Artifact Guide

A Path to DOT: Formalizing Fully Path-Dependent Types

MARIANNA RAPOPORT, University of Waterloo, Canada
ONDŘEJ LHOTÁK, University of Waterloo, Canada

The Dependent Object Types (DOT) calculus aims to formalize the Scala programming language with a focus on *path-dependent types* — types such as $x.a_1 \ldots a_n.T$ that depend on the runtime value of a *path* $x.a_1 \ldots a_n$ to an object. Unfortunately, existing formulations of DOT can model only types of the form $x.A$ which depend on *variables* rather than general paths. This restriction makes it impossible to model nested module dependencies. Nesting small components inside larger ones is a necessary ingredient of a modular, scalable language. DOT's variable restriction thus undermines its ability to fully formalize a variety of programming-language features including Scala's module system, family polymorphism, and covariant specialization.

This paper presents the pDOT calculus, which generalizes DOT to support types that depend on paths of arbitrary length, as well as singleton types to track path equality. We show that naive approaches to add paths to DOT make it inherently unsound, and present necessary conditions for such a calculus to be sound. We discuss the key changes necessary to adapt the techniques of the DOT soundness proofs so that they can be applied to pDOT. Our paper comes with a Coq-mechanized type-safety proof of pDOT. With support for paths of arbitrary length, pDOT can realize DOT's full potential for formalizing Scala-like calculi.

1 GETTING STARTED GUIDE

This artifact presents the Coq formalization of the pDOT type-safety proof as presented in Section 5 of our paper.

Our Coq proof can be either found

- preinstalled on the following VirtualBox VM:

  https://drive.google.com/open?id=1XsQGPywn2ElOE9VGmh09pwBLPSftxTQk

- or in the following Github repository:

  https://git.io/dot-with-paths

1.1 Compiling the Proof

1.1.1 On VirtualBox. To use the VM,

1) start up VirtualBox;
2) select New VM;
3) add a new Linux (Ubuntu 64-bit) machine with the provided .vdi file as the hard disk;
4) start the VM;
5) it should log in automatically as user osboxes (password is osboxes.org if necessary);

Authors' addresses: Marianna Rapoport, University of Waterloo, Canada, mrapoport@uwaterloo.ca; Ondřej Lhoták, olhotak@uwaterloo.ca, University of Waterloo, Canada.

2475-1421/2019/10-ART1
https://doi.org/

You will find the directory containing a README.html file and the artifact on the Desktop. You may open this file from the VM using Firefox. For IDE options for Coq the VM comes preinstalled with the Proof General Emacs interface.

To compile the proof open a terminal and run:

```
cd dot-calculus/src/extensions/paths
make
```

This will compile the proof and re-generate the documentation.

1.1.2 On your machine. System requirements:

- make
- an installation of Coq 8.9.0, preferably through opam
- the TLC library which can be installed through

```
opam repo add coq-released http://coq.inria.fr/opam/released
opam install -j4 coq-tlc
```

To check out the repository and compile the proof do the following:

```
git clone https://github.com/amaurremi/dot-calculus
cd dot-calculus/src/extensions/paths
make
```

This will compile the proof and generate the documentation.

2 STEP BY STEP INSTRUCTIONS

2.1 Overview

The Coq development presented in this artifact formalizes the type-safety proof of the pDOT calculus as presented in our paper. Specifically, it defines the calculus itself (its abstract syntax, type system, and operational semantics) and its type safety proof. The Type Soundness Theorem proves that well-typed terms in pDOT either diverge (i.e. run forever) or reduce to a normal form, which includes values (functions and objects) or paths. Since the operational semantics does not reduce paths we present an Extended Type Soundness Theorem defined in terms of the reduction relation extended with the lookup operation that looks up paths in the runtime environment. This theorem states that a well-typed term either diverges or reduces to a value (which does not include paths).

