

OPEN

PROBLEMS

OF
SESSION π
TYPE

Us ∈ **M**obility **R**esearch **G**roup



MobilityReadingGroup

π -calculus, Session Types research at Imperial College

[Home](#) [People](#) [Publications](#) [Grants](#) [Talks](#) [Tools](#) [Awards](#) [Kohel Honda](#)

NEWS

Our recent work [Fencing off Go: Liveness and Safety for Channel-based Programming](#) was summarised on [The Morning Paper](#) blog.

2 Feb 2017

Weizhen passed her viva today, congratulations Dr. Yang!

24 Jan 2017

Mariangiola Dezani-Ciancaglini, a long-term collaborator with our group working on Session Types turns 70 today, more details [here](#).

23 Dec 2016

Rumyana passed her viva today.

SELECTED PUBLICATIONS

2017

Raymond Hu , Nobuko Yoshida : [Explicit Connection Actions in Multiparty Session Types](#). *To appear in FASE 2017* .

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : [Fencing off Go: Liveness and Safety for Channel-based Programming](#). POPL 2017 .

Rumyana Neykova , Nobuko Yoshida : [Let It Recover: Multiparty Protocol-Induced Recovery](#). CC 2017 .

Julien Lange , Nobuko Yoshida : [On the Undecidability of Asynchronous Session Subtyping](#). *To appear in FoSSaCS 2017* .

Academic Staff

Nobuko Yoshida

Research Associate

Raymond Hu

Julien Lange

Nicholas Ng

Xinyu Niu

Alceste Scalas

Bernardo Toninho

PhD Student

Assel Altayeva

Juliana Franco

Rumyana Neykova

Weizhen Yang

<http://mrg.doc.ic.ac.uk/>

OOI Collaboration

OOI OCEAN OBSERVATORY INITIATIVE

Location
CURRENT LOCATION

SEARCH

RESOURCES

- All Resources
- Data Products
- Observatories
- Platforms
- Instruments

Welcome to Release 2 of the Ocean Observing Initiative (OOI). You already have access to many OOI features and real-time data. You can also customize the home page on this page for your use of the OOI as our guest.

For personalized services, such as setting up notifications and personalizing settings for your next visit, create a free account by clicking on "Create Account" at the top of the page.

RECENT UPDATES

DATA CENTER	RECORD	RECENT UPDATES	DATE	TYPE	OPERATION	STATUS
Temperature	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
Salinity	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
Chlorophyll	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
Currents	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
Sea Surface Height (SSH)	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
10m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
15m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
20m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
25m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
30m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
35m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
40m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
45m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
50m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
55m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
60m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
65m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
70m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
75m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
80m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
85m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
90m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
95m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good
100m depth	✓	1 Hour	2013-01-01 23:00:00	Temp	Observation	good

Dashboard
RECENT IMAGES

- Clider
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00
- Gorgonian Coral
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00
- Acoustic Release
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00

POPULAR RESOURCES

- Seafloor COT
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00
- Marine Capital
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00
- Surface Buoy
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00

UPCOMING EVENTS

- Oregon Coast Wave Hinge
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00
- Water Surface Elevation
Last Modified: 2013-01-01 23:00:00
Last Viewed: 2013-01-01 23:00:00
Last Updated: 2013-01-01 23:00:00

FACEBOOK **RELATED** **COMPOSITE** **STATUS**

- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16:** Multiparty Session Actors. Rumyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Rumyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Rumyana Neykova , Nicholas Ng

Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

Online tool : <http://scribble.doc.ic.ac.uk/>

```
1  module examples;
2
3  global protocol HelloWorld(role Me, role World) {
4    hello() from Me to World;
5    choice at World {
6      goodMorning1() from World to Me;
7    } or {
8      goodMorning1() from World to Me;
9    }
10 }
11
```

Load a sample 

Check

Protocol:

Role:

Project

Generate Graph

End-to-End Switching Programme by DCC



1. All design work takes place in ABACUS, DCC's enterprise architecture tool. This can export standard XMI files (an open standard for UML5)

