Multiparty Asynchronous Session Types: A Mechanised Proof of Subject Reduction (Artifact)

Dawit Tirore
IT University of Copenhagen, Denmark

Jesper Bengtson
IT University of Copenhagen, Denmark

Marco Carbone
IT University of Copenhagen, Denmark

Abstract

Session types offer a type-based approach to describing the message exchange protocols between participants in communication-based systems. Initially, they were introduced in a binary setting, specifying communication patterns between two components. With the advent of multiparty session types (MPST), the typing discipline was extended to arbitrarily many components. In MPST, communication patterns are given in terms of global types, an Alice-Bob notation that gives a global view of how components interact. A central theorem of MPST is subject reduction: a well-typed system remains well-typed after reduction. The literature contains some formulations of MPST with proofs of subject reduction that have later been shown to be incorrect. In this paper, we show that the subject reduction proof of the original formulation of MPST by Honda et al. contains some flaws. Additionally, we provide a restriction to the theory and show that, for this fragment, subject reduction does indeed hold. Finally, we use subject reduction to show that well-typed processes never go wrong. All of our proofs are mechanised using the Coq proof assistant.

This artifact accompanies our paper [1]. It contains the Coq mechanisation of the theory described therein.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Type structures; Theory of computation → Program specifications

Keywords and phrases Session types, Multiparty, Mechanisation, Coq

Digital Object Identifier 10.4230/DARTS.11.2.3

Funding This work was supported by DFF-Research Project 1 Grant no. 1026-00385A, from The Danish Council for Independent Research for the Natural Sciences (FNU)

Related Article Dawit Tirore, Jesper Bengtson, and Marco Carbone, "Multiparty Asynchronous Session Types: A Mechanised Proof of Subject Reduction", in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 31:1–31:30, 2025.
`https://doi.org/10.4230/LIPIcs.ECOOP.2025.31`

Related Conference 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact accompanies the paper Multiparty Asynchronous Session Types: A Mechanised Proof of Subject Reduction. It consists of a Coq (version 8.15.0) mechanisation of the subject reduction theorem and communication safety for multiparty asynchronous session types. The artifact is both functional and reusable. In particular, our mechanisation of multiparty asynchronous session types with explicit channels provides a solid foundation for future mechanised developments. Several components of the development are self-contained and can be reused independently. These include the theory of global types and their projection to local types, the formal correspondence between global and local semantics, the typing infrastructure, the proof of subject reduction, and the proof of communication safety. The development can also serve educational purposes, offering a comprehensive example of how to formalise a non-trivial type system and its meta-theory in Coq.

2 Content

The artifact package includes:
- All the Coq code carrying out all the proofs of the paper
- A Docker container that does not require any installation (besides Docker)

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://github.com/Tirore96/subject_reduction/tree/ECOOP2025`.
A Docker container version is available at: `https://doi.org/10.5281/zenodo.15291182`.

4 Tested platforms

The provided artifact was tested on a Mac OS X Sequoia 15.4.1 machine (Apple M4 Pro with 24 GiB of memory). Also, the Docker container was tested using Docker Desktop 4.40.0 for OS X.

5 License

The artifact is available under the The MIT License (`https://opensource.org/license/mit`).

6 MD5 sum of the artifact

259a3bb15bd95021e4570feb01312e77

7 Size of the artifact

2.9 GiB

A Compiling the Coq Code

Compiling with Docker. A Docker image containing the Coq code and its dependencies has been made available. To spin up a container based on this image, first load the image:

```
docker load -i ecoop25_subject_reduction_docker.tar.gz
```

And then run it:

```
docker run -it ecoop25_subject_reduction_docker /bin/bash
```

This opens an interactive shell inside the container. Now, navigate to the `subject_reduction` folder by running `cd subject_reduction` and checkout the correct branch with `git checkout ECOOP2025`. The Coq code can now be compiled by running `make -f CoqMakefile`.

For the expected output please see the section "Expected output" below.

Installing from Scratch. This section provides installation instructions without the use of a docker image.
Make sure opam version is 2.1.0 or above (`opam -version`).
If opam has not been used before, it must be initialised

```
opam init
eval $(opam env)
```

Next run the following commands from the `subject_reduction/` directory:

```
opam switch create ARTIFACT 4.10.2
eval $(opam env --switch=ARTIFACT)
opam pin add coq 8.15.0
opam repo add coq-released https://coq.inria.fr/opam/released
opam install coq-mathcomp-ssreflect.1.17.0
opam install coq-paco.4.2.0
opam install coq-mathcomp-zify.1.3.0+1.12+8.13
opam install coq-deriving.0.1.1
opam install coq-equations.1.3+8.15
coq_makefile -f _CoqProject -o CoqMakeFile
make -f CoqMakeFile
```

You can go back to your own switch with `eval $(opam env -switch=NAMEOFYOURSWITCH)`

Expected Output. The last part of SubjectRed4.v is three print statements

```
Print Assumptions OFT_cong.
Print Assumptions subject_reduction_final.
Print Assumptions subject_reduction_empty.
```

They print the additional axioms (besides standard CIC axioms) that our Subject Reduction theorem (Theorem 26 in the paper) relies on. Common for all three statements in the theorem, is that they rely only on functional extensionality, which is printed three times.

