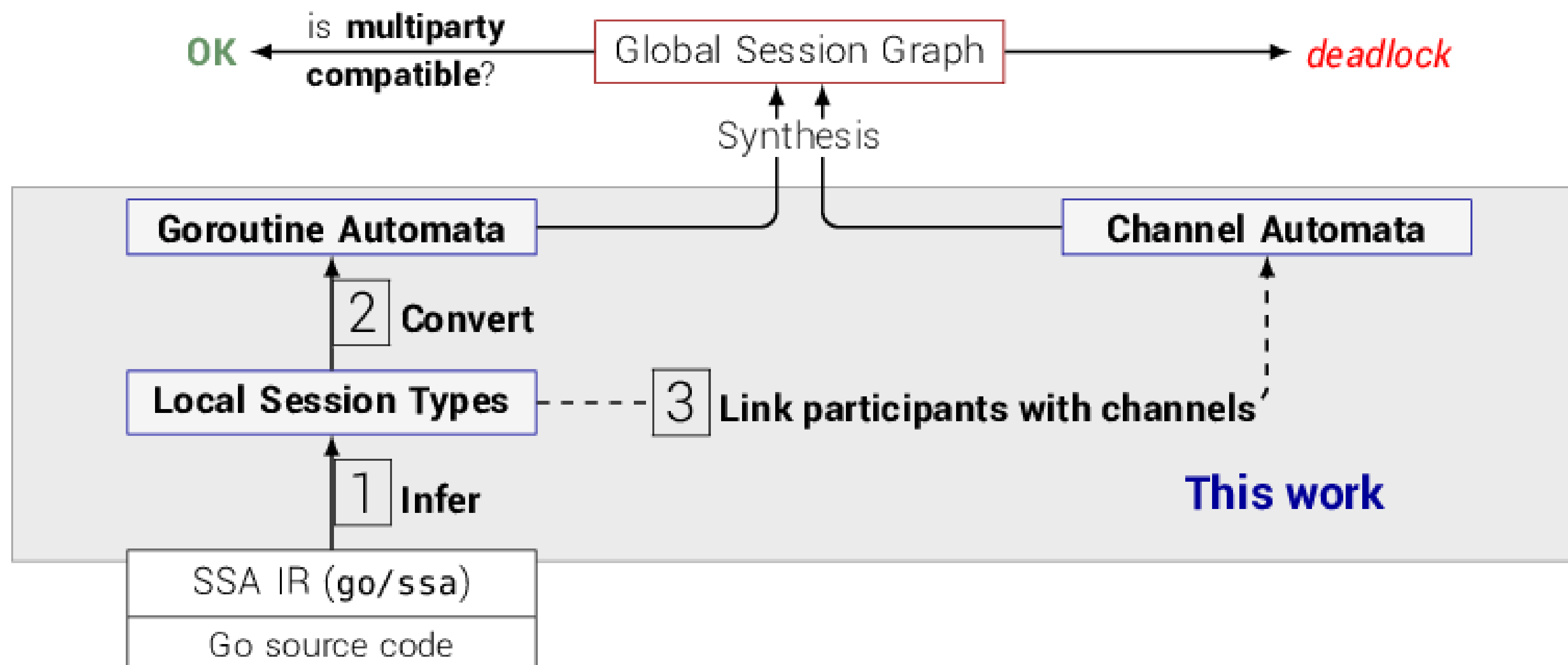
Static Deadlock Detection for Go by Global Session Graph Synthesis

Nicholas Ng & Nobuko Yoshida

Department of Computing
Imperial College London

Contributions

- Static deadlock detection tool *dingo-hunter*
- Deadlock detection based on session types
- **Infer** session types as Communicating Automata
- **Synthesise** global session graphs from CA



Go and Concurrency

- Developed by Google for multi-core programming
- Concurrency model built on CSP (process calculi)
- Message-passing **communication** over channels

" *Do not communicate by sharing memory; instead, share memory by communicating.* "
-- Effective Go (developer guide)

Go concurrency: Goroutines and Channels

- Goroutines: lightweight threads
- Channels: bounded queues (default size 0)
- Go concurrency = compose goroutines & coordinate with channels

```
func deepThought(replyCh chan int) {
    time.Sleep(75 * time.Millisecond)
    replyCh <- 42
}

func main() {
    ch := make(chan int)
    go deepThought(ch)
    answer := <-ch
    fmt.Printf("The answer is %d\n", answer)
}
```

Run

Concurrency Problems

- Deadlocks ☹
- *Some* goroutines are blocked waiting forever

```
func deepThought(replyCh chan int) {
    time.Sleep(75 * time.Millisecond)
    replyCh <- 42
}

func main() {
    ch := make(chan int)
    go deepThought(ch)
    answer := <-ch
    fmt.Printf("The answer is %d\n", answer)
}
```

Run

fatal error: all goroutines are asleep - deadlock!