2.2 How to Review this Artifact

2.2.1 Inspecting Source files. The documentation can be accessed through the Readme.html file, or directly through the dot-calculus/src/extensions/paths/doc directory. The README.html file lists links to pretty printed Coq source files, but the raw .v files can be found in the dot-calculus/src/extensions/paths directory. In the pretty-printed versions, the proof scripts are hidden by default; you may click on "Show Proofs" at the top of the page to display all the proofs, or click under the Lemma or Theorem statements to display their proofs.

2.2.2 Verifying Correctness and Paper Correspondence. Successful compilation using make indicates a correct proof.

You can grep for strings like admit and Admitted in the proof files to verify that we proved all the theorems. You can also browse the code in Emacs using the Proof General mode and see what assumptions or hypotheses have used by adding the following Coq command:

```
Print Assumptions <lemma name>.
```

For example, to see what assumptions the Extended Soundness Theorem uses, add the command Print Assumptions extended_safety. in Safety.v on Line 557 (after the proof of the extended_safety theorem).

2.3 Used Libraries and Axioms

The pDOT calculus is a generalization of the simple DOT proof by Rapoport et al. [2017]. The Coq formalization of this proof can be found in the dot-calculus/src/simple-proof/proof folder.

The pDOT calculus is formalized using the locally nameless representation with cofinite quantification [Aydemir et al. 2008] in which free variables are represented as named variables, and bound variables are represented as de Bruijn indices. We use the TLC library developed by Arthur Charguéraud that provides useful infrastructure for metatheory proofs. We include the Sequences library by Xavier Leroy into our development to reason about the reflexive, transitive closure of binary relations.

We configure Coq with the following axioms:

- functional extensionality
- propositional extensionality
- indefinite description
- law of excluded middle
- John Major's equality

We use John Major's equality to do dependent induction in Coq. We inherit the remaining axioms from the TLC library.

2.4 Paper Correspondence

The correspondence between the paper and Coq formalization is documented in Tables 1, 2, and 3.

Table 1. Correspondence of Definitions

Definition	In paper	File	Paper notations	Proof notations	In proof
Abstract Syntax	Fig. 1	Definitions.v			
- variable	Fig. 1	Definitions.v			avar
- term member	Fig. 1	Definitions.v			trm_label
- type member	Fig. 1	Definitions.v			typ_label
- path	Fig. 1	Definitions.v	$x.a.b.c$	p_sel x (c::b::a::nil)	path
			$p.a$	p·a	
			$p.b$	p··b	
- term	Fig. 1	Definitions.v			trm
- stable term	Fig. 1	Definitions.v			def_rhs
- value	Fig. 1	Definitions.v	$\nu(x\colon T)d$	ν(T)d	val
			$\lambda(x\colon T)\,t$	λ(T)t	
- definition	Fig. 1	Definitions.v	$\{a = t\}$	{a := t}	def
			$\{A = T\}$	{A := T}	
- type	Fig. 1	Definitions.v	$\{a\colon T\}$	{a : T}	typ
			$\{A\colon T..U\}$	{A >: T <: U}	
			$\forall(x\colon T)\,U$	∀(T)U	
			$p.A$	p↓A	
			p.type	{{p}}	
			$\mu\,(x\colon T)$	μ(T)	
			$T \wedge U$	T ∧ U	
			$\top$	⊤	
			$\bot$	⊥	
Type System					
- term typing	Fig. 2	Definitions.v	$\Gamma \vdash t\colon T$	Γ ⊢ t : T	ty_trm

