A Formalization of the Reversible Concurrent Calculus CCSK^{P} in Beluga

Gabriele Cecilia
Augusta University

20 June 2025

ICE 2025

Table of Contents

Reversible Concurrent Calculi

Concurrent Calculi:

- Abstract models for concurrent systems
- Examples: CCS, π-calculus

Reversible Concurrent Calculi

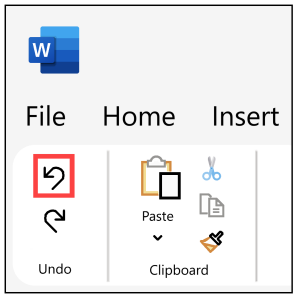
Concurrent Calculi:

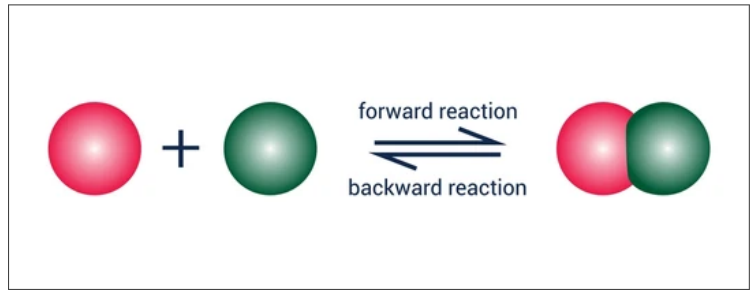
- Abstract models for concurrent systems
- Examples: CCS, π-calculus

Reversible Concurrent Calculi:

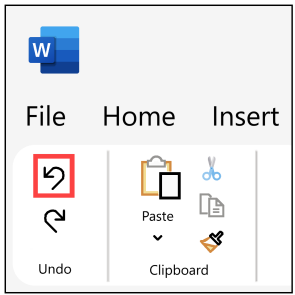
- Abstract models for concurrent systems *in which every action can be undone*
- Examples: CCSK, RCCS, CCSK$^{\mathrm{P}}$

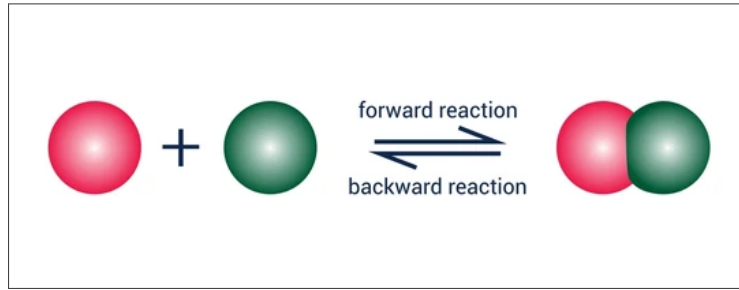
Reversibility





Reversibility





- Accurate representation of reversible systems
- Computational efficiency: chips, debuggers, quantum computing, ...

Formalization

How one line of code caused a $60 million loss

60,000 people lost full phone service, half of AT&T's network was down, and 500 airline flights were delayed

NOV 13, 2023

On January 15th, 1990, AT&T's New Jersey operations center detected a widespread system malfunction, shown by a plethora of red warnings on their network display.

Despite attempts to rectify the situation, the network remained compromised for 9 hours, leading to a 50% failure rate in call connections.

AT&T lost over **$60 million** as a result with **over 60,000 of Americans left with fully disconnected phones**.

How a single line of code brought down a half-billion euro rocket launch

It's Tuesday, June 4th, 1996, and the European Space Agency is set to launch its new Ariane 5 rocket for the first time. This is the culmination of a decade of design, testing and a budget spending billions of euros.

Formalization

Mechanized Metatheory for the Masses: The PoplMark Challenge

Brian E. Aydemir[1], Aaron Bohannon[1], Matthew Fairbairn[2], J. Nathan Foster[1], Benjamin C. Pierce[1], Peter Sewell[2], Dimitrios Vytiniotis[1], Geoffrey Washburn[1], Stephanie Weirich[1], and Steve Zdancewic[1]

[1] Department of Computer and Information Science, University of Pennsylvania
[2] Computer Laboratory, University of Cambridge

Abstract. How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?