- Go has a *runtime* deadlock detector (http://github.com/golang/go/blob/d2c81ad84776edfb4c790666f1d80554b4393d46/src/runtime/proc.go#L3243)
- No. of running goroutines: if `run > 0` ⟶ OK ✓; if `run == 0` ⟶ deadlock ☹

```
// From Go source code - https://github.com/golang/go
// File runtime/proc.go


func checkdead() {
...
    // -1 for sysmon
    run := sched.mcount - sched.nmidle - sched.nmidlelocked - 1
    if run > 0 {
        return
    }
...
    getg().m.throwing = -1 // do not dump full stacks
    throw("all goroutines are asleep - deadlock!")
}
```

Runtime deadlock detection

- No false positive
- Only if deadlock during execution
- Only **global** deadlocks: if ALL goroutines are blocked

Defeating runtime deadlock detection

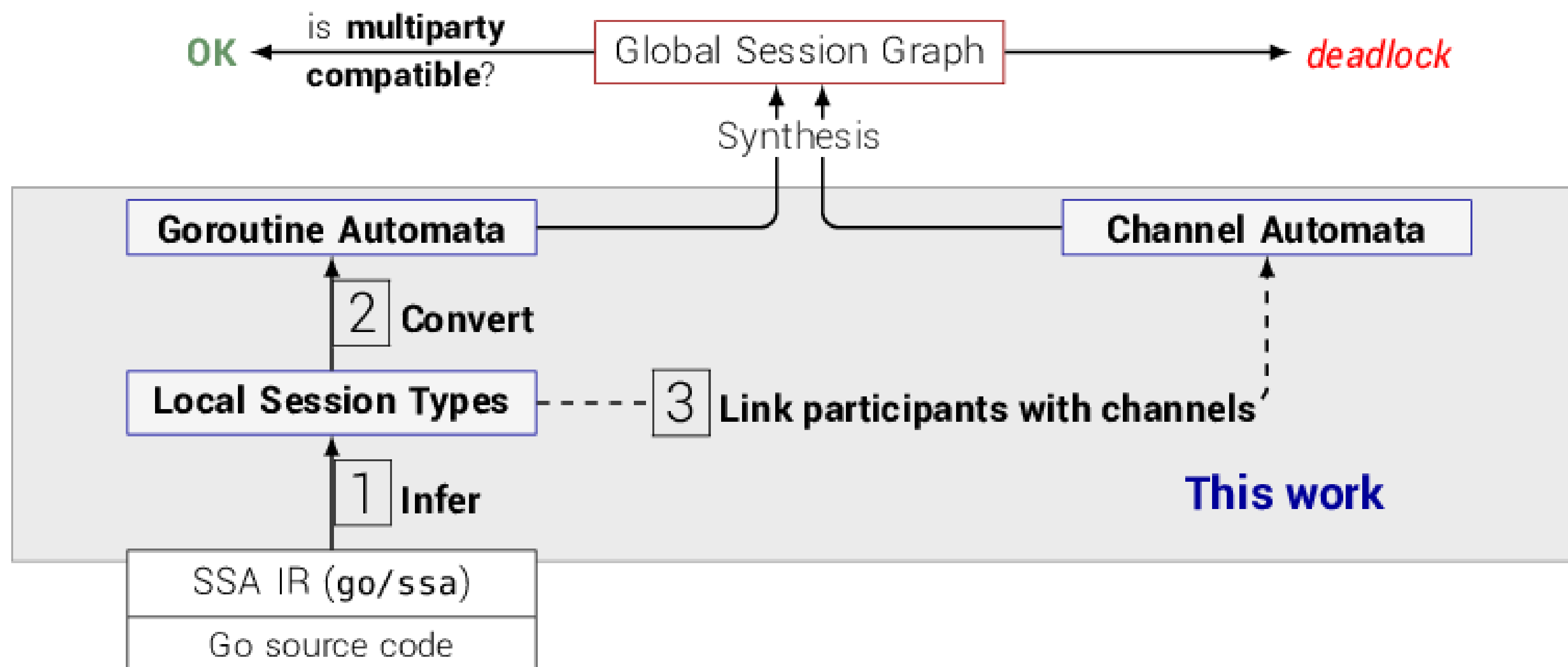
```
func Send(ch chan<- int)                         { ch <- 42 }              // Send
func RecvAck(ch <-chan int, done chan<- int) { v := <-ch; done <- v } // Recv then Send

func main() {
    ch, done := make(chan int), make(chan int)
    go Send(ch)
    go RecvAck(ch, done)
    go RecvAck(ch, done) // OOPS

    // Anonymous goroutine: Some long running work (e.g. http service)
    go func() {
        for i := 0; i < 3; i++ {
            fmt.Println("Working #", i)
            time.Sleep(1 * time.Second)
        }
    }()
    result = <-done
    result = <-done // OOPS
    fmt.Println("Result is ", result)
}
```

Run

This will be our running example

Static Analysis & Deadlock Detection by Global Graph Synthesis

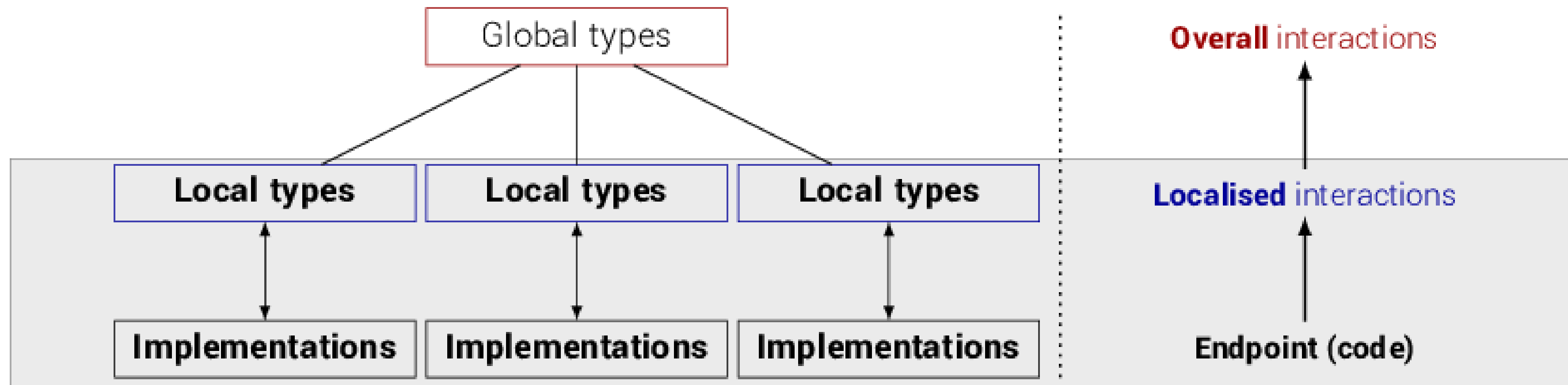
Static Analysis & Deadlock Detection by Global Graph Synthesis



