Type Tailoring (Artifact)

Ashton Wiersdorf
University of Utah, Salt Lake City, UT, USA

Stephen Chang
University of Massachusetts Boston, MA, USA

Matthias Felleisen
Northeastern University, Boston, MA, USA

Ben Greenman
University of Utah, Salt Lake City, UT, USA

Abstract

Type systems evolve too slowly to keep up with the quick evolution of libraries – especially libraries that introduce abstractions. Type tailoring offers a lightweight solution by equipping the core language with an API for modifying the elaboration of surface code into the internal language of the typechecker. Through user-programmable elaboration, tailoring rules appear to improve the precision and expressiveness of the underlying type system. Furthermore, type tailoring cooperates with the host type system by expanding to code that the host then typechecks. In the context of a hygienic metaprogramming system, tailoring rules can even harmoniously compose with one another.

Type tailoring has emerged as a theme across several languages and metaprogramming systems, but never with direct support and rarely in the same shape twice. For example, both OCaml and Typed Racket enable forms of tailoring, but in quite different ways. This paper identifies key dimensions of type tailoring systems and tradeoffs along each dimension. It demonstrates the usefulness of tailoring with examples that cover sized vectors, database queries, and optional types. Finally, it outlines a vision for future research at the intersection of types and metaprogramming.

2012 ACM Subject Classification Software and its engineering → Extensible languages

Keywords and phrases Types, Metaprogramming, Macros, Partial Evaluation

Digital Object Identifier 10.4230/DARTS.10.2.24

Related Article Ashton Wiersdorf, Stephen Chang, Matthias Felleisen, and Ben Greenman, "Type Tailoring", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 44:1–44:27, 2024.
`https://doi.org/10.4230/LIPIcs.ECOOP.2024.44`

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 purpose of this artifact is to provide running code to support the claims in section 2 and subsection 3.8 of the paper. The programs demonstrate type tailoring in a variety of languages and settings.

2 Content

The artifact package includes:

- README file containing instructions on how to replicate the claims in the paper.
- Folder `claims` containing subdirectories for different categories of claims.

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 and associated instructions are also available at: `https://zenodo.org/doi/10.5281/zenodo.10578596`. On Zenodo we also provide a Docker image as a tarball which packages up all of the dependencies for the artifact. We recommend using this where possible.

4 Tested platforms

The artifact is known to work on the following platforms:

- AMD Ryzen 5 machine with 16 GB of RAM running Debian 12 (source & Docker container)
- Apple M1 Pro machine with 32 GB of RAM and running macOS 14 (source only)

16 GB of RAM is likely overkill for this artifact.

We recommend the Docker container (available on Zenodo and DockerHub), as it includes all dependencies, but because we built the Docker on an x86 machine it may not run on Apple Silicon. When we tested on an Apple M1 Pro, the Julia section failed. Everything else was okay, but ran relatively slowly.

5 License

The artifact is available under the MIT license.

6 MD5 sum of the artifact

06f8aa83ea549c491b1bc58ee78a9ac4

7 Size of the artifact

1.4 MB