On Julia's Efficient Algorithm for Subtyping Union Types and Covariant Tuples (Artifact)

Benjamin Chung
Northeastern University, Boston, MA, USA
bchung@ccs.neu.edu

Francesco Zappa Nardelli
Inria, France
francesco.zappa_nardelli@inria.fr

Jan Vitek
Northeastern University, Boston, MA, USA
Czech Technical University in Prague, Czech Republic
j.vitek@neu.edu

Abstract

The key claim in our paper Julia's efficient algorithm for subtyping unions and covariant tuples is that our algorithm works. This artifact provides support for that claim through two means: a Coq proof of the algorithm's correctness and an implementation of the algorithm. The single-file proof contains three proof-generating implementations of subtyping, based on normalization and two variations on the subtyping algorithm we describe. The implementation consists of a web interface to an OCaml implementation of our algorithm, which checks subtyping between any two arbitrary user-defined types and generates both the answer and a trace of algorithm execution.

2012 ACM Subject Classification Theory of computation → Type theory; Software and its engineering → General programming languages

Keywords and phrases Type systems, subtyping, algorithmic type systems, distributive unions

Digital Object Identifier 10.4230/DARTS.5.2.8

Related Article Benjamin Chung, Francesco Zappa Nardelli, and Jan Vitek, "Julia's Efficient Algorithm for Subtyping Unions and Covariant Tuples", in 33rd European Conference on Object-Oriented Programming (ECOOP 2019), LIPIcs, Vol. 134, pp. 24:1–24:15, 2019.
`https://dx.doi.org/10.4230/LIPIcs.ECOOP.2019.24`

Related Conference 33rd European Conference on Object-Oriented Programming (ECOOP 2019), July 15–19, 2019, London, United Kingdom

1 Introduction

This is the artifact for the pearl paper "On Julia's efficient algorithm for subtyping union types and covariant tuples." It consists of two primary components:

- `index.html`: An implementation of the subtyping algorithm running in a webpage. This implementation is modified only slightly from the one described in the paper to enable visualization. For sources, see the `web-impl` directory.
- `julia-iterators.v`: The Coq source code for the proofs referenced in our paper.

This document is a worse-formatted and non-executable version of `index.html`. We suggest the the online version (at `https://benchung.github.io/subtype-artifact/`) or simply open `index.html` from the artifact archive for information on the artifact and to try out our algorithm. The website version of the artifact is tested to work in Google Chrome, and should work in all modern browsers.

2 Proof

The proof script (found in `proof/julia-iterators.v`) depends on Coq 8.9.0. A detailed description of our proof can be found in section 3 of the paper. The proof is standalone, and has no library dependencies.

It relies on the standard library provided axiom `Eqdep.Eq_rect_eq.eq_rect_eq`, which establishes the invariance under substitution of dependent equality. In our formalization, structural type iterators are dependent upon the type over which they iterate. We rely on this axiom to decide when two iterators are iterating over the same or different types. It is an axiom in our system as it is independent of the calculus of constructions.

3 Implementation

We include a web implementation of our algorithm. To use it, please see `index.html` for the running implementation and instructions on its use and compilation.

Compiling the Implementation

The implementation is written in OCaml and compiled using js_of_ocaml. It requires:

- OCaml 4.07.0 or later
- opam 2.0.4 or later

To compile the OCaml to Javascript, run

```
make deps
make
```

in the `web-impl` subdirectory, which should update the file `web-impl/js/subtype.js`.

4 License

5 MD5 sum of the artifact

4d0356d94cd8d21f42e6b73de886f871

6 Size of the artifact

746657 bytes