- Explore all branches (vs. only execution path for runtime checker)
- Approach based on **Multiparty Session Types** (MPST) [1]
- Guarantee communication-safety & **deadlock-freedom** for n-party interactions

[1]: Honda, Yoshida, Carbone, *Multiparty Asynchronous Session Types* , POPL'08, J. ACM

The Multiparty Session Types (MPST) framework



- **Global Graph Synthesis** [2] local type(s) ⟶ global type
- Local types as *Communicating Automata* [3]
- Global types as *graphical choreographies*
- Safety guaranteed by **multiparty compatibility** property (global)

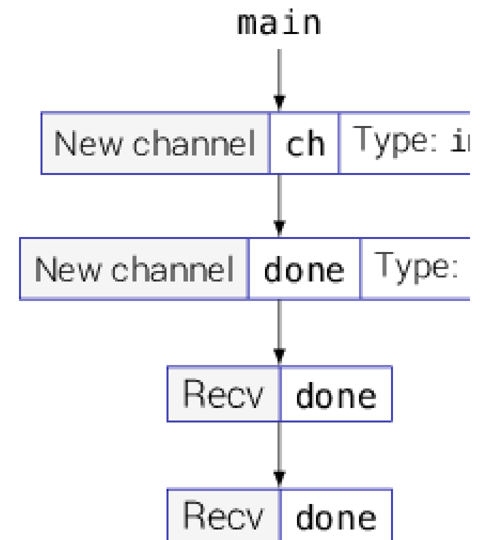
[2]: Lange, Tuosto, Yoshida, *From Communicating Machines to Graphical Choreographies* , POPL'15

[3]: Brand, Zafiropulo, *On communicating finite-state machines* , J. ACM Vol. 30 No. 2, 1983

Type inference from Go code

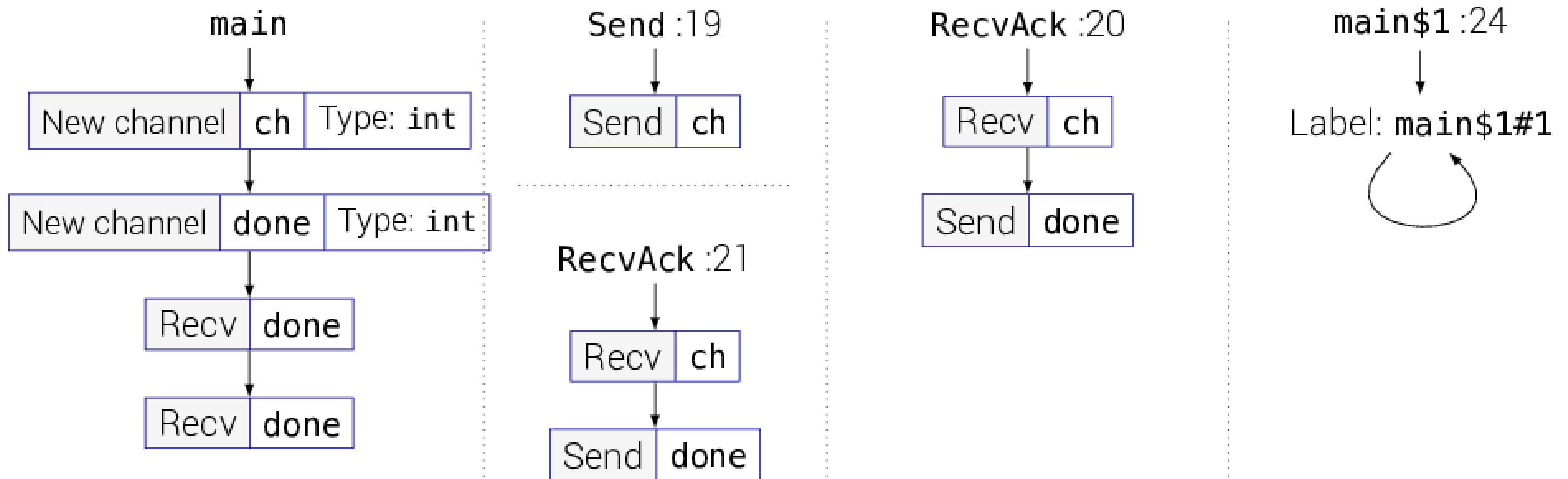
- Control flow graph as Finite State Machine (FSM), per goroutine
- + Communication primitives: `make(chan T)`, send, receive

```
func main() {
    ch, done := make(chan int), make(chan int)
    go Send(ch)
    go RecvAck(ch, done)
    go RecvAck(ch, done) // OOPS

    // Anonymous goroutine: Some long running work (e.g. http service)
    go func() {
        for i := 0; i < 3; i++ {
            fmt.Println("Working #", i); time.Sleep(1 * time.Second)
        }
    }()
    result = <-done
    result = <-done // OOPS
    fmt.Println("Result is ", result)
}
```



