OPEN PROBLEMS OF SESSION TYPES
Us ∈ Mobility Research Group

MobilityReadingGroup
π-calculus, Session Types research at Imperial College

NEWS

SELECTED PUBLICATIONS

2017


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


http://mrg.doc.ic.ac.uk/
• **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 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.
module examples;

global protocol HelloWorld(role Me, role World) {
    hello() from Me to World;
    choice at World {
        goodMorning1() from World to Me;
    } or {
        goodMorning1() from World to Me;
    }
}
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

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

www.estafet.com

Estafet Managed Service
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

Scope of the demo
7. Generate exception report and send back to DCC

Estafet Managed Service

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

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 #pwiconf, I want to learn more.

DoC researcher to speak at Golang UK conference
by Vicky Kapogianni
20 July 2016

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

@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

The Golang UK Conference
Interactions with Industries

**F#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

**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)
Selected Publications 2016/2017

- [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- [POPL’16] Dominic Orchard, NY: Effects as sessions, sessions as effects.
Selected Publications 2016/2017

- [CC’16] Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
HOW to

- derive theories to practices
- make theories understandable
- meet theoretical challenges (concurrency, distributions)
- communicate people
Behavioural Type-Based Static Verification Framework for Go

Julian Lange  Nicholas Ng  Bernardo Toninho  Nobuko Yoshida
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)".
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

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

- Dynamic, 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
- 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, extensible
  - 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

Check safety and liveness

Create input model and formula

(2) Model checking

(3) Termination checking

Transform and verify

Behavioural types

Address type and process gap

Pass to termination prover

(1) Type inference

SSA IR

Go source code

Nobuko Yoshida
Open Problems of Session Types
mrg.doc.ic.ac.uk
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

```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.
```

Run program:

```
$ go run main.go
fatal error: all goroutines are asleep - deadlock!
```
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; ; i++ { // infinite
            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

Deadlock detection

Concurrency in Go

Nobuko Yoshida
Open Problems of Session Types

mrg.doc.ic.ac.uk
Concurrency in Go
Deadlock detection

```go
func main() {
    ch, done := make(chan int), make(chan int)
    go send(ch) // Spawn as goroutine.
    go func() {
        for i := 0; ; i++ { // infinite
            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.

Deadlock NOT detected (some goroutines are running)
```
Go has a runtime deadlock detector, panics (crash) if deadlock
Deadlock if all goroutines are blocked
Some packages (e.g. \texttt{net} for networking) dis\textbf{a}bles it

```go
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

Nobuko Yoshida
Open Problems of Session Types
Concurrent 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

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

func send(ch chan int) { ch <- 1 }
```

Deadlock **NOT** detected
Go Programs as Processes

Go Program

\[
P, Q \ := \ \pi; P \quad \quad \pi \ := \ u!\langle e \rangle \mid u?(y) \mid \tau
\]
Go Programs as Processes

**Go Program**

\[
P, Q \ := \ \pi; P \\
\mid \ \text{close } u; P
\]

\[
\pi \ := \ u!(e) \mid u?(y) \mid \tau
\]
Go Programs as Processes

<table>
<thead>
<tr>
<th>Go Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P, Q ) := ( \pi; P ) ( \pi ) := ( u!\langle e \rangle \mid u?(y) \mid \tau )</td>
</tr>
<tr>
<td>close ( u ); ( P )</td>
</tr>
<tr>
<td>select( { \pi_i; P_i }_{i \in I} )</td>
</tr>
</tbody>
</table>

Nobuko Yoshida
Open Problems of Session Types
Go Programs as Processes

Go Program

\[ P, Q \quad ::= \quad \pi; P \quad \quad \pi \quad ::= \quad u!(e) \mid u?(y) \mid \tau \]
\[ \quad \mid \quad \text{close} \ u; P \]
\[ \quad \mid \quad \text{select}\{\pi_i; P_i\}_{i \in I} \]
\[ \quad \mid \quad \text{if} \ e \ \text{then} \ P \ \text{else} \ Q \]
## Go Programs as Processes

