# Pure Methods for roDOT (Artifact)

Vlastimil Dort 🖂 回

Charles University, Prague, Czech Republic

Yufeng Li 🖂 University of Cambridge, UK

Ondřej Lhoták ⊠©

University of Waterloo, Canada

Pavel Parízek 🖂 💿 Charles University, Prague, Czech Republic

#### — Abstract -

The artifact for the paper Pure methods for roDOT (ECOOP 2024) contains the Coq mechanization of the theorems appearing in the paper, and the necessary definitions and lemmata. Additionally, the artifact contains a mechanization of the roDOT calculus presented in an earlier paper Reference mutability for DOT (ECOOP 2020). We used the

calculus from this paper as the baseline for our paper, but it has not been mechanized before.

The functionality of the artifact is the ability to verify the correctness of the theorems by running Coq.

Our code is based on a mechanization of a soundness proof for Field Mutable DOT.

2012 ACM Subject Classification Software and its engineering  $\rightarrow$  Formal language definitions; Software and its engineering  $\rightarrow$  Object oriented languages

Keywords and phrases type systems, DOT calculus, pure methods

Digital Object Identifier 10.4230/DARTS.10.2.6

Funding This work was supported by the Czech Science Foundation project 23-06506S, and by the Czech Ministry of Education, Youth and Sports project LL2325 of the ERC.CZ programme. This research was also supported by the Natural Sciences and Engineering Research Council of Canada.

Related Article Vlastimil Dort, Yufeng Li, Ondřej Lhoták, and Pavel Parízek, "Pure Methods for roDOT", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 13:1–13:29, 2024.

https://doi.org/10.4230/LIPIcs.ECOOP.2024.13

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

#### 1 Scope

The Coq code contains mechanizations of the theorems and the necessary definitions and lemmata in the submitted paper and its extended version [2]. Additionally, the artifact contains a mechanization of the roDOT calculus presented in the paper Reference mutability for DOT [1]. The code is based on a mechanization of soundness proof for Field Mutable DOT [3].

- Figure 1
  - **Variables** The mechanization uses a *locally nameless representation of variables*, see definition avar in Syntax/Vars.v. The kinds of variables (location, reference, parameter) are defined as varkind in Syntax/Vars.v. We use the typing context to associate the kind with a variable.
  - Terms are defined as trm and lit in Syntax/Terms.v.



© Vlastimil Dort, Yufeng Li, Ondřej Lhoták, and Pavel Parízek; licensed under Creative Commons License CC-BY 4.0 Dagstuhl Artifacts Series, Vol. 10, Issue 2, Artifact No. 6, pp. 6:1-6:8 Dagstuhl Artifacts Series



DAGSTUHL Dagstuhl Artifacts Series ARTIFACTS SERIES Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# 6:2 Pure Methods for roDOT (Artifact)