Existing Concurrent Calculi Formalizations

Author, Year	Publication	Technique
Nesi 94	A Formalization of the Process Algebra CCS in HOL	Named syntax
Melham 94	A Mechanized Theory of the π-Calculus in HOL	Named syntax
Hirschkoff 97	A Full Formalisation of π-Calculus Theory in the Calculus of Constructions	De Bruijn indexes
Bengtson 09	Formalizing Process Calculi	Nominal techniques
Miller et al. 99	Foundational Aspects of Syntax	HOAS
Honsell et al. 01	π-Calculus in (Co)Inductive Type Theory	HOAS

Existing Concurrent Calculi Formalizations

Author, Year	Publication	Technique
Nesi 94	A Formalization of the Process Algebra CCS in HOL	Named syntax
Melham 94	A Mechanized Theory of the π-Calculus in HOL	Named syntax
Hirschkoff 97	A Full Formalisation of π-Calculus Theory in the Calculus of Constructions	De Bruijn indexes
Bengtson 09	Formalizing Process Calculi	Nominal techniques
Miller et al. 99	Foundational Aspects of Syntax	HOAS
Honsell et al. 01	π-Calculus in (Co)Inductive Type Theory	HOAS

... but no reversible concurrent calculi formalizations.

Table of Contents

CCS with Keys and Proof labels (CCSK^{P})

CCSK: CCS with Keys

- A reversible extension of CCS
- Phillips & Ulidowski, 2007
- Processes and transitions enriched with communication keys

CCS with Keys and Proof labels ($CCSK^P$)

CCSK: CCS with Keys

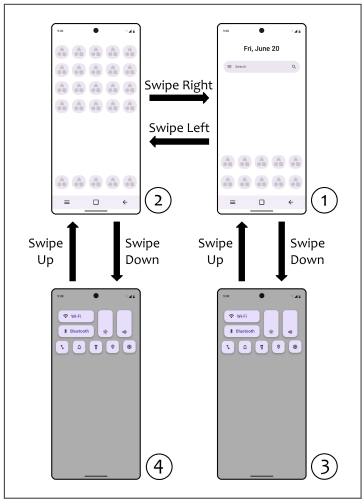
- A reversible extension of CCS
- Phillips & Ulidowski, 2007
- Processes and transitions enriched with communication keys

$CCSK^P$: CCS with Keys and Proof labels

- A proved transition system for CCSK
- Aubert, 2024
- Semantics enriched with proof labels and causality relations

Example: Smartphone

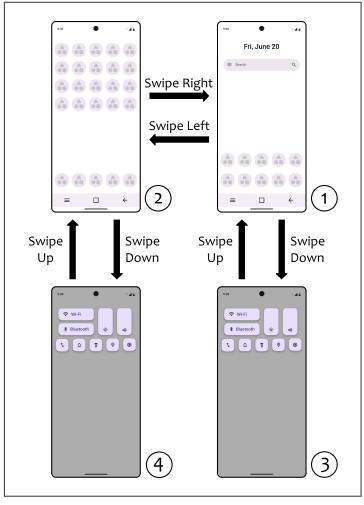
Example: Smartphone



Representation in CCS:

$$P_1 \stackrel{\text{def}}{=} \text{left}.P_2 + \text{down}.P_3$$
$$P_2 \stackrel{\text{def}}{=} \text{right}.P_1 + \text{down}.P_4$$
$$P_3 \stackrel{\text{def}}{=} \text{up}.P_1$$
$$P_4 \stackrel{\text{def}}{=} \text{up}.P_2$$

Example: Smartphone



Representation in CCSK$^{\mathsf{P}}$:

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

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

Syntax

- Infinite set of names N: $a, b, \ldots$
- Complementary names $\overline{\mathrm{N}}$: $\overline{a}, \overline{b}, \ldots$
- Symbol for interactions τ
- Infinite set of keys: $k, m, \ldots$

} Labels, ranged over by $\alpha, \beta, \ldots$

Syntax

- Infinite set of names N: $a, b, \ldots$
- Complementary names $\overline{\mathrm{N}}$: $\overline{a}, \overline{b}, \ldots$
- Symbol for interactions τ
- Infinite set of keys: $k, m, \ldots$

} Labels, ranged over by $\alpha, \beta, \ldots$

Definition

The set of processes is defined by the following syntax:

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

Notation:
keys(X): set of keys occurring in X
std(X): predicate true when X has no keys (X is said to be *standard*)

Semantics

Given by a Labelled Transition System (LTS).
Transitions are labelled by *proof labels*.

