Reversible event structures, and Controlled and Uncontrolled Reversibility in CCSK

Eva Graversen & Nobuko Yoshida

April 29

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

Mobility Research Group

π-calculus, Session Types research at Imperial College

Home | People | Publications | Grants | Talks | Tutorials | Tools | Awards | Kohei Honda

NEWS

The paper *Multiparty asynchronous session types* by Kohei Honda, Nobuko Yoshida, and Marco Carbone, published in POPL 2008 has been awarded the ACM SIGPLAN Most Influential POPL Paper Award today at POPL 2018.

» more

10 Jan 2018

Estafet has published a page on their usage of the Scribble language developed in our group with RedHat and other industry partners.

» more

25 Sep 2017

Nick spoke at Golang UK 2017 on applying behavioural types to verify concurrent Go programs.

SELECTED PUBLICATIONS

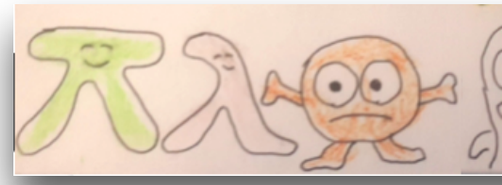
2018

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : A Static Verification Framework for Message Passing in Go using Behavioural Types. *To appear in* ICSE 2018 .

Bernardo Toninho , Nobuko Yoshida : Depending On Session Typed Process. *To appear in* FoSSaCS 2018 .

Bernardo Toninho , Nobuko Yoshida : On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings. *To appear in* ESOP 2018 .

Rumyana Neykova , Raymond Hu , Nobuko Yoshida , Fahd Abdeljallal : Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#. *To appear in* CC 2018 .



Post-docs:

Simon CASTEL

David CASTRO

Francisco FERR

Raymond HU

Rumyana NEY

Nicholas NG

Alceste SCALA

PhD Students:

Assel ALTAYEV

Juliana FRANC

Eva GRAVERSE

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

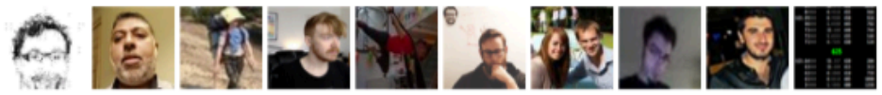
Interactions with Industries

CC'18

F#unctional Londoners Meetup G

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

ECOOP'17

…ributed Systems vs. Compositionality

Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

actyx

ECOOP'16

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)

Behavioural Type-Based Static Verification Framework for Go

Imperial College London

Home | College and Campus | Science | Engineering | Health | Business | Search here... | Go ▸

News

Go concurrency verification research at DoC grabs headline

POPL'17

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)".

ICSE'18

the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

A static verification framework for message passing in Go using behavioural types

JANUARY 25, 2018

tags: Concurrency, Programming Languages

A static verification framework for message passing in Go using behavioural types Lange et al., *ICSE 18*

With thanks to Alexis Richardson who first forwarded this paper to me.

We're jumping ahead to ICSE 18 now, and a paper that has been accepted for publication there later this year. It fits with the theme we've been exploring this week though, so I thought I'd cover it now. We've seen verification techniques applied in the context of **Rust** and **JavaScript**, looked at the integration of **linear types in Haskell**, and today it is the turn of Go!

Selected Publications 2017/2018

- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY, 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.

Selected Publications 2017/2018

- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types.
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION'17]** Keigo Imai, NY, 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.

Motivation

- Reversibility can be used for:
  - Modelling naturally reversible systems
  - Debugging
  - State-space exploration
- Event structures have been used to define semantics for CCS, π-calculus, LOTOS, etc. in forwards-only setting.
- Categorical definitions can help define choice and parallel composition.

