Imperial College London

Building Graphical Choreographies from Communicating Machines

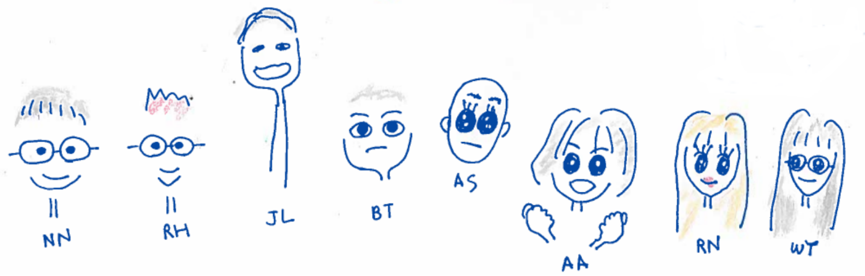
Principles and Applications

Julien Lange

based on works with L. Bocchi, N. Ng, E. Tuosto, *and* N. Yoshida

May 2017

Session Type Mobility Group



www.mrg.doc.ic.ac.uk

Us ∈ Mobility Research Group

MobilityReadingGroup

π-calculus, Session Types research at Imperial College

Home | People | Publications | Grants | Talks | Tools | Awards | Kohei 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 .

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

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

OOI Collaboration

- **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

www.scribble.org

Home | Getting Started | Downloads | Documentation | Community

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/

```
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;
  }
}
```

Load a sample | Check | Protocol: examples.HelloWorld | Role: Me | Project | Generate Graph

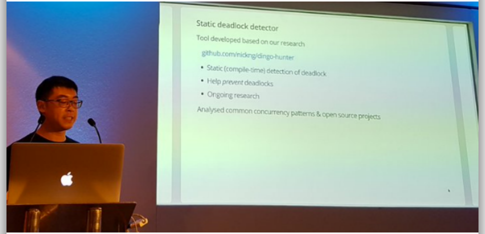
Interactions with Industries

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.

Nobuko Yoshida
Imperial College, London

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

Click here to add content

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

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*

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, 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 .

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.

Imperial College London

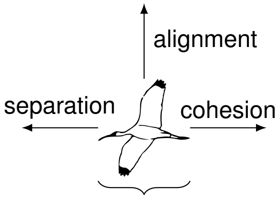
Building Graphical Choreographies from Communicating Machines

Principles and Applications

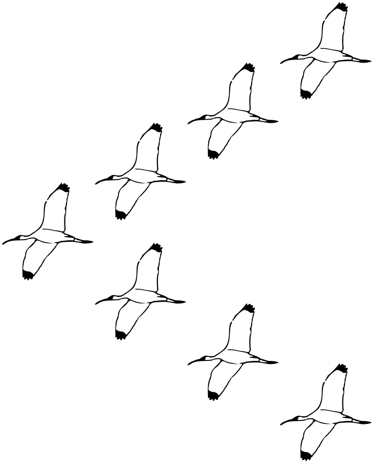
Julien Lange

based on works with L. Bocchi, N. Ng, E. Tuosto, *and* N. Yoshida

May 2017



Single bird $\simeq$ local behaviour $\simeq$ CFSM

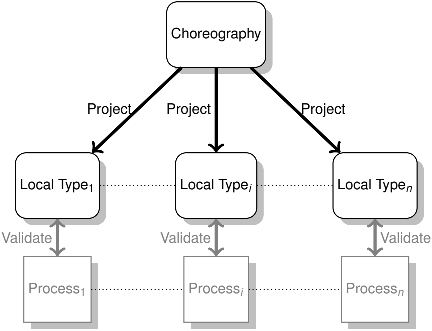


Flock $\simeq$ global behaviour $\simeq$ choreography

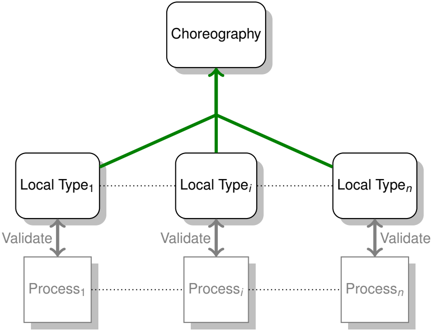
Introduction

- Parts of distributed systems change/evolve, not always in a coordinated way,
- these changes are often *not* documented.
- Service oriented systems are sometimes composed dynamically,
- it is often unclear how complex the overall system has become.
- Cognizant's Zero Deviation Lifecyle.

A *global* point of view of a distributed system is *essential* for top-level management.

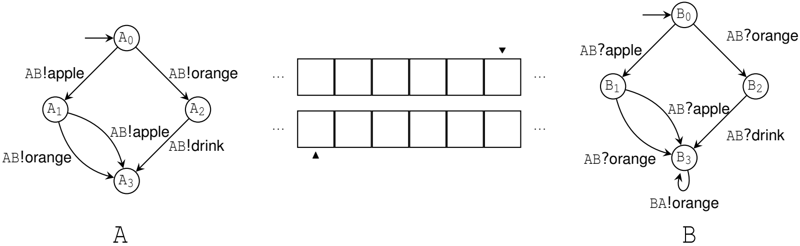


- Choreography-driven development, cf. Multiparty Session Types top-down approach (POPL'08 & ESOP'12)
- Not applicable without *a priori knowledge* of a choreography



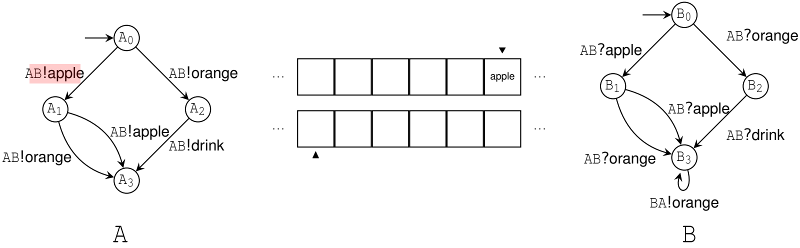
- Choreography-driven development, cf. Multiparty Session Types top-down approach (POPL'08 & ESOP'12)
- Not applicable without *a priori knowledge* of a choreography
- Our goal: from *Communicating Finite-State Machines* to *Global Graphs*

Background: CFSMs



"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)

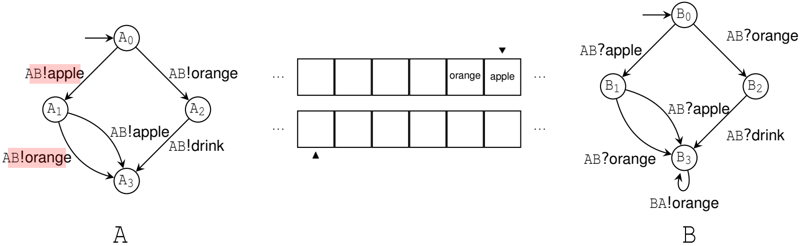
Background: CFSMs



“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Imperial College London

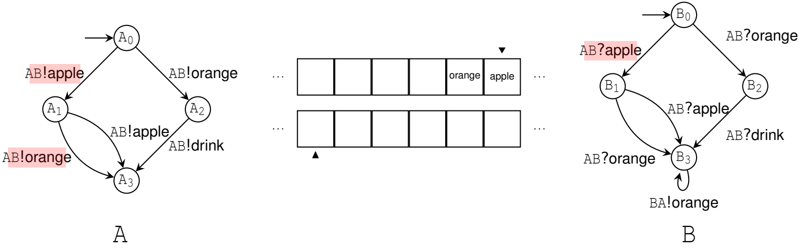
Background: CFSMs



"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)

