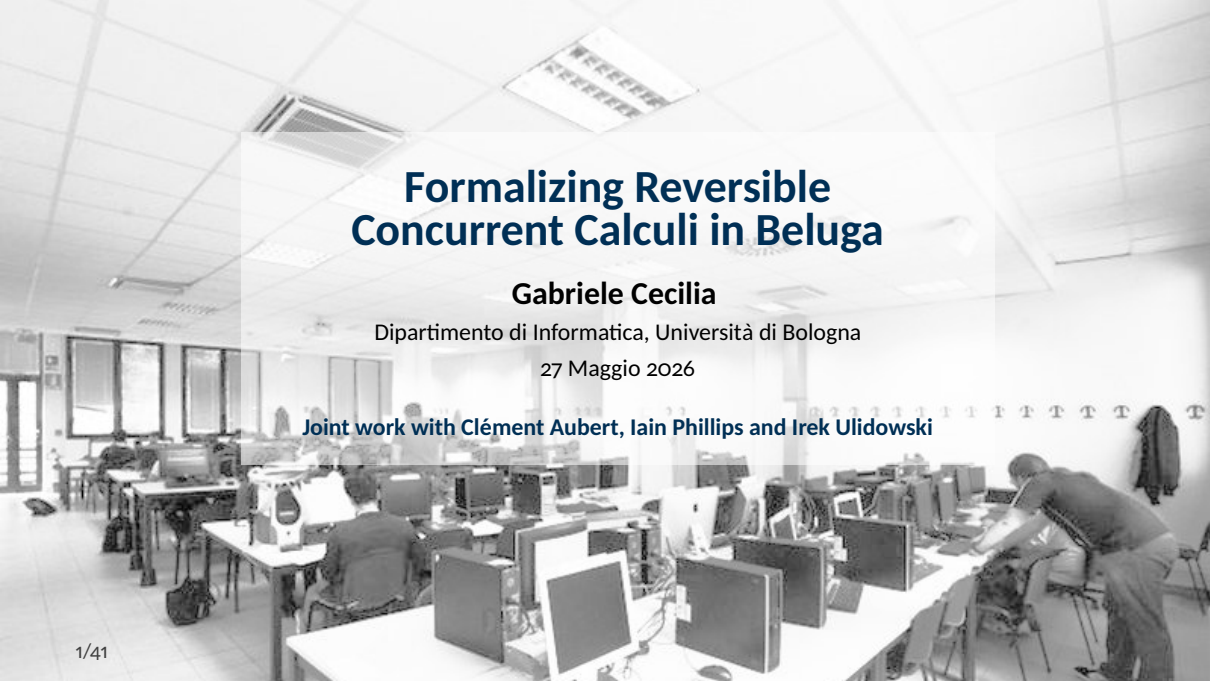




Formalizing Reversible Concurrent Calculi in Beluga

Gabriele Cecilia

Dipartimento di Informatica, Università di Bologna

27 Maggio 2026

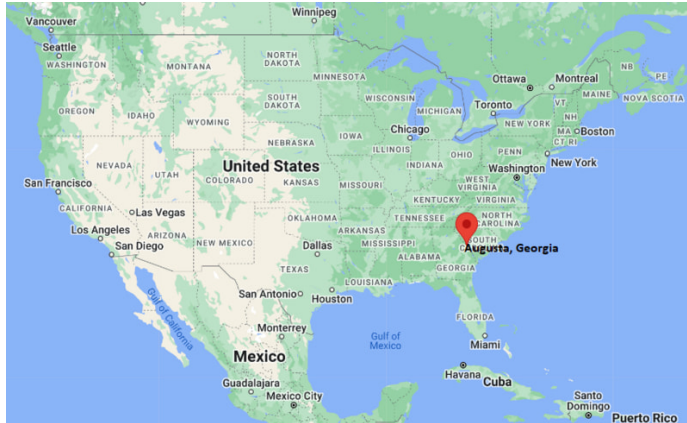
Joint work with Clément Aubert, Iain Phillips and Irek Ulidowski

A brief personal introduction

Ph.D. student in Computer Science
since January 2025



A brief personal introduction



A brief personal introduction



A brief personal introduction



A brief personal introduction



Dr. Clément Aubert
Associate Professor in Computer Science

My research area

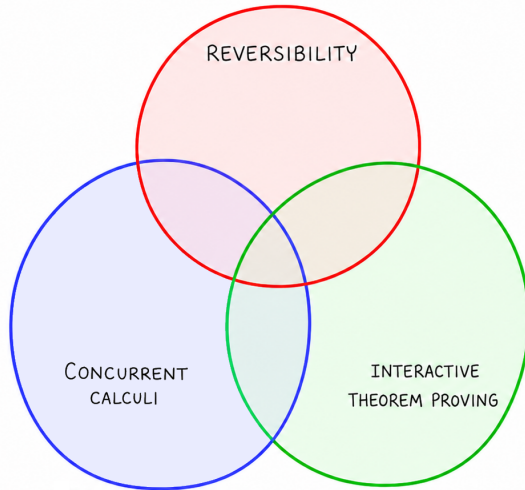


Table of Contents

► Introduction

► CCSK and CCSK^P

► Beluga Formalization

- Closed and open transitions
- The axiomatic approach for CCSK

► Conclusions

In the previous talk...

π -calculus: an example of *concurrent calculus*

In the previous talk...

π -calculus: an example of *concurrent calculus*

Concurrent Calculi: abstract models of concurrent systems



In the previous talk...

π -calculus: an example of *concurrent calculus*

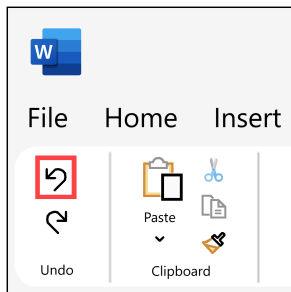
In this talk: **reversible** concurrent calculi

What is reversibility?

