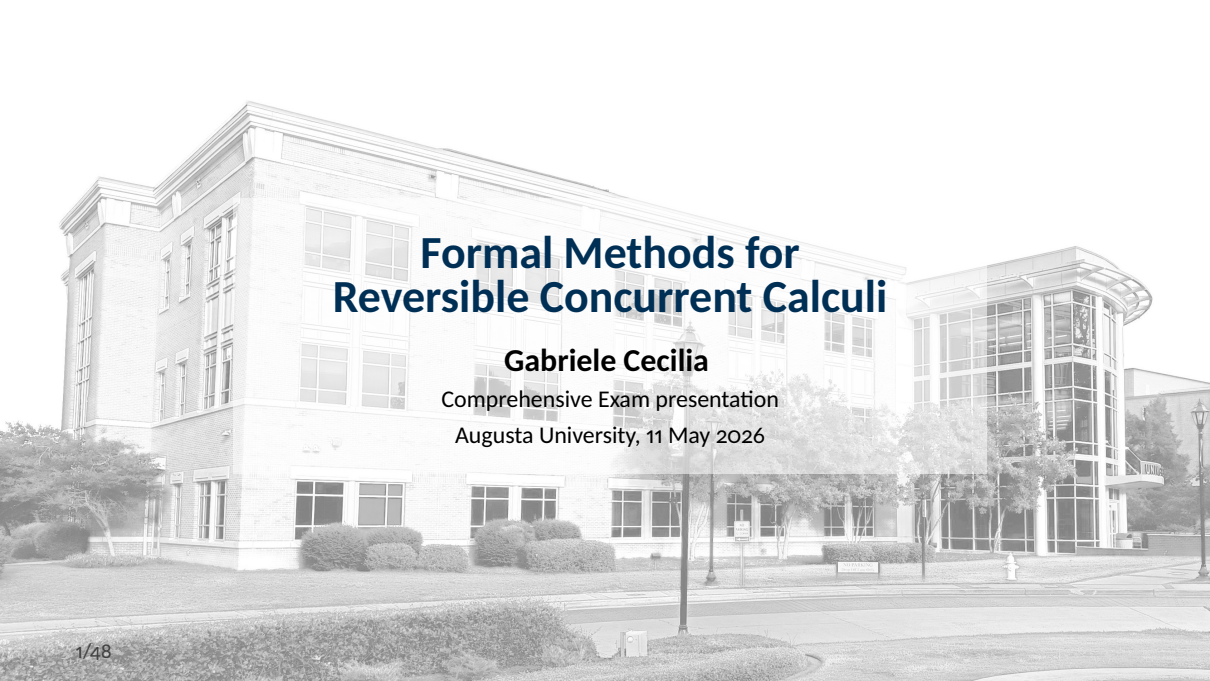
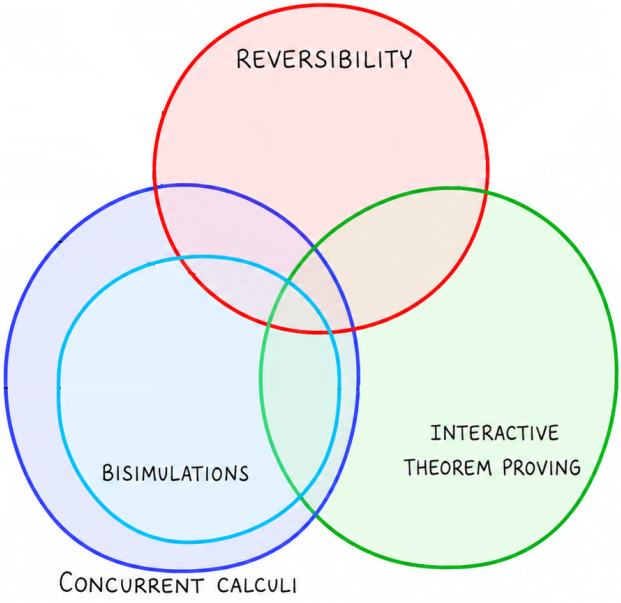