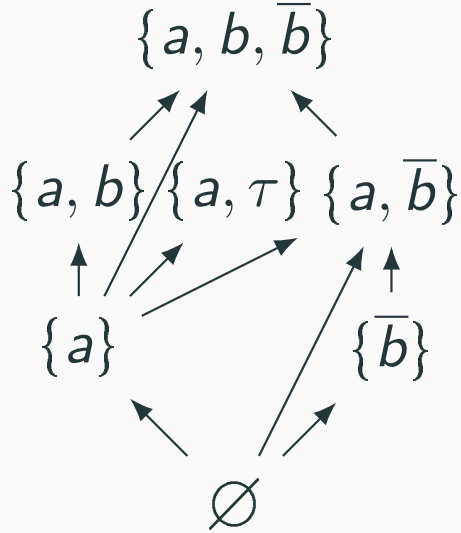
CCSK

CCSK (Phillips et al. 06) uses keys to denote past actions and which actions they have synchronised with:

$$a.b \mid \overline{a} \xrightarrow{\tau[m]} a[m].b \mid \overline{a}[m] \xrightarrow{b[n]} a[m].b[n] \mid \overline{a}[m] \overset{b[n]}{\rightsquigarrow} a[m].b \mid \overline{a}[m]$$

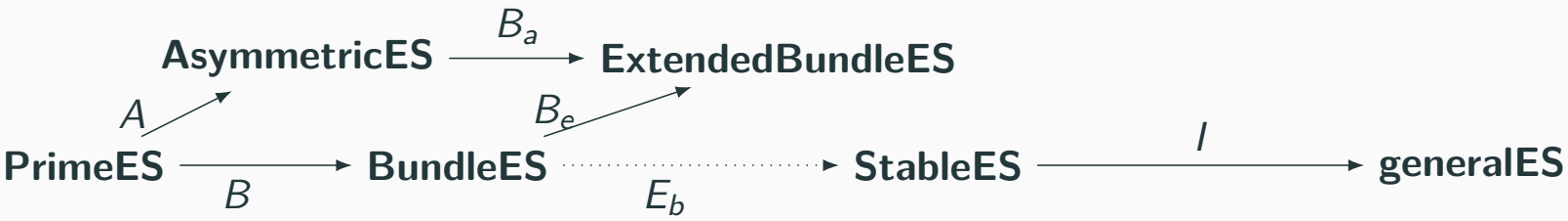
Forwards CCS Process as Prime Event Structure (Winskel 1982)

$a.b \mid \overline{b} \quad \rightarrow \quad E = \{a, b, \overline{b}, \tau\}, \; a < b, \; a < \tau, \; b \sharp \tau, \; \tau \sharp \overline{b} \quad \rightarrow$