The ability to undo actions/computations, restoring the system to a previous state

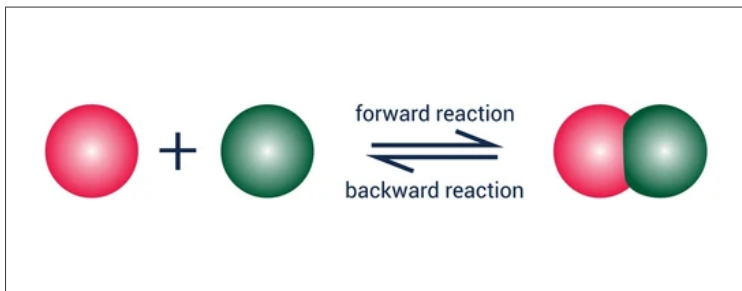
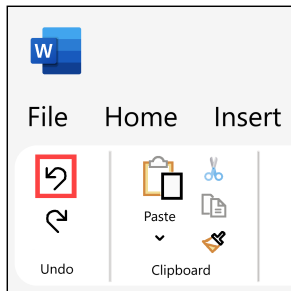
What is reversibility?

The ability to undo actions/computations, restoring the system to a previous state



What is reversibility?

The ability to undo actions/computations, restoring the system to a previous state



Why reversibility?

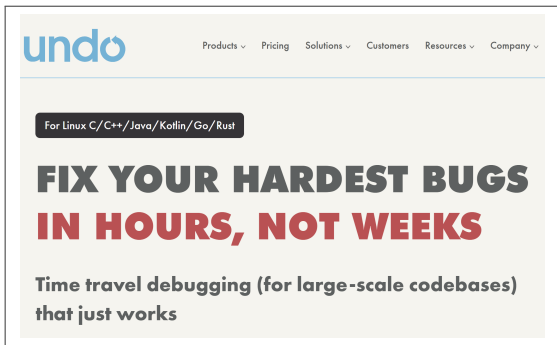
- It allows identifying causal dependencies on processes and actions

Why reversibility?

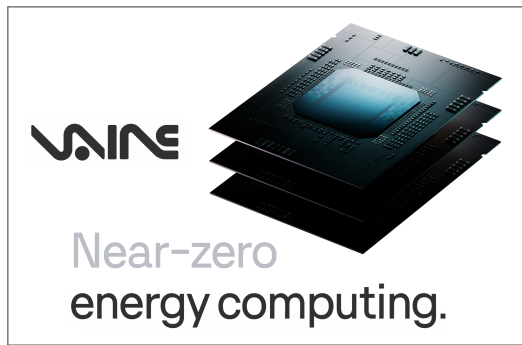
- It allows identifying causal dependencies on processes and actions
- Landauer's principle: irreversible computation implies heat dissipation

Why reversibility?

- It allows identifying causal dependencies on processes and actions
- Landauer's principle: irreversible computation implies heat dissipation



The screenshot shows the top of the Undo website. At the top left is the 'undo' logo in blue. To its right is a navigation menu with links: 'Products', 'Pricing', 'Solutions', 'Customers', 'Resources', and 'Company', each followed by a small downward arrow. Below the navigation bar is a dark grey button with white text that reads 'For Linux C/C++/Java/Kotlin/Go/Rust'. The main headline is 'FIX YOUR HARDEST BUGS' in large, bold, dark grey letters, followed by 'IN HOURS, NOT WEEKS' in large, bold, red letters. Below the headline is a sub-headline in smaller, dark grey text: 'Time travel debugging (for large-scale codebases) that just works'.



The image shows the VINE logo on the left, which consists of a stylized 'V' followed by the word 'VINE' in a bold, sans-serif font. To the right of the logo is a 3D rendering of three stacked circuit boards, with the top board featuring a glowing blue square component. Below the logo and image, the text 'Near-zero energy computing.' is written in a large, grey, sans-serif font.

Reversibility + Concurrent calculi

Reversible concurrent calculi: abstract models of concurrent systems where every action can be undone

Reversibility + Concurrent calculi

Reversible concurrent calculi: abstract models of concurrent systems where **every action can be undone**

Challenges addressed:

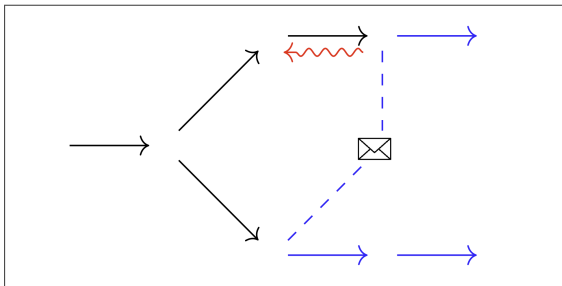
- Recovering inputs from outputs in a memory-efficient way

Reversibility + Concurrent calculi

Reversible concurrent calculi: abstract models of concurrent systems where **every action can be undone**

Challenges addressed:

- Recovering inputs from outputs in a memory-efficient way
- Complexity of undoing computations in concurrent systems



Reversibility + Concurrent calculi

How to make a concurrent calculus reversible? Two approaches:

- Tag actions with **communication keys** and store them in processes
→ CCSK (CCS with Keys), CCSK^P (CCS with Keys and Proof labels)

Reversibility + Concurrent calculi

How to make a concurrent calculus reversible? Two approaches:

- Tag actions with **communication keys** and store them in processes
→ CCSK (CCS with Keys), CCSK^P (CCS with Keys and Proof labels)
- Equip threads with a **memory stack**, storing information to backtrack
→ RCCS (Reversible CCS)

“An Axiomatic Theory for Reversible Computation”, I. Lanese, I. Phillips and I. Ulidowski

Definition 3.1 (Basic Axioms). We say an LTSI satisfies

SP if whenever $t : P \xrightarrow{\alpha} Q$, $u : P \xrightarrow{\beta} R$ with $t \iota u$ then there are cofinal transitions $u' : Q \xrightarrow{\beta} S$ and $t' : R \xrightarrow{\alpha} S$;