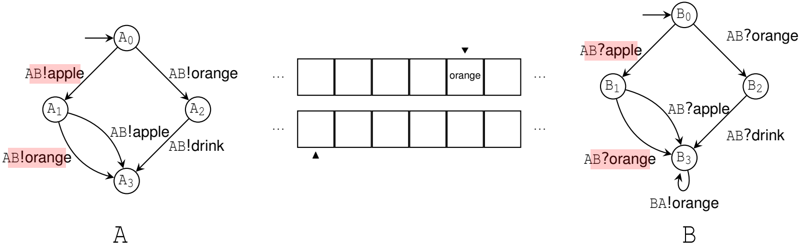
Imperial College London

Background: CFSMs



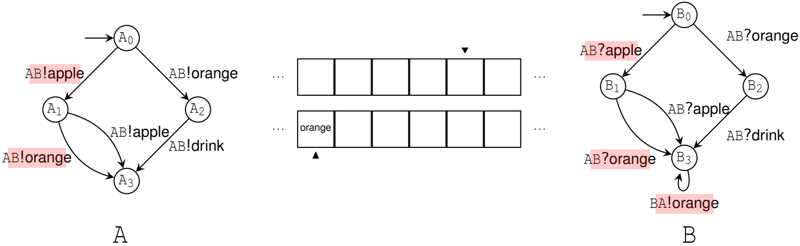
“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Background: CFSMs



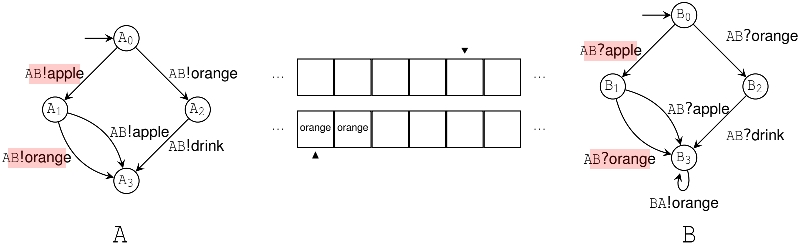
“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Background: CFSMs



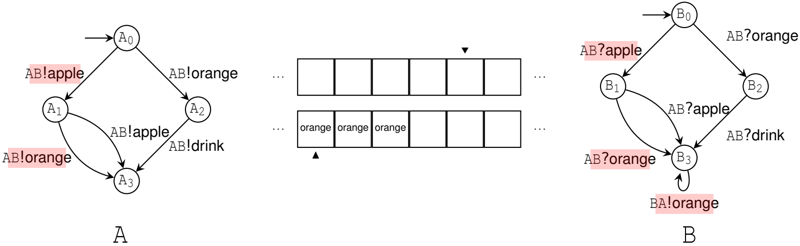
“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Background: CFSMs



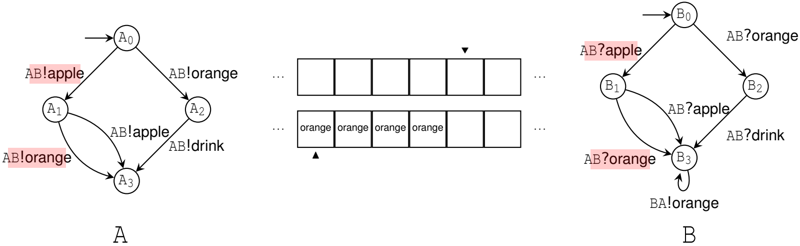
“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Background: CFSMs



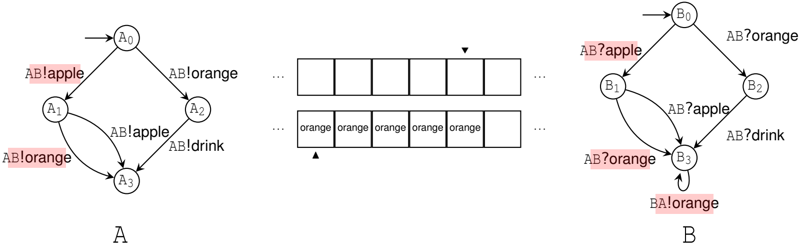
“On Communicating Finite-State Machines”, Brand & Zafiropulo (’83)

Background: CFSMs



"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)

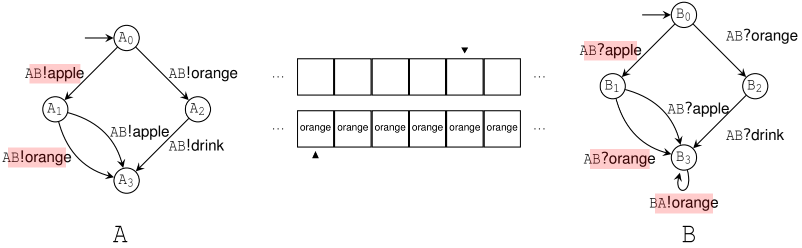
Background: CFSMs



"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)

Imperial College London

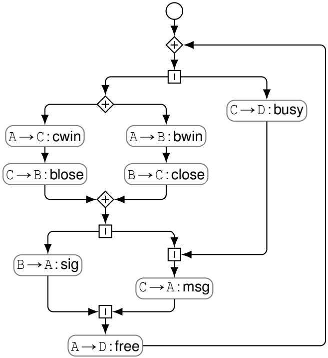
Background: CFSMs



"On Communicating Finite-State Machines", Brand & Zafiropulo ('83)