Inferred local session types

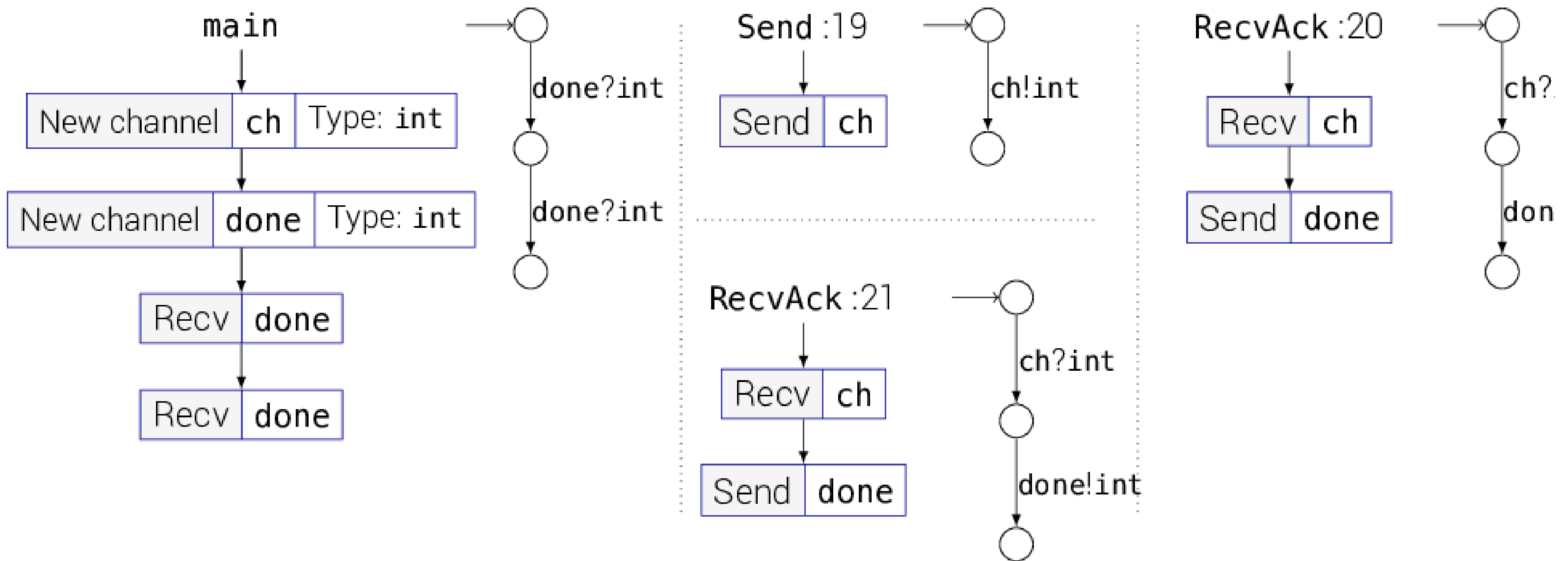
- For simplicity: real variable names
- Note: `main$1:24` (work anonymous function) has no communication



Variable names: `ch = main.t0`, `done = main.t1`

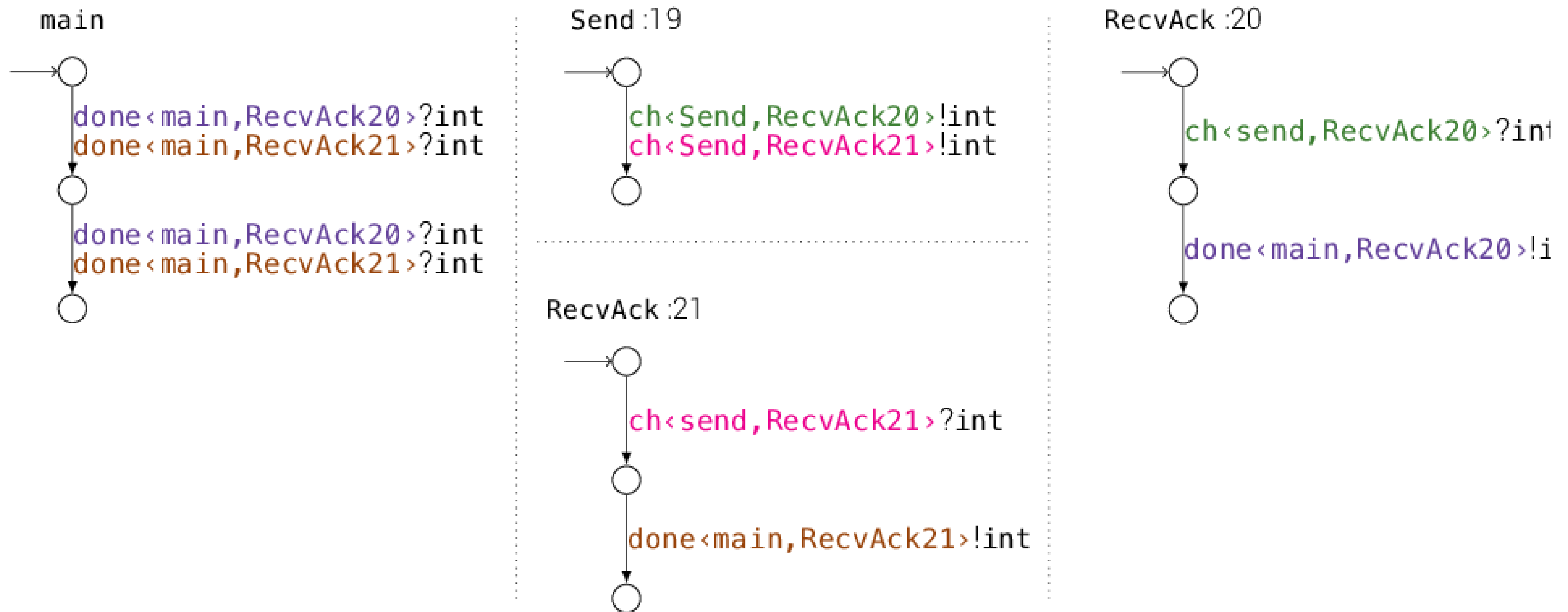
Inferred Local Types to Goroutine Automata