2. XMI is converted into OpenTracing format for consumption by managed service



7. Generate exception report and send back to DCC



OPENTRACING



3. OpenTracing files are combined to build a model in Scribble

4. Model holds types rather than instances to understand behaviour

5. Scribble compiler identifies inconsistency, change & design flaws

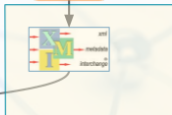
6. Issues highlighted graphically in Eclipse

End-to-End Switching Programme by DCC



Caveats:

1. Using earlier implementation of Scribble (CDL), because we already have those tools
2. Using earlier plugin to Eclipse - we'd want to improve this
3. We're not going via OpenTracing - this is part of the bid costs



7. Generate exception report and send back to DCC

Scope of the demo



OPENTRACING

3. OpenTracing files are combined to build a model in Scribble



4. Model holds types rather than instances to understand behaviour



5. Scribble compiler identifies inconsistency, change & design flaws



6. Issues highlighted graphically in Eclipse

Interactions with Industries

Strange Loop

SEPTEMBER 15-17 2016 / PEABODY OPERA HOUSE / ST. LOUIS, MO



Nobuko Yoshida
Imperial College, London

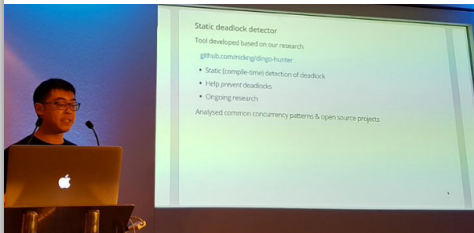


Adam Bowen @adamnbowen · Sep 15

I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at [#pwlconf](#), I want to learn more.

DoC researcher to speak at Golang UK conference

by Vicky Kapogianni
20 July 2016



Static deadlock detector
Tool developed based on our research
[@github.com/inking/dingo-hunter](#)

- Static (compile-time) detection of deadlock
- Help prevent deadlocks
- Ongoing research

Analysed common concurrency patterns & open source projects

DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

[Click here to add content](#)



[@nicholascwng](#) rocking on [@GolangUKconf](#) about static deadlock detection in [#golang](#) [#gouk16](#)



The Golang UK Conference

Interactions with Industries

#unctional Londoners Meetup Group

6 days ago · 6:30 PM

Session Types with Fahd Abdeljallal



43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... [LEARN MORE](#)

Distributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — CTO of Actyx

actyx

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

Go concurrency verification research at DoC grabs headline

A paper by DoC researchers at POPL on Go concurrency verification was featured in a tech blog and generates a buzz outside of the research community.

A [paper](#) by researchers at the department was recently featured in the morning paper, a [blog](#) by venture capitalist Adrian Colye, which summarises an important, influential, topical or otherwise interesting paper in the field of computer science every weekday in an easily digestible way by non-researchers. On the [2 Feb 2017 issue](#) of the morning paper, It was highlighted as "the true spirit of POPL (Principles of Programming Languages)".

Selected Publications 2016/2017



- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornella Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange , NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu , NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Romyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects .

Selected Publications 2016/2017



- [ECOOP'17] Alceste Scala, Raymond Hu, Ornela Darda, NY :A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- [COORDINATION'17] Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- [FoSSaCS'17] Julien Lange , NY : On the Undecidability of Asynchronous Session Subtyping.
- [FASE'17] Raymond Hu , NY : Explicit Connection Actions in Multiparty Session Types.
- [CC'17] Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- [POPL'17] Julien Lange , Nicholas Ng , Bernardo Toninho , NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- [FPL'16] Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk: EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- [ECOOP'16] Alceste Scala, NY: Lightweight Session Programming in Scala
- [CC'16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- [FASE'16] Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- [TACAS'16] Julien Lange, NY: Characteristic Formulae for Session Types.
- [ESOP'16] Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- [POPL'16] Dominic Orchard, NY: Effects as Sessions, Sessions as Effects.