To compile the proofs, run the command make -f CoqMakefile in the root of the directory. The expected output is:

```
....
COQC theories/Process/SubjectRed.v
COQC theories/Process/linearity_equiv.v
COQC theories/Process/SubjectRed2.v
COQC theories/Process/SubjectRed3.v
COQC theories/Process/SubjectRed4.v
COQC theories/Process/Safety.v
Axioms:
FunctionalExtensionality.functional_extensionality_dep
  : forall (A : Type) (B : A -> Type) (f g : forall x : A, B x),
    (forall x : A, f x = g x) -> f = g
Axioms:
FunctionalExtensionality.functional_extensionality_dep
   : forall (A : Type) (B : A -> Type) (f g : forall x : A, B x),
    (forall x : A, f x = g x) -> f = g
```

```
Axioms:
FunctionalExtensionality.functional_extensionality_dep
   : forall (A : Type) (B : A -> Type) (f g : forall x : A, B x),
    (forall x : A, f x = g x) -> f = g
Axioms:
FunctionalExtensionality.functional_extensionality_dep
  : forall (A : Type) (B : A -> Type) (f g : forall x : A, B x),
    (forall x : A, f x = g x) -> f = g
```

To remove the additional files generated by compilation, run the command `make -f CoqMakefile clean` in the root of the directory.

Mapping definitions. This is a comprehensive list of all the main definitions and theorems from the paper. All these theorems are also directly linked to the online repository (`https://github.com/Tirore96/subject_reduction/tree/ECOOP2025/theories`) via the rooster icon in the paper. The root is the theories folder.

- Section 2
  - Process syntax: Process/processSyntax.v as process
  - Congruence: Process/Congruence.v as Cong
  - Reduction: Process/SubjectRed.v as Sem
- Section 3
  - Syntax: IndTypes/syntax.v where global types are defined as gType and local types as lType (Note we only have message type bool, and the message type int used in the paper is there to illustrate examples)
  - Projection: Projection of Tirore et al. Their projection implementation is contained in Projection/, IndTypes/andCoTypes/*and the procedure isprojinProjection/indProj.v
  - Semantics
    - Global types: linearity.v where the relation is step
    - Local types and environments: harmony.v where the relations are Estep and EnvStep
  - Linearity
    - Trace: linearity.v as Tr
    - Input/output dependence: linearity.v as exists_dep InDep and exists_dep OutDep. The paper presentation of dependence has been simplified. In Process/linearity_equiv.v we prove this definition equivalent to the coinductive formulation of linearity
    - Coinductive linearity definition: linearDecide.v as LinearCo
    - Decidability of Linearity: linearDecide.v where the lemma is Linear_decidable
  - Projection Theorem
    - Coinductive Equality: CoTypes/coLocal.v where the relation is EQ2
    - Decidability of Coinductive Equality: CoTypes/bisim.v as bisim
    - The theorem: harmony.v as harmony_sound and as harmony_complete
- Section 4
  - Queue types and queue contexts
    - Queue type and environment: Congruence.v as qType and q_env
    - Decomposition: Congruence.v as split_set
  - Coherence
    - Unstuck: Auxiliary step predicate for unstuckness in Process/Congruence.v as the predicate canStep. The full unstuck predicate as goodG.

    - Coherence definition: Process/Congruence.v where coherence of global types is coherentG, coherence of projected global types is coherent. Process/SubjectRed4.v where decomposed local type environment is coherent as weak_coherent_as
- Section 5
  - Typing judgment: Process/Congruence.v as OFT
  - Partitioning: Process/Congruence.v, as the function filter_q which splits the typing environment for queues q_env
  - Typing of declarations: Process/Congruence.v as DefTyping
  - Substitution lemma: Process/Congruence.v as OFT_subst
  - Example typing of process using declarations: Process/Example.v as typing
  - Subject Congruence and Reduction: (1) in Process/Congruence.v as OFT_cong, (2) and (3) in Process/SubjectRed4.v as subject_reduction_final and subject_reduction_empty
  - Definition of error, Communication Safety theorem and corollary: Process/Safety.v as Error_struct, OFT_not_error_struct and OFT_not_error_sem

References

1 Dawit Tirore, Jesper Bengtson, and Marco Carbone. Multiparty asynchronous session types: A mechanised proof of subject reduction. In Jonathan Aldrich and Alexandra Silva, editors, *39th European Conference on Object-Oriented Programming, ECOOP 2025, June 30–July 4, 2025, Bergen, Norway*, volume 333 of *LIPIcs*, pages 31:1–31:30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ECOOP.2025.31.