Formal Methods for Reversible Concurrent Calculi

Gabriele Cecilia

Comprehensive Exam presentation

Augusta University, 11 May 2026


REVERSIBILITY
BISIMULATIONS
INTERACTIVE
THEOREM PROVING
CONCURRENT CALCULI

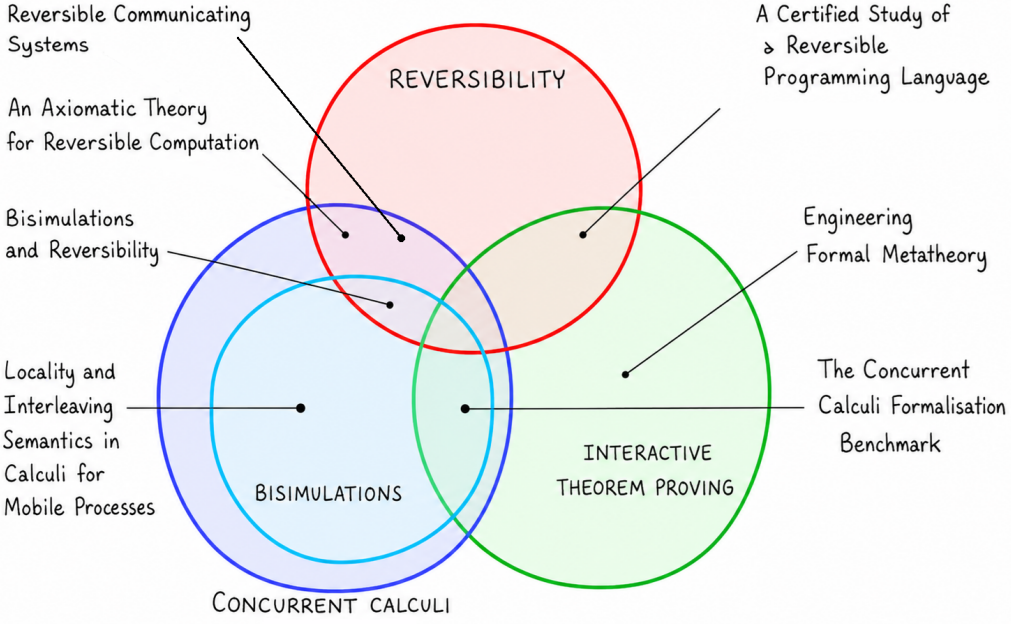

Reversible Communicating Systems
An Axiomatic Theory for Reversible Computation
Bisimulations and Reversibility
Locality and Interleaving Semantics in Calculi for Mobile Processes
A Certified Study of a Reversible Programming Language
Engineering Formal Metatheory
The Concurrent Calculi Formalisation Benchmark
REVERSIBILITY
BISIMULATIONS
INTERACTIVE THEOREM PROVING
CONCURRENT CALCULI

Table of Contents

What Are Concurrent Calculi?

Concurrent systems: multiple processes interacting within a shared system

Concurrent calculi: abstract models of concurrent systems

Why Concurrent Calculi?

Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif

Bruno Blanchet[1]

[1] *INRIA Paris, France; Bruno.Blanchet@inria.fr*

ABSTRACT

ProVerif is an automatic symbolic protocol verifier. It supports a wide range of cryptographic primitives, defined by rewrite rules or by equations. It can prove various security properties: secrecy, authentication, and process equivalences, for an unbounded message space and an unbounded number of sessions. It takes as input a description of the protocol to verify in a dialect of the applied pi calculus, an extension of the pi calculus with cryptography. It automatically translates this protocol description into Horn clauses and determines whether the desired security properties hold by resolution on these clauses. This survey presents an overview of the research on ProVerif.

Why Concurrent Calculi?

Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif

Bruno Blanchet[1]

[1] *INRIA Paris, France; Bruno.Blanchet@inria.fr*

ABSTRACT

ProVerif is an automatic symbolic protocol verifier. It supports a wide range of cryptographic primitives, defined by rewrite rules or by equations. It can prove various security properties: secrecy, authentication, and process equivalences, for an unbounded message space and an unbounded number of sessions. It takes as input a description of the protocol to verify in a dialect of the applied pi calculus, an extension of the pi calculus with cryptography. It automatically translates this protocol description into Horn clauses and determines whether the desired security properties hold by resolution on these clauses. This survey presents an overview of the research on ProVerif.

Breaking Unlinkability of the ICAO 9303 Standard for e-Passports using Bisimilarity

Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith

Computer Science and Communications, University of Luxembourg

Abstract. We clear up confusion surrounding privacy claims about the ICAO 9303 standard for e-passports. The ICAO 9303 standard includes a Basic Access Control (BAC) protocol that should protect the user from being traced from one session to another. While it is well known that there are attacks on BAC, allowing an attacker to link multiple uses of the same passport, due to differences in implementation; there still remains confusion about whether there is an attack on unlinkability directly on the BAC protocol as specified in the ICAO 9303 standard. This paper clarifies the nature of the debate, and sources of potential confusion. We demonstrate that the original privacy claims made are flawed, by uncovering attacks on a strong formulation of unlinkability. We explain why the use of the bisimilarity equivalence technique is essential for uncovering our attacks. We also clarify what assumptions lead to proofs of formulations of unlinkability using weaker notions of equivalence. Furthermore, we propose a fix for BAC within the scope of the standard, and prove that it is correct, again using a state-of-the-art approach to bisimilarity.

Ingredients of Concurrent Calculi

- Syntax: what processes are
- Semantics: how processes behave
- Bisimulations: when processes are equivalent

Example: Home Wi-Fi Network - Syntax

Router: $R_1 \stackrel{\text{def}}{=} \text{req}.\overline{\text{resp}}.R_1$

Prefix: $\alpha.P$

Example: Home Wi-Fi Network - Syntax

Router: $R_1 \stackrel{\text{def}}{=} \text{req}.\overline{\text{resp}}.R_1$

Device: $D_1 \stackrel{\text{def}}{=} \overline{\text{req}}.\text{resp}.C_1 + \text{local}.D_1$

Prefix: $\alpha.P$

Sum: $P + Q$

Example: Home Wi-Fi Network - Syntax

Router: $R_1 \stackrel{\text{def}}{=} \text{req}.\overline{\text{resp}}.R_1$

Prefix: $\alpha.P$

Device: $D_1 \stackrel{\text{def}}{=} \overline{\text{req}}.\text{resp}.C_1 + \text{local}.D_1$