HOW to

- derive theories to practices
- make theories understandable
- meet theoretical challenges (concurrency . distributions)
- Communicate people

GO

programming language @ Google (2009)

- ▶ **Message - Passing** based multicore PL, successor of C
- ▶ **Do not communicate by shared memory;**
instead, share memory by **communicating**

Go Lang Proverb

- ▶ **Explicit channel-based concurrency**
 - Buffered I/O communication channels
 - Lightweight thread spawning - **goroutines**
 - Selective **send/receive**

CSP_{80'}

FUN

Dropbox, Netflix, Docker, CoreOS

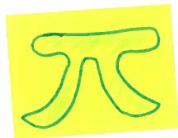
- ▶ **GO** has a **runtime deadlock detector**
- ▶ How can we detect **partial deadlock** and **channel errors** for **realistic programs**?
- ▶ Use **behavioural types** in process calculi
e.g. [ACM Survey, 2016] **185** citations, 6 pages
- ▶ Dynamic channel creations, unbounded thread creations, recursions, ..
- ▶ **Scalable** (synchronous/asynchronous) **Modular, Refinable**

▶ GO has a runtime deadlock detector

▶ How can we detect partial deadlock and channel errors for realistic programs?

▶ Use behavioural types in process calculi

e.g. [ACM Survey, 2016] **185** citations, 6 pages



▶ Dynamic channel creations, unbounded thread creations, recursions, ..

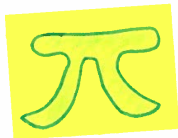
▶ Scalable (synchronous/asynchronous) Modular, Refinable

▶ **GO** has a **runtime deadlock detector**

▶ How can we detect **partial deadlock** and **channel errors** for realistic programs?

▶ Use **behavioural types** in process calculi

e.g. [ACM Survey, 2016] **185** citations, 6 pages



▶ Dyn.  ne , unbounded thread creations, recursions, ..

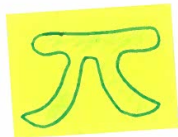
▶ **Scalable** (synchronous/asynchronous) **Modular, Refinable**

▶ **GO** has a **runtime deadlock detector**

▶ How can we detect **partial deadlock** and **channel errors** for realistic programs?

▶ Use **behavioural types** in process calculi

e.g. [ACM Survey, 2016] **185** citations, 6 pages



186 ??

▶ channel creations, unbounded thread creations, recursions, ..

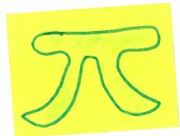
▶ **Scalable** (synchronous/asynchronous) **Modular, Refinable**

▶ **GO** has a runtime deadlock detector

▶ How can we detect *partial deadlock* and *channel errors* for realistic programs?

▶ Use *behavioural types* in process calculi

e.g. [ACM Survey, 2016] **185** citations, 6 pages



▶ Dynamic channel creations, unbounded thread creations, ...

▶ **Scalable** (synchronous/asynchronous) **Modular**, **retinable**

Understandable

Our Framework

STEP 1 Extract Behavioural Types

- ▶ (Most) Message passing features of GO
- ▶ Tricky primitives: selection, channel creation

STEP 2 Check Safety/Liveness of Behavioural Types

- ▶ Model-Checking (Finite Control)

STEP 3

- ▶ Relate Safety/Liveness of Behavioural Types and GO Programs
 - ▶ 3 Classes [POPL'17]
 - ▶ Termination Check

Our Framework

STEP 1 Extract Behavioural Types

- ▶ (Most) Message passing features of GO
- ▶ Tricky primitives: selection, channel creation

STEP 2 Check Safety/Liveness of Behavioural Types

- ▶ Model-Checking (Finite Control)