- **Definitions** are defined as def and defs in Syntax/Terms.v.
- **Stack** is defined as stack in Syntax/Stack.v.
- **Types** are defined as typ and dec in Syntax/Types.v.
- **Type names** are defined as typ\_label in Syntax/Labels.v.
- **Heap** is defined as heap in Syntax/Heap.v.
- Configuration is defined as config in Syntax/AbstractMachine.v.
- **Environment** is defined as renv in Syntax/Env.v.
- Figure 2 contains selected rules from Figures 8 and 10 in the appendix see the corresponding entries.
- Figure 3 Mutable reachability (Rea) is defined as mut\_reach in Mutability/MutableReachability.v.
- Theorem 1 soundness\_initial in Safety.v
- **Theorem 2** immutability\_guarantee in Mutability/ImmutabilityGuarantee.v.
- Definition 8 does not have a direct counterpart in the Coq code, subtyping is defined by subtyp in GeneralTyping/GeneralTyping.v.
- Lemma 9 reference\_nocap\_type in CanonicalForms/ReferenceTypes.v.
- Definition 11 does not have a direct counterpart in the Coq code. It is used unfolded in the premises of Theorem 16 and Definition 26.
- **Figure 4** 
  - $\texttt{ The first rule (TS-N) is defined as ty_incap_nocap in \texttt{GeneralTyping/GeneralTyping.v.}}$
  - The second rule (ST-NM) is defined as subtyp\_nocap\_mut\_top\_boundless in GeneralTyping/GeneralTyping.v.
- **Figure 5** 
  - **General subtyping** is defined as subtyp in GeneralTyping/GeneralTyping.v.
  - **Tight subtyping** is defined as subtyp\_t in CanonicalForms/TightTyping.v.
  - Precise typing is defined using precise\_flow in CanonicalForms/PreciseTyping.v.
  - **General typing** is defined as ty\_var in GeneralTyping/GeneralTyping.v.
  - **Tight typing** is defined as ty\_trm\_t in CanonicalForms/TightTyping.v.
  - Invertible typing is defined as ty\_var\_inv in CanonicalForms/FlatInvertibleTyping.v.
- Lemma 12 ty\_incap\_nocap and ty\_incap\_diag\_nocap\_sup in GeneralTyping/GeneralTyping.v.
- Lemma 13 invertible\_flat\_typing\_closure\_tight in CanonicalForms/FlatInvertibleTyping.v.
- Lemma 14 invertible\_flat\_typing\_or\_inv in CanonicalForms/FlatInvertibleTyping.v.
- Figure 7 contains selected rules from Figures 18 and 19 in appendix see the corresponding entries.
- Definition 15 does not have a direct counterpart in the Coq code. It is used unfolded in the conclusion of Theorem 16.
- Theorem 16 SefG\_I in Mutability/SefGuarantee.v
- Lemma 17 imm\_transport in Mutability/SefGuarantee.v
- Lemma 18 sef\_neq\_elim\_ty\_hole and sef\_neq\_elim\_heap\_corr in Mutability/SefGuarantee.v
- Lemma 19 sef\_neq\_ro\_ty\_config in Mutability/SefGuarantee.v
- Lemma 20 ref\_elim\_similar\_conf\_sef in Similarity/ConfigSefSimilarity.v
- Definition 21 heap\_flds\_sim in Mutability/SefGuarantee.v
- Lemma 22 red\_heap\_flds\_sim in Mutability/SefGuarantee.v

