Union Types with Disjoint Switches (Artifact)

Baber Rehman
The University of Hong Kong, China

Xuejing Huang
The University of Hong Kong, China

Ningning Xie
University of Cambridge, UK

Bruno C. d. S. Oliveira
The University of Hong Kong, China

Abstract

This artifact contains the mechanical formalization of the calculi associated with the paper Union Types with Disjoint Switches. All of the metatheory has been formalized in Coq theorem prover. We provide a docker image as well the code archive.

The paper studies a union calculus (λ_u). Primary idea of λ_u calculus is a type based disjoint switch construct for the elimination of union types. We also study several extensions of the λ_u calculus including intersection types, distributive subtyping, nominal types, parametric polymorphism and an extension for the empty types.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Union types, switch expression, disjointness, intersection types

Digital Object Identifier 10.4230/DARTS.8.2.17

Funding This research was funded by the University of Hong Kong and Hong Kong Research Grants Council projects number 17209519, 17209520 and 17209821.

Acknowledgements We thank the anonymous reviewers for their helpful and constructive comments.

Related Article Baber Rehman, Xuejing Huang, Ningning Xie, and Bruno C. d. S. Oliveira, "Union Types with Disjoint Switches", in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 25:1–25:31, 2022.
`https://doi.org/10.4230/LIPIcs.ECOOP.2022.25`

Related Conference 36th European Conference on Object-Oriented Programming (ECOOP 2022), June 6–10, 2022, Berlin, Germany

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2022 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

All of the the metatheory and theorems stated in paper can be found in the artifact. Please follow the steps under *Getting the artifact* to access the artifact and look into the directory hierarchy to verify the claims. Next, we explain the code structure in artifact and its correlation with the paper.

1.1 Code Structure

There are 4 sub-folders in the artifact in *code* folder:

1. section3
2. section4
3. section51
4. section52

Each folder contains Coq formalization for a variant of our calculus discussed in paper. Organization of folders is:

Correlation of folders in artifact and sections in paper

Folder	**System**	**Reference in paper**
section3	Union calculus	Discussed in section 3
section4	Union calculus with various advance features	Discussed in section 4
section51	An extension with parametric polymorphism	Discussed in section 5.1
section52	An extension with generalized subtyping	Discussed in section 5.2

Note. Note that section 5 in extended version of the paper corresponds to appendix A in ECOOP 2022 publication.

1.1.1 section3

- *syntax.v* contains syntax and disjointness properties of the union calculus.
- *typing.v* contains semantics and properties related to type-safety and determinism.

Correlation of section 3 in paper and folder section3 in artifact

Lemma in Paper	**Coq file**	**Lemma(s) in Coq File**
Lemma 3	syntax.v	BL_soundness and BL_completeness
Theorem 5	syntax.v	Disj_soundness and Disj_completeness
Theorem 7	typing.v	preservation
Theorem 8	typing.v	progress
Theorem 10	typing.v	determinism
Lemma 12	syntax.v	disj1_disj

1.1.2 section4

- *disjointness.v* contains disjointness properties of the union calculus with intersection types, nominal types and distributive subtyping.
- *typing.v* contains semantics and properties related to type-safety and determinism.
- *equivalence.v* contains distributive subtyping.

Correlation of section 4 in paper and folder section4 in artifact		
Lemma in Paper	**Coq file**	**Lemma(s) in Coq File**
Lemma 13	equivalence.v	algo_sub_arrow_inv
Lemma 14	equivalence.v	dsub2asub
Lemma 15	equivalence.v	decidability
Theorem 18	disjointness.v	Disj_soundness and Disj_completeness
Lemma 19	disjointness.v	elem_in_findsubtypes_sub
Lemma 20	disjointness.v	ord_in_findsubtypes
Theorem 21	typing.v	preservation
Theorem 22	typing.v	progress
Theorem 23	typing.v	determinism

1.1.3 section51

- *syntax.v* contains syntax and disjointness properties of the union calculus with parametric polymorphism, intersection types and nominal types.
- *typing.v* contains semantics and properties related to type-safety and determinism.

Correlation of section 5.1 in paper and folder section51 in artifact		
Lemma in Paper	**Coq file**	**Lemma(s) in Coq File**
Lemma 25	typing.v	disj_subst
Lemma 26	typing.v	disj_narrowing

1.1.4 section52

- *syntax.v* contains syntax and disjointness properties of the union calculus with intersection types, nominal types and general subtyping rule.
- *typing.v* contains semantics and properties related to type-safety and determinism.