BTI if whenever $t : P \xrightarrow{a} Q$ and $t' : P \xrightarrow{b} Q'$ and $t \neq t'$ then $t \iota t'$;

WF if there is no infinite reverse computation—that is, we do not have P_i (not necessarily distinct) such that $P_{i+1} \xrightarrow{a_i} P_i$ for all $i = 0, 1, \dots$

“An Axiomatic Theory for Reversible Computation”, I. Lanese, I. Phillips and I. Ulidowski

Definition 3.1 (Basic Axioms). We say an LTSI satisfies

SP if whenever $t : P \xrightarrow{\alpha} Q$, $u : P \xrightarrow{\beta} R$ with $t \iota u$ then there are cofinal transitions $u' : Q \xrightarrow{\beta} S$ and $t' : R \xrightarrow{\alpha} S$;

BTI if whenever $t : P \xrightarrow{a} Q$ and $t' : P \xrightarrow{b} Q'$ and $t \neq t'$ then $t \iota t'$;

WF if there is no infinite reverse computation—that is, we do not have P_i (not necessarily distinct) such that $P_{i+1} \xrightarrow{a_i} P_i$ for all $i = 0, 1, \dots$

→ To prove the soundness of a reversible system, verifying the basic axioms and properties is enough. All other results are inherited for free

Interactive theorem provers

- Computer programs for constructing machine-checked mathematical proofs
- Scope: programming languages, compilers, software, mathematics, ...
- Guarantee the correctness of definitions and proofs, proof search, increase comprehension of systems

Concurrent calculi formalizations

Author, Year	Publication	Technique
Nesi 94	<i>A Formalization of the Process Algebra CCS in HOL</i>	Named syntax
Melham 94	<i>A Mechanized Theory of the π-Calculus in HOL</i>	Named syntax
Hirschhoff 97	<i>A Full Formalisation of π-Calculus Theory in the Calculus of Constructions</i>	De Bruijn indices
Miller et al. 99	<i>Foundational Aspects of Syntax</i>	HOAS
Honsell et al. 01	<i>π-Calculus in (Co)Inductive Type Theory</i>	HOAS
Bengtson et al. 09	<i>Formalizing the π-Calculus using Nominal Logic</i>	Nominal logic

→ Concurrent Calculi Formalisation Benchmark, 2024

Reversible concurrent calculi formalizations

- *A Formalization of the Reversible Concurrent Calculus CCSK^P in Beluga*, Cecilia, 2025

Reversible concurrent calculi formalizations

- *A Formalization of the Reversible Concurrent Calculus CCSK^P in Beluga*, Cecilia, 2025

However, more papers are upcoming:

- *A Lean Mechanization of Reversible Occurrence Nets*, Dávalos and Melgratti, 2026
- *Characterising and Formalising Independence and Causality in (Reversible) Concurrent Calculi*, Aubert, Cecilia, Phillips and Ulidowski
- *A mechanization of An Axiomatic Theory for Reversible Computation*, Lanese

Table of Contents

► Introduction

► CCSK and CCSK^P

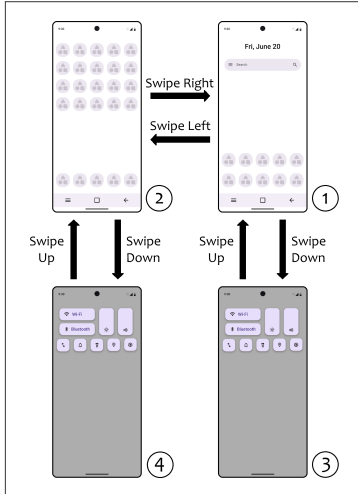
► Beluga Formalization

- Closed and open transitions
- The axiomatic approach for CCSK

► Conclusions

Example: smartphone

Example: smartphone



Representation in CCS:

$$P_1 \stackrel{\text{def}}{=} \text{left}.P_2 + \text{down}.P_3$$

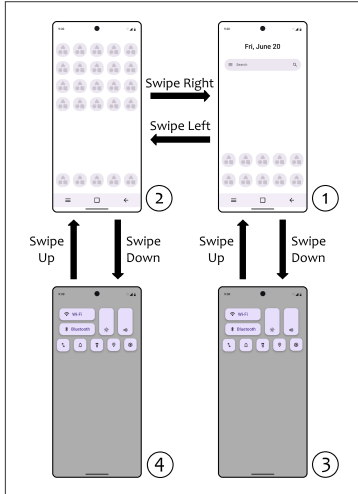
$$P_2 \stackrel{\text{def}}{=} \text{right}.P_1 + \text{down}.P_4$$

$$P_3 \stackrel{\text{def}}{=} \text{up}.P_1$$

$$P_4 \stackrel{\text{def}}{=} \text{up}.P_2$$

$$P_1 \xrightarrow{\text{left}} P_2$$

Example: smartphone



Representation in CCSK and CCSK^P:

$X = \text{left.down} + \text{down}$

$X \xleftrightarrow{\quad} \text{left}[k].\text{down} + \text{down}$

Example: smartphone

In CCSK:

$$X \xrightarrow{\text{left}[k]} \text{left}[k].\text{down} + \text{down}$$



In CCSK^P:

$$X \xrightarrow{+_L \text{left}[k]} \text{left}[k].\text{down} + \text{down}$$



Syntax

- Infinite set of **names** N : $a, b, \dots$
 - Complementary names $\bar{N}$: $\bar{a}, \bar{b}, \dots$
 - Symbol for interactions τ
 - Infinite set of **keys**: $k, m, \dots$
- } **Labels**, ranged over by $\alpha, \beta, \dots$

Syntax

- Infinite set of **names** N : $a, b, \dots$
 - Complementary names $\bar{N}$: $\bar{a}, \bar{b}, \dots$
 - Symbol for interactions τ
 - Infinite set of **keys**: $k, m, \dots$
- } **Labels**, ranged over by $\alpha, \beta, \dots$

Definition

The set of **processes** is defined by the following syntax:

$$X, Y ::= \mathbf{0} \mid \alpha.X \mid \alpha[k].X \mid X + Y \mid (X \mid Y) \mid X \setminus a$$

Semantics

Given by a **Labelled Transition System** (LTS).