### Go Program

<table>
<thead>
<tr>
<th>$P, Q$</th>
<th>$\pi; P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>close $u; P$</td>
<td>$\pi := u!\langle e \rangle \mid u?y \mid \tau$</td>
</tr>
<tr>
<td>select{$\pi_i; P_i$}, $i \in I$</td>
<td></td>
</tr>
<tr>
<td>if $e$ then $P$ else $Q$</td>
<td></td>
</tr>
<tr>
<td>newchan($y:\sigma$); $P$</td>
<td></td>
</tr>
</tbody>
</table>
Go Programs as Processes

<table>
<thead>
<tr>
<th>Go Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P, Q ) := ( \pi; P ) | ( \pi := u!(e) | u?(y) | \tau )</td>
</tr>
<tr>
<td>| close ( u ); ( P )</td>
</tr>
<tr>
<td>| select( { \pi_i; P_i }_{i \in I} )</td>
</tr>
<tr>
<td>| if ( e ) then ( P ) else ( Q )</td>
</tr>
<tr>
<td>| newchan( (y: \sigma) ); ( P )</td>
</tr>
<tr>
<td>| ( P</td>
</tr>
</tbody>
</table>
Go Programs as Processes

Go Program

\[
P, Q := \pi; P \\
| \text{close } u; P \\
| \text{select}\{\pi_i; P_i\}_{i \in I} \\
| \text{if } e \text{ then } P \text{ else } Q \\
| \text{newchan}(y:\sigma); P \\
| P \mid Q \mid 0 \mid (\nu c)P \\
| X\langle \tilde{e}, \tilde{u} \rangle \\
\]

\[
D := X(\tilde{x}) = P \\
P := \{D_i\}_{i \in I} \text{ in } P
\]

\[
\pi := u!(e) \mid u?(y) \mid \tau
\]
Go Programs as Processes

Go Program

\[
\begin{align*}
P, Q & := \pi; P \\
& \quad \text{close } u; P \\
& \quad \text{select}\{\pi_i; P_i\}_{i \in I} \\
& \quad \text{if } e \text{ then } P \text{ else } Q \\
& \quad \text{newchan}(y:\sigma); P \\
& \quad P \mid Q \mid 0 \mid (\nu c)P \\
& \quad X\langle\tilde{e}, \tilde{u}\rangle \\
D & := X(\tilde{x}) = P \\
P & := \{D_i\}_{i \in I} \text{ in } P
\end{align*}
\]
### 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)**
Channel Safety

- Channel is closed at most once.
- Can only input from a closed channel (default value).
- Others raise an error and crash.
Mi Go Liveness / Safety

Channel Safety

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

\[ P \Downarrow \alpha \]

Barb [Milner & Sangioergi 92]

\( P \) is channel safe if \( P \xrightarrow{*} (\neg \exists) Q \) and \( Q \Downarrow \text{close}(a) \)

\[ \neg (Q \Downarrow \text{end}(a)) \land \neg (Q \Downarrow \overline{a}) \]

never closing never send
Migo Liveness/Safety

- Liveness

All reachable actions are eventually performed

\[ P \text{ is live if } P \xrightarrow{\delta} Q \]

\[ Q \downarrow a \Rightarrow Q \downarrow \tau \text{ at } a \]

\[ Q \downarrow \overline{a} \Rightarrow Q \downarrow \tau \text{ at } a \]
Select

\[ P_1 = \text{select } \{ a!, b? \}, z, P \]  

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

\[ R_1 = a? \]

\[ \text{Time Out} \]

- if \( P \) is live
  - \( P_1 \) is live
  - \( P_2 \) is not live
  - \( P_2 | R_2 \) is
Select

\[ P_1 = \text{select}\ \{ a!, b?, z \cdot P \} \]