Imperial College London

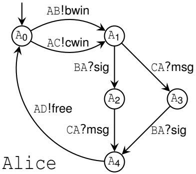
Global Graphs

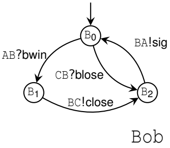


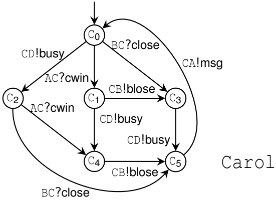
Four Player Game

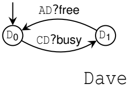
Imperial College
London

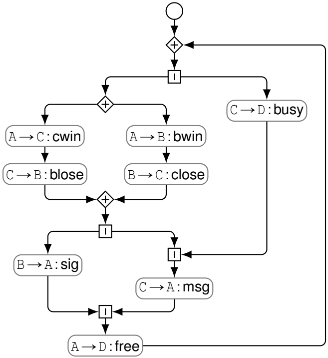
Global Graphs





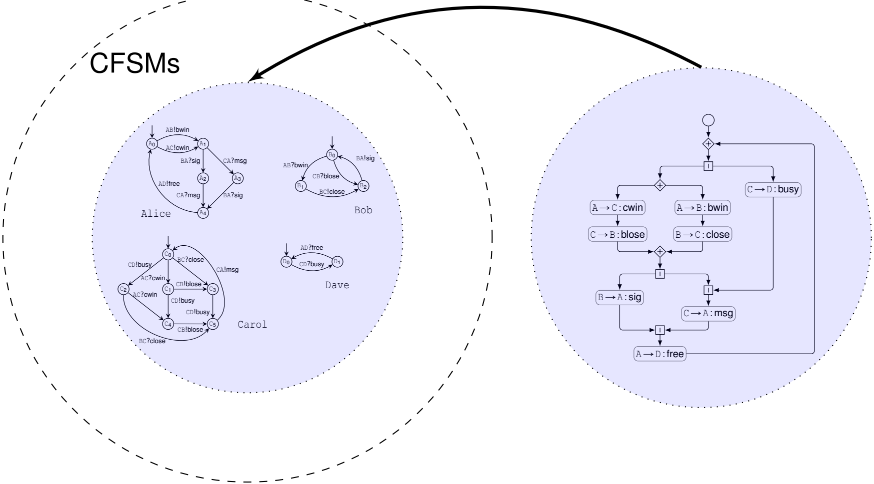




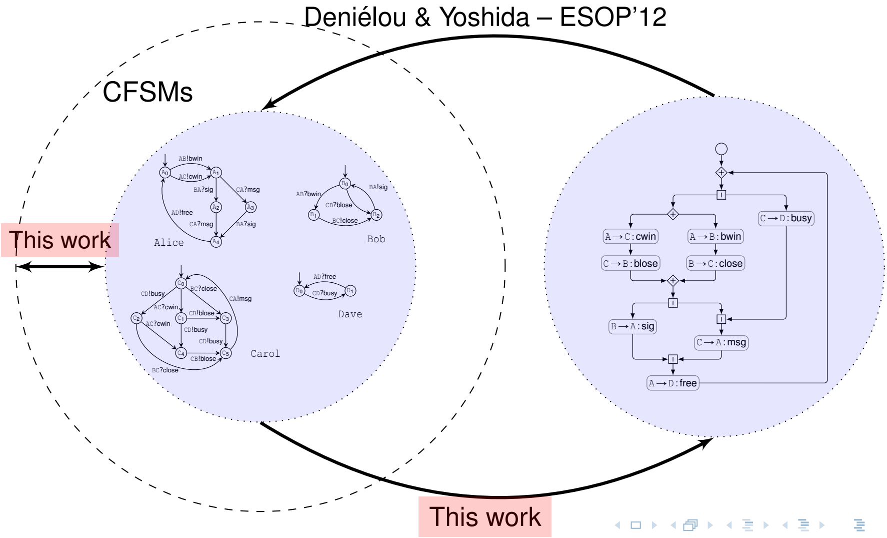


Four Player Game

Deniélou & Yoshida – ESOP'12



Imperial College
London



Objectives

Two main objectives:

- **Sound Condition for Safety:** generalised multiparty compatibility

  If $S = (M_1, \ldots, M_k)$ is *compatible* then S is "safe", i.e., every sent message is eventually received and no deadlock.

- **Construction of a Global Graph:**

  If G is the global graph constructed from S, then

$$S = (M_1, \ldots, M_k) \equiv (G{\downarrow}_1, \ldots, G{\downarrow}_k)$$

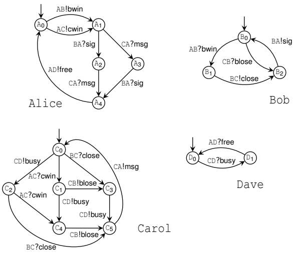
The Plan

1. Build $\mathrm{TS}(S)$, the transition system of all *synchronous* executions
2. Check for safety on $\mathrm{TS}(S)$ to
   - ensure equivalence between original system and the projections of the choreography,
   - guarantee safety (no deadlock, no orphan message)
3. Build a choreography (global graph) from $\mathrm{TS}(S)$, relying on
   - the theory of regions, and
   - Petri nets.

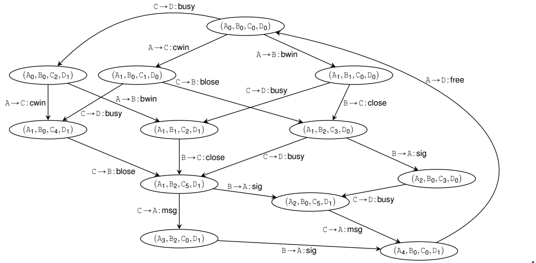
Imperial College London

1. Synchronous Transition System of CFSMs

Synchronous Transition System (TS(S))



Four Player Game (CFSMs)



Synchronous Transition System (TS(S))

Imperial College London

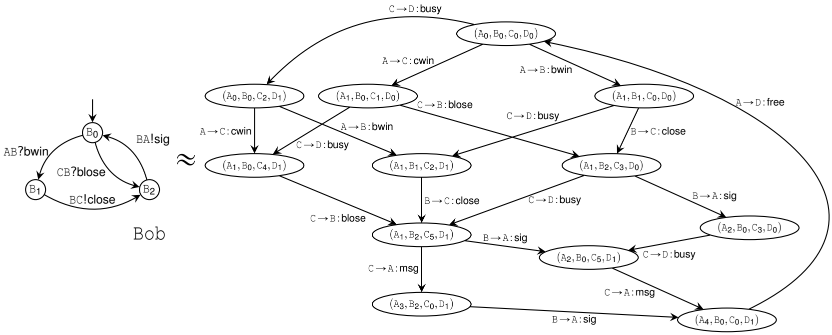
2. Check for Safety: *Generalised Multiparty Compatibility* (GMC)

1. Representability
2. Branching Property

Checking Compatibility (i) – Representability

Representability

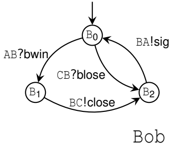
- For each participant: projection of TS(S) $\approx$ original machine



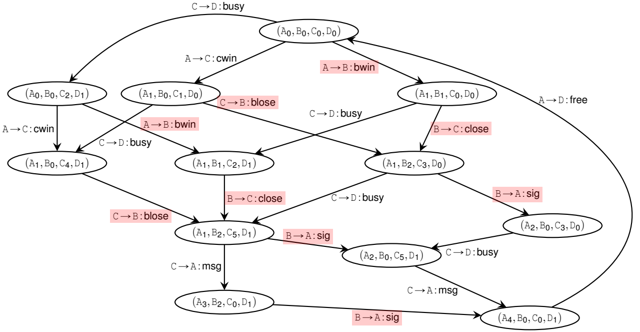
Checking Compatibility (i) – Representability

Representability

- For each participant: projection of $\mathrm{TS}(S) \approx$ original machine



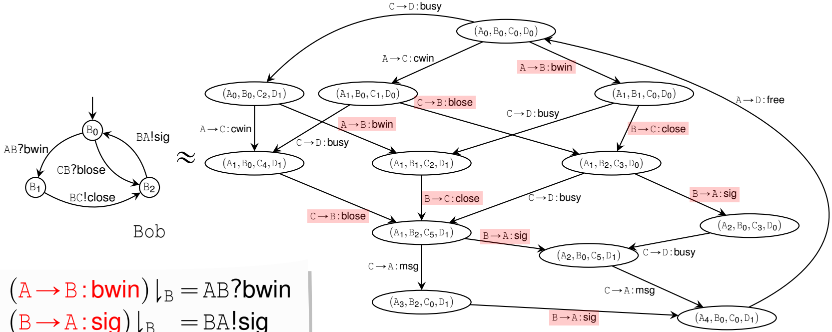
$\approx$



Checking Compatibility (i) – Representability

Representability

- For each participant: projection of $\mathsf{TS}(S) \approx$ original machine



$(\mathtt{A} \to \mathtt{B} : \mathsf{bwin}) \downharpoonright_{\mathtt{B}} = \mathtt{AB}?\mathsf{bwin}$

$(\mathtt{B} \to \mathtt{A} : \mathsf{sig}) \downharpoonright_{\mathtt{B}} = \mathtt{BA}!\mathsf{sig}$

$(\mathtt{C} \to \mathtt{D} : \mathsf{busy}) \downharpoonright_{\mathtt{B}} = \varepsilon$