- Local session types ⟹ *Communicating Automata*
- **Receive** is ? transition, **Send** is ! transition



Channel Automata

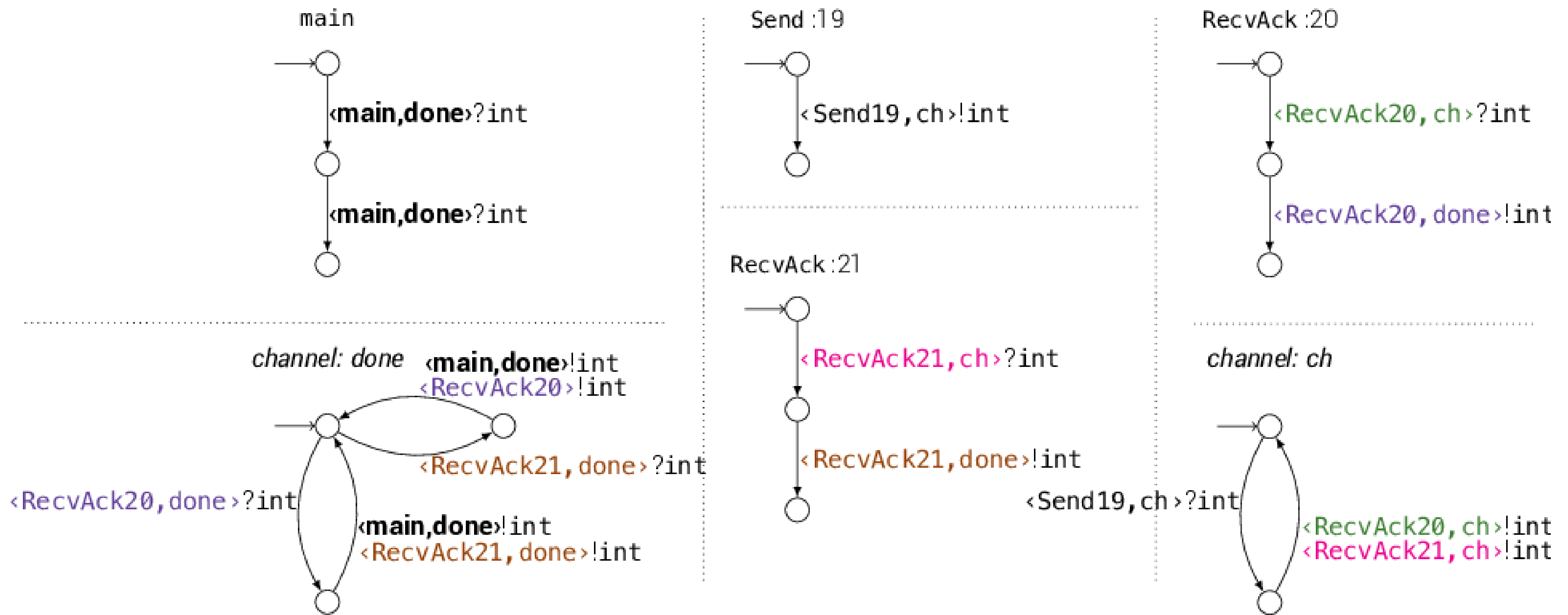
- Channels in **Communicating Automata** - *fixed* point-to-point links
- **Go channels** - *shared*, do not represent *fixed* endpoint
- 2 goroutines writing to same channel valid ✓



Note: Go channels valid regardless of the deadlock

Channel Automata

- **Channel Automata** defers selection to a machine (for each channel)
- **Unbuffered channel** q0 → receive → send → q0



Channel Automata *done* & *ch* (only essential transitions)

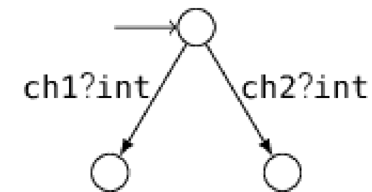
Bonus: Select non-deterministic choice

- Switch-case for communication

```go
func main() {
    ch1, ch2 := make(chan int), make(chan int)
    go calc(ch1)
    go calc(ch2)
    select {
    case ans := <-ch1:
        fmt.Println("Answer from ch1: ", ans)
    case ans := <-ch2:
        fmt.Println("Answer from ch2: ", ans)
    }
}
```

