



Defining Name Accessibility Using Scope Graphs (Artifact)

Aron Zwaan  

Delft University of Technology, The Netherlands

Casper Bach Poulsen  

Delft University of Technology, The Netherlands

Abstract

Many programming languages allow programmers to regulate *accessibility*; i.e., annotating a declaration with keywords such as `export` and `private` to indicate where it can be accessed. Despite the importance of name accessibility for, e.g., compilers, editor auto-completion and tooling, and automated refactorings, few existing type systems provide a formal account of name accessibility.

We present a declarative, executable, and language-parametric model for name accessibility, which provides a formal specification of name accessibility in Java, C#, C++, Rust, and Eiffel. We

achieve this by defining name accessibility as a predicate on *resolution paths* through *scope graphs*. Since scope graphs are a language-independent model of name resolution, our model provides a uniform approach to defining different accessibility policies for different languages.

Our model is implemented in Statix, a logic language for executable type system specification using scope graphs. We evaluate its correctness on a test suite that compares it with the C#, Java, and Rust compilers, and show we can synthesize access modifiers in programs with holes accurately.

2012 ACM Subject Classification Software and its engineering → Compilers; Software and its engineering → Language features; Theory of computation → Program constructs

Keywords and phrases access modifier, visibility, scope graph, name resolution

Digital Object Identifier 10.4230/DARTS.10.2.27

Related Article Aron Zwaan and Casper Bach Poulsen, “Defining Name Accessibility Using Scope Graphs”, in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 47:1–47:29, 2024.

<https://doi.org/10.4230/LIPIcs.ECOOP.2024.47>

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 artifact supports the following contributions from the paper:

- We present a specification of (various versions of) accessibility on modules (Section 5), subclasses (Section 6), and their conjunctive and disjunctive combination (Section 7); and
- we extend our specification with accessibility-restricting inheritance (Section 8).

The artifact contains a specification of AML (the language in which these access modifiers are formalized), written in Statix [3, 2].

- We implement our specification in Statix, and compare it with the standard compilers of Java, C#, and Rust. Moreover, we show access modifiers can be synthesized accurately using Statix-based Code Completion [1] (Section 10).

Facilities to specify and execute tests are included in the artifact, including scripts to access them.



© Aron Zwaan and Casper Bach Poulsen;
licensed under Creative Commons License CC-BY 4.0
Dagstuhl Artifacts Series, Vol. 10, Issue 2, Artifact No. 27, pp. 27:1–27:3



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



2 Content

The artifact package includes:

- The specification of AML (Access Modifier Language) presented in the paper. The specification is included in the `src/statics/access` directory, relative to the `aml` root. Each of the files corresponds (roughly) to a figure in the paper:
 - `base.stx`: Figure 7
 - `public.stx`: Figure 7, A-PUB and AP-PUB rules.
 - `auxiliary.stx`: Figure 9
 - `internal.stx`: Figure 10
 - `private.stx`: Figure 14
 - `protected.stx`: Figure 15
 - `private-protected.stx`: Figure 17
 - `protected-internal.stx`: Figure 17
 - `path.stx`: Figure 18
- A Spoofax language definition that makes it executable.
- Tests that validate the specification.

All these are included in the `/home/myuser/aml` directory of the included docker image.

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 is also available at Zenodo: <https://doi.org/10.5281/zenodo.11179594>.

4 Tested platforms

Platform Requirements:

- CPU: x86, Intel Core i7/i9 (virtualization)
- Memory: 6GB RAM
- Disk: 32GB Hard Drive
- Software: zip, gzip, docker

Running on an ARM chips (such as Apple Silicon), possibly with virtualization software and or the `--platform linux/amd64` argument to the `docker run` commands, may work, but is not tested nor officially supported by the artifact authors.

5 License

The artifact is available under the Apache License 2.0.

6 MD5 sum of the artifact

0ba292c4ccb4f5e844f8133c08085dc3

7 Size of the artifact

2.54 GiB

References

- 1 Daniël A. A. Pelsmaeker, Hendrik van Antwerpen, Casper Bach Poulsen, and Eelco Visser. Language-parametric static semantic code completion. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA):1–30, 2022. doi:10.1145/3527329.
- 2 Arjen Rouvoet, Hendrik van Antwerpen, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Knowing when to ask: sound scheduling of name resolution in type checkers derived from declarative specifications. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA), 2020. doi:10.1145/3428248.
- 3 Hendrik van Antwerpen, Casper Bach Poulsen, Arjen Rouvoet, and Eelco Visser. Scopes as types. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA), 2018. doi:10.1145/3276484.