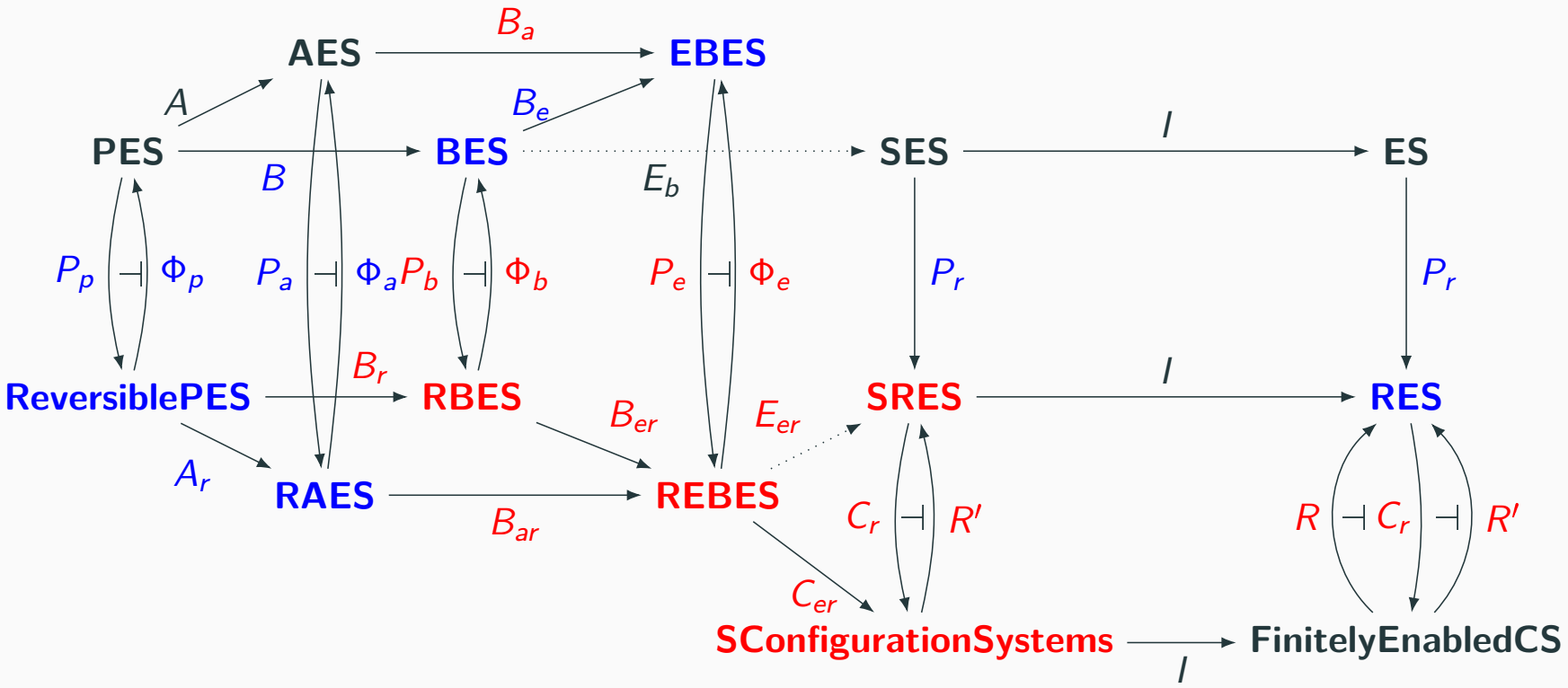


Process Event Structure Configurations

Forwards-only Category Overview



Category Overview



Previously defined Morphisms defined by us Entirely defined by us

Outline

Configuration Systems

Reversible Bundle Event Structures

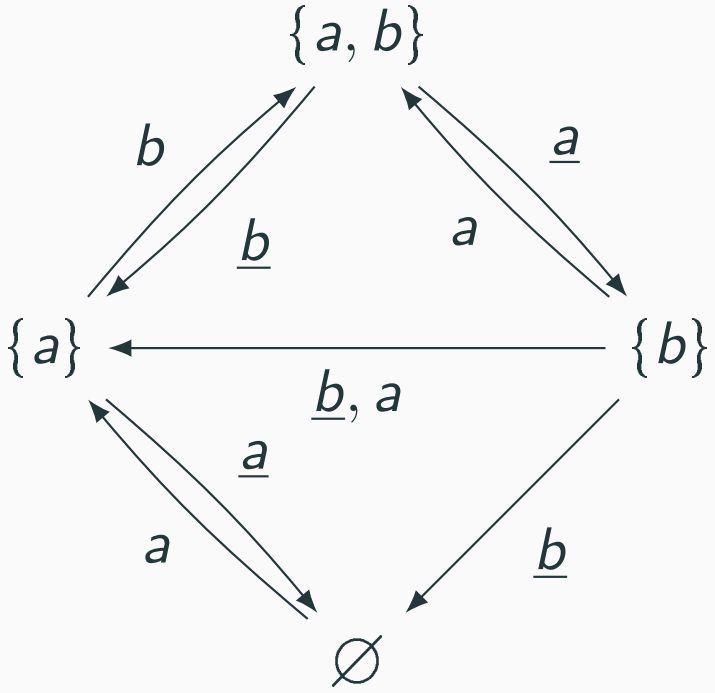
Event structure semantics of CCSK

Roll-CCSK

Configuration Systems

Configuration System

A configuration system has a set of events $E = \{a, b\}$, a set of reversible events $F = \{a, b\}$, which can be undone $\underline{a}$, $\underline{b}$, a set $\mathsf{C} \subseteq 2^E$ configurations of events, and a set $\rightarrow \subseteq \mathsf{C} \times (E \cup \underline{F}) \times \mathsf{C}$ transitions between them:



Morphism

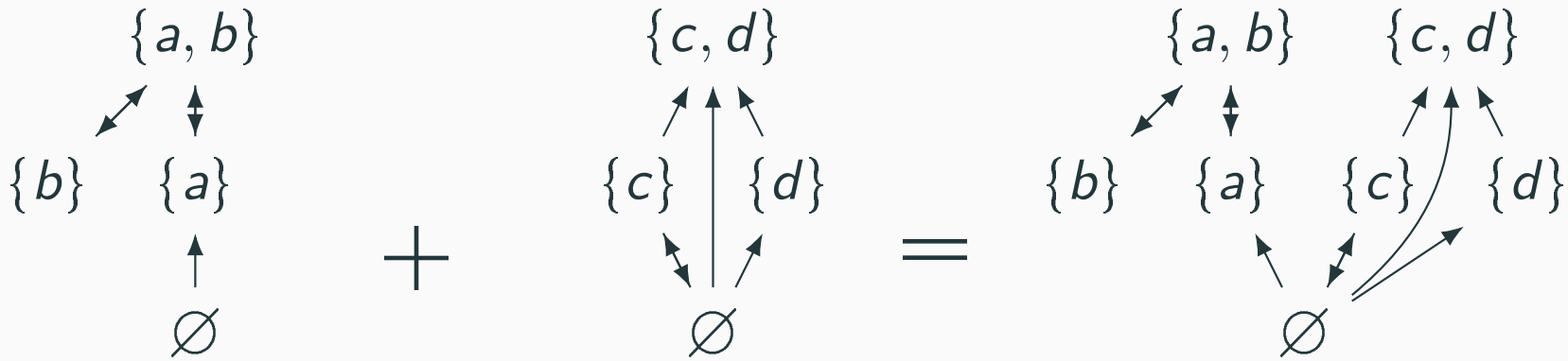
Definition (Configuration system-morphism)

Let $(E_0, F_0, \mathsf{C}_0, \rightarrow_0)$ and $(E_1, F_1, \mathsf{C}_1, \rightarrow_1)$ be configuration systems.