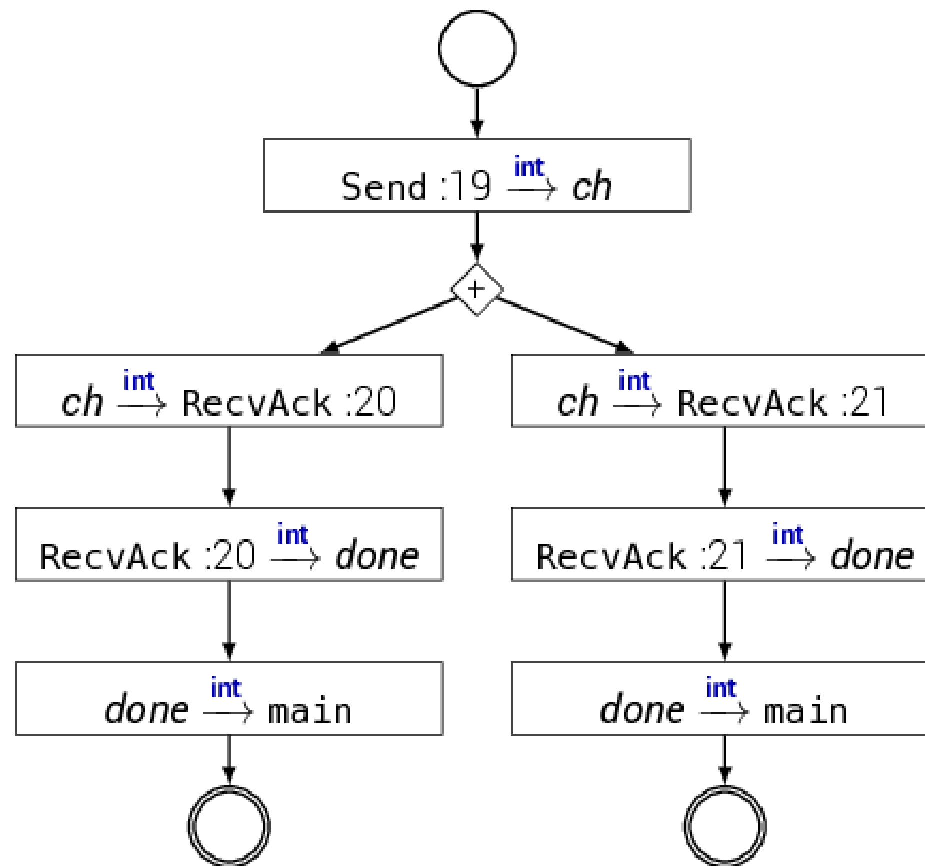
Run

Select (non-deterministic choice)



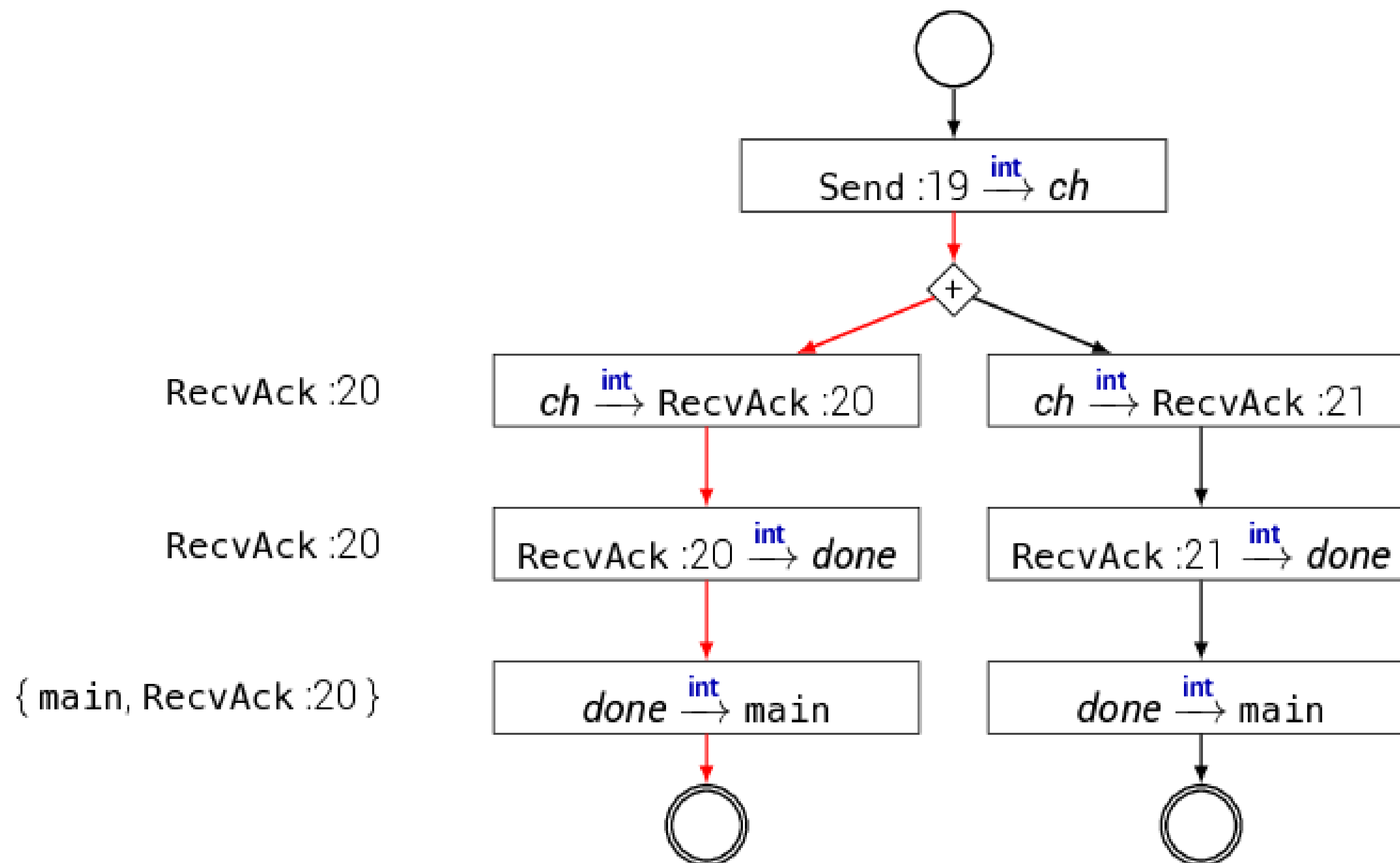
Global Graph Synthesis

- Join both automata to get overall **global** graph
- All synchronous transitions, A: `ch!int` + B: `ch?int` becomes A → B: int



Global Graph Synthesis: What is safe?

