Row and Bounded Polymorphism via Disjoint Polymorphism (Artifact)

Ningning Xie
The University of Hong Kong, Hong Kong, China
nnxie@cs.hku.hk

Bruno C. d. S. Oliveira
The University of Hong Kong, Hong Kong, China
bruno@cs.hku.hk

Xuan Bi
The University of Hong Kong, Hong Kong, China
xbi@cs.hku.hk

Tom Schrijvers
KU Leuven, Belgium
tom.schrijvers@cs.kuleuven.be

Abstract

The artifact contains the Coq formalization of the elaboration from row polymorphism and bounded polymorphism to disjoint polymorphism, as described in the paper "row and bounded polymorphism via disjoint polymorphism". We document in detail how to build and compile the Coq proofs from scratch, as well as the proof and structure. Moreover, we have provided a docker image that can be downloaded and used to check the proofs.

2012 ACM Subject Classification Theory of computation → Type theory; Software and its engineering-Object oriented languages; Software and its engineering → Polymorphism

Keywords and phrases Intersection types, bounded polymorphism, row polymorphism

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2020.23

1 Building Instructions

Our Coq proofs are verified in *Coq 8.8.2*. We rely on two Coq libraries: *metalib* for the locally nameless representation in our proofs; and *coq-equations* for defining logical relations using pattern matching.

We provide two ways to evaluate the artifact. Section 1.1 describes how to download the docker image. The docker image has all dependencies installed, and thus we can simply run the proofs. Section 1.2 describes how to build the proofs from scratch.

1.1 Docker Image

1.1.1 Getting a Docker Image

1. Open terminal
2. Type `docker pull ecoop2020/artifact`
   Sha245-code: `sha256:90a15bb8130a535e09a360d8c724fd55a92f661efe1939c822d5fddeb0b32048`.
3. Type `docker run -it ecoop2020/artifact`
4. The artifact is located in directory `/home/coq/proof/`

1.1.2 Build and Compile the Proofs

1. Type `cd /home/coq/proof`

2. There are two Coq directories, *elaborations* and *operational-semantics*
3. Go to either directory
4. Type `make` in the terminal to build and compile the proofs.
5. You should see something like the following (suppose `>` is the prompt):

```
> make
COQC file1.v
COQC file2.v
......
COQC filen.v
```

It takes roughly half a hour to finish the compilation.

1.2 Build from Scratch

1.2.1 Prerequisites

1. Install *Coq 8.8.2*. The recommended way to install Coq is via OPAM. Refer to `https://coq.inria.fr/opam/www/using.html` for detailed steps. Or one could download the pre-built packages for Windows and MacOS via `https://github.com/coq/coq/releases/tag/V8.8.2`. Choose a suitable installer according to your platform.
2. Make sure Coq is installed (type `coqc` in the terminal, if you see "command not found" this means you have not properly installed Coq), then install *metalib*, checking out the commit that is compatible with Coq 8.8.2:
   a. Open terminal
   b. Type `git clone https://github.com/plclub/metalib`
   c. Type `cd metalib/Metalib`
   d. Type `git checkout fca72de3d`
   e. Type `make install`
3. Install `coq-equations.1.0+8.8`, refer to `https://github.com/mattam82/Coq-Equations#installation`.
   a. Open terminal
   b. Type `opam repo add coq-released https://coq.inria.fr/opam/released`
   c. Type `opam install coq-equations.1.0+8.8`

1.2.2 Build and Compile the Proofs

1. Unzip the archive file
2. There are two Coq directories, *elaborations* and *operational-semantics*
3. Go to either directory
4. Type `make` in the terminal to build and compile the proofs.
5. You should see something like the following (suppose `>` is the prompt):

```
> make
COQC file1.v
COQC file2.v
......
COQC filen.v
```

It takes roughly half a hour to finish the compilation.

2 Proof Structure

We have two directories for the Coq proof.

1. `./elaborations`: All lemmas related to elaborations.
2. `./operational-semantics`: All lemmas related to coherence in operational semantics.

The differences are as follows.

1. Elaboration focuses only on type-preservation lemmas, while operational-semantics contains the elaboration from F_i^+ to F_{co}, and the coherence lemma for elaboration.
2. Operational-semantics contains more files related to logical relation and contextually equivalence.
3. Operational-semantics contains only the predicative subset of the systems, as the logical relation is only defined for F_i^+'s predicative subset (please refer to Paper footnote 4).

Other lemmas should be the same in these two versions.

2.1 Elaborations

`./elaborations`

- Utility: `LibTactics.v`
- Syntax of F_i^+ and $\lambda^{||}$: `Syntax_ott.v`
- F_i^+:
  - `Fii_inf.v`: infrastructure, mostly generated by Metalib.
  - `Infrastructure.v`: more advanced properties.
  - `Disjoint.v`: disjointness lemmas.
- $\lambda^{||}$:
  - `Row_inf.v`: infrastructure, mostly generated by Metalib.
  - `Row_Properties.v`: more advanced properties.
  - `Row_Elaboration.v`: elaboration function definitions and proofs.
- The intuitive elaboration scheme for the modified row type system:
  - `Row_Intuitive_Syntax.v`: definition and elaboration scheme.
  - `Row_Intuitive_Inf.v`: infrastructure, mostly generated by Metalib.
  - `Row_Intuitive_Elaboration.v`: elaboration proof.
- kernel $\mathsf{F}_{<:}$:
  - `FSub_Definition.v`: definition.
  - `FSub_Infrastructure.v`: infrastructure.
  - `FSub_Lemma.v`: more advanced properties.
  - `FSub_Elaboration.v`: elaboration function definitions and proofs.