Examples of transitions:

- $l.d + d \xmapsto{+_L l[k]} l[k].d + d$
- $l.d + d \xmapsto{+_R d[m]} l.d + d[m]$
- $a \mid b[m] \xmapsto{|_R b[m]} a \mid b$
- $a \mid b[m] \xmapsto{|_L a[k]} a[k] \mid b[m]$

Semantics

Definition

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

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

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

Semantics

Definition

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

$$\mathrm{std}(X) \quad \frac{}{\alpha.X \overset{\alpha[k]}{\longmapsto} \alpha[k].X} \quad \text{pref} \qquad \mathrm{std}(X) \quad \frac{}{\alpha[k].X \overset{\alpha[k]}{\rightsquigarrow} \alpha.X} \quad \underline{\text{pref}}$$

$$\not\kappa(\theta) \neq k \quad \frac{X \overset{\theta}{\longmapsto} X'}{\alpha[k].X \overset{\theta}{\longmapsto} \alpha[k].X'} \quad \text{kpref} \qquad \not\kappa(\theta) \neq k \quad \frac{X' \overset{\theta}{\rightsquigarrow} X}{\alpha[k].X' \overset{\theta}{\rightsquigarrow} \alpha[k].X} \quad \underline{\text{kpref}}$$

Notation:

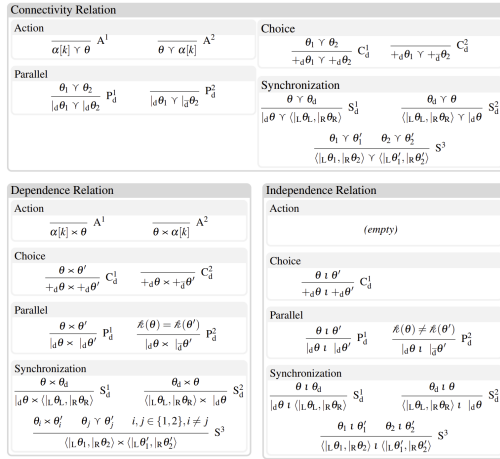
Given a combined transition t: $X \overset{\theta}{\mapsto\!\!\!\!\rightarrow} X'$, X and X' are said the *source* and *target* of t. Two transitions t_1 and t_2 are *connected* if there exists a *path* (i.e., a sequence of transitions) between the source of t_1 and the target of t_2.

Causality Relations

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

- Dependence ($\asymp$)
- Independence (ι)
- Connectivity ($\curlyvee$)

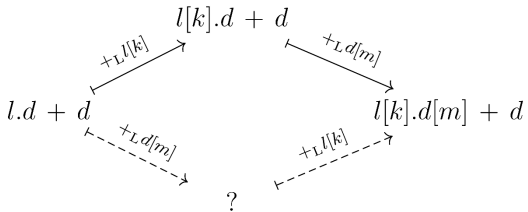
Introduced by Aubert et al. in "*Independence and Causality in the Reversible Concurrent Setting*" (2025). ($\star$)



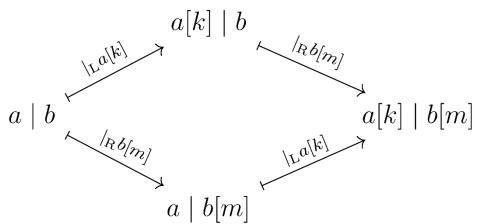
Causality Relations

Examples:

- $+_{\mathrm{L}} l[k] \asymp +_{\mathrm{L}} d[m]$:



- $|_{\mathrm{L}} a[k] \iota |_{\mathrm{R}} b[m]$:



Properties of Causality Relations

Main results proven in Section 3-4 of ($\star$):