Sum: $P + Q$

Network: $R_1 \mid R_2 \mid D_1 \mid D_2 \mid D_3 \mid D_4$

Parallel: $P \mid Q$

Example: Home Wi-Fi Network - Syntax

Router:	$R_1 \stackrel{\text{def}}{=} \text{req}.\overline{\text{resp}}.R_1$	Prefix: α.P
Device:	$D_1 \stackrel{\text{def}}{=} \overline{\text{req}}.\text{resp}.C_1 + \text{local}.D_1$	Sum: P + Q
Network:	$R_1 \mid R_2 \mid D_1 \mid D_2 \mid D_3 \mid D_4$	Parallel: P $\mid$ Q
Private Network:	$(R_1 \mid R_2 \mid D_1 \mid D_2 \mid D_3 \mid D_4) \setminus \{\text{req},\text{resp}\}$	Restriction: P \a

Example: Home Wi-Fi Network - Semantics

Router: $\mathrm{req}.\overline{\mathrm{resp}}.\mathrm{R}_1 \xrightarrow{\mathrm{req}} \overline{\mathrm{resp}}.\mathrm{R}_1$

Example: Home Wi-Fi Network - Semantics

Router: $\mathsf{req}.\overline{\mathsf{resp}}.\mathsf{R}_1 \xrightarrow{\mathsf{req}} \overline{\mathsf{resp}}.\mathsf{R}_1$

Device: $\overline{\mathsf{req}}.\mathsf{resp}.\mathsf{C}_1 + \mathsf{local}.\mathsf{D}_1 \xrightarrow{\overline{\mathsf{req}}} \mathsf{resp}.\mathsf{C}_1$

Example: Home Wi-Fi Network - Semantics

Router: $\mathrm{req}.\overline{\mathrm{resp}}.\mathrm{R}_1 \xrightarrow{\mathrm{req}} \overline{\mathrm{resp}}.\mathrm{R}_1$

Device: $\overline{\mathrm{req}}.\mathrm{resp}.\mathrm{C}_1 + \mathrm{local}.\mathrm{D}_1 \xrightarrow{\overline{\mathrm{req}}} \mathrm{resp}.\mathrm{C}_1$

Network: $\mathrm{R}_1 \mid \mathrm{D}_1 \xrightarrow{\tau} \overline{\mathrm{resp}}.\mathrm{R}_1 \mid \mathrm{resp}.\mathrm{C}_1$

Why Behavioral Equivalence?

- Abstraction from irrelevant details:
  $R_1 \mid D_1 \sim D_1 \mid R_1, \quad (a \mid \overline{a}) \setminus a \sim (b \mid \overline{b}) \setminus b$

Why Behavioral Equivalence?

- Abstraction from irrelevant details:
  $R_1 \mid D_1 \sim D_1 \mid R_1, \quad (a \mid \overline{a}) \backslash a \sim (b \mid \overline{b}) \backslash b$
- Security and distinguishability:
  one must be able to distinguish a malicious from a non-malicious process

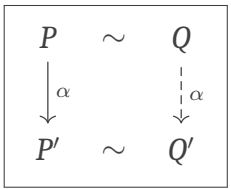
Why Behavioral Equivalence?

- Abstraction from irrelevant details:
  $R_1 \mid D_1 \sim D_1 \mid R_1, \quad (a \mid \overline{a}) \setminus a \sim (b \mid \overline{b}) \setminus b$
- Security and distinguishability:
  one must be able to distinguish a malicious from a non-malicious process
- Safe replacement of components:
  a process can be used in place of another equivalent process

Notions of Bisimulation

Definition

A symmetric binary relation $\mathcal{R}$ on processes is a strong bisimulation iff, whenever $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$.
P and Q are strongly bisimilar ($P \sim Q$) iff there exists a strong bisimulation $\mathcal{R}$ such that $P \mathcal{R} Q$.



Q can simulate every action that P can perform, and vice versa

Notions of Bisimulation

However, depending on the context, this definition may relate too few or too many processes

- $a.b \not\sim a.\tau.b \quad \rightarrow \quad$ *weak bisimulations*
- $a \mid b \sim a.b + b.a \quad \rightarrow \quad$ *interleaving/non-interleaving bisimulations*

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Setting: polyadic π-calculus

Contributions:

- Defining *location bisimulation* for the π-calculus, a non-interleaving equivalence

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Setting: polyadic π-calculus

Contributions:

- Defining *location bisimulation* for the π-calculus, a non-interleaving equivalence
- Characterizing it in terms of *observation equivalence*, an interleaving equivalence, via a fully abstract encoding

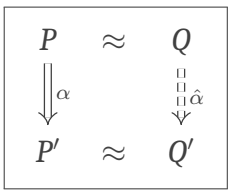
Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Notation: $\overset{\alpha}{\Rightarrow} ::= \Rightarrow\overset{\alpha}{\rightarrow}\Rightarrow$; $\overset{\hat{\alpha}}{\Rightarrow} ::= \overset{\alpha}{\Rightarrow}$ if $\alpha \neq \tau$, $\Rightarrow$ otherwise

Definition

A symmetric binary relation $\mathcal{R}$ on processes is a weak bisimulation iff, whenever $P\,\mathcal{R}\,Q$ and $P \overset{\alpha}{\Rightarrow} P'$, there exists Q' such that $Q \overset{\hat{\alpha}}{\Rightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$.

P and Q are observation equivalent ($P \approx Q$) iff there exists a weak bisimulation $\mathcal{R}$ such that $P\mathcal{R}Q$.



Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Locations capture spatial dependencies on processes.

$$\mathsf{a} \mid \mathsf{b} \xrightarrow[l_1]{\mathsf{a}} l_1 :: 0 \mid \mathsf{b} \xrightarrow[l_2]{\mathsf{b}} l_1 :: 0 \mid l_2 :: 0$$

$$\mathsf{a.b} + \mathsf{b.a} \xrightarrow[l_1]{\mathsf{a}} l_1 :: \mathsf{b} \xrightarrow[l_1 l_2]{\mathsf{b}} l_1 :: l_2 :: 0$$

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Locations capture spatial dependencies on processes.

$$\mathsf{a} \mid \mathsf{b} \xrightarrow[l_1]{\mathsf{a}} l_1 :: 0 \mid \mathsf{b} \xrightarrow[l_2]{\mathsf{b}} l_1 :: 0 \mid l_2 :: 0$$

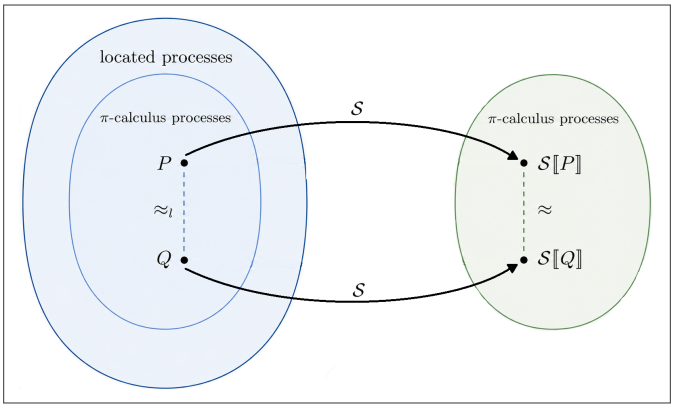
$$\mathsf{a.b} + \mathsf{b.a} \xrightarrow[l_1]{\mathsf{a}} l_1 :: \mathsf{b} \xrightarrow[l_1 l_2]{\mathsf{b}} l_1 :: l_2 :: 0$$

A location bisimulation is a weak bisimulation on located processes in which the locations appearing in transitions must match.

Location bisimilarity is denoted as $\approx_l$.

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Correspondence between location bisimulation and observation equivalence: a fully abstract encoding



Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Discussion:

- Different bisimulations capture different observational distinctions

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

Discussion:

- Different bisimulations capture different observational distinctions
- The correspondence allows to use the simpler theory of observation equivalence to reason about location bisimulation

Paper 1: *"Locality and Interleaving Semantics in Calculi for Mobile Processes"*, D. Sangiorgi

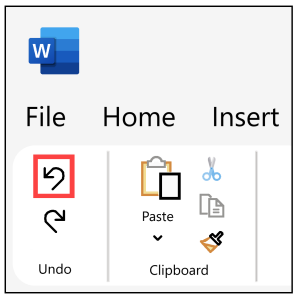
Discussion:

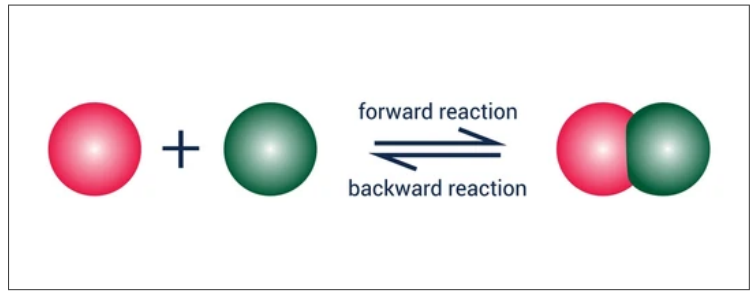
- Different bisimulations capture different observational distinctions
- The correspondence allows to use the simpler theory of observation equivalence to reason about location bisimulation
- Non-interleaving semantics, as well as spatial/causal dependencies on processes, are important also in the reversible setting

Table of Contents

What Is Reversibility?

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





Why Reversibility?

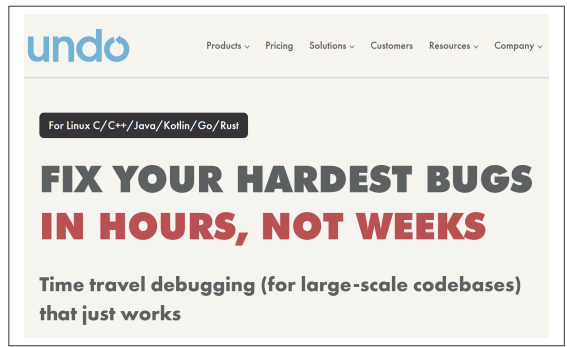
- It allows identifying causal dependencies on processes and actions

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



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:

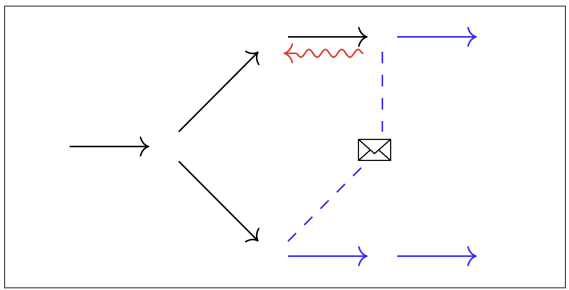
- Recovering inputs from outputs in a memory-efficient way

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
  $\rightarrow$ "*Reversing Algebraic Process Calculi*", I. Phillips and I. Ulidowski, 2007