- **Multiparty Compatibility** property on global graph
- **Representability** All [Goroutine] automata are "represented" in global graph
- **Branching condition** Branches are propagated to all machines in choice

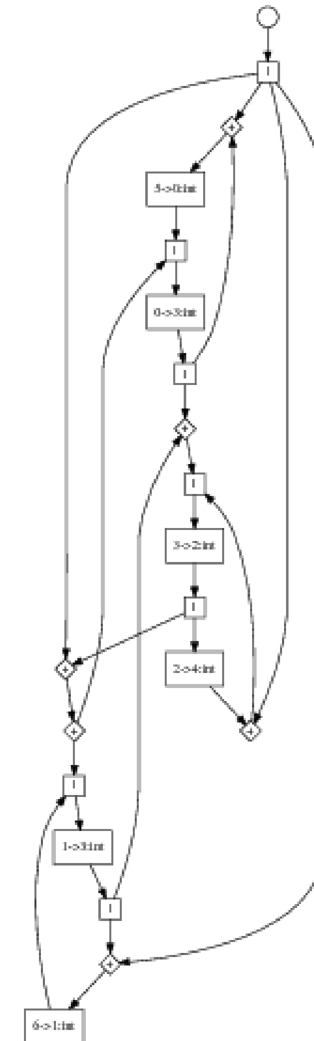


Fan-in pattern

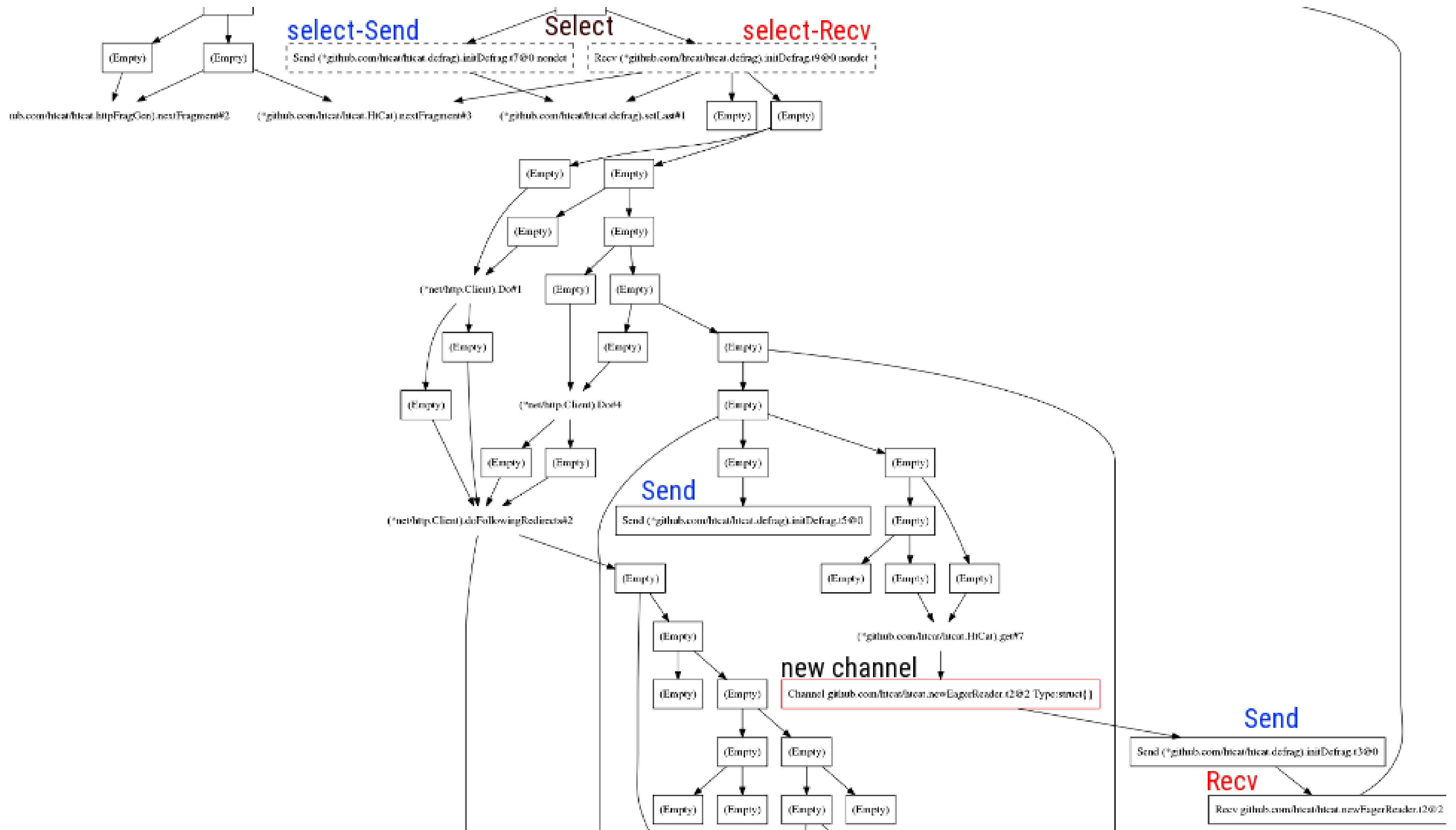
- Merging pattern with `select`

```
func fanin(input1, input2 <-chan int) <-chan int {
    c := make(chan int)
    go func() {
        for {
            select {
            case s := <-input1:
                c <- s
            case s := <-input2:
                c <- s
            }
        }
    }()
    return c
}
```

Run



Bigger example: htcat - Concurrent HTTP GETs



github.com:htcat/htcat (https://github.com/htcat/htcat) (721 LoC)

Bigger example: htcat - Concurrent HTTP GETs

- 7xx lines
- 9632 nodes
- 11 Goroutine Automata / 8 Channel Automata
- Safe ✓
- Error handling code: conditional new goroutine

```
if err != nil {
    go cat.d.cancel(err)
    return
}
```