Transitions in CCSK are labelled by *keyed labels*.

Transitions in CCSK^P are labelled by *proof labels*.

Semantics

Definition

Keyed labels, denoted as $\alpha[k]$, are pairs of a label and a key.

The set of **proof labels** is defined by the following syntax:

$$\theta ::= v\alpha[k] \quad | \quad v\langle |_{\text{L}}v_1\lambda[k], |_{\text{R}}v_2\bar{\lambda}[k] \rangle$$

where λ ranges over $\mathbf{N} \cup \overline{\mathbf{N}}$ and v, v_1 and v_2 range over strings of symbols $\{|_{\text{L}}, |_{\text{R}}, +_{\text{L}}, +_{\text{R}}\}$.

One can also define the label $\ell(\theta)$ and the key $k(\theta)$ of a proof label θ .

Semantics

Definition

Semantics is given by a **combined LTS**, made of the union of forward ($\mapsto$) and backward ($\rightsquigarrow$) transition rules such as the following:

$$\text{std}(X) \frac{}{\alpha.X \mapsto^{\alpha[k]} \alpha[k].X} \text{pref}$$

$$\text{std}(X) \frac{}{\alpha[k].X \rightsquigarrow^{\alpha[k]} \alpha.X} \text{pref}$$

$$\mathcal{K}(\theta) \neq k \frac{X \xrightarrow{\theta} X'}{\alpha[k].X \xrightarrow{\theta} \alpha[k].X'} \text{kpref}$$

$$\mathcal{K}(\theta) \neq k \frac{X' \rightsquigarrow^{\theta} X}{\alpha[k].X' \rightsquigarrow^{\theta} \alpha[k].X} \text{kpref}$$

Notation:

Given a combined transition $t: X \xrightarrow{\theta} X'$, X and X' are said the *source* and *target* of t .

Loop lemma

By construction, the LTSes of CCSK and CCSK^P satisfy the loop lemma: each transition can be reversed

Loop lemma

$$X \xrightarrow{a} Y \text{ iff } Y \xrightarrow{\bar{a}} X$$

We denote as $\bar{t}$ the inverse transition of t .

Bijection between CCSK and CCSK^P

Definition

We define the *proof forgetful* ($\cdot^\circ$) and *proof enrichment* ($\cdot^\dagger$) mappings between transitions as follows:

$$\cdot^\circ : (X_1 \xrightarrow{\theta} X_2) \mapsto (X_1 \xrightarrow{\ell(\theta)[\mathcal{K}(\theta)]} X_2) \quad (\text{CCSK}^P \text{ to CCSK})$$

$$\cdot^\dagger : (X_1 \xrightarrow{\alpha[k]} X_2) \mapsto (X_1 \xrightarrow{\theta} X_2) \quad \text{s.t. } \ell(\theta) = \alpha, \mathcal{K}(\theta) = k \quad (\text{CCSK to CCSK}^P)$$

Bijection between CCSK and CCSK^P

Definition

We define the *proof forgetful* ($\cdot^\circ$) and *proof enrichment* ($\cdot^\dagger$) mappings between transitions as follows:

$$\cdot^\circ : (X_1 \xrightarrow{\theta} X_2) \mapsto (X_1 \xrightarrow{\ell(\theta)[\mathcal{K}(\theta)]} X_2) \quad (\text{CCSK}^P \text{ to CCSK})$$

$$\cdot^\dagger : (X_1 \xrightarrow{\alpha[k]} X_2) \mapsto (X_1 \xrightarrow{\theta} X_2) \quad \text{s.t. } \ell(\theta) = \alpha, \mathcal{K}(\theta) = k \quad (\text{CCSK to CCSK}^P)$$

Theorem

The mappings $\cdot^\circ$ and $\cdot^\dagger$ are mutual inverses.

Why proof labels?

- Independence and dependence of transitions in terms of their labels

Why proof labels?

- Independence and dependence of transitions in terms of their labels
- Independence and dependence of transitions defined separately, not by complementarity

Why proof labels?

- Independence and dependence of transitions in terms of their labels
- Independence and dependence of transitions defined separately, not by complementarity
- (Forward) label determinism: given a transition with fixed source and label, its target is uniquely determined

Causality relations

We can define three binary relations on proof labels characterizing causality of transitions:

- Dependence ($\times$)
- Independence ($\wr$)
- Connectedness (Υ)

Connectedness Relation	
Action $\frac{}{\alpha[k] \Upsilon \theta} A^1 \quad \frac{}{\theta \Upsilon \alpha[k]} A^2$	Choice $\frac{\theta_1 \Upsilon \theta_2}{+_d \theta_1 \Upsilon +_d \theta_2} C_d^1 \quad \frac{}{+_d \theta_1 \Upsilon +_d \theta_2} C_d^2$
Parallel $\frac{\theta_1 \Upsilon \theta_2}{ _d \theta_1 \Upsilon _d \theta_2} P_d^1 \quad \frac{}{ _d \theta_1 \Upsilon _d \theta_2} P_d^2$	Synchronization $\frac{\theta \Upsilon \theta_d}{ _d \theta \Upsilon \langle _L \theta_L, _R \theta_R \rangle} S_d^1 \quad \frac{\theta_d \Upsilon \theta}{\langle _L \theta_L, _R \theta_R \rangle \Upsilon _d \theta} S_d^2$ $\frac{\theta_1 \Upsilon \theta'_1 \quad \theta_2 \Upsilon \theta'_2}{\langle _L \theta_1, _R \theta_2 \rangle \Upsilon \langle _L \theta'_1, _R \theta'_2 \rangle} S^3$