Theorem 1 (Characterization of connectivity of proof labels)

(i) If $t_1 : X_1 \overset{\theta_1}{\mapsto\!\!\!\!\rightarrow} X_1'$ and $t_2 : X_2 \overset{\theta_2}{\mapsto\!\!\!\!\rightarrow} X_2'$ are connected, then $\theta_1 \curlyvee \theta_2$.
(ii) If $\theta_1 \curlyvee \theta_2$, then there exist $t_1 : X_1 \overset{\theta_1}{\mapsto\!\!\!\!\rightarrow} X_1'$ and $t_2 : X_2 \overset{\theta_2}{\mapsto\!\!\!\!\rightarrow} X_2'$ such that t_1 and t_2 are connected.

Theorem 2 (Complementarity of dependence and independence)

(i) If $\theta_1 \iota \theta_2$ then $\theta_1 \curlyvee \theta_2$.
(ii) If $\theta_1 \asymp \theta_2$ then $\theta_1 \curlyvee \theta_2$.
(iii) If $\theta_1 \curlyvee \theta_2$ then either $\theta_1 \iota \theta_2$ or $\theta_1 \asymp \theta_2$, but not both.

Properties of Causality Relations

- *Soundness* of the causality relations
- Dependence and independence are usually *defined by complementarity*.
  Separate axioms $\rightarrow$ easier to deal with them

Table of Contents

Beluga

Developed at the Complogic group at McGill University, Canada

→ Two-level system (LF level, computation level)

→ Encoding of object-level binding constructs through Higher-Order Abstract Syntax (HOAS)

→ Terms are paired with the *contexts* that give them meaning

→ *Curry-Howard isomorphism*: proofs as recursive functional programs, propositions as types

Higher-Order Abstract Syntax (HOAS)

Bound variables of the object language as arguments of meta-language functions

$\rightarrow$ α-renaming and capture-avoiding substitutions managed by the meta-language

$\rightarrow$ Focus on the development of the target system, no technical details of names handling

Useful for encoding systems with a complex binders infrastructure (e.g., π-calculus)

Encoding of Syntax

Names, Keys, Labels and Processes

```
LF names: type =;
LF keys: type =
  | z: keys
  | s: keys → keys;
LF labels: type =
  | inp: names → labels
  | out: names → labels
  | tau: labels;
```

```
LF proc: type =
  | null: proc                                  % 0
  | pref: labels → proc → proc                  % A.X
  | kpref: labels → keys → proc → proc          % A[k].X
  | sum: proc → proc → proc                     % X+Y
  | par: proc → proc → proc                     % X|Y
  | nu: (names → proc) → proc;                  % X\a
```

Examples:

- $l.d + d \quad \rightarrow \quad$ `sum (pref l (pref d zero)) (pref d zero)`
- $a \mid b[m] \quad \rightarrow \quad$ `par (pref a zero) (kpref b m zero)`
- $(\overline{a}.b) \setminus b \quad \rightarrow \quad$ `nu \b.(pref (out a) (pref (inp b) zero))`

Encoding of Syntax

Terms are paired with contexts, containing assumptions. Contexts are classified through schemas

We need a context of the form "`x:names`":

Context declaration

```
schema ctx = names;
```

Consequence: we can define *contextual* processes `[g ⊢ X]`, where `g` contains the free variables occurring in the open process `X`.

Encoding of Semantics

Proof Labels

```
LF pr_lab: type =
   | pr_base: labels → keys → pr_lab          % α[k]
   | pr_suml: pr_lab → pr_lab                 % +_L θ
   | pr_sumr: pr_lab → pr_lab                 % +_R θ
   | pr_parl: pr_lab → pr_lab                 % |_L θ
   | pr_parr: pr_lab → pr_lab                 % |_R θ
   | pr_sync: pr_lab → pr_lab → pr_lab;       % ⟨|_L θ1, |_R θ2⟩
```

Examples:

- $+_{\mathrm{L}} l[k] \quad \rightarrow \quad$ `pr_suml (pr_base l k)`
- $|_{\mathrm{R}} b[m] \quad \rightarrow \quad$ `pr_parr (pr_base b m)`