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

Barb \ \downarrow \tilde{a}

\[ \pi_i \downarrow q_i \]

\[ \text{select} \ \{ \pi_i \cdot P_i \} \downarrow \tilde{a} \]

\[ P \downarrow \tilde{a} \]

\[ Q \downarrow \tilde{a} \Rightarrow Q \downarrow z \text{ at } a_i \]
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/c\texttt{close} on closed channel)
  - ✔ Liveness (partial deadlock)
  - ✔ Eventual reception
    - Require additional guarantees
the \( \mu \)-calculus

encoding properties with barbs

Global Deadlock

Channel Safety

Liveness

\( \bigwedge a \in C \ (\downarrow a \lor \downarrow \overline{a}) \Rightarrow \langle \alpha \rangle T \)

\( \bigwedge a \in C \ \downarrow \text{close } a \Rightarrow \neg (\downarrow \overline{a} \lor \downarrow \text{close } a) \)

\( \bigwedge a \in C \ (\downarrow a \lor \downarrow \overline{a}) \Rightarrow \Phi (\langle [a] \rangle T) \land \)

\( \bigwedge \overline{a} \in C^m \ \downarrow \overline{a} \Rightarrow \Phi (V a e \overline{a} \langle [a] \rangle T) \)

[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

```go
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*
Relating Programs and Types

Program

\[ F(n, i, o) \triangleq i?(x); \text{if } (x \% n \neq 0) \text{ then } o!(x); F(n, i, o) \text{ else } F(n, i, o) \]

Type

\[ \text{filter}(i, o) \triangleq i; (\overline{o}; t_F(i, o) \oplus t_F(i, o)) \]

- Identify 3 classes (Liveness)
  1. May Terminate
  2. Without infinitely running conditionals
  3. Non-deterministic conditional

Channel Safety Programs = Types
Relating Programs and Types

Program

\[ F(n, i, o) \triangleq \begin{cases} i?(x); \text{if } (x \% n) \neq 0!\langle x \rangle; F(n, i, o) \text{ else } F(n, i, o) \end{cases} \]

Type

\[ \text{filter}(i, o) \triangleq i; (\overline{o}; t_F(i, o) \oplus t_F(i, o)) \]

- Identify 3 classes (Liveness)
  1. May Terminate
  2. Without infinitely running conditionals
  3. Non-deterministic conditional

Channel Safety Programs = Types
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

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

- Type: Live
- Program: NOT live

Nobuko Yoshida
Open Problems of Session Types
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

Diagram:

- Go source code
- SSA IR
- Behavioural types
- Type inference
- Transform and verify
- Model checking
- Termination checking

Nobuko Yoshida
Open Problems of Session Types

mrg.doc.ic.ac.uk
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
most programs use traditional imperative control flow features such as for loops, for-range loops (i.e. loops over a fixed finite data structure) and for-select loops (i.e. an infinite loop with a select that can break the loop – the Consumer function of Figure 1) instead of recursion; we assume that loop indices are not modified in loop bodies and that no goto-like constructs are used in a loop.

Since the analysis only takes into account loop parameters, a loop that indefinitely blocks (e.g. due to communication) may be identified as terminating. However, if our analysis identifies the inferred types as live and the termination check validates the program, both termination and program liveness are guaranteed.

6 EVALUATION

Table 3 lists several benchmarks of our tool against other static deadlock detection tools for Go (a detailed comparison of these tools is given in § 7). The benchmarks were run with go1.8.3 on an 8-core Intel i7-3770 machine with 16GB RAM on a 64-bit Linux. The model checker we used was mcrl2 v201707.1.

The results for Godel Checker are shown in columns 3–11. Column 3 shows the number of states in the input LTS as a measurement of the relative complexity of each program (proportional to the number of concurrency-related operations rather than the number of lines of code). Columns 4–7 show the core properties of Figure 6 in § 4, i.e. no global deadlock ($\psi_\infty$), liveness ($\psi_\infty$), channel safety ($\psi_\eta$) and eventual reception ($\psi_e$). Columns 8–10 list the running time of Godel Checker, where Column 8 lists the inference time, Columns 9 and 10 are the model checking times for liveness, and both liveness and channel safety, respectively. The total run time can be obtained by adding Column 8 to Column 9 or 10. Unless otherwise stated, all times are in milliseconds. Column 11 (Term) shows the result of the termination check, which proves the termination of loops in the given program, or times out after 15s. A program that times out is conservatively assumed not to terminate.

The times include both type inference and analysis stages, which does not support Programs 6, 7, and 19 due to dynamic spawning goroutines, while gopherlyzer does not support them due to a nested select statement. GoInfer/Gong analyses them correctly, but is much slower than Godel Checker.