Table 1. Correspondence of Definitions

Definition	In paper	File	Paper notations	Proof notations	In proof
- definition typing	Fig. 2	Definitions.v	$p; \Gamma \vdash d : T$	`x; bs; Γ ⊢ d : T` (single definition) `x; bs; Γ ⊢ d :: T` (multiple definitions) Here, p=`x.bs`, i.e. `x` is p's receiver, and `bs` are p's fields in reverse order	`ty_def` `ty_defs`
- tight bounds	Fig. 2	Definitions.v			`tight_bounds`
- subtyping	Fig. 2	Definitions.v	$\Gamma \vdash T <: U$	`Γ ⊢ T <: U`	`subtyp`
Operational semantics	Fig. 3	Reduction.v	$\gamma \mid t \longmapsto \gamma' \mid t'$ $\gamma \mid t \longmapsto^* \gamma' \mid t'$	`(γ, t) ⟼ (γ', t')` `(γ, t) ⟼* (γ', t')`	`red`
Path lookup	Fig. 4	Lookup.v	$\gamma \vdash p \rightsquigarrow s$ $\gamma \vdash s \rightsquigarrow^* s'$	`γ ⟦ p ⇝ s ⟧` `γ ⟦ s ⇝* s' ⟧`	`lookup_step`
Extended reduction	Sec. 5	Safety.v	$\gamma \mid t \twoheadrightarrow \gamma' \mid t'$ $\gamma \mid t \twoheadrightarrow^* \gamma' \mid t'$	`(γ, t) ↠ (γ', t')` `(γ, t) ↠* (γ', t')`	`extended_red`
Inert and record types	Fig. 5	Definitions.v	inert T inert Γ		`inert_typ` `inert`
Well-formed environments	Sec. 5.2.1	PreciseTyping.v			`wf`
Correspondence between a value and type environment	Sec. 5	Definitions.v	$\gamma : \Gamma$	`γ ∶ Γ`	`well_typed`

Table 2. Correspondence of Lemmas and Theorems

Theorem	File	In proof
Theorem 5.1 (Soundness)	Safety.v	`safety`
Theorem 5.2 (Extended Soundness)	Safety.v	`extended_safety`
Lemma 5.3 (Progress)	Safety.v	`progress`
Lemma 5.4 (Preservation)	Safety.v	`preservation`
Lemma 5.5	CanonicalForms.v	`canonical_forms_fun`

Table 3. Correspondence of Examples

Example	In paper	File
List example	Figure 6 a	ListExample.v
Compiler example	Figure 6 b	CompilerExample.v
Singleton type example	Figure 6 c	SingletonTypeExample.v

2.5 Proof Organization

2.5.1 Safety Proof. The Coq proof is split up into the following modules:

- *Definitions.v*: Definitions of pDOT's abstract syntax and type system.
- *Reduction.v*: Normal forms and the operational semantics of pDOT.
- *Safety.v*: *Final safety theorem* through Progress and Preservation.
- Lookup.v: Definition of path lookup and properties of lookup.
- Binding.v: Lemmas related to opening and variable binding.
- SubEnvironments.v: Lemmas related to subenvironments.
- Weakening.v: Weakening Lemma.
- RecordAndInertTypes.v: Lemmas related to record and inert types.
- Replacement.v: Properties of equivalent types.
- Narrowing.v: Narrowing Lemma.
- PreciseFlow.v and PreciseTyping.v: Lemmas related to elimination typing. In particular, reasons about the possible precise types that a path can have in an inert environment.
- TightTyping.v: Defines tight typing and subtyping.
- Substitution.v: Proves the Substitution Lemma.
- InvertibleTyping.v and ReplacementTyping.v: Lemmas related to introduction typing.
- GeneralToTight.v: Proves that in an inert context, general typing implies tight typing.
- CanonicalForms.v: Canonical Forms Lemma.
- Sequences.v: A library of relation operators by Xavier Leroy.

2.5.2 Examples.

- CompilerExample.v: The dotty-compiler example that contains paths of length greater than one.
- ListExample.v: A covariant-list implementation.
- SingletonTypeExample.v: Method chaining through singleton types.
- ExampleTactics.v: Helper tactics to prove the above examples.

Figure 1 shows a dependency graph between the Coq modules.

REFERENCES

Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. 2008. Engineering formal metatheory. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*. 3–15.

Marianna Rapoport, Ifaz Kabir, Paul He, and Ondřej Lhoták. 2017. A simple soundness proof for dependent object types. *PACMPL* 1, OOPSLA (2017), 46:1–46:27.