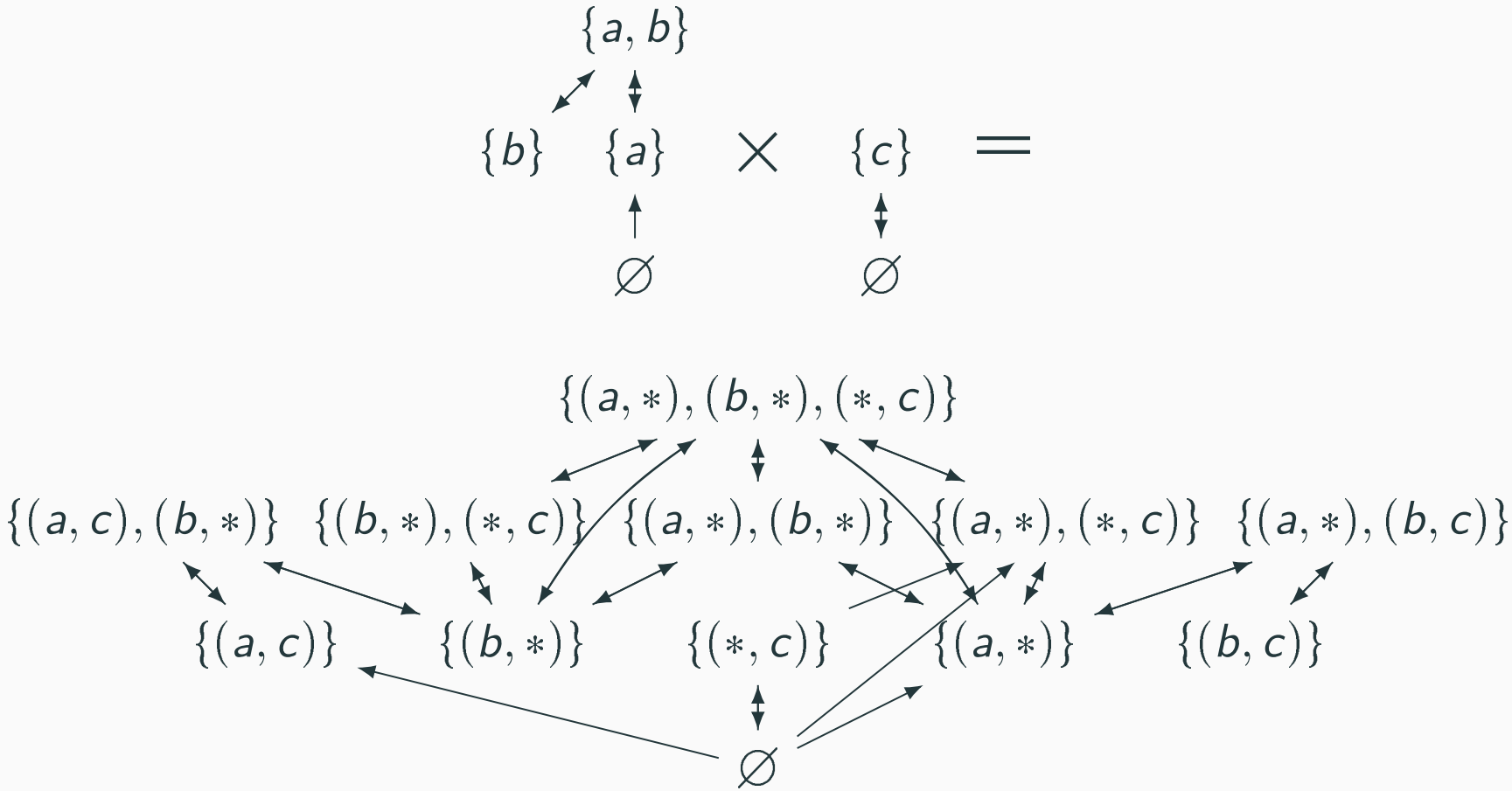
A *configuration system morphism* is a partial function $f : E_0 \rightarrow E_1$ such that

1. for any $X, Y \in \mathsf{C}_0$, $A \subseteq E_0$, and $B \subseteq F_0$, if $X \xrightarrow{A \cup \underline{B}}_0 Y$ then $f(X) \xrightarrow{f(A) \cup f(\underline{B})}_1 f(Y)$
2. for any $X \in \mathsf{C}_0$, $f(X) \in \mathsf{C}_1$
3. for all $e_0, e_0' \in E_0$, if $f(e_0) = f(e_0') \neq \perp$ and $e_0 \neq e_0'$ then there exists no $X \in \mathsf{C}_0$ such that $e_0, e_0' \in X$

Coproduct (Choice)



Partially Synchronous Product (Parallel Composition)