Reversibility + Concurrent Calculi

How to make a concurrent calculus reversible? Two approaches:

- Tag actions with communication keys and store them in processes
  $\rightarrow$ "*Reversing Algebraic Process Calculi*", I. Phillips and I. Ulidowski, 2007
- Equip threads with a memory stack, storing information to backtrack
  $\rightarrow$ "*Reversible Communicating Systems*", V. Danos and J. Krivine, 2004

Paper 2: "*Reversible Communicating Systems*", V. Danos and J. Krivine

Contributions:

- RCCS (Reversible CCS), a reversible extension of CCS (Calculus of Communicating Systems), by equipping processes with a memory
- Proof of fundamental reversibility properties: loop lemma, square lemma, characterization of causal equivalent traces
- Integration of irreversible actions in RCCS

Paper 2: "*Reversible Communicating Systems*", V. Danos and J. Krivine

Monitored processes: $m \triangleright P$, with P CCS process, m memory

Memories:	$m ::=$		
		$\langle\rangle$	Empty memory
	$\mid$	$\langle 1\rangle \cdot m$	Left-Fork
	$\mid$	$\langle 2\rangle \cdot m$	Right-Fork
	$\mid$	$\langle *, \alpha, P\rangle \cdot m$	Semi-Synch
	$\mid$	$\langle m, \alpha, P\rangle \cdot m$	Synch

Example of transitions:

$$\langle\rangle \triangleright \alpha.P + Q \underset{\langle\rangle:\alpha_*}{\overset{\langle\rangle:\alpha}{\rightleftarrows}} \langle *, \alpha, Q\rangle \cdot \langle\rangle \triangleright P$$

Paper 2: “*Reversible Communicating Systems*”, V. Danos and J. Krivine

Example of property: loop lemma

Lemma 6 (Loop). *For any forward transition* $t : R \xrightarrow{\mu:\alpha} S$*, there exists a backward transition* $t_* : S \xrightarrow{\mu:\alpha_*} R$ *and conversely.*

Paper 2: “*Reversible Communicating Systems*”, V. Danos and J. Krivine

Discussion:

- First paper to introduce a reversible concurrent calculus

Paper 2: "*Reversible Communicating Systems*", V. Danos and J. Krivine

Discussion:

- First paper to introduce a reversible concurrent calculus
- Many of its definitions and results have become standard in the field

Paper 2: "*Reversible Communicating Systems*", V. Danos and J. Krivine

Discussion:

- First paper to introduce a reversible concurrent calculus
- Many of its definitions and results have become standard in the field
- Causal-consistent reversibility: an action cannot be undone unless all its consequences have already been undone

Paper 3: *"An Axiomatic Theory for Reversible Computation"*, I. Lanese, I. Phillips and I. Ulidowski

When studying concrete reversible concurrent formalisms, most researchers provide similar but unrelated proofs of the same results

Paper 3: *"An Axiomatic Theory for Reversible Computation"*, I. Lanese, I. Phillips and I. Ulidowski

When studying concrete reversible concurrent formalisms, most researchers provide similar but unrelated proofs of the same results

Contributions:

- An axiomatic system for reversible computation
- Two new properties (causal safety and causal liveness), characterizing causal-consistent reversibility
- Case studies of application of the axiomatic system

Paper 3: *"An Axiomatic Theory for Reversible Computation"*, I. Lanese, I. Phillips and I. Ulidowski

Their proposed framework is as abstract as possible:

Definition 2.1. A *labelled transition system (LTS)* is a structure $(\mathsf{Proc}, \mathsf{Lab}, \rightarrow)$, where Proc is the set of states (or processes), Lab is the set of action labels and $\rightarrow \subseteq \mathsf{Proc} \times \mathsf{Lab} \times \mathsf{Proc}$ is a transition relation.

Definition 2.2 (Labelled Transition System with Independence (LTSI)). We say that $(\mathsf{Proc}, \mathsf{Lab}, \rightarrow, \iota)$ is an LTSI if $(\mathsf{Proc}, \mathsf{Lab}, \rightarrow)$ is an LTS and ι is an irreflexive symmetric binary relation on transitions.

Definition 2.3 (Reverse and Combined LTS). Given an LTS $(\mathsf{Proc}, \mathsf{Lab}, \rightarrow)$, let the *reverse LTS* be $(\mathsf{Proc}, \mathsf{Lab}, \rightsquigarrow)$, where $P \overset{a}{\rightsquigarrow} Q$ iff $Q \overset{a}{\rightarrow} P$. It is convenient to combine the two LTSs (forward and reverse): let the reverse labels be $\underline{\mathsf{Lab}} = \{\underline{a} : a \in \mathsf{Lab}\}$, and define the *combined LTS* to be $\rightarrow \subseteq \mathsf{Proc} \times (\mathsf{Lab} \cup \underline{\mathsf{Lab}}) \times \mathsf{Proc}$ by $P \overset{a}{\rightarrow} Q$ iff $P \overset{a}{\rightarrow} Q$ and $P \overset{\underline{a}}{\rightarrow} Q$ iff $P \overset{a}{\rightsquigarrow} Q$.

Paper 3: *"An Axiomatic Theory for Reversible Computation"*, I. Lanese, I. Phillips and I. Ulidowski

Example of axioms:

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 $t : P \overset{a}{\rightsquigarrow} Q$ and $t' : P \overset{b}{\rightsquigarrow} Q'$ 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 $i = 0, 1, \ldots$.