Checking Compatibility (ii) – Branching Property

Each branching $\begin{tikzcd}[column sep=small] & n_1 \arrow[dl, "e"'] \arrow[dr, "e'"] & \\ {} & & {} \end{tikzcd}$ in TS must be either

Checking Compatibility (ii) – Branching Property

Each branching
$$\begin{tikzcd} & n_1 \arrow[dl, "e"'] \arrow[dr, "e'"] & \\ {} & & {} \end{tikzcd}$$
in TS must be either

- commuting:
$$\begin{tikzcd} & n_1 \arrow[dl, "e"'] \arrow[dr, "e'"] & \\ n_2 \arrow[dr, "e'"'] & & n_3 \arrow[dl, "e"] \\ & n_4 & \end{tikzcd}$$

Checking Compatibility (ii) – Branching Property

Each branching $\begin{tikzcd}[column sep=small] & n_1 \arrow[dl, "e"'] \arrow[dr, "e'"] & \\ {} & & {} \end{tikzcd}$ in TS must be either

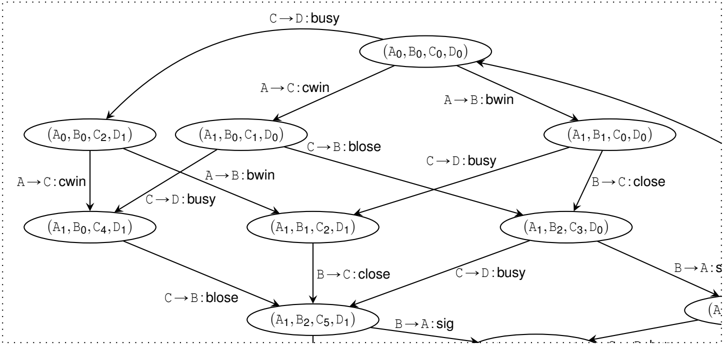
Last node, reachable from n_1, from which e and e' can be fired.

- or, each *last node* n_k

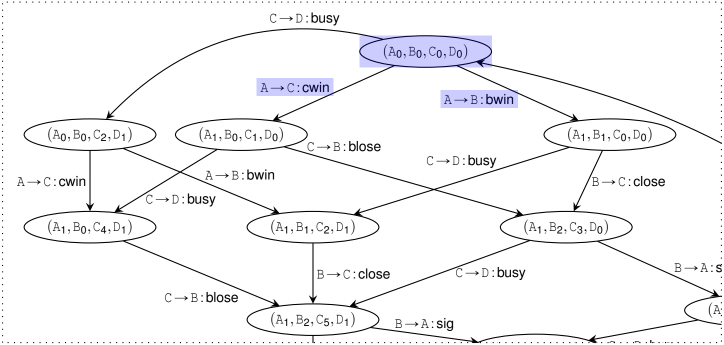
$$\begin{tikzcd}[column sep=small] & n_1 \arrow[dl, "e"'] \arrow[dr, "e'"] \arrow[rr, "e_1"] & & n_2 \arrow[dl, "e"'] \arrow[dr, "e'"] \arrow[rr, dotted, "e_{k-1}"] & & \boxed{n_k} \arrow[dl, "e"'] \arrow[dr, "e'"] & \\ {} & & {} & & {} & & {} \end{tikzcd}$$

must be a "well-formed" choice, i.e.,

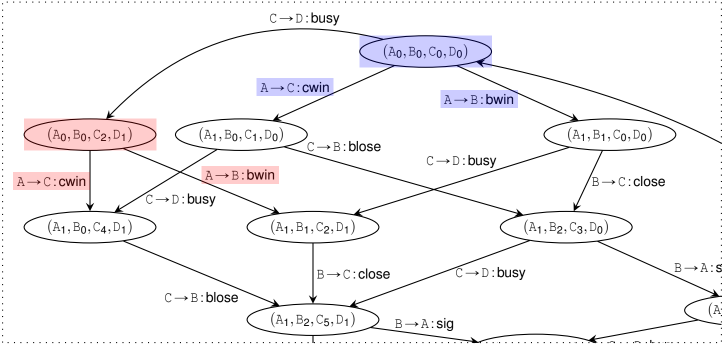
- for each participant
  - it receives a different message in each branch, or
  - it is not involved in the choice
- there is a unique sender


C → D : busy
(A0, B0, C0, D0)
A → C : cwin
A → B : bwin
(A0, B0, C2, D1)
(A1, B0, C1, D0)
C → B : blose
(A1, B1, C0, D0)
C → D : busy
A → C : cwin
A → B : bwin
B → C : close
C → D : busy
(A1, B0, C4, D1)
(A1, B1, C2, D1)
(A1, B2, C3, D0)
B → C : close
C → D : busy
C → B : blose
(A1, B2, C5, D1)
B → A : sig