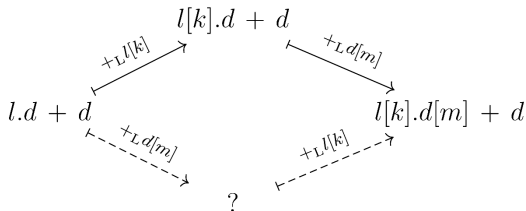
Dependence Relation	
Action $\frac{}{\alpha[k] \times \theta} A^1 \quad \frac{}{\theta \times \alpha[k]} A^2$	
Choice $\frac{\theta \times \theta'}{+_d \theta \times +_d \theta'} C_d^1 \quad \frac{}{+_d \theta \times +_d \theta'} C_d^2$	
Parallel $\frac{\theta \times \theta'}{ _d \theta \times _d \theta'} P_d^1 \quad \frac{\cancel{\theta}(\theta) = \cancel{\theta}(\theta')}{ _d \theta \times _d \theta'} P_d^2$	
Synchronization $\frac{\theta \times \theta_d}{ _d \theta \times \langle _L \theta_L, _R \theta_R \rangle} S_d^1 \quad \frac{\theta_d \times \theta}{\langle _L \theta_L, _R \theta_R \rangle \times _d \theta} S_d^2$ $\frac{\theta_i \times \theta'_i \quad \theta_j \Upsilon \theta'_j \quad i, j \in \{1, 2\}, i \neq j}{\langle _L \theta_1, _R \theta_2 \rangle \times \langle _L \theta'_1, _R \theta'_2 \rangle} S^3$	

Independence Relation	
Action $(empty)$	
Choice $\frac{\theta \wr \theta'}{+_d \theta \wr +_d \theta'} C_d^1$	
Parallel $\frac{\theta \wr \theta'}{ _d \theta \wr _d \theta'} P_d^1 \quad \frac{\cancel{\theta}(\theta) \neq \cancel{\theta}(\theta')}{ _d \theta \wr _d \theta'} P_d^2$	
Synchronization $\frac{\theta \wr \theta_d}{ _d \theta \wr \langle _L \theta_L, _R \theta_R \rangle} S_d^1 \quad \frac{\theta_d \wr \theta}{\langle _L \theta_L, _R \theta_R \rangle \wr _d \theta} S_d^2$ $\frac{\theta_1 \wr \theta'_1 \quad \theta_2 \wr \theta'_2}{\langle _L \theta_1, _R \theta_2 \rangle \wr \langle _L \theta'_1, _R \theta'_2 \rangle} S^3$	

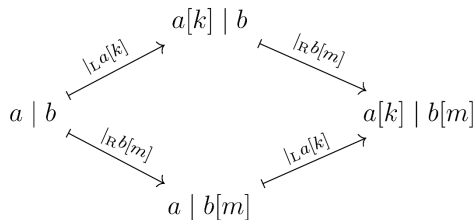
Causality relations

Examples:

- $+_L l[k] \times +_L d[m]$:



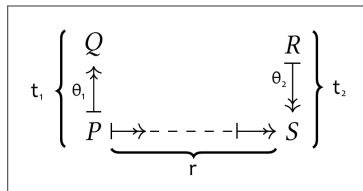
- $|_L a[k] \wr |_R b[m]$:



Connectedness of transitions

Definition

Two transitions t_1 and t_2 are **connected** if there exists a *path* (i.e., a sequence of transitions) between the source of t_1 and the target of t_2 .



Theorem

- (i) If $t_1 : X_1 \xrightarrow{\theta_1} X'_1$ and $t_2 : X_2 \xrightarrow{\theta_2} X'_2$ are connected, then $\theta_1 \curlyvee \theta_2$.
- (ii) If $\theta_1 \curlyvee \theta_2$, then there exist $t_1 : X_1 \xrightarrow{\theta_1} X'_1$ and $t_2 : X_2 \xrightarrow{\theta_2} X'_2$ such that t_1 and t_2 are connected.

Independence of transitions

Definition

- Given $t_1 : X_1 \xrightarrow{\theta_1} X'_1$ and $t_2 : X_2 \xrightarrow{\theta_2} X'_2$ in CCSK^P , t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\theta_1 \not\iota \theta_2$.

Independence of transitions

Definition

- Given $t_1 : X_1 \xrightarrow{\theta_1} X'_1$ and $t_2 : X_2 \xrightarrow{\theta_2} X'_2$ in CCSK^P , t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\theta_1 \iota \theta_2$.
- Given $t_1 : X_1 \xrightarrow{\alpha[k]} X'_1$ and $t_2 : X_2 \xrightarrow{\beta[m]} X'_2$ in CCSK , t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\text{lbl}(t_1^\dagger) \iota \text{lbl}(t_2^\dagger)$, where $\text{lbl}(t : X \xrightarrow{\theta} X') = \theta$.

Independence of transitions

Definition

- Given $t_1 : X_1 \xrightarrow{\theta_1} X'_1$ and $t_2 : X_2 \xrightarrow{\theta_2} X'_2$ in CCSK^P , t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\theta_1 \iota \theta_2$.
- Given $t_1 : X_1 \xrightarrow{\alpha[k]} X'_1$ and $t_2 : X_2 \xrightarrow{\beta[m]} X'_2$ in CCSK , t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\text{lbl}(t_1^\dagger) \iota \text{lbl}(t_2^\dagger)$, where $\text{lbl}(t : X \xrightarrow{\theta} X') = \theta$.

Theorem

The axiomatic theory holds for the LTSes of CCSK^P and CCSK .

Table of Contents

► Introduction

► CCSK and CCSK^P

► Beluga Formalization

- Closed and open transitions
- The axiomatic approach for CCSK

► Conclusions

Restrictions

In CCSK and CCSK^P, restrictions $(\cdot) \setminus a$ are **binders**

When formalizing binders, one has to deal with α -renaming and *capture-avoiding substitutions*.