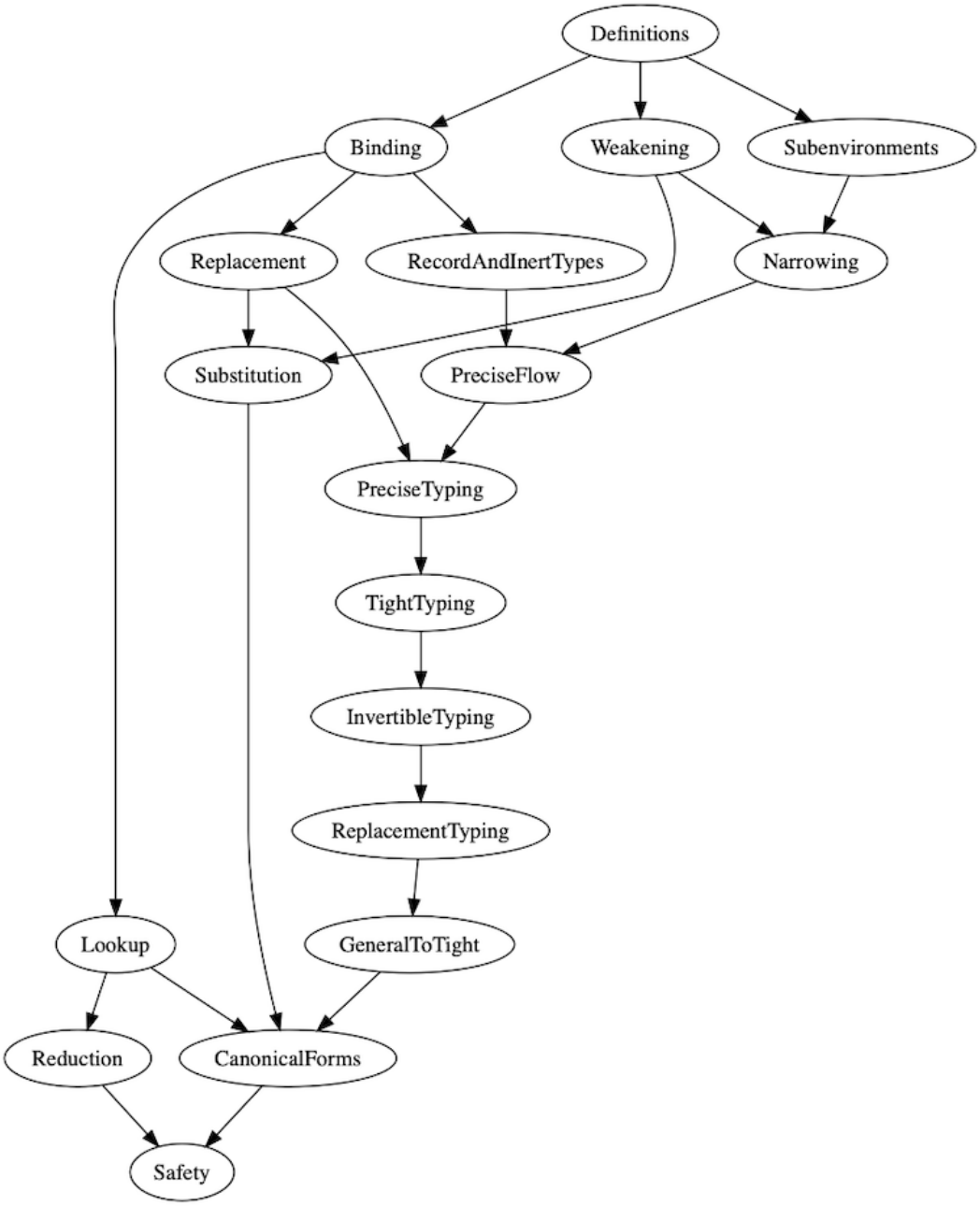

Fig. 1. Dependency between Coq proof files