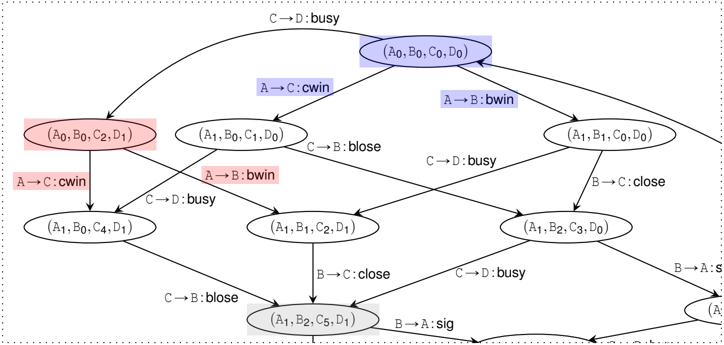
Imperial College London

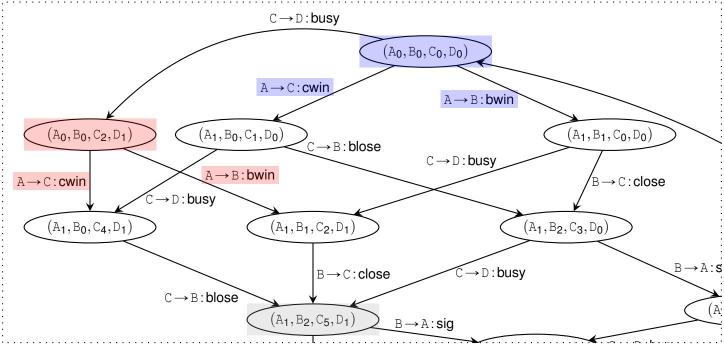


Imperial College
London



Imperial College
London





A : AB!bwin $\neq$ AC!cwin

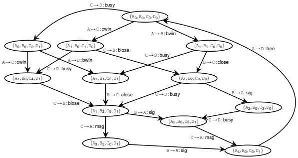
B : AB?bwin $\neq$ CB?blose

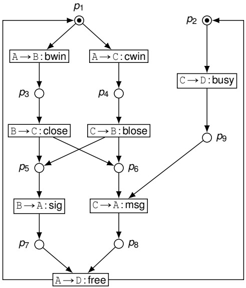
D : not involved in the choice

C : AC?cwin $\neq$ BC?close

3. Build a global graph

1: From TS to Petri Net

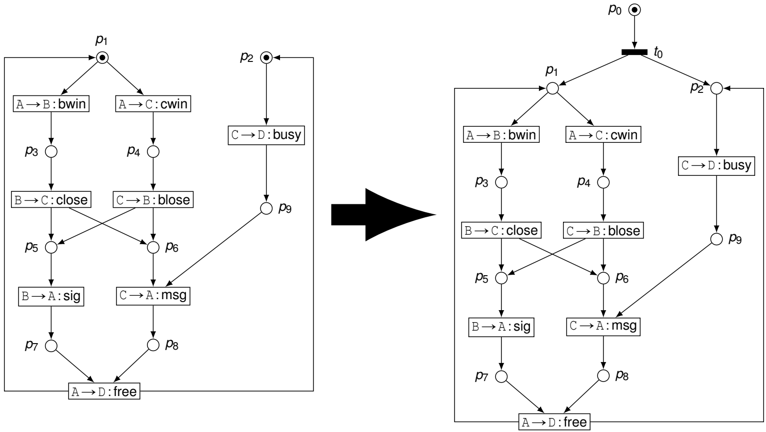




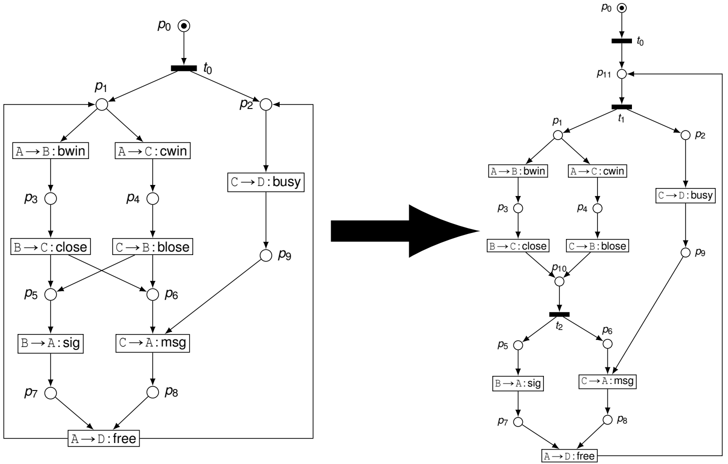
We use the work of Cortadella et al. (1998), based on the **theory of regions**, to derive a *safe* and *extended free-choice* Petri net from the Synchronous Transition System.

Imperial College
London

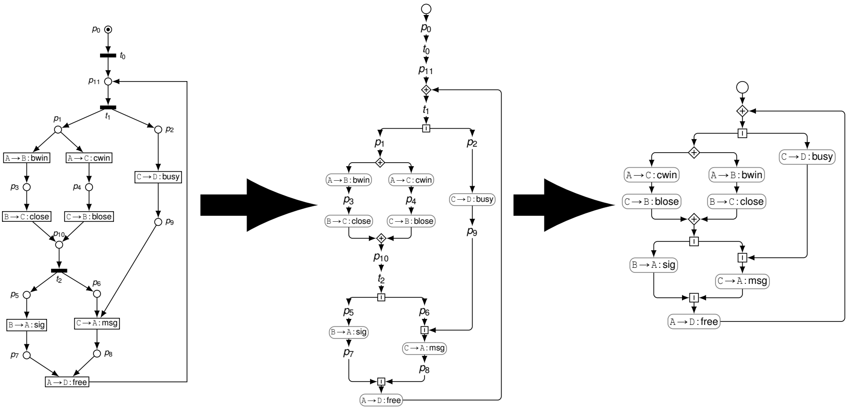
2: From Petri Net to One-source Petri Net



3: From One-source Petri Net to Joined Petri Net



4 & 5 : From Joined Petri Net to Global Graph



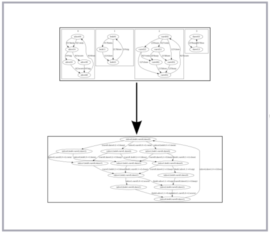
Prototype Implementation

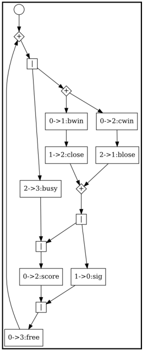
```
.outputs
.state graph
alice0 1 ! bwin alice1
alice0 2 ! cwin alice1
alice1 1 ? sig alice2
alice1 2 ! score alice3
alice2 2 ! score alice4
alice3 1 ? sig alice4
alice4 3 ! free alice0
.marking alice0
.end

.outputs
.state graph
bob0 0 ? bwin bob1
bob1 2 ! close bob2
bob0 2 ? blose bob2
bob2 0 ! sig bob0
.marking bob0
.end

.outputs
.state graph
carol0 1 ? close carol3
carol0 3 ! busy carol2
carol0 0 ? cwin carol1
carol1 3 ! busy carol4
carol1 1 ! blose carol3
carol2 1 ? close carol5
carol2 0 ? cwin carol4
carol3 3 ! busy carol5
carol4 1 ! blose carol5
carol5 0 ? score carol0
.marking carol0
.end

.outputs
.state graph
dave0 2 ? busy dave1
dave1 0 ? free dave0
.marking dave0
.end
```





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

Applications of choreography synthesis

1. *Timed systems*:

   Meeting Deadlines Together (CONCUR 2015)
   Laura Bocchi, Julien Lange & Nobuko Yoshida

2. *Deadlock detection in Go*:

   Static Deadlock Detection for Go by Global Session Graph Synthesis (CC 2016)
   Nicholas Ng & Nobuko Yoshida:

3. *Analysis of Erlang libraries*:

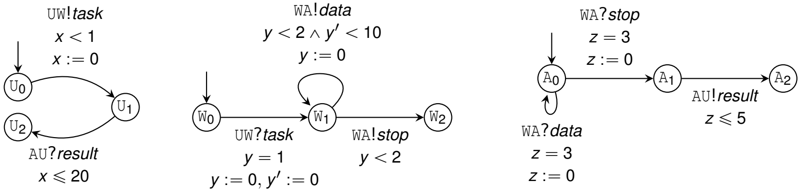
   Choreography-Based Analysis of Distributed Message Passing Programs (PDP 2016)
   Ramsay Taylor, Emilio Tuosto, Neil Walkinshaw, & John Derrick

Imperial College London

Meeting Deadlines Together (CONCUR 2015)

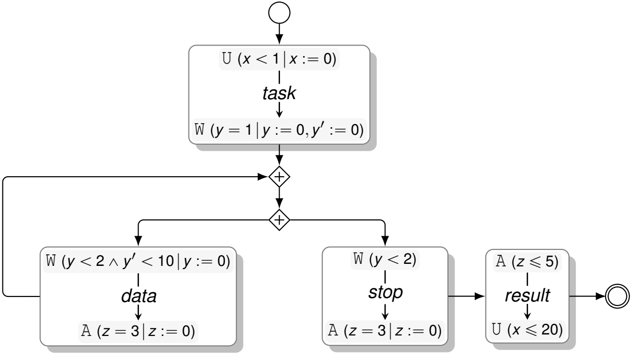
Laura Bocchi, Julien Lange & Nobuko Yoshida

From Communicating Timed Automata to Timed Choreographies



User ($\mathtt{U}$) Worker ($\mathtt{W}$) Aggregator ($\mathtt{A}$)

- Each machine owns several clocks (not shared)
- Time elapses at the same pace for each clock



- Construction as in the untimed setting
- Additional safety checks for time constraints

Imperial College London

Safety checks for CTAs

- MC: (Generalised) Multiparty Compatibility
- IE: Interaction Enabling
- CE: Cycle Enabling

Property	$S \sim (\mathrm{TS}(S)\downharpoonright_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$	Safety	Progress	Non-Zeno	Eventual Reception
MC	✓	✓	✗	✗	✗
MC+IE	✓	✓	✓	✗	✗
MC+CE	✓	✓	✗	✓	✗
MC+IE+CE	✓	✓	✓	✓	✓