#### V. Dort, Y. Li, O. Lhoták, and P. Parízek

- Definition 23 Transform in Transformation/Transformation.v
- Definition 24 does not have a direct counterpart in the Coq code. It is used unfolded in the conclusions of Theorem 25 and Lemma 58. The similarity transformation is defined as trf\_similarity in Transformation/TransformationSimilarity.v.
- Theorem 25 transformation\_preserved\_if\_terminates in Transformation/TransformationLiftingPreservation.v
- Definition 26 trf\_swap\_calls\_local in Transformation/TransformationSwapCalls.v
- Theorem 27 swap\_calls\_transformation\_guarantee in Transformation/TransformationSwapCallsGuarantee.v
- Figure 8 ty\_trm in GeneralTyping/GeneralTyping.v
- Figure 9 ty\_var in GeneralTyping/GeneralTyping.v
- Figure 10 subtyp in GeneralTyping/GeneralTyping.v
- Figure 11 ty\_incap and typ\_capbnd in GeneralTyping/GeneralTyping.v
- Figure 12 ty\_def and ty\_defs in GeneralTyping/GeneralTyping.v
- Figure 13 red in OperationalSemantics/OperationalSemantics.v
- Figure 14
  - Heap correspondence is defined as heap\_correspond in CanonicalForms/HeapCorrespondence.v.
  - Environment correspondence is defined as renv\_corr in CanonicalForms/EnvCorrespondence.v.
  - Stack typing is defined as ty\_stack in CanonicalForms/ConfigTyping.v.
  - **Configuration typing** is defined as ty\_config in CanonicalForms/ConfigTyping.v. This definition has several assumptions about well formedness of the configuration (using variables of the correct kind and no unbound variables).
- Figure 15 ty\_inv\_atomic in CanonicalForms/LayeredTyping/LayeredTypingAtomic.v.
- Figure 16 ty\_inv\_basic in CanonicalForms/LayeredTyping/LayeredTypingBasic.v.
- Figure 17 ty\_inv\_basic\_closure in CanonicalForms/LayeredTyping/LayeredTypingBasicClosure.v.
- Figure 18 ty\_inv\_logic in CanonicalForms/LayeredTyping/LayeredTypingLogic.v.
- Figure 19 ty\_inv\_main in CanonicalForms/LayeredTyping/LayeredTypingMain.v.
- Figure 20 subtyp\_atomic in CanonicalForms/LayeredTyping/LayeredSubtypingAtomic.v.
- Figure 21 ty\_has\_M and ty\_has\_N\_atomic in CanonicalForms/LayeredTyping/LayeredCapabilityBasic.v.
- Figure 22 ty\_has\_M\_closure and ty\_has\_N\_closure in CanonicalForms/LayeredTyping/LayeredCapabilityBasicClosure.v.
- Lemma 28 invertible\_main\_to\_precise\_typ\_dec in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Lemma 29 invertible\_main\_to\_precise\_fld\_dec in CanonicalForms/InvertibleTyping.v.
- Lemma 30 invertible\_main\_to\_precise\_met\_dec in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.

# 6:4 Pure Methods for roDOT (Artifact)

- Lemma 31 tight\_to\_invertible\_main in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Lemma 32 invertible\_main\_typing\_closure\_tight in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Lemma 33 ty\_inv\_main\_lc\_and in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Lemma 34 ty\_inv\_main\_lc\_and\_invl and ty\_inv\_main\_lc\_and\_invr in CanonicalForms/LayeredTyping/LayeredTypingMain.v.
- Lemma 35 ty\_inv\_main\_lc\_or\_diag in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Lemma 36 ty\_inv\_main\_lc\_nocap\_main\_replace in CanonicalForms/LayeredTyping/LayeredTypingMain.v.
- Lemma 37 ty\_inv\_main\_tight in CanonicalForms/LayeredTyping/LayeredTypingMainCount.v.
- Lemma 38 ty\_inv\_main\_lc\_N\_cases in CanonicalForms/LayeredTyping/LayeredTypingMainHard.v.
- Definitions 39 to 43 do not have a conterpart in the Coq code.
- Definition 44 We use a pair of lists of variables (list var).
- Definition 45 similar\_trm, similar\_avar, similar\_item, and similar\_stack in Similarity/SyntaxSimilarity.v
- Definition 46 similar\_conf\_sef in Similarity/ConfigSefSimilarity.v
- Lemma 47 conf\_sim\_red\_append in Similarity/ConfigSefSimilarity.v
- Lemma 48 conf\_sim\_red\_create in Similarity/ConfigSefSimilarity.v
- Definition 49 TransformCondTypeIdentical in Transformation/Transformation.v, applied using TransformEnsures
- Definition 50 TransformCondTyping in Transformation/TransformationTerm.v, applied using TransformRequires
- Figure 23 trf\_local\_idtyp\_trm, trf\_local\_idtyp\_lit, trf\_local\_idtyp\_def, and trf\_local\_idtyp\_defs in Transformation/TransformationLifting.v
- Figure 24 trf\_focus\_idtyp\_conf and trf\_lift\_idtyp\_conf in Transformation/TransformationConfig.v
- Definition 52 TransformWeakeningCompat in Transformation/TransformationTerm.v
- Lemma 53 WeakeningCompatTransformationTrm in Transformation/TransformationLifting.v
- Definition 54 Expressed by composing TransformPreserves in Transformation/Transformation.v with TransformIsAnswer in Transformation/TransformationConfig.v
- Lemma 55 transformation\_preserved in Transformation/TransformationLiftingPreservation.v
- Definition 56 TransformConfigReducesToShortIfTerminates in Transformation/TransformationConfig.v, applied to trf\_similarity in Transformation/TransformationSimilarity.v.
- Lemma 57 TransformConfigReducesTo1Similarity in Transformation/TransformationSimilarity.v,
- Lemma 58 swap\_calls\_transformation\_preservation in Transformation/TransformationSwapCallsGuarantee.v
- Lemma 59 TransformSwapCallsWeakening in Transformation/TransformationSwapCalls.v
- Lemma 60 transformation\_swap\_calls\_reduces\_to\_similarity in Transformation/TransformationSwapCallsPreservation.v