Encoding of Semantics

Forward and Backward Transitions (and auxiliary predicates)

```
LF key: pr_lab → keys → type = ...        LF std: proc → type = ...

LF fstep: proc → pr_lab → proc → type =
  | fs_pref: std X → fstep (pref A X) (pr_base A K) (kpref A K X)
  ...

LF bstep: proc → pr_lab → proc → type =
  | bs_pref: std X → bstep (kpref A K X) (pr_base A K) (pref A X)
  ...
```

Idea: $X \overset{\theta}{\mapsto} Y$ holds $\Leftrightarrow$ `(fstep X T Y)` is inhabited

Analogously, causality relations are encoded by three type families `conn`, `dep` and `indep`.

Writing proofs in Beluga

Proofs in Beluga:

- Total (recursive) functions
- Proof term written by the user, without tactics: interactivity through computation holes
- Lack of syntactic sugar for existentials and conjunctions
  $\rightarrow$ additional type families to encode proof statements

Proof of Theorems 1 and 2: key insights and findings

- Basic properties of keys, proof labels and transitions to be encoded
  $\rightarrow$ 15 additional lemmas

  Examples: decidability of equality of keys, standard processes have no keys, loop lemma (each transition can be reversed), ...
- **Theorem 2 (complementarity):** direct and uneventful encoding

Proof of Theorems 1 and 2: key insights and findings

Theorem 1 (connectivity):

- Some auxiliary lemmas are not required
- Some statements have been slightly refined
- Lengthy proofs, due to many nested pattern matchings
- Low-level details to fill out $\rightarrow$ 20 additional lemmas
- Non-constructive subcase $\rightarrow$ new proof strategy (+ 500 lines of code)

Example of proof in Beluga

Lemma: for all keys K, K does not occur in a standard process

```
rec no_key_in_std: (g:ctx) {K:[⊢ keys]} [g ⊢ std X] → [g ⊢ notin K[] X] =
  / total d (no_key_in_std _ _ _ d) /
mlam K ⇒ fn d ⇒ case d of
  | [g ⊢ std_null] ⇒ [g ⊢ not_null]
  | [g ⊢ std_pref D] ⇒ let [g ⊢ N] = no_key_in_std [⊢ K] [g ⊢ D] in
    [g ⊢ not_pref N]
  | [g ⊢ std_sum D1 D2] ⇒ let [g ⊢ N1] = no_key_in_std [⊢ K] [g ⊢ D1] in
    let [g ⊢ N2] = no_key_in_std [⊢ K] [g ⊢ D2] in [g ⊢ not_sum N1 N2]
  | [g ⊢ std_par D1 D2] ⇒ let [g ⊢ N1] = no_key_in_std [⊢ K] [g ⊢ D1] in
    let [g ⊢ N2] = no_key_in_std [⊢ K] [g ⊢ D2] in [g ⊢ not_par N1 N2]
  | [g ⊢ std_nu \a.D] ⇒
    let [g,a:names ⊢ N] = no_key_in_std [⊢ K] [g,a:names ⊢ D] in
    [g ⊢ not_nu \a.N];
```

Table of Contents

Evaluation

Technical overview:

- Size of the encoding: $\sim$2000 lines of code
- Informal theorems and lemmas covered: 13
- Technical and auxiliary lemmas: 36

Evaluation

Benefits of using Beluga:

- HOAS $\rightarrow$ no handling of bound names
- Proof term matches the informal proof

Drawbacks of using Beluga:

- Lack of syntactic sugar for existentials and conjunctions
- Limited support for custom notation
- No abstraction mechanism for repeated proof patterns

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

Conclusions

Results:

- First formalization of a reversible concurrent calculus
- Verified the correctness of Sections 3-4 of ($\star$): causality relations are sound
- Filled out details and provided a new proof strategy

Future Work

- Journal paper version of ($\star$), with more results formalized
- Covering subsystems of $\mathrm{CCSK}^{\mathrm{P}}$ (CCS, CCSK)
- Formalizing other (results about) reversible concurrent calculi, such as: *"An Axiomatic Theory for Reversible Computation"* by Lanese et al.

Thank you for listening!
Any questions?