Reversible Bundle Event Structures

Reversible Bundle Event Structure

$\mathcal{E} = (E, F, \mapsto, \#, \rhd)$ where

$E = \{a, b, c\}$,

$F = \{a, b\}$,

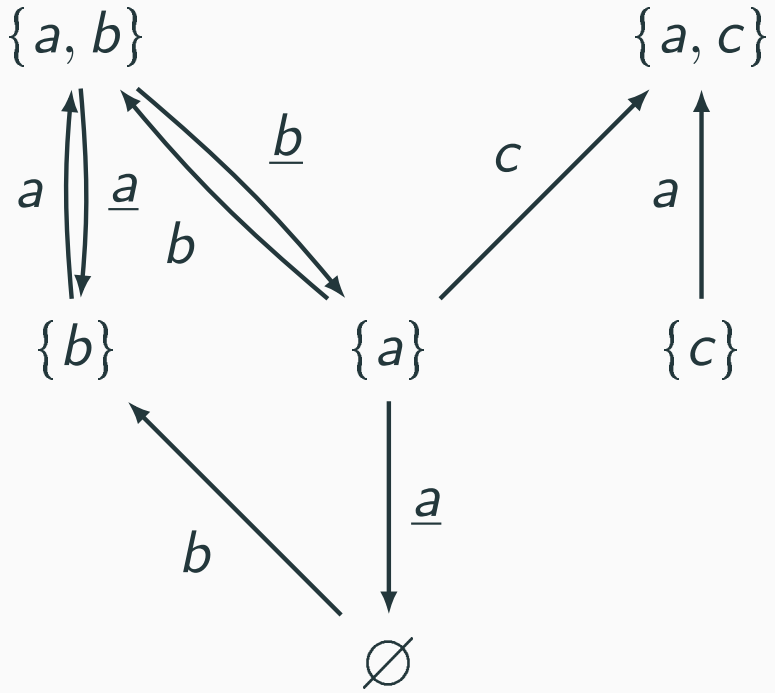
$\{c, b\} \mapsto a$,

$\{a\} \mapsto c$,

$b \# c$,

$\{a\} \mapsto \underline{b}$,

$c \rhd \underline{a}$



Causal

If a CS or RBES is causal, an event can be reversed if and only if every event caused by it has been reversed.

In a causal CS any reachable configuration is forwards-reachable.

Most reversible process calculi are causal.

The previous RBES and CS were not causal.

Causal CS and RBES

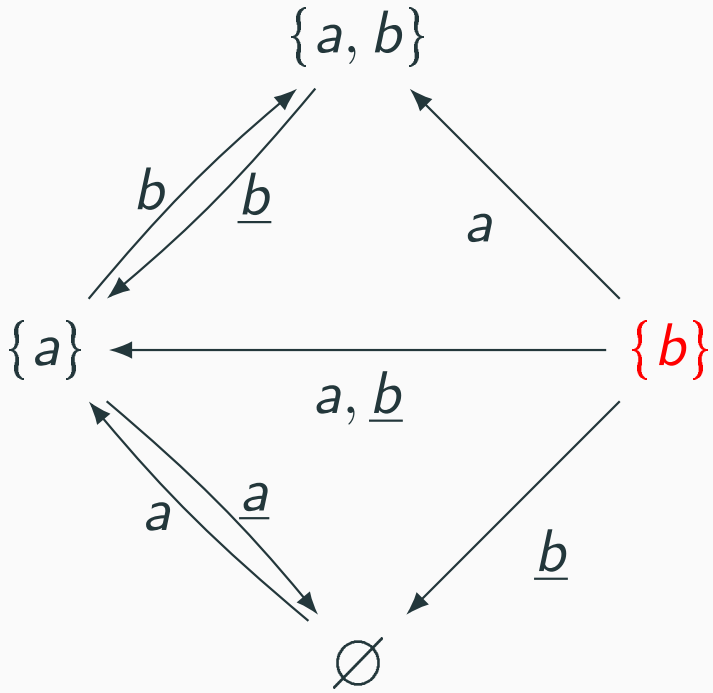
$\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ where

$E = \{a, b\}$

$F = \{a, b\}$

$\{a\} \mapsto b$,

$b \rhd \underline{a}$



Event structure semantics of CCSK

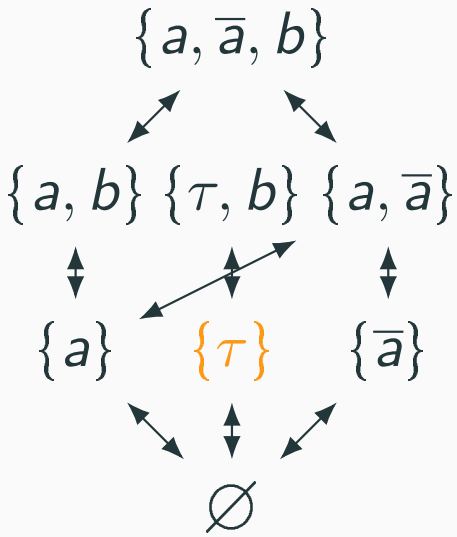
CCSK

CCSK uses keys to denote past actions and which actions they have synchronised with:

$$a.b \mid \overline{a} \xrightarrow{\tau[m]} a[m].b \mid \overline{a}[m] \xrightarrow{b[n]} a[m].b[n] \mid \overline{a}[m] \overset{b[n]}{\rightsquigarrow} a[m].b \mid \overline{a}[m]$$