Example: $(a \mid \bar{a}) \setminus a \equiv_{\alpha} (b \mid \bar{b}) \setminus b$

Restrictions

In CCSK and CCSK^P, restrictions $(\cdot) \setminus a$ are **binders**

When formalizing binders, one has to deal with α -renaming and *capture-avoiding substitutions*.

Example: $(a \mid \bar{a}) \setminus a \equiv_{\alpha} (b \mid \bar{b}) \setminus b$

→ **Higher-Order Abstract Syntax** (HOAS): bound names of the object language as arguments of meta-level functions

Restrictions - syntax

Encoding of restrictions

```
LF proc: type =  
  | nu: (names → proc) → proc  
  ...  
  ;
```

Example: $(\bar{a}.b) \setminus b \rightarrow \text{nu } \setminus b. (\text{pref } (\text{out } a) (\text{pref } (\text{inp } b) \text{ zero}))$

Restrictions - CCSK semantics

$$\alpha \notin \{a, \bar{a}\} \quad \frac{X \xrightarrow{\alpha[k]} X'}{X \setminus a \xrightarrow{\alpha[k]} X' \setminus a} \text{ res} \quad \frac{X \xrightarrow{\lambda[k]} X' \quad Y \xrightarrow{\bar{\lambda}[k]} Y'}{X \mid Y \xrightarrow{\tau[k]} X' \mid Y'} \text{ syn}$$

Encoding of the LTS rule for restrictions

```

LF fstep: proc → key_lab → proc → type =
  | fs_nu: ({a:names} fstep (X a) (kl_base α K) (X' a))
           → fstep (nu X) (kl_base α K) (nu X')
  ...

```

Example: $(a \mid \bar{a}) \setminus a \xrightarrow{\tau[k]} (a[k] \mid \bar{a}[k]) \setminus a$

Restrictions - CCSK^P semantics

$$\ell(\theta) \notin \{a, \bar{a}\} \quad \frac{X \xrightarrow{\theta} X'}{X \setminus a \xrightarrow{\theta} X' \setminus a} \text{ res} \quad \frac{X \xrightarrow{\nu_L \lambda[k]} X' \quad Y \xrightarrow{\nu_R \bar{\lambda}[k]} Y'}{X \mid Y \xrightarrow{\langle |_{L\nu_L \lambda[k]}, |_{R\nu_R \bar{\lambda}[k]} \rangle} X' \mid Y'} \text{ syn}$$

Encoding of the LTS rule for restrictions

```
LF fstep: proc → pr_lab → proc → type =
  | fs_nu: ({a:names} fstep (X a) θ (X' a)) → fstep (nu X) θ (nu X')
```

Example: $(a \mid \bar{a}) \setminus a \xrightarrow{\langle |_{L a[k]}, |_{R \bar{a}[k]} \rangle} (a[k] \mid \bar{a}[k]) \setminus a$

$\ell(\langle |_{L a[k]}, |_{R \bar{a}[k]} \rangle) = \tau \notin \{a, \bar{a}\}$, but a occurs in $\langle |_{L a[k]}, |_{R \bar{a}[k]} \rangle$

New attempt: mirror the informal LTS rule

$$\ell(\theta) \notin \{a, \bar{a}\} \quad \frac{X \xrightarrow{\theta} X'}{X \setminus a \xrightarrow{\theta} X' \setminus a} \text{ res}$$

Possible encoding of the LTS rule for restrictions

```
LF fstep: proc → pr_lab → proc → type =  
  | fs_nu: ({a:names} fstep (X a) (θ a) (X' a)) → ({a:names} lab (θ a) L)  
    → fstep (nu X) θ (nu X')
```

It does not type-check: θ has both type `pr_lab` and `(names → pr_lab)`

New attempt: mirror the informal LTS rule

Possible encoding of the LTS rule for restrictions

```
LF fstep: proc → pr_lab → proc → type =  
  | fs_nu: ({a:names} fstep (X a) (θ a) (X' a)) → ({a:names} lab (θ a) L)  
           → fstep (nu X) (θ ?) (nu X')
```

If ? is a metavariable A: we lose uniqueness of derivations of transitions, and we lose the bijection with CCSK

$$\begin{array}{ll} (a \mid \bar{a}) \setminus a \xrightarrow{\tau[k]} (a[k] \mid \bar{a}[k]) \setminus a & (a \mid \bar{a}) \setminus a \xrightarrow{\langle |_L a[k], |_R \bar{a}[k] \rangle} (a[k] \mid \bar{a}[k]) \setminus a \\ (b \mid \bar{b}) \setminus b \xrightarrow{\tau[k]} (b[k] \mid \bar{b}[k]) \setminus b & (b \mid \bar{b}) \setminus b \xrightarrow{\langle |_L b[k], |_R \bar{b}[k] \rangle} (b[k] \mid \bar{b}[k]) \setminus b \end{array}$$

New attempt: mirror the informal LTS rule

Possible encoding of the LTS rule for restrictions

```
LF fstep: proc → pr_lab → proc → type =  
  | fs_nu: ({a:names} fstep (X a) (θ a) (X' a)) → ({a:names} lab (θ a) L)  
           → fstep (nu X) (θ ?) (nu X')
```

If ? is a constant c of type `names` (a placeholder): no way to prevent the usage of c in prefixes $c.X$

Closed and open transitions

Solution:

- Consider some proof labels up to α -equivalence:

$$(a \mid \bar{a}) \setminus a \xrightarrow{\langle |_L a[k], |_R \bar{a}[k] \rangle \setminus a} (a[k] \mid \bar{a}[k]) \setminus a$$

Encoding of closed and open proof labels

```
LF pr_lab_cl: type = % Closed proof labels
...;

LF pr_lab_op: type = % Open proof labels
| open: (names → pr_lab_cl) → pr_lab_op;
```

Closed and open transitions

Solution:

- Two types for transitions, respectively using proof labels and their α -equivalence classes

Encoding of closed and open transitions

```
LF fstep_cl: proc → pr_lab_cl → proc → type = % Closed forw. transitions
| fc_nu: ({a:names} fstep_cl (X a) T (X' a))
    → fstep_cl (nu X) T (nu X')
...
LF fstep_op: proc → pr_lab_op → proc → type = % Open forw. transitions
| fo_nu_cl: ({a:names} occurs a (T a)) → ({a:names} lab (T a) A)
    → ({a:names} fstep_cl (X a) (T a) (X' a))
    → fstep_op (nu X) (open T) (nu X')
| fo_nu_op: ({a:names} fstep_op (X a) T (X' a))
    → fstep_op (nu X) T (nu X')
```

Does this encoding deliver?