L. Bocchi, J. Lange, and N. Yoshida. *Meeting Deadlines Together.*
More at `www.doc.ic.ac.uk/~jlange/cta/`

Static Deadlock Detection for Go by Global Session Graph Synthesis (CC 2016)

Nicholas Ng & Nobuko Yoshida:

Choreography Synthesis for Go (*i*)

Some Go facts:

- 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)

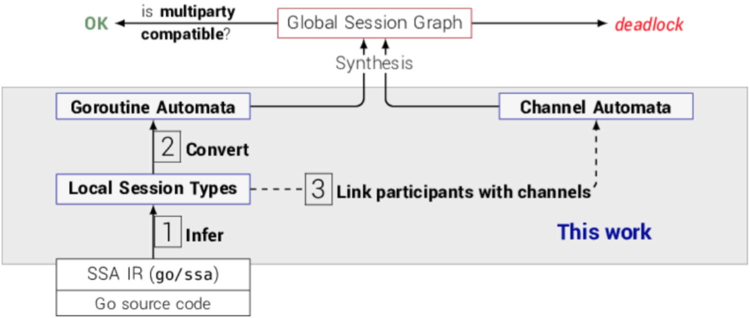
Choreography Synthesis for Go (*ii*)

```
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)
}
```

Go has a **runtime** deadlock detector, but it is very limited!

Choreography Synthesis for Go (*iii*)

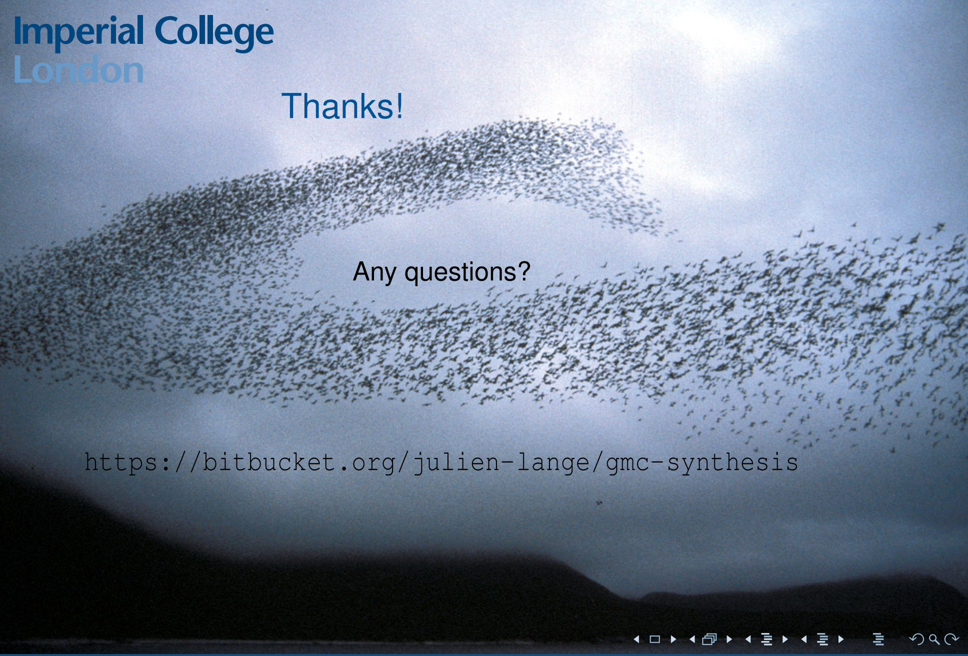


https://www.doc.ic.ac.uk/~cn06/pub/2016/dingo

Summing up

- An effective way of checking properties of CFSMs, and whether one can construct a global graph from them.
- An algorithm based on the theory of regions.
- A CFSMs characterisation of well-formed *generalised global types*.
- Applications:
  - An extension for communicating timed automata
  - Deadlock detection in Go
  - Analysis of Erlang libraries

Imperial College London



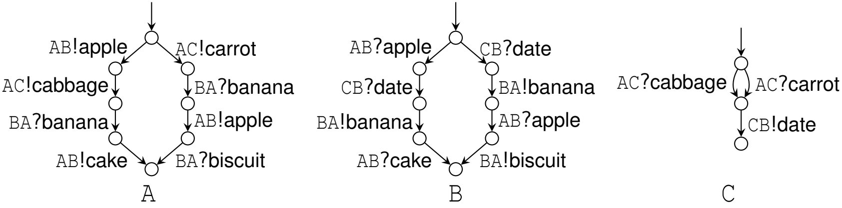
Thanks!

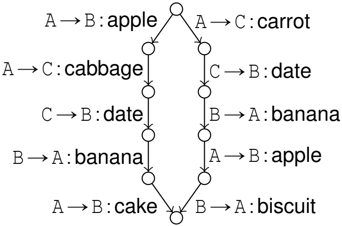
Any questions?