Paper 3: *"An Axiomatic Theory for Reversible Computation"*, I. Lanese, I. Phillips and I. Ulidowski

Discussion:

- To prove the soundness of a reversible system, verifying the basic axioms is enough. All other results are inherited for free
- Benefits of a potential formalization with a proof assistant: certify the correctness of the axiomatic system; develop a mechanized framework to check the soundness of reversible models

Paper 4: “*Bisimulations and Reversibility*”, C. Aubert, I. Phillips and I. Ulidowski

Setting: CCSK (CCS with Keys) and KCS (Keyed Configuration Structures)

Paper 4: *"Bisimulations and Reversibility"*, C. Aubert, I. Phillips and I. Ulidowski

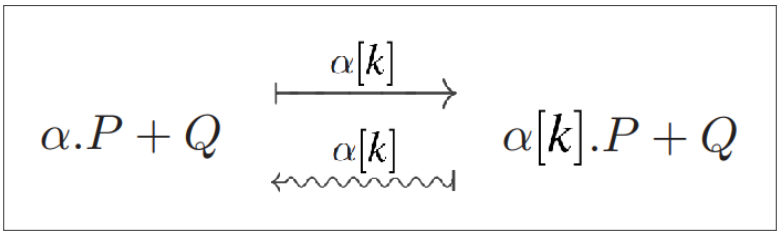
Setting: CCSK (CCS with Keys) and KCS (Keyed Configuration Structures)

Contributions:

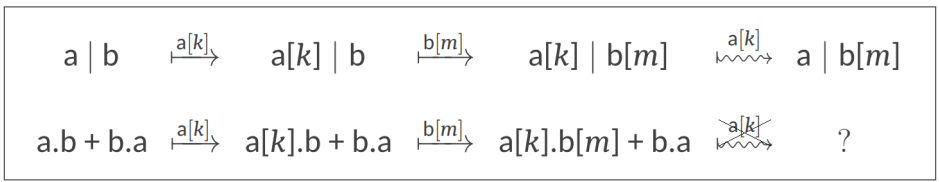
- Presentation of CCSK, KCS, and the equivalence of their operational semantics
- Characterizations of HHP (hereditary history-preserving) and HP (history-preserving) bisimulations in terms of several different bisimulations, defined only in terms of forward and backward transitions

Paper 4: *"Bisimulations and Reversibility"*, C. Aubert, I. Phillips and I. Ulidowski

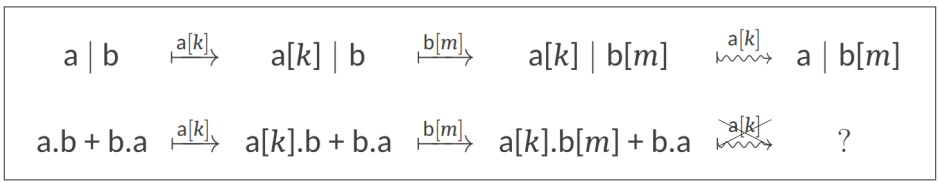
CCSK: communication keys identify forward transitions and are stored in processes



Paper 4: *"Bisimulations and Reversibility"*, C. Aubert, I. Phillips and I. Ulidowski



Paper 4: *"Bisimulations and Reversibility"*, C. Aubert, I. Phillips and I. Ulidowski



Definition 4.9 (FR on CCSK [25,26]). *Let X, Y etc,. range over* CCSK *processes. A relation $\mathcal{R}$ is a* forward-reverse (FR) bisimulation *if whenever $\mathcal{R}(X,Y)$ then*

- *if* $X \xrightarrow{\alpha[i]} X'$ *then* $\exists Y'.\ Y \xrightarrow{\alpha[i]} Y'$ *and* $\mathcal{R}(X',Y')$;
- *if* $Y \xrightarrow{\alpha[i]} Y'$ *then* $\exists X'.\ X \xrightarrow{\alpha[i]} X'$ *and* $\mathcal{R}(X',Y')$;
- *if* $X \xrightarrow{\alpha[i]}\!\!\!\!\!\!\sim\ X'$ *then* $\exists Y'.\ Y \xrightarrow{\alpha[i]}\!\!\!\!\!\!\sim\ Y'$ *and* $\mathcal{R}(X',Y')$;
- *if* $Y \xrightarrow{\alpha[i]}\!\!\!\!\!\!\sim\ Y'$ *then* $\exists X'.\ X \xrightarrow{\alpha[i]}\!\!\!\!\!\!\sim\ X'$ *and* $\mathcal{R}(X',Y')$.

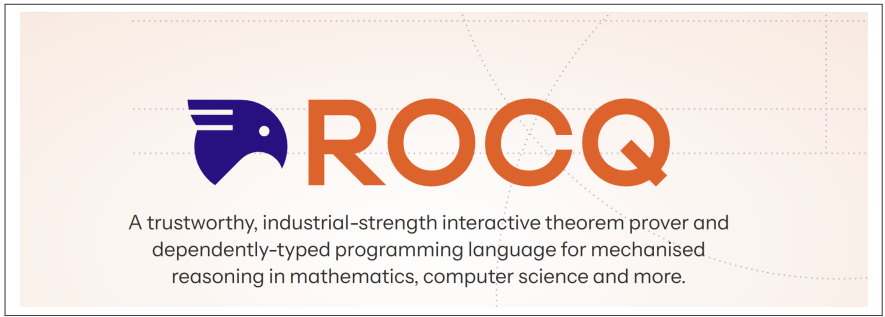
Paper 4: *"Bisimulations and Reversibility"*, C. Aubert, I. Phillips and I. Ulidowski