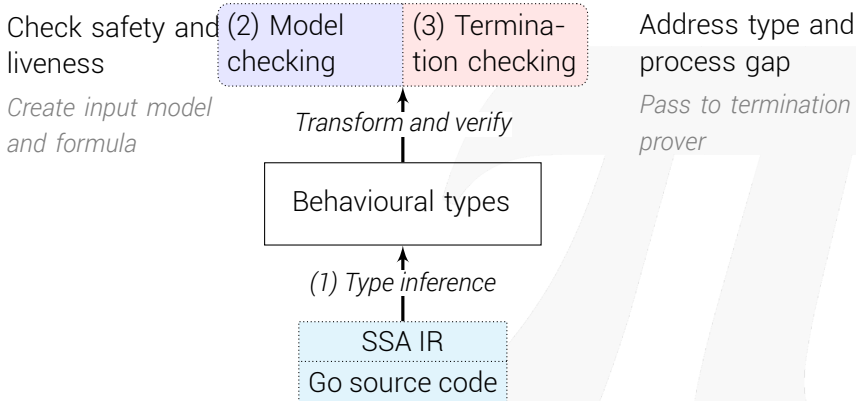


STEP 3

- ▶ Relate Safety/Liveness of Behavioural Types and GO Programs
 - ▶ 3 Classes [POPL'17]
 - ▶ Termination Check

Verification framework for Go

Overview



Concurrency in Go

```
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; i < 2; i++ {
            print("Working...")
        }
    }()
    go recv(ch, done)
    go recv(ch, done) // Who is ch receiving from?
    print("Done:", <-done, <-done) // 2 receivers, 2 replies
}

func send(ch chan int) { ch <- 1 } // Send to channel.
func recv(in, out chan int) { out <- <-in } // Fwd in to out.
```

- Send/receive blocks goroutines if channel full/empty resp.
- Close a channel `close(ch)`
- Guarded choice `select { case <-ch:; case <-ch2: }`

Concurrency in Go

Deadlock detection

```
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; i < 2; i++ {
            print("Working...")
        }
    }()
    go recv(ch, done)
    go recv(ch, done) // Who is ch receiving from?
    print("Done:", <-done, <-done) // 2 receivers, 2 replies
}

func send(ch chan int) { ch <- 1 } // Send to channel.
func recv(in, out chan int) { out <- <-in } // Fwd in to out.
```

Run program:

```
$ go run main.go
fatal error: all goroutines are asleep - deadlock!
```

Concurrency in Go

Deadlock detection

```
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; ; i++ { // infinite loop
            print("Working...")
        }
    }()
    go recv(ch, done)
    go recv(ch, done) // Who is ch receiving from?
    print("Done:", <-done, <-done) // 2 receivers, 2 replies
}

func send(ch chan int) { ch <- 1 } // Send to channel.
func recv(in, out chan int) { out <- <-in } // Fwd in to out.
```

Change to infinite loop

Concurrency in Go

Deadlock detection

```
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; ; i++ { // infinite loop
            print("Working...")
        }
    }()
    go recv(ch, done)
    go recv(ch, done) // Who is ch receiving from?
    print("Done:", <-done, <-done) // 2 receivers, 2 replies
}

func send(ch chan int) { ch <- 1 } // Send to channel.
func recv(in, out chan int) { out <- <-in } // Fwd in to out.
```

Change to infinite loop

Deadlock **NOT** detected (some goroutines are running)

Concurrency in Go

Deadlock detection

- Go has a runtime deadlock detector, panics (crash) if deadlock
- Deadlock if all goroutines are blocked
- Some packages (e.g. net for networking) **disables** it

```
import _ "net" // Load "net" package
func main() {
    ch := make(chan int)
    send(ch)
    print(<-ch)
}
func send(ch chan int) { ch <- 1 }
```

Add benign import

Concurrency in Go

Deadlock detection

- Go has a runtime deadlock detector, panics (crash) if deadlock
- Deadlock if all goroutines are blocked
- Some packages (e.g. net for networking) **disables** it

```
import _ "net" // Load "net" package
func main() {
    ch := make(chan int)
    send(ch)
    print(<-ch)
}
func send(ch chan int) { ch <- 1 }
```