Correlation of section 5.2 in paper and folder section52 in artifact		
Lemma in Paper	**Coq file**	**Lemma(s) in Coq File**
Lemma 27	syntax.v	bot_sub_all
Lemma 28	syntax.v	disj_bot_like

2 Content

The artifact package includes:

- Coq formalization of the calculi discussed in paper
- Docker Image
- README file
- Extended version of ECOOP publication

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/baberrehman/disjoint-switches`.

We provide two alternatives to access and run the artifact:

1. Docker Image
2. Build From Scratch

3.1 Docker Image

This is the easiest way to run the artifact. We provide a docker image with all the dependencies installed in it. This section explains how to pull the docker image of artifact from docker repo and use it. Run the following commands one by one in terminal:

- docker pull baberrehman/switches
- docker run -it baberrehman/switches

The artifact is located in the directory: */home/coq/code*

- cd /home/coq/code

There are 4 sub-folders in the artifact, with make file in each.

1. section3 $\longrightarrow$ Discussed in section 3 in paper
2. section4 $\longrightarrow$ Discussed in section 4 in paper
3. section51 $\longrightarrow$ Discussed in section 5.1 in paper
4. section52 $\longrightarrow$ Discussed in section 5.2 in paper

Note. Note that section 5 in extended version of the paper corresponds to appendix A in ECOOP 2022 publication. Go to each folder and run make:

3.1.1 section3

- cd /home/coq/code/section3
- make

3.1.2 section4

- cd /home/coq/code/section4
- make

Note that compilation of *equivalence.v* takes a few minutes to complete.

3.1.3 section51

- cd /home/coq/code/section51
- make

3.1.4 section52

- cd /home/coq/code/section52
- make

Please feel free to go through the code in each section. *vim* and *cat* commands are available to look into the files. Recommended way to look into the files is by downloading the code archive from the given git repo. This completes the evaluation of artifact following docker image.

3.2 Build from Scratch

This section explains how to build the artifact from scratch.

3.2.1 Prerequisites

We tested all the Coq files using Coq version 8.13.1. Please use same version for the sake of consistency. We recommend installing Coq using the opam package installer.

- opam install coq.8.13.1

Refer to this link for more information and installation steps: `https://coq.inria.fr/opam-using.html`

3.2.2 Required Libraries

Coq TLC and Coq Metatheory libraries are required to compile the code. Next, we explain briefly how to install each. TLC library can also be installed using the opam package installer. Run the following commands one by one to install TLC by opam package installer:

- opam repo add coq-released http://coq.inria.fr/opam/released
- opam install coq-tlc.20210316

Please refer to this link for detailed compilation and installation of Coq TLC: `https://github.com/charguer/tlc/tree/20210316`. Metatheory can be installed by following the instructions at this link: `https://github.com/plclub/metalib`. Metatheory library from branch **coq8.10** must be installed.

3.2.3 Getting the files

Use the following commands to clone our git repo.

- git clone https://github.com/baberrehman/disjoint-switches.git

You should be able to see all the Coq files inside folder *code* after cloning the repo. Alternatively you can download the zip file from repo and you should be able to see all the Coq files after unzipping it.

There are 4 sub-folders in the *code* folder, with make file in each.

1. section3 $\longrightarrow$ Discussed in section 3 in paper
2. section4 $\longrightarrow$ Discussed in section 4 in paper
3. section51 $\longrightarrow$ Discussed in section 5.1 in paper
4. section52 $\longrightarrow$ Discussed in section 5.2 in paper

Note. Please make sure to run **eval $(opam env)** if Coq is installed using opam. This command can be skipped otherwise. Also note that section 5 in extended version of the paper corresponds to appendix A in ECOOP 2022 publication.

Open the terminal in each folder and run make:

3.2.4 section3

- eval $(opam env) (optional)
- make

Similarly, section4, section51 and section52 can be compiled by opening the terminal in each respective folder and running the **make** command. Please feel free to go through the code in each section. This completes the evaluation of artifact following build from scratch.

4 Tested platforms

We tested all the Coq files using Coq version 8.13.1. Please use same version for the sake of consistency. Coq TLC and Coq Metatheory libraries are also required to run the artifact.

5 License

The artifact is available under Creative Common License on DARTS.

6 MD5 sum of the artifact

3da9190bf71bb5ec64106a174932c900

7 Size of the artifact

437 kB