Discussion:

- The presented bisimulations are non-interleaving and capture causal dependencies on processes
- The authors consider bisimulations which differ w.r.t. the presence of a bijection preserving labels and causal dependencies, the invariance under key renaming, the inclusion of backward moves in the bisimulation game. However, several other angles are yet to be explored

Table of Contents

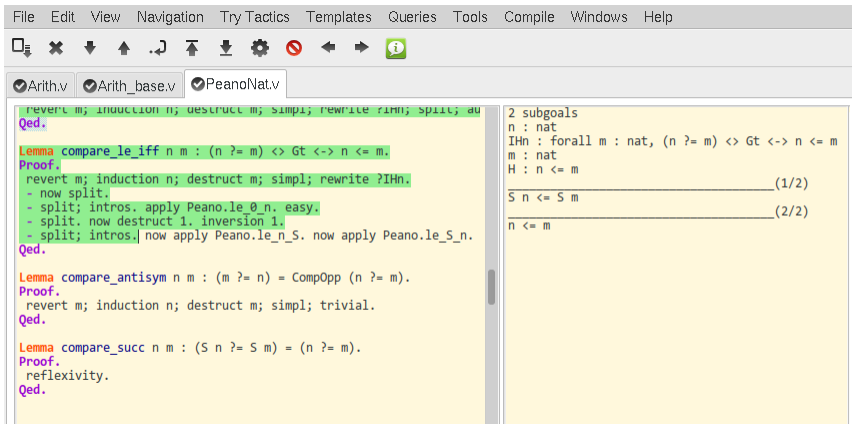
What are Interactive Theorem Provers?



Computer programs for constructing machine-checked mathematical proofs

Scope: programming languages, compilers, software, mathematics, …

What are their features?



- Interaction with the user (tactics, proof state, computation holes . . .)

What are their features?

- Interaction with the user (tactics, proof state, computation holes . . .)
- Automation

What are their features?

- Interaction with the user (tactics, proof state, computation holes . . .)
- Automation
- Libraries

What are their features?

- Interaction with the user (tactics, proof state, computation holes . . .)
- Automation
- Libraries
- Proofs checked by a trusted kernel

Why using them?

- Guarantee correctness of proofs

Why using them?

- Guarantee correctness of proofs
- Proof search and counterexamples search

Why using them?

- Guarantee correctness of proofs
- Proof search and counterexamples search
- Proofs with numerous cases, mostly not deep

Why using them?

- Guarantee correctness of proofs
- Proof search and counterexamples search
- Proofs with numerous cases, mostly not deep
- Increase comprehension of systems

Applications

Proof assistants can be useful in many domains, including:

- Metatheory of programming languages
  $\rightarrow$ "*Engineering Formal Metatheory*", B. Aydemir et al.

Applications

Proof assistants can be useful in many domains, including:

- Metatheory of programming languages
  → “*Engineering Formal Metatheory*”, B. Aydemir et al.
- Concurrency
  → “*The Concurrent Calculi Formalisation Benchmark*”, M. Carbone et al.

Applications

Proof assistants can be useful in many domains, including:

- Metatheory of programming languages
  → "*Engineering Formal Metatheory*", B. Aydemir et al.
- Concurrency
  → "*The Concurrent Calculi Formalisation Benchmark*", M. Carbone et al.
- Reversibility
  → "*A Certified Study of a Reversible Programming Language*", L. Paolini, M. Piccolo and L. Roversi

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Contributions:

- An overview of the existing binding encoding techniques at the time of publication
- A new binding encoding approach based on locally nameless variables representation + cofinite quantification
- Examples of applications of their new method and a comparison with other techniques

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Most modern programming languages include variables and binding constructs.

Binder: a piece of syntax introducing a variable and defining its scope

Example:

```
let X = 5 in (X + Y)
```

`X` is bound in `(X + Y)`, `Y` is free in `(X + Y)`

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Most modern programming languages include variables and binding constructs.

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

Example:

```
let X = 3 in (X - 2)
let Y = 3 in (Y - 2)
```

$\rightarrow$ Different approaches: De Bruijn indices, nominal techniques, locally named/nameless, HOAS, ...

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Locally nameless:

- Bound variables encoded as De Bruijn indices $\rightarrow$ α-renaming for free
- Free variables encoded using a type of names $\rightarrow$ statements involving free variables match their informal version
- Separation between free and bound names $\rightarrow$ no risk of variable capture

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Discussion:

- The provided method has been successful; however, there is no universal best binding encoding technique

Paper 5: *"Engineering Formal Metatheory"*, B. Aydemir et al.

Discussion:

- The provided method has been successful; however, there is no universal best binding encoding technique
- The problem of encoding binders is central in formalizations of (reversible) concurrent calculi as well

Paper 6: "*The Concurrent Calculi Formalisation Benchmark*", M. Carbone et al.

Contributions:

A set of three benchmark problems for the formalization of concurrent calculi

Goals:

Assess current state of the art, highlight open research problems, identify best practices and techniques

Setting:

Different fragments of the π-calculus

Paper 6: “*The Concurrent Calculi Formalisation Benchmark*”, M. Carbone et al.