→ Not elegant

Does this encoding deliver?

→ Not elegant

→ Cumbersome: duplication of types and proofs, restrictions subcases are harder to deal with

Does this encoding deliver?

- Not elegant
- Cumbersome: duplication of types and proofs, restrictions subcases are harder to deal with
- Effective: bijection with CCSK, the axiomatic theory holds for CCSK^P

Proving the axiomatic approach for CCSK

Recall:

Definition

Given $t_1 : X_1 \xrightarrow{\alpha[k]} X'_1$ and $t_2 : X_2 \xrightarrow{\beta[m]} X'_2$ in CCSK, t_1 and t_2 are **independent** iff t_1 and t_2 are connected and $\text{lbl}(t_1^\dagger) \iota \text{lbl}(t_2^\dagger)$, with:

$\cdot^\dagger : (X_1 \xrightarrow{\alpha[k]} X_2) \mapsto (X_1 \xrightarrow{\theta} X_2) \quad \text{s.t. } \ell(\theta) = \alpha, \mathcal{K}(\theta) = k;$
 $\text{lbl}(t : X \xrightarrow{\theta} X') = \theta.$

Proving the axiomatic approach for CCSK

In the literature, the strategy is the following:

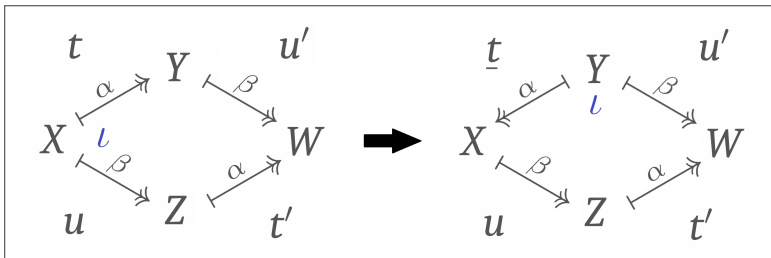
- Prove that the axioms and properties (SP, BTI, WF, ...) for CCSK^P ;
- Leverage the bijection between CCSK^P and CCSK to conclude that they hold for CCSK.

However, the latter is not always as straightforward as it seems.

Example: propagation of coinital independence (PCI)

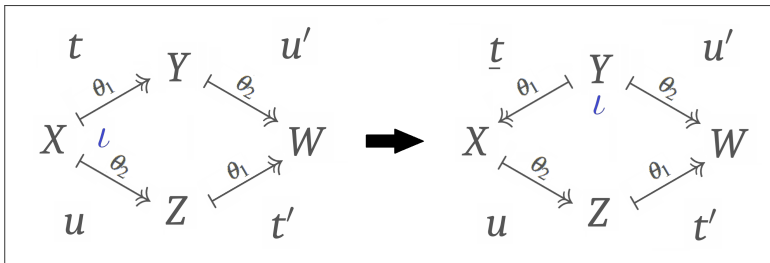
Propagation of coinital independence

Whenever $t : X \xrightarrow{\alpha} Y$, $u : X \xrightarrow{\beta} Z$, $u' : Y \xrightarrow{\beta} W$ and $t' : Z \xrightarrow{\alpha} W$ with $t \iota u$, then $u' \iota \underline{t}$.



In CCSK^P ...

...this is immediate:

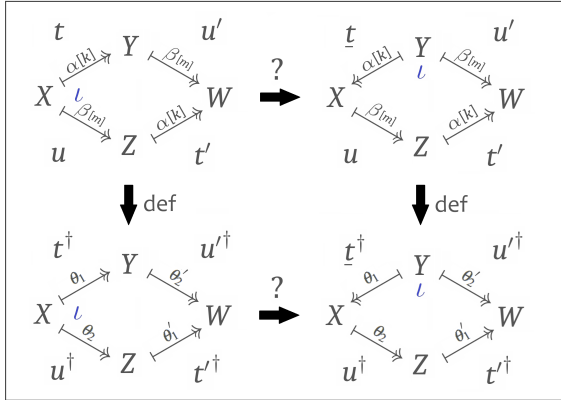


$t \iota u$ means $\theta_1 \iota \theta_2$.

$u' \iota \underline{t}$ means $\theta_2 \iota \theta_1$.

In CCSK ...

...this is not immediate:



$t \iota u$ means $\text{lbl}(t^\dagger) = \theta_1 \iota \theta_2 = \text{lbl}(u^\dagger)$. $u' \iota \underline{t}$ means $\text{lbl}(u'^\dagger) = \theta'_2 \iota \theta_1 = \text{lbl}(\underline{t}^\dagger)$.

But we don't know whether $\theta_2 = \theta'_2$.

How to prove PCI for CCSK?

We want to prove that, given u and u' as in the previous diagram, then $u^\dagger$ and $u'^\dagger$ have the same proof label.

This is not true in general: given any two CCSK transitions with the same keyed label, their enriched versions may not have the same proof label.

Example:

$$\begin{array}{ll} a + a \xrightarrow{a[k]} a[k] + a & \rightarrow \quad a + a \xrightarrow{+La[k]} a[k] + a \\ a + a \xrightarrow{a[k]} a + a[k] & \rightarrow \quad a + a \xrightarrow{+Ra[k]} a + a[k] \end{array}$$

→ The presence of the diamond as in the PCI statement is fundamental

How to prove PCI for CCSK? Alternatives:

- “Brute force”: by case analysis on all the transitions in the CCSK and CCSK^P diagram.