Conclusion

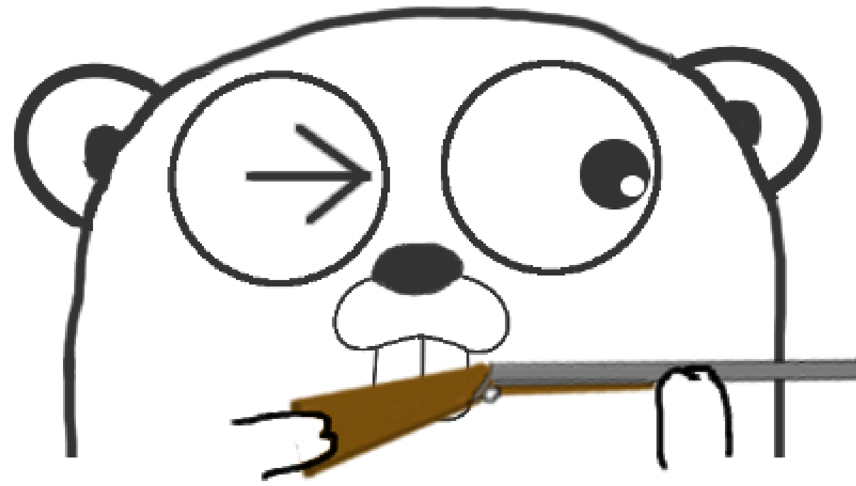
- Static analysis tool for Go
- Detect deadlock through global session graph synthesis
- Applied to open source code base

Static analysis tool

- github.com/nickng/dingo-hunter (https://github.com/nickng/dingo-hunter)

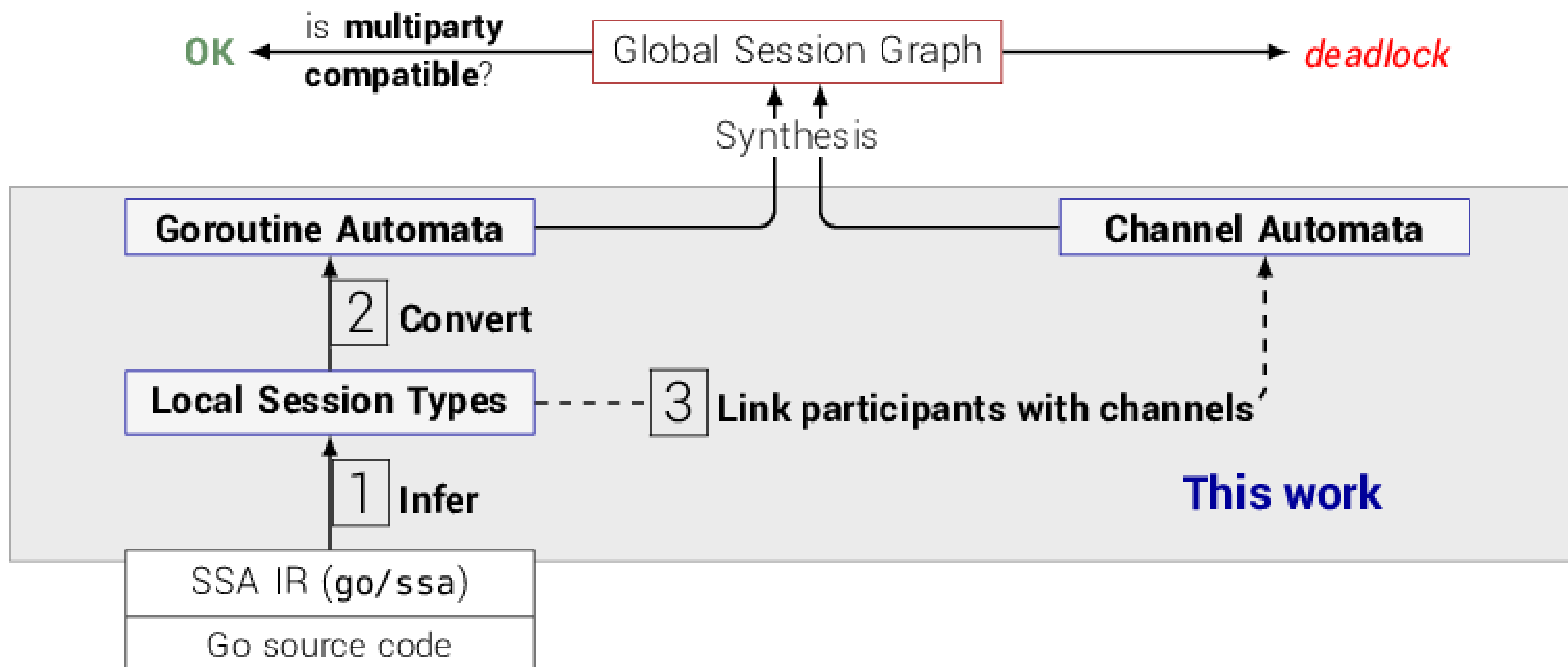
Synthesis tool (POPL'15 artifact)

- bitbucket.org/julien-lange/gmc-synthesis (https://bitbucket.org/julien-lange/gmc-synthesis)



Future Work

- **Buffered channels** asynchronous communication over channels
- **Dynamic patterns** Expand bounded ranges (e.g. `for i:=0; i<10; i++ {...}`)



Thank you

Nicholas Ng & Nobuko Yoshida

Department of Computing
Imperial College London
{nickng,n.yoshida}@imperial.ac.uk (mailto:%7bnickng,n.yoshida%7d@imperial.ac.uk)

http://mrg.doc.ic.ac.uk (http://mrg.doc.ic.ac.uk)