The problems address in isolation three main issues arising when formalizing concurrent calculi:

- *Linearity:* each channel is used by processes exactly once
- *Scope extrusion:* restricted names can be shared, thus extending their scope
- *Coinductive reasoning:* coinduction is needed to define and reason about potentially infinite objects. Bisimulations are an example of coinductive definition

Paper 6: "*The Concurrent Calculi Formalisation Benchmark*", M. Carbone et al.

Discussion:

- Mechanizing concurrent calculi is of interest, presents known issues, and there are open problems
- Solutions to the first and second challenge have already been proposed[1]. A solution to the third challenge is currently under development[2].

[1] "*A Beluga Formalization of the Harmony Lemma in the π-Calculus*", G. Cecilia and A. Momigliano

[2] "*Barbed Similarity for the π-Calculus in Beluga: A Case Study in Coinductive Reasoning*", L. Trogni, G. Cecilia and A. Momigliano

Paper 7: *"A Certified Study of a Reversible Programming Language"*, L. Paolini, M. Piccolo and L. Roversi

Contributions:

- Description of a big-step operational semantics and a denotational semantics for Janus
- Proof of full abstraction between the two semantics
- Certified proofs of all results in Matita

Paper 7: *"A Certified Study of a Reversible Programming Language"*, L. Paolini, M. Piccolo and L. Roversi

Janus: a sequential reversible programming language

Every state has a unique predecessor state $\rightarrow$ programs can also be executed backwards

Paper 7: *"A Certified Study of a Reversible Programming Language"*, L. Paolini, M. Piccolo and L. Roversi

Janus: a sequential reversible programming language

Every state has a unique predecessor state $\rightarrow$ programs can also be executed backwards

Example: reversible if statement

```
if x = 0 then
    y += 1
else
    y -= 1
fi y > 0
```

Paper 7: *"A Certified Study of a Reversible Programming Language"*, L. Paolini, M. Piccolo and L. Roversi

Big-step operational semantics: the forward/backward evaluation of a statement in a certain state returns the following/preceding state

Denotational semantics: each statement is interpreted as a partial injective function

Full abstraction: the two semantics are equivalent

Matita: developed at University of Bologna. Based on the Calculus of (Co)Inductive Constructions, the dependent type theory underlying Rocq

Paper 7: *"A Certified Study of a Reversible Programming Language"*, L. Paolini, M. Piccolo and L. Roversi

Discussion:

- Not only reversibility and interactive theorem proving, but also category theory: properties of partial injective functions
- One of the few formalizations of sequential reversible models. So far, the only formalization of a reversible concurrent model is my encoding of $CCSK^P$ [3].

[3] *"A Formalization of the Reversible Concurrent Calculus $CCSK^P$ in Beluga"*, G. Cecilia

Table of Contents

Formalizing Reversible Concurrent Calculi

- Concurrent calculi: several formalizations

Formalizing Reversible Concurrent Calculi

- Concurrent calculi: several formalizations
- Reversible sequential models: few formalizations

Formalizing Reversible Concurrent Calculi

- Concurrent calculi: several formalizations
- Reversible sequential models: few formalizations
- Reversible concurrent calculi: one formalization

Formalizing Reversible Concurrent Calculi

- Concurrent calculi: several formalizations
- Reversible sequential models: few formalizations
- Reversible concurrent calculi: one formalization

$\rightarrow$ Formalizing reversible concurrent calculi
In particular *"An Axiomatic Theory for Reversible Computation"*, to provide a mechanized framework to check their soundness

Reversible Notions of Bisimulation and Congruence

Several notions of bisimulation have been studied for reversible concurrent calculi, covering several aspects (invariance under key renaming, presence of a bijection preserving labels and causal dependencies, etc.)

Reversible Notions of Bisimulation and Congruence

Several notions of bisimulation have been studied for reversible concurrent calculi, covering several aspects (invariance under key renaming, presence of a bijection preserving labels and causal dependencies, etc.)

However, other dimensions are yet to be explored:

- Power of observation on processes (reduction/barbed bisimulation)
- Abstraction from internal transitions (weak bisimulations)
- Invariance under contexts

Reversible Notions of Bisimulation and Congruence

Several notions of bisimulation have been studied for reversible concurrent calculi, covering several aspects (invariance under key renaming, presence of a bijection preserving labels and causal dependencies, etc.)

However, other dimensions are yet to be explored:

- Power of observation on processes (reduction/barbed bisimulation)
- Abstraction from internal transitions (weak bisimulations)
- Invariance under contexts

Moreover, formal proofs regarding bisimulation require dealing with coinductive reasoning

Benchmark Challenges for the Mechanization of Concurrent Models

The problems of the *"Concurrent Calculi Formalisation Benchmark"* are still open. In particular, the third challenge is still unsolved

$\rightarrow$ *"Barbed Similarity for the π-Calculus in Beluga: A Case Study in Coinductive Reasoning"*, L. Trogni, G. Cecilia and A. Momigliano

Benchmark Challenges for the Mechanization of Concurrent Models

The problems of the *"Concurrent Calculi Formalisation Benchmark"* are still open. In particular, the third challenge is still unsolved

$\rightarrow$ *"Barbed Similarity for the π-Calculus in Beluga: A Case Study in Coinductive Reasoning"*, L. Trogni, G. Cecilia and A. Momigliano

Moreover, the development of a set of benchmark problems for the formalization of reversible models of computation could be a promising research direction

Thank you for listening!
Any questions?