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

Imperial College
London

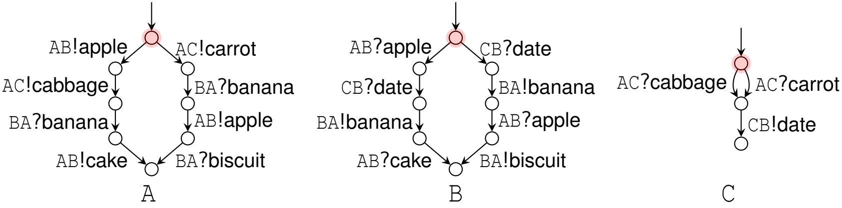
Violating the no-race condition

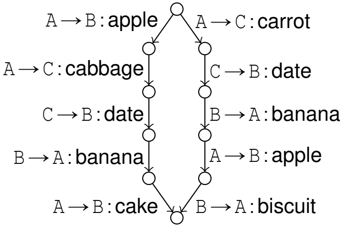




AB	:	ε
CB	:	ε
AC	:	ε
BA	:	ε

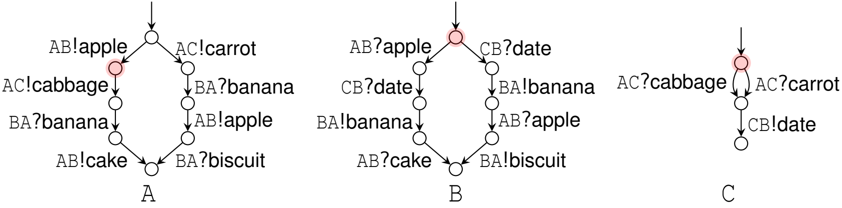
Violating the no-race condition

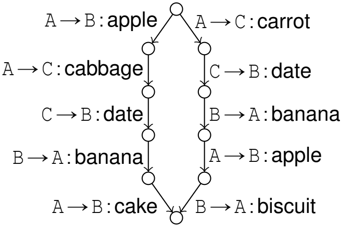




AB	:	ε
CB	:	ε
AC	:	ε
BA	:	ε

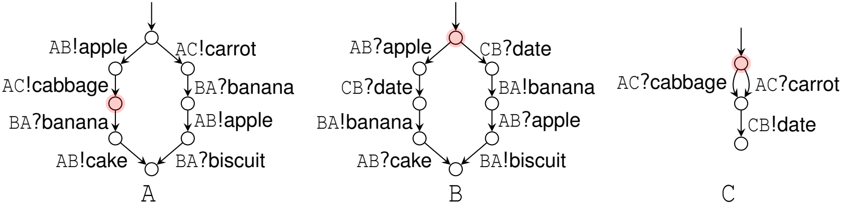
Violating the no-race condition

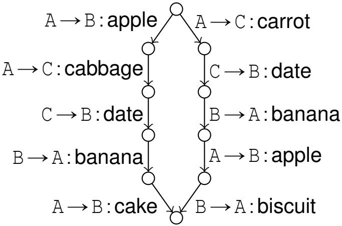




AB	:	apple
CB	:	ε
AC	:	ε
BA	:	ε

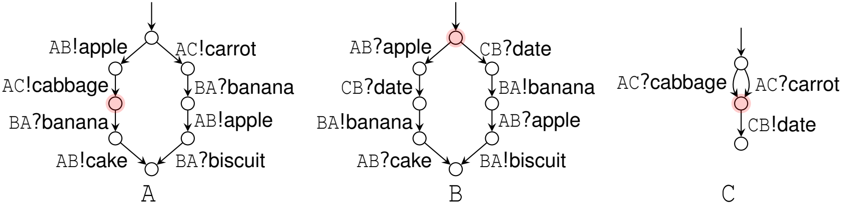
Violating the no-race condition

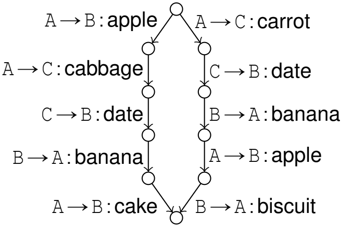




AB	:	apple
CB	:	ε
AC	:	cabbage
BA	:	ε

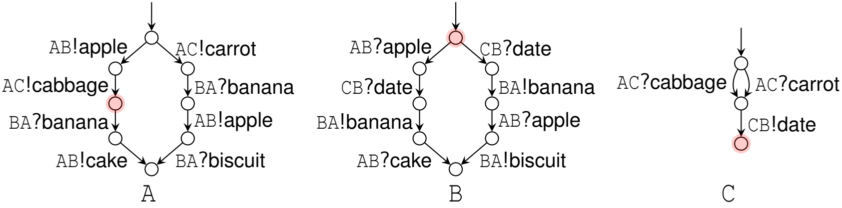
Violating the no-race condition

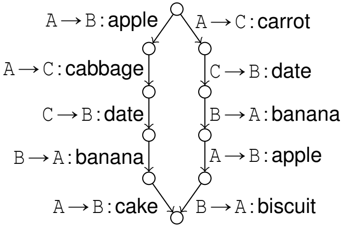




AB	:	apple
CB	:	ε
AC	:	ε
BA	:	ε

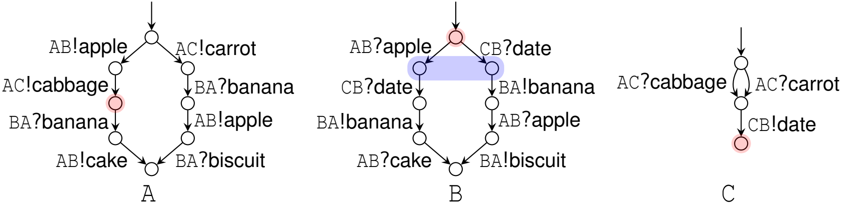
Violating the no-race condition

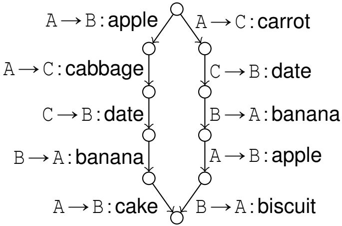




AB	:	apple
CB	:	date
AC	:	ε
BA	:	ε

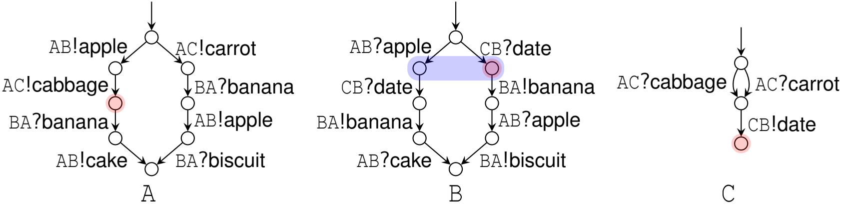
Violating the no-race condition

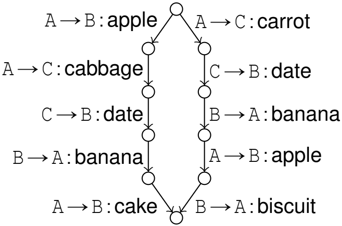




AB	:	apple
CB	:	date
AC	:	ε
BA	:	ε

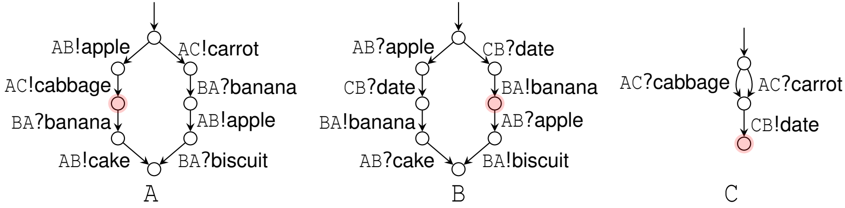
Violating the no-race condition

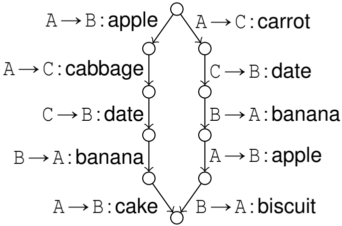




AB	:	apple
CB	:	ε
AC	:	ε
BA	:	ε

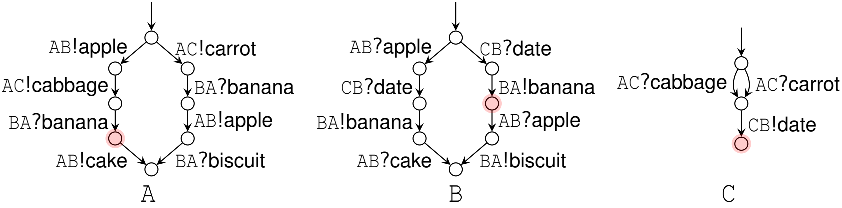
Violating the no-race condition

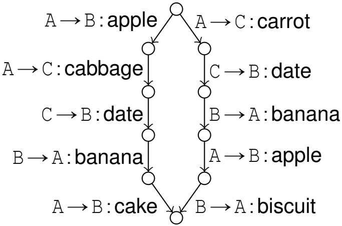




AB	:	apple
CB	:	ε
AC	:	ε
BA	:	banana

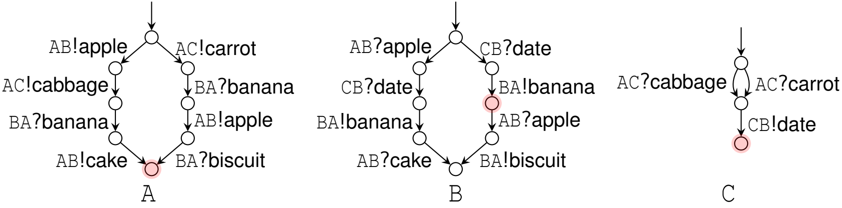
Violating the no-race condition

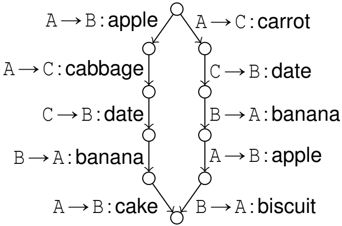




AB	:	apple
CB	:	ε
AC	:	ε
BA	:	ε

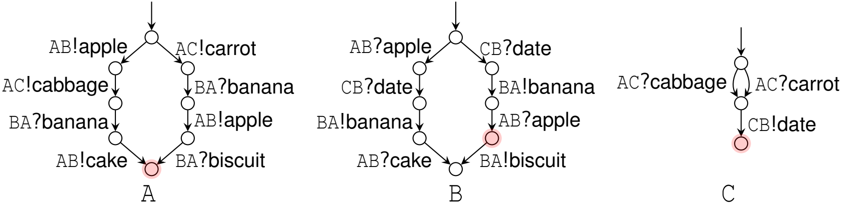
Violating the no-race condition

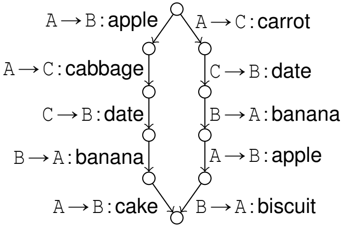




AB	:	apple · cake
CB	:	ε
AC	:	ε
BA	:	ε

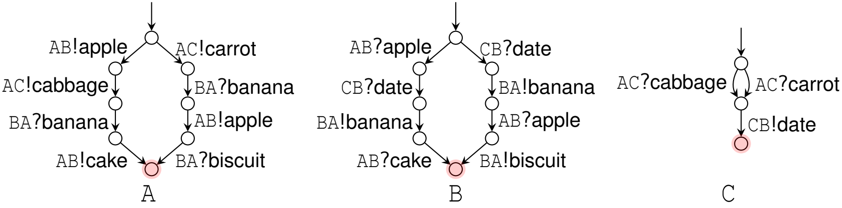
Violating the no-race condition

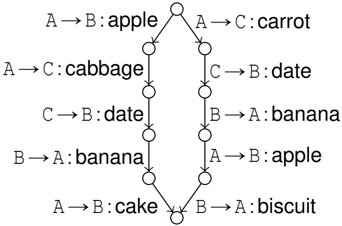




AB	:	cake
CB	:	ε
AC	:	ε
BA	:	ε

Violating the no-race condition

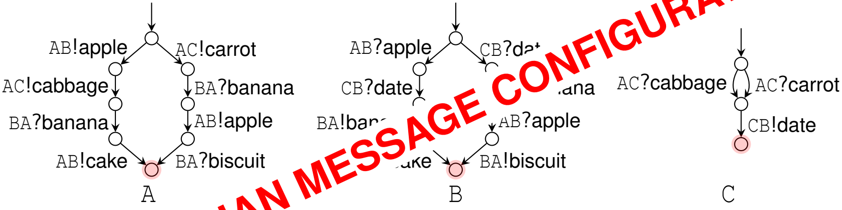


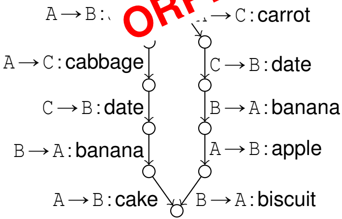


AB	:	cake
CB	:	ε
AC	:	ε
BA	:	biscuit

Violating the no-race condition