Add benign import

Deadlock **NOT** detected

Go Programs as Processes

Go Program

$$P, Q ::= \pi; P$$
$$\pi ::= u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$P, Q := \pi; P$$
$$| \quad u; P$$
$$\pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q := \pi; P \\ \quad | \quad u; P \\ \quad | \quad \{\pi_i; P_i\}_{i \in I} \end{array} \qquad \pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q \quad := \quad \pi; P \\ \quad \quad | \quad u; P \\ \quad \quad | \quad \{\pi_i; P_i\}_{i \in I} \\ \quad \quad | \quad e P Q \end{array} \qquad \pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q := \pi; P \\ \quad | \quad u; P \\ \quad | \quad \{\pi_i; P_i\}_{i \in I} \\ \quad | \quad e P Q \\ \quad | \quad (y:\sigma); P \end{array} \qquad \pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q := \pi; P \\ \quad | \quad u; P \\ \quad | \quad \{\pi_i; P_i\}_{i \in I} \\ \quad | \quad e P Q \\ \quad | \quad (y:\sigma); P \\ \quad | \quad P \mid Q \mid \mathbf{0} \mid (\nu c)P \end{array} \qquad \pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q := \pi; P \qquad \qquad \qquad \pi := u!\langle e \rangle \mid u?(y) \mid \tau \\ \quad | \quad u; P \\ \quad | \quad \{\pi_i; P_i\}_{i \in I} \\ \quad | \quad e P Q \\ \quad | \quad (y:\sigma); P \\ \quad | \quad P \mid Q \mid \mathbf{0} \mid (\nu c)P \\ \quad | \quad X\langle \tilde{e}, \tilde{u} \rangle \\ D := X(\tilde{x}) = P \\ \mathbf{P} := \{D_i\}_{i \in I} P \end{array}$$

Go Programs as Processes

Go Program

$$\begin{array}{l} P, Q := \pi; P \\ \quad | u; P \\ \quad | \{\pi_i; P_i\}_{i \in I} \\ \quad | e P Q \\ \quad | (y:\sigma); P \\ \quad | P | Q | \mathbf{0} | (\nu c)P \\ \quad | X(\tilde{e}, \tilde{u}) \\ D := X(\tilde{x}) = P \\ \mathbf{P} := \{D_i\}_{i \in I} P \end{array} \quad \pi := u!\langle e \rangle \mid u?(y) \mid \tau$$

Abstracting Go with Behavioural Types

Types

$$\begin{aligned}\alpha &::= \bar{u} \mid u \mid \tau \\ T, S &::= \alpha; T \mid T \oplus S \mid \&\{\alpha_i; T_i\}_{i \in I} \mid (T \mid S) \mid \mathbf{0} \\ &\quad \mid (\mathbf{new} \ a)T \mid \mathbf{close} \ u; T \mid \mathbf{t}\langle \tilde{u} \rangle \\ \mathbf{T} &::= \{\mathbf{t}\langle \tilde{y}_i \rangle = T_i\}_{i \in I} \mathbf{in} \ S\end{aligned}$$

- Types of a CCS-like process calculus
- Abstracts Go concurrency primitives
 - Send/Recv, new (channel), parallel composition (spawn)
 - Go-specific: Close channel, Select (guarded choice)

MiGo Liveness / Safety

$P \Downarrow a$

Barb
[Milner 8
Sangiorgi 92]

Channel Safety

- ▶ Channel is closed at most once
- ▶ Can only input from a closed channel (default value)
- ▶ Others raise an error and **crash**

MiGo Liveness / Safety

$P \Downarrow a$

Barb
[Milner 8
Sangiorgi 92]

Channel Safety

- ▶ Channel is closed at most once
- ▶ Can only input from a closed channel (default value)
- ▶ Others raise an error and **crash**

P is channel safe if $P \xrightarrow{*} (\nu \bar{c}) Q$ and $Q \Downarrow \text{close}(a)$