#### V. Dort, Y. Li, O. Lhoták, and P. Parízek

# 2 Content

The artifact package includes:

The main component of the artifact is a Coq project containing the definitions of the type system in the paper and proofs of its soundness and the guarantees. The project contains the Coq source code. In the provided Docker image, the source code can be found under the directory /root/rodot.



# 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://hub.docker.com/r/rodotcalculus/ecoop2024-artifact.



# **Tested platforms**

Running the artifact requires an  $x86_64$  machine with Linux and Docker.

#### 5 License

The artifact is available under the MIT license.

# 6 MD5 sum of the artifact

a32d393a3275275c52e0ae40115360c6

#### 7 Size of the artifact

677.5 KiB

#### A Building the project

Compiling the code requires  $\text{Coq}^1$  version 8.10.2, and the TLC library<sup>2</sup> version 20181116.

The following commands can be used to install Coq and TLC using the OCaml Package Manager<sup>3</sup>:

```
opam init --compiler=4.09.1 --disable-sandboxing -a
opam pin add coq 8.10.2 -y
opam repo add coq-released http://coq.inria.fr/opam/released
opam pin add coq-tlc 20181116 -y
```

To build the project and verify the correctness of the proof, run the following command in the rodot directory of the artifact:

make -j4

<sup>&</sup>lt;sup>1</sup> https://coq.inria.fr/

<sup>&</sup>lt;sup>2</sup> https://www.chargueraud.org/softs/tlc/

<sup>&</sup>lt;sup>3</sup> https://opam.ocaml.org/, https://coq.inria.fr/opam-using.html

# 6:6 Pure Methods for roDOT (Artifact)

# B Quick-start guide for the Docker image

1. Run the docker image. To run the Docker image on your Linux machine, execute the following command.

sudo docker run -it --rm rodotcalculus/ecoop2024-artifact

- 2. Find the source code. When you run the docker image, a shell will start in the directory /root/rodot. The \*.v files in this directory and its subdirectories constitute the source code.
- 3. Build the project. To build the project and verify the correctness of the proof, run make -j4. You can omit the -j4 option or use a different number to control the level of parallelism of the build.

# C Re-creating the Docker image locally

The docker image can also be re-created locally from the artifact source code and the provided **Dockerfile**, using the following command:

sudo docker build -t ecoop2024-artifact .

# D Typing modes

Our Coq definitions and theorems support extensibility, which allows extending the calculus with additional typing rules, and verifying multiple versions of the calculus simultaneously. This is achieved by parameterizing the definitions and theorems with the parameter typing\_mode. Possible values of typing\_mode are defined in GeneralTyping/TypingMode.v. Each value represent a version of the calculus, where each version can have a different set of features. Some typing rules and theorems are only available if a specific feature is enabled. This is achieved by a feature check in the rule or theorem, for example mode\_has\_mutability typing\_mode ->.

- For the mechanization of the related paper, the mode representing this version of the calculus is rodot\_sef.
- For the mechanization of [1], the mode representing this version of the calculus is rodot.

#### **E** Differences between the paper presentation and mechanization

There are several differences between the presentation in the paper and the mechanization, in order to improve readability of the paper and improve organization and extensibility of the code.