CCSK Process as Reversible Bundle Event Structure

$$a[m].b \mid \overline{a}[m] \longrightarrow \begin{array}{l} E = \{a, \overline{a}, \tau, b\} \\ \{a, \tau\} \mapsto b \\ a \sharp \tau \sharp \overline{a} \\ b \rhd \underline{a} \\ b \rhd \underline{\tau} \end{array} \longrightarrow$$



Process Event Structure Configurations

Roll-CCSK

Rollback

Rollback (Lanese et al., 11) is a way to control when the process reverses, only letting tagged actions and actions caused by them reverse when the roll is reached:

$$a[m].b[n] \mid \overline{a}_\gamma[m].c[o]\text{roll } \gamma \overset{\text{roll } \gamma}{\rightsquigarrow} a.b \mid \overline{a}_\gamma.c.\text{roll } \gamma$$

Rollback

Rollback (Lanese et al., 11) is a way to control when the process reverses, only letting tagged actions and actions caused by them reverse when the roll is reached:

$$a[m].b[n] \mid \overline{a}_\gamma[m].c[o]\text{roll } \gamma \overset{\text{roll } \gamma}{\rightsquigarrow} a.b \mid \overline{a}_\gamma.c.\text{roll } \gamma$$

Using this method,

$$(a_\gamma.(\overline{d}.0 \mid c.\text{roll } \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\text{roll } \gamma') \mid \overline{a} \mid \overline{b})\backslash\{a, b, c, d\}$$

is not able to roll all the way back to the beginning, as executing one roll undoes an action preceding the other.

We split the roll into two actions, one triggering the roll, which is reversed when a previous action is rolled back, and one executing the roll:

$$(a_\gamma[m].(\overline{d}[n].0 \mid c[o].\text{roll } \gamma) \mid b_{\gamma'}[p].(\overline{c}[o] \mid d[n].\text{roll } \gamma') \mid \overline{a}[m] \mid \overline{b}[p])\backslash\{a, b, c, d\}$$

start roll γ

$$(a_\gamma[m].(\overline{d}[n].0 \mid c[o].\text{rolling } \gamma) \mid b_{\gamma'}[p].(\overline{c}[o] \mid d[n].\text{roll } \gamma') \mid \overline{a}[m] \mid \overline{b}[p])\backslash\{a, b, c, d\}$$

start roll γ'

$$(a_\gamma[m].(\overline{d}[n].0 \mid c[o].\text{rolling } \gamma) \mid b_{\gamma'}[p].(\overline{c}[o] \mid d[n].\text{rolling } \gamma') \mid \overline{a}[m] \mid \overline{b}[p])\backslash\{a, b, c, d\}$$

roll γ'

$$(a_\gamma[m].(\overline{d}.0 \mid c.\text{rolling } \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\text{roll } \gamma') \mid \overline{a}[m] \mid \overline{b})\backslash\{a, b, c, d\}$$

roll γ

$$(a_\gamma.(\overline{d}.0 \mid c.\text{roll } \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\text{roll } \gamma') \mid \overline{a} \mid \overline{b})\backslash\{a, b, c, d\}$$