Unfeasible: transitions have 4 rules for parallel compositions. With 8 transitions in total, this amounts to $4^8 (\sim 65k)$ subcases

How to prove PCI for CCSK? Alternatives:

- “Brute force”: by case analysis on all the transitions in the CCSK and CCSK^P diagram.

Unfeasible: transitions have 4 rules for parallel compositions. With 8 transitions in total, this amounts to $4^8 (\sim 65k)$ subcases

- Proof with the axiomatic approach (events, parabolic lemma, causal consistency)

Hard to prove in Beluga: inductions on the number of transitions/transition pairs occurring in paths

How to prove PCI for CCSK? Alternatives:

- “Brute force”: by case analysis on all the transitions in the CCSK and CCSK^P diagram.

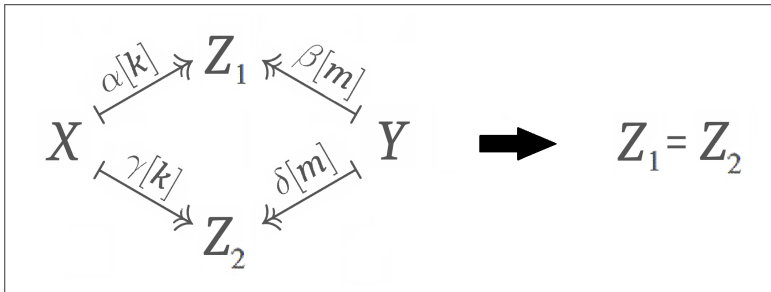
Unfeasible: transitions have 4 rules for parallel compositions. With 8 transitions in total, this amounts to $4^8 (\sim 65k)$ subcases

- Proof with the axiomatic approach (events, parabolic lemma, causal consistency)

Hard to prove in Beluga: inductions on the number of transitions/transition pairs occurring in paths

- Using the axiomatic approach, equivalent properties of diamonds
Mostly unfeasible, except for one

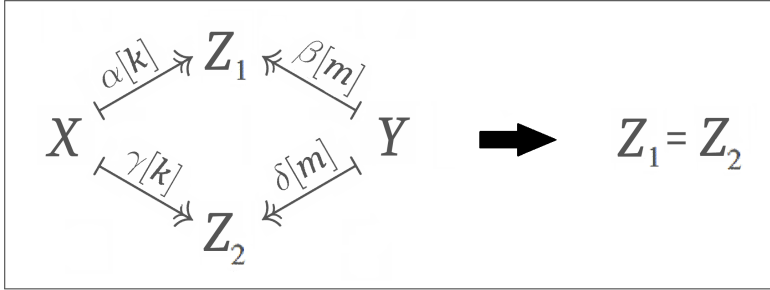
Solution:



Lemma

Given the combined CCSK transitions in figure, if $k \neq m$, then $Z_1 = Z_2$.

Solution:



Lemma

Given the combined CCSK transitions in figure, if $k \neq m$, then $Z_1 = Z_2$.

Proved by case analysis on the CCSK transitions, for ~ 1700 LOC

Table of Contents

- ▶ Introduction
- ▶ CCSK and CCSK^P
- ▶ Beluga Formalization
 - Closed and open transitions
 - The axiomatic approach for CCSK
- ▶ Conclusions

Code structure and size

- `shared`: definitions (processes, keys, ...) and proofs (properties of keys, ...)
- `ccsk`:
 - definitions and basic properties (unique derivation of transitions, ...)
 - `axioms`: proof of the axiomatic approach
- `ccskp`:
 - definitions and basic properties (unique derivation of transitions, ...)
 - properties of causality relations (characterization of connectedness, ...)
 - `axioms`: proof of the axiomatic approach
- `bijection`: bijection between CCSK and CCSK^P
- `examples`: examples of processes, transitions, ...

Code structure and size

- `shared` : ~ 200 LOC
- `ccsk` : ~ 2500 LOC
 - definitions and basic properties: ~ 200 LOC
 - `axioms` : ~ 2300 LOC
- `ccskp` : ~ 5900 LOC
 - definitions and basic properties: ~ 1400 LOC
 - properties of causality relations: ~ 2000 LOC
 - `axioms` : ~ 2500 LOC
- `bijection` : ~ 500 LOC
- `examples` : ~ 500 LOC

Total: ~ 9600 LOC

Evaluation - Beluga

- HOAS $\rightarrow$ no lemmas about handling names and substitutions
 $\rightarrow$ duplication of types and proofs (closed/open transitions)
- Proof term entirely written by the user
 $\rightarrow$ Proof term matches the informal proof; more control
 $\rightarrow$ Automation could prove useful (Harpoon? coding agents?)
- Lack of syntactic sugar for existentials and conjunctions

Overall, other proof assistants may be a better fit for this system.

Evaluation - Beluga

Then, why Beluga?

Evaluation - Beluga

Then, why Beluga?

Prior familiarity with the proof assistant and with HOAS

Evaluation - Beluga

Then, why Beluga?

Prior familiarity with the proof assistant and with HOAS

→ I tried using it, and initially there was no issue

Evaluation - Beluga

Then, why Beluga?

Prior familiarity with the proof assistant and with HOAS

→ I tried using it, and initially there was no issue

In the end, it proved to be a viable option for this system

Conclusions

- First formalization of reversible concurrent calculi
- The results about CCSK and CCSK^P are correct; the proofs sometimes are not
- The first formalization choices can affect months/years of work
- I have more expertise in reversible concurrent calculi and Beluga. However, I may choose other proof assistants for future projects

Future work

- Journal article: *Characterising and Formalising Independence and Causality in (Reversible) Concurrent Calculi*, Aubert, Cecilia, Phillips and Ulidowski
- Extending the formalization: CCS^P , events, key-based equivalences
- Bisimulations for reversible concurrent calculi

Thank you for listening!
Any questions?