KafKa: Gradual Typing for Objects (Artifact)

Benjamin Chung
Northeastern University, Boston, MA, USA

Paley Li
Czech Technical University, Prague, Czech Republic

Francesco Zappa Nardelli
INRIA, Paris, France

Jan Vitek
Northeastern University, Boston, MA, USA

Abstract

A wide range of gradual type systems have been proposed, providing many languages with the ability to mix typed and untyped code. However, hiding under language details, these gradual type systems have fundamentally different ideas of what it means to be typed. In this paper, we show that four of the most common gradual type systems provide distinct guarantees, and we give a formal framework for comparing gradual type systems for object-oriented languages. First, we show that the different gradual type systems are practically distinguishable via a three-part litmus test. Then, we present a formal framework for defining and comparing gradual type systems. Within this framework, different gradual type systems become translations between a common source and target language, allowing for direct comparison of semantics and guarantees.

2012 ACM Subject Classification Software and its engineering → Semantics

Keywords and phrases Gradual typing, object-orientation, language design, type systems

Digital Object Identifier 10.4230/DARTS.4.3.10

Related Article Benjamin Chung, Paley Li, Francesco Zappa Nardelli, and Jan Vitek, "KafKa: Gradual Typing for Objects", in Proceedings of the 32nd European Conference on Object-Oriented Programming (ECOOP 2018), LIPIcs, Vol. 109, pp. 12:1–12:25, 2018.
https://dx.doi.org/10.4230/LIPIcs.ECOOP.2018.12

Related Conference 32nd European Conference on Object-Oriented Programming (ECOOP 2018), July 19–21, 2018, Amsterdam, Netherlands

1 Scope

The scope of the artifact includes a complete implementation of the Kafka VM, the translation from native code, and a complete Coq proof of Kafka.

Our Kafka and native translation implementations are written in F#. The folder kafkaimpl contains the translation from the surface source language to kafka, the translation from kafka to CIL, and the type checker for Kafka. The folder kafkatests contains unit tests for the kafka implementation. The implementation of Kafka follows the same syntax and semantics as those presented in figure 3 and 4 of the paper. The implementation of our translation from the source language to Kafka follows the rules presented in section 5 of the paper. The result of each litmus test under each gradual semantics reflects the behavior expressed in section 3 of our paper.

The mechanized proof of type soundness for Kafka in Coq is found in the proof directory. There are three holes in the proof, in kafka.v:

- Transitivity of structural recursive subtyping (line number: 342)
- Soundness of subtyping (line number: 350)
- That subtyping still holds when the class table is expanded (line number: 642)

The first two components are well-known prior work (e.g. [1]). The third property simply requires that pre-existing subtyping judgments still hold when the class table is expanded.

2 Content

The artifact package includes:

- Kafka's .Net implementation (netimpl directory)
- Kafka's Coq proof (proof directory)
- Litmus tests in each of the four native languages and our source language (examples directory)

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 to DROPS, the artifact is also available at: `https://github.com/BenChung/GradualComparisonArtifact`.

4 Tested platforms

The Coq component of the artifact has no platform dependencies beyond requiring Coq 8.6. The .Net implementation is easiest to build on Windows (see instructions), and has been reported to be build on Linux under Mono.

4.1 Building the KafKa Implementation

To build the artifact under Windows with Visual Studio, open the kafkaimpl.sln file inside of the netImpl/kafkaimpl subdirectory in Visual Studio 2017 or later with F# installed. Restore the NuGet packages (right click on references under kafkaimpl in the solution explorer, click on "manage NuGet packages", then click "restore" on the bar at the top), and then build the solution.

Installation on Linux and related platforms is untested, though has been reported to work. Install mono-complete, nuget, and fsharp, then run "nuget restore" in the kafkaimpl subdirectory, followed by "msbuild".

4.2 Using the Kafka Implementation

The KafKa implementation takes two command line arguments: the semantics to use and the file to execute. The following table shows the arguments to use for each semantics:

Optional	opt
Transient	tra
Behavioral	beh
Concrete	con

The second argument is the path to the file to execute. Source language versions of the litmus tests are found in the examples/source subdirectory, and exhibit the same behavior as described in the paper when ran using our implementation.

4.3 Native Litmus Test Implementations

In this section, we discuss how to install each of the native languages that are required to run the native litmus tests.

4.3.1 TypeScript

For TypeScript, all the information regarding the language and the process of installment can be found at `https://www.typescriptlang.org/`.

4.3.2 Thorn

Unfortunately, there does not exist a public implementation of Thorn that is readily available. The Thorn skeleton, which we obtained through private means, was heavily savaged from bit rot and decay, and was not easily installable nor it contain all the necessary components, such as the thorn type checker.

4.3.3 Typed Racket

For Typed Racket, first you would need to install the Racket IDE called DrRacket, which can be found at https://docs.racket-lang.org/getting-started/index.html. In order to write a Typed Racket module within DrRacket you would be required to follow the three steps outlined at: `https://docs.racket-lang.org/ts-guide/quick.html`.

4.3.4 Reticulated Python

The information regarding the installment and running of Reticulated Python can be found at: `https://github.com/mvitousek/reticulated`. It requires Python version 3.5 or older.

5 License

The artifact is available under the Apache license.

6 MD5 sum of the artifact

5186a2242726e810f5acac714a827a6c

7 Size of the artifact

0.230 GiB

Acknowledgments. I want to thank Aviral Goal for helping to test this artifact.

References

1 Timothy Jones and David J. Pearce. A mechanical soundness proof for subtyping over recursive types. In *Proceedings of the 18th Workshop on Formal Techniques for Java-like Programs*, FTfJP'16, pages 1:1–1:6, New York, NY, USA, 2016. ACM. URL: `http://doi.acm.org/10.1145/2955811.2955812`, `doi:10.1145/2955811.2955812`.