
Step 1	Substitute	$\{\langle G2, 18\rangle, \langle G1, 11\rangle\}$
Step 2	Substitute	$\{\langle G2, 18\rangle, \langle G1, 12\rangle\}$
Step 3	Augment	$\{\langle G2, 18\rangle, \langle G1, 12\rangle, \langle G3, 17\rangle\}$
Step 4	Flip	$\{\langle G2, 18\rangle, \langle G3, 17\rangle, \langle G1, 12\rangle\}$
Step 5	Substitute	$\{\langle G2, 18\rangle, \langle G3, 17\rangle, \langle G1, 11\rangle\}$