2.2 Operational Semantics

`./operational-semantics`

- Utility: `LibTactics.v`
- Syntax of F_i^{+} and $\lambda^{||}$: `Syntax_ott.v`
- Axioms: `Assumed.v`
- F_i^{+}:
  - `Fii_inf.v`: infrastructure, mostly generated by Metalib.
  - `Infrastructure.v`: more advanced properties.
  - `TypeSystems`: type systems, including elaboration to F_{co}.
  - `Disjoint.v`: disjointness lemmas.
  - `LR.v`: logical relation definition.
  - `Compatibility.v`: compatibility lemmas for logical relation.
  - `SourceProperty.v`: lemmas related to elaboration to F_{co}.
  - `Coherence.v`: lemmas for contextual equivalence.
- F_{co}:
  - `SystemF_inf`: infrastructure, mostly generated by Metalib.
  - `TargetProperty.v`: more advanced properties.
- kernel $\mathsf{F}_{<:}$:
  - `FSub_Definition.v`: definition.
  - `FSub_Infrastructure.v`: infrastructure.
  - `FSub_Lemma.v`: more advanced properties.
  - `FSub_Elaboration.v`: elaboration function definitions and proofs.
  - `FSub_Operational.v`: coherence proofs.
- $\lambda^{||}$:
  - `Row_inf.v`: infrastructure, mostly generated by Metalib.
  - `Row_Properties.v`: more advanced properties.
  - `Row_Elaboration.v`: elaboration function definitions and proofs.
  - `Row_Operational.v`: coherence proofs.

3 Correspondence

3.1 Definitions

Paper	Definition	File	Name of formalization
F_i^+			
Figure 1	Expression	`Syntax_ott.v`	`Inductive sexp`
	Type	`Syntax_ott.v`	`Inductive sty`
	Subtyping	`Syntax_ott.v`	`Inductive sub`
	Typing	`Syntax_ott.v`	`Inductive has_type`
Figure 2	Top-like	`Syntax_ott.v`	`Inductive TopLike`
	Disjointness	`Syntax_ott.v`	`Inductive disjoint`
$\lambda^{\|\|}$			
Figure 3	Expression	`Syntax_ott.v`	`Inductive rexp`
	Type	`Syntax_ott.v`	`Inductive rt`
	Well-formedness	`Syntax_ott.v`	`Inductive wftc`
	Compatibility	`Syntax_ott.v`	`with cmp`
	Constraint list satisfaction	`Syntax_ott.v`	`Inductive cmpList`
	Type equivalence	`Syntax_ott.v`	`with teq`
	Constraint list equivalence	`Syntax_ott.v`	`with ceq`
Figure 4	Type-directed elaboration	`Row_Elaboration.v`	`Inductive wtt`
Figure 5	Translation of types	`Row_Intuitive_Inf.v`	`Fixpoint trans_rt`
	Type-directed elaboration	`Row_Intuitive_Elaboration.v`	`Inductive wtt`
Figure 6	Translation of types	`Row_Elaboration.v`	`Inductive trans_rt`
kernel $\mathsf{F}_{<:}$			
Figure 7	Expression	`FSub_Definitions.v`	`Inductive exp`
	Type	`FSub_Definitions.v`	`Inductive typ`
	Subtyping	`FSub_Definitions.v`	`Inductive fsub`
	Typing	`FSub_Elaboration.v`	`Inductive typing`
	Elaboration of types	`FSub_Elaboration.v`	`Inductive fsub_trans_typ`
	Elaboration of contexts	`FSub_Elaboration.v`	`Inductive fsub_trans_D`
	Elaboration of contexts	`FSub_Elaboration.v`	`Inductive fsub_trans_G`

3.2 Lemmas in Elaborations

./elaborations

Paper Lemma	File	Name of formalization
F_i^+		
Lemma 1	`Disjoint.v`	`Lemma disjoint_sub`
$\lambda^{\|\|}$		
Theorem 6	`Row_Intuitive_Elaboration.v`	`Theorem type_safe`
Lemma 10	`Row_Elaboration.v`	`Lemma trans_rt_substitution_distributivity`
Lemma 11	`Row_Elaboration.v`	`Lemma trans_teq`
Lemma 12	`Row_Elaboration.v`	`Lemma trans_r_trans_r_cmp`
		`Lemma bot_trans_r_bot_trans_r_cmp`
		`Lemma bot_trans_r_trans_r_cmp_and_trans_r_bot_trans_r_c`
Lemma 13	`Row_Elaboration.v`	`Lemma cmp_record_r`
Theorem 14	`Row_Elaboration.v`	`Lemma translation_well_formed`
kernel $\mathsf{F}_{<:}$		
Lemma 16	`FSub_Elaboration.v`	`Lemma fsub_trans_typ_exists`
		`Lemma fsub_trans_typ_uniq`
Lemma 18	`FSub_Elaboration.v`	`Lemma trans_subst_fsub_general`
Theorem 19	`FSub_Elaboration.v`	`Lemma trans_typing`

3.3 Lemmas in Operational Semantics

./operational-semantics

Paper Lemma	File	Name of formalization
F_i^+		
Theorem 2	`Coherence.v`	`Theorem coherence`
$\lambda^{\|\|}$		
Theorem 15	`Row_Operational.v`	`Theorem translation_coherence`
kernel $\mathsf{F}_{<:}$		
Theorem 20	`FSub_Operational.v`	`Theorem translation_coherence`

Note Theorem 21 (Simulation) a paper proof and is given in Appendix D (please refer to the paragraph after Theorem 21).