ORPHAN MESSAGE CONFIGURATION!

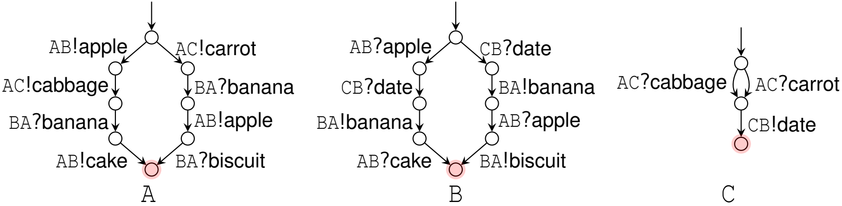


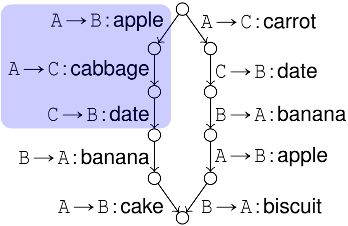


AB	:	cake
CB	:	ε
AC	:	ε
BA	:	biscuit

Imperial College
London

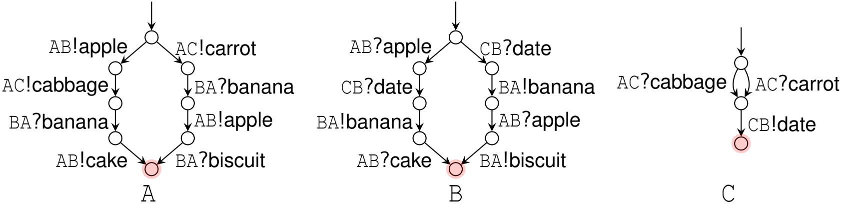
Violating the no-race condition

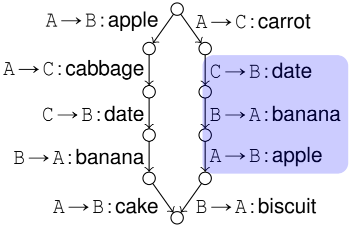




AB	:	cake
CB	:	ε
AC	:	ε
BA	:	biscuit

Violating the no-race condition





AB	:	cake
CB	:	ε
AC	:	ε
BA	:	biscuit

Imperial College London

References

- Julien Lange, Emilio Tuosto, and Nobuko Yoshida. “From Communicating Machines to Graphical Choreographies”. In: *POPL*. 2015
- Laura Bocchi, Julien Lange, and Nobuko Yoshida. “Meeting Deadlines Together”. In: *CONCUR*. 2015
- Ramsay Taylor et al. “Choreography-Based Analysis of Distributed Message Passing Programs”. In: *PDP 2016*
- Nicholas Ng and Nobuko Yoshida. “Static deadlock detection for concurrent go by global session graph synthesis”. In: *CC 2016*

Related work (*i*)

- Pierre-Malo Deniélou and Nobuko Yoshida. "Multiparty Session Types Meet Communicating Automata". In: *ESOP*. LNCS. Springer, 2012
- Julien Lange and Emilio Tuosto. "Synthesising Choreographies from Local Session Types". In: *CONCUR*. LNCS. Springer, 2012
- Pierre-Malo Deniélou and Nobuko Yoshida. "Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types". In: *ICALP*. LNCS. 2013
- Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. "Timed Multiparty Session Types". In: *CONCUR 2014*. 2014
- Pavel Krcal and Wang Yi. "Communicating Timed Automata: The More Synchronous, the More Difficult to Verify". In: *CAV*. LNCS. 2006

Related work (*ii*)

- Kohei Honda, Nobuko Yoshida, and Marco Carbone. “Multiparty asynchronous session types”. In: *POPL*. ACM, 2008
- Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. “Global Principal Typing in Partially Commutative Asynchronous Sessions”. In: *ESOP*. LNCS. Springer, 2009
- Samik Basu, Tevfik Bultan, and Meriem Ouederni. “Synchronizability for Verification of Asynchronously Communicating Systems”. In: *VMCAI*. LNCS. Springer, 2012
- Samik Basu, Tevfik Bultan, and Meriem Ouederni. “Deciding choreography realizability”. In: *POPL*. ACM, 2012