Programs 1–7 are typical concurrent programs from the literature. The sieve program is not finite control (it spawns an infinite number of threads), thus it can only be analysed by GoInfer/Gong. Program 6 is a (three) dining philosophers program where the first fork can be released, while Program 7 is the traditional deadlock-free version (Program 19 is as Program 6 but with 5 philosophers). dingo-hunter does not support Programs 6, 7, and 19 due to dynamically spawned goroutines, while gopherlyzer does not support them due to a nested select statement. GoInfer/Gong analyses them correctly, but is much slower than Godel Checker.

Programs 8–12 consist of idiomatic Go patterns which are all handled correctly and quickly by our tool. Program 13 is a publicly available program which is not live. Program 14 is an implementation of the alternating bit protocol. Program 15 is the Producer-Consumer example from § 1, which is not live. All tools were able to verify this simple program. Program 16 demonstrates the mismatch between type and program liveness, where the type is live but due to an erroneous loop the program does not terminate and causes a partial deadlock. The termination check identifies this as possibly non-terminating, while GoInfer/Gong incorrectly identifies it as live. Program 17 closes a channel twice which flags a violation of channel safety in Godel Checker and GoInfer/Gong. Interestingly, dingo-hunter detects a deadlock (a false alarm) due to its representation of channel closure as a message exchange, but not due to the double close. gopherlyzer also detects a deadlock incorrectly due to the same reason. Program 18 is a program that violates the eventual reception property by sending an asynchronous message that is never received – none of the earlier tools can detect this.

Table 3: Go programs verified by our framework and comparison with existing static deadlock detection tools.

<table>
<thead>
<tr>
<th>Program</th>
<th># states</th>
<th>$\psi_\infty$</th>
<th>$\psi_\infty$</th>
<th>$\psi_\eta$</th>
<th>$\psi_e$</th>
<th>Godel Checker</th>
<th>Termin</th>
<th>dingo-hunter [35]</th>
<th>Term</th>
<th>gopherlyzer [39]</th>
<th>Term</th>
<th>GoInfer/Gong [30]</th>
<th>Time</th>
<th>CS</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mismatch</td>
<td>53</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>6390</td>
<td>3956.41</td>
<td>616.78</td>
</tr>
<tr>
<td>2 fixed</td>
<td>16</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>4653</td>
<td>40859.24</td>
<td>609.95</td>
</tr>
<tr>
<td>3 forselect</td>
<td>40</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>40859</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>4 prob</td>
<td>61</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>40859</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>5 prodcons</td>
<td>32</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>40859</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>6 cond-recur</td>
<td>15</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>130.4</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>7 consys</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>267</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>8 sieve</td>
<td>11</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>267</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>9 al-bit</td>
<td>112</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>10 double-close</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>11 double-cons</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>12 stackmsg</td>
<td>4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>13 concys</td>
<td>15</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>783.21</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>14 double-close</td>
<td>17</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>783.21</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>15 double-cons</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>16 al-bit</td>
<td>112</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>17 double-close</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>18 ProdCons</td>
<td>106</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>19 ProdCons</td>
<td>106</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>20 ProdCons</td>
<td>106</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>21 ProdCons</td>
<td>106</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
<tr>
<td>22 ProdCons</td>
<td>106</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>634</td>
<td>996.02</td>
<td>996.02</td>
</tr>
</tbody>
</table>

CS: Channel Safe, Term: Termination check, DF: Deadlock-free, timeout: Termination check timeout (likely does not terminate), SF: False Alarm, UN: Undetected liveness error.