Roll-CCSK Process as Event Structure

$a_\gamma.\text{roll } \gamma \rightarrow$

$E = \{a, \text{start roll } \gamma, \text{roll } \gamma\}$

$\{a\} \mapsto \text{start roll } \gamma$

$\{\text{start roll } \gamma\} \mapsto \text{roll } \gamma$

$\{\text{roll } \gamma\} \mapsto \underline{\text{start roll } \gamma}$

$\{\text{roll } \gamma\} \mapsto \underline{a}$

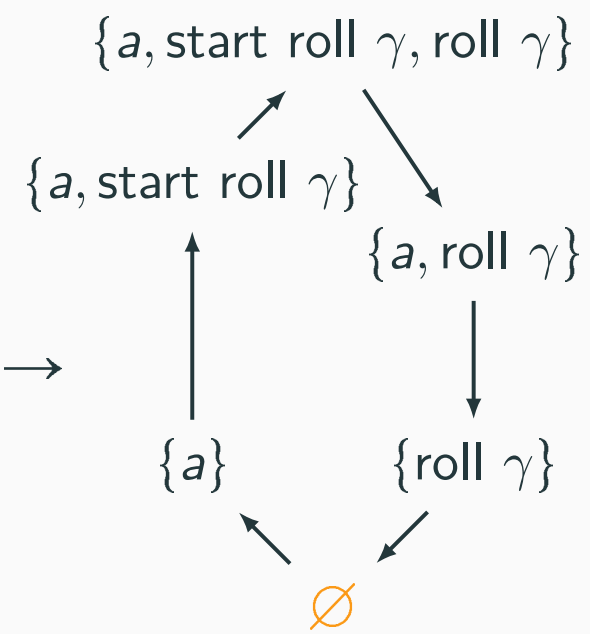
$a \rhd \underline{\text{roll } \gamma}$

$\text{start roll } \gamma \rhd \underline{a}$

$\text{roll } \gamma \rhd a$

$\text{roll } \gamma \rhd \text{start roll } \gamma$

$\rightarrow$



Process Event Structure Configurations

Splitting events in Roll-CCSK

Sometimes reversal of events is caused by multiple events combining. To model this, events must be spilt.

$$a[m].b[n] \mid \overline{a}_\gamma[m].\text{rolling } \gamma \xrightarrow{\text{roll } \gamma} a.b \mid \overline{a}_\gamma.\text{roll } \gamma$$
$$a[m'].b[n] \mid \overline{a}_\gamma[m].\text{rolling } \gamma \xrightarrow{\text{roll } \gamma} a[m'].b[n] \mid \overline{a}_\gamma.\text{roll } \gamma$$

Splitting events in Roll-CCSK

Sometimes reversal of events is caused by multiple events combining. To model this, events must be spilt.

$$a[m].b[n] \mid \overline{a}_\gamma[m].\text{rolling } \gamma \xrightarrow{\text{roll } \gamma} a.b \mid \overline{a}_\gamma.\text{roll } \gamma$$
$$a[m'].b[n] \mid \overline{a}_\gamma[m].\text{rolling } \gamma \xrightarrow{\text{roll } \gamma} a[m'].b[n] \mid \overline{a}_\gamma.\text{roll } \gamma$$

This requires splitting b as it is only rolled back by roll γ if the as synchronised, meaning

$\{a\} \mapsto b_a$
$\{\tau\} \mapsto b_\tau$
$\{\text{roll } \gamma\} \mapsto \underline{b_\tau}$

Result

Given a process P, which generates an event structure $\mathcal{E}$ and an initial state Init,

1. $P \xrightarrow{\mu} P'$ for a P' which generates an event structure $\mathcal{E}'$ and initial state Init$'$, if and only if there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ and an event e in $\mathcal{E}$ that is enabled in Init and labelled μ, such that $f(\text{Init} \cup \{e\}) = \text{Init}'$
2. $P \xrightarrow{\text{roll } \gamma} P'$ for a P' which generates an event structure $\mathcal{E}'$ and initial state Init$'$, if and only if there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ and a sequence of transitions in the configuration system of $\mathcal{E}$, $\text{Init} \xrightarrow{e_0} X_0 \overset{e_1}{\rightsquigarrow} \ldots X_{n-1} \overset{e_n}{\rightsquigarrow} X_n \overset{e_0}{\rightsquigarrow} X$ such that e_0 is labelled roll γ and $f(X) = \text{Init}'$

Conclusion

- Causal reversible bundle event structures can describe the semantics of CCSK
- Roll-CCSK extends CCSK to control reversibility by using rollback
- Non-causal reversible extended bundle event structures can model Roll-CCSK.

Future Work

π-calculus: $(\nu n)(\overline{a}\langle n\rangle \mid \overline{b}\langle n\rangle \mid n(y))$

- Non-structural causation
- Traditionally non-stable
- Past actions must be stored in separate memories