- The mechanized syntax uses a locally nameless representation, where variables bound in terms and literals are represented by deBruijn indices rather than variable names.
- Definitions of an object are represented by a list of definitions (while they are structured by a binary intersection in the paper definition, the structure is irrelevant).
- The mechanization represents multiple variants of the calculus, where the syntax is shared, but the differences in typing rules are achieved through the typing mode mechanism described above.
- The mechanized syntax contains additional constructors that are not part of roDOT, such as trm\_apply, lit\_fun, ctx\_stop, item\_fun. However, the typing rules related to these constructors are disabled by the typing mode feature checks explained above. Since the theorems involve typed terms and machine configurations, these additional constructors are irrelevant.

#### V. Dort, Y. Li, O. Lhoták, and P. Parízek

- The kind of a variable (reference/location, etc.), which is determined by the variable letter in the paper, is kept track of in the typing context in the mechanization.
- The special handling of the self-reference in the typing of object literals and objects on the heap is implemented using the objctx parameter of typing. This is a parameter of each typing judgment in addition to the usual typing context. This allows typing object literals and objects on the heap with the same typing rules, passing objctx\_lit for typing object literals and objctx\_heap for typing heap items. For term typing, the objctx is empty.
- To avoid conflicts of variable names, the mechanized definitions and theorems contain additional well-formedness and freshness premises where necessary. These premises are implicitly assumed and not shown in the paper version.
- For the purpose of generating fresh variables and ensuring no conflict of variable names between entries of the machine configuration and types in the typing context, machine configurations carry an additional entry Lv, which stores all the variable names that appeared at any step of the execution so far.
- Definition types and object literals are not part of the definition of terms (trm) and types (typ) directly, but use separate definitions (lit, def) used through constructors typ\_rcd, lit\_obj.
- We use additional definitions to avoid code repetition and to improve extensibility.
- The definitions of transformations and similarity are structured using the type classes feature of Coq, to allow reusing definitions and lemmas for different syntactic elements.
- F Checking theorem assumptions

Coq code can use additional axioms, which are assumed true, so the correctness of the evaluation depends on correctness of the used axioms.

This artifact uses the following axioms of classical logic:

```
LibAxioms.prop_ext (from TLC library)
LibAxioms.indefinite_description (from TLC library)
LibAxioms.fun_ext_dep (from TLC library)
Classical_Prop.classic
```

In order to check what axioms are used (and that no additional axioms are used), we provide the script print-assumptions.sh. Its first argument names a module, and its second argument names a theorem to check.

To check the axioms used by the main theorems, run the following commands **after building the project**:

```
./print-assumptions.sh Safety soundness_initial
./print-assumptions.sh \
Mutability.ImmutabilityGuarantee immutability_guarantee
./print-assumptions.sh Mutability.SefGuarantee SefG_I
./print-assumptions.sh \
Transformation.TransformationSwapCallsGuarantee \
swap_calls_transformation_guarantee
```

#### - References -

- Vlastimil Dort and Ondřej Lhoták. Reference mutability for DOT. In Robert Hirschfeld and Tobias Pape, editors, 34th European Conference on Object-Oriented Programming, ECOOP 2020, November 15-17, 2020, Berlin, Germany (Virtual Conference), volume 166 of LIPIcs, pages 18:1–18:28. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ECOOP.2020.18.
- 2 Vlastimil Dort, Yufeng Li, Ondřej Lhoták, and Pavel Parízek. Pure methods for ro-

DOT (an extended version). Technical Report D3S-TR-2024-01, Dep. of Distributed and Dependable Systems, Charles University, 2024. URL: https://d3s.mff.cuni.cz/files/publications/dort\_pure\_report\_2024.pdf.

3 Ifaz Kabir. themaplelab/dot-public: A simpler syntactic soundness proof for dependent object types. https://github.com/themaplelab/ dot-public/tree/master/dot-simpler.