$$\neg(Q \Downarrow \text{end}(a)) \wedge \neg(Q \Downarrow \bar{a})$$

never closing

never send

a closed

Migo Liveness / Safety

► Liveness

All reachable actions are eventually performed

P is live if $P \xrightarrow{*}_{(v\bar{c})} Q$

$$Q \downarrow a \Rightarrow Q \downarrow \tau \text{ at } a$$

$$Q \downarrow \bar{a} \Rightarrow Q \downarrow \tau \text{ at } a$$

Reduction
(τ)
at a

Select



$P_1 = \text{select } \{a!, b?, z.P\}$

Time
Out

if P is live
 P_1 is live

$P_2 = \text{select } \{a!, b?\}$

$R_1 = a?$

P_2 is not
live
 $P_2 \mid R_2$ is

Select



$P_1 = \text{select } \{a!, b?, z.P\}$

Time
Out

if P is live
 P_1 is live

$P_2 = \text{select } \{a!, b?\}$

$R_1 = a?$

P_2 is not
live
 $P_2 | R_2$ is

Barb $\downarrow \tilde{a}$

$\pi_i \downarrow a_i$

$\text{select } \{\pi_i. P_i\} \downarrow \tilde{a}$

$P \downarrow \tilde{a}$

$Q \downarrow \bar{a}_i$

$P | Q \downarrow [a_i]$

Liveness

$Q \downarrow \tilde{a} \Rightarrow Q \Downarrow z \text{ at } a_i$

Verification framework for Go

Model checking with mCRL2

Generate LTS model and formulae from types

- Finite control (no parallel composition in recursion)
- Properties (formulae for model checker):
 - ✓ Global deadlock
 - ✓ Channel safety (no send/**close** on closed channel)
 - ✓ Liveness (partial deadlock)
 - ✓ Eventual reception
 - Require additional guarantees

the λ -calculus

encoding properties with barbs

Global Deadlock

$$\bigwedge a \in C (\downarrow a \vee \downarrow \bar{a}) \Rightarrow \langle \alpha \rangle T$$

Channel Safety

$$\bigwedge a \in C \downarrow \text{close } a \Rightarrow \neg (\downarrow \bar{a} \vee \downarrow \text{close } a)$$

Liveness

$$\bigwedge a \in C (\downarrow a \vee \downarrow \bar{a}) \Rightarrow \Phi (\langle [a] \rangle T) \wedge$$

$$\bigwedge \tilde{a} \in C^m \downarrow \tilde{a} \Rightarrow \Phi (\bigvee_{a \in \tilde{a}} \langle [a] \rangle T)$$

[Lange & NY
TACAS'17]

Verification framework for Go

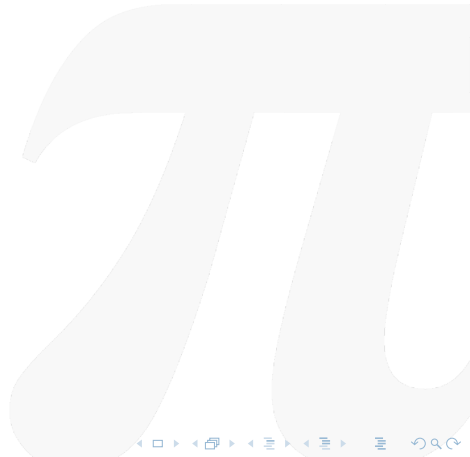
Termination checking with KITTeL

- Extracted types do not consider *data* in process
- Type liveness != program liveness
 - Especially when involving iteration
 - Check for loop termination
- Properties:
 - ✓ Global deadlock
 - ✓ Channel safety (no send/`close` on closed channel)
 - ✓ Liveness (partial deadlock)
 - ✓ Eventual reception

```
func main() {  
    ch := make(chan int)  
    go func() {  
        for i := 0; i < 10; i-- {  
            // Does not terminate  
        }  
        ch <- 1  
    }()  
    <-ch  
}
```

- Type: **Live**
- Program: **NOT** live

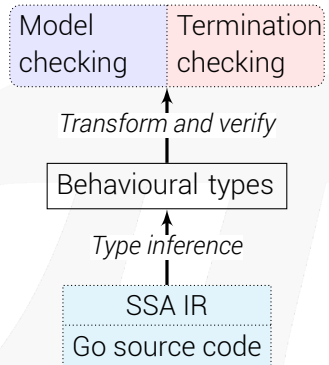
Tool demo



Conclusion

Verification framework based on
Behavioural Types

- Behavioural types for Go concurrency
- Infer types from Go source code
- Model check types for safety/liveness
- + termination for iterative Go code



Future work

- Extend framework to support more properties
- Unlimited possibilities!
 - Different verification techniques
 - e.g. [POPL'17], Choreography synthesis [CC'15]
 - Different concurrency issues
 - Other synchronisation mechanisms
 - Race conditions

