Gallimaufry: An Automated Framework for Proving Type-Safety

Anne Mulhern [1]

Computer Sciences Department
University of Wisconsin-Madison
Madison, WI USA

Abstract

Gallimaufry is a novel language extension development framework with an integrated type-safety component. Its core component is a translator which translates programs written in a simple object-oriented language to semantically equivalent programs in a lambda calculus. A proof of the correctness of the translator is then constructed using an automated proof assistant, Coq. A user of Gallimaufry will experiment with a language feature by specifying the syntax and translation rules for the feature. Gallimaufry will then automatically generate a proof of the type-safety of the feature or will indicate where the proof fails.

Key words: type-safety, language design, translational semantics, object-oriented language, automated proof assistant, lambda-calculus, $\mathcal{SOOL}$, Coq

1 Introduction

For many years, type-safety has been considered an essential property of a language. Type-safety was an important goal for the designers of the Java language and is one of its primary selling points. Generally, Java programmers can expect to find many of their bugs revealed as type errors at compile time; in contrast, C programmers experience such bugs as inexplicable behavior or crashes at runtime. Java is also considered to be far more secure because of its type-safety. Even the initial version of Java was quite complicated, and no formal proof was given for its type-safety. Proofs of Java's type-safety were subsequently developed using a variety of techniques [?, ?, ?, ?, ?, ?, ?, ?], and Java is now generally considered to have been proven type-safe.

Around any reasonably popular language there will arise a large population of variants. These variants may arise to address limitations in the original language's expressiveness or to redress flaws in its design. Java is an outstanding example. There were many contenders

[1] Email: mulhern@cs.wisc.edu

for a version of generics for Java; a version based on GJ [?] has now been incorporated into Java 1.5 (Tiger) [?]. At the time that GJ was designed, it was not demonstrated to be type-safe. A proof of the type-safety of GJ followed only subsequently, and, in fact, work on the proof exposed at least one bug in the original design of GJ.

Gallimaufry [?] is a new kind of framework for experimenting with language extensions. The novel aspect of this framework is the support that it provides for proving type-safety of language extensions. The proof is based on a translational semantics developed in detail by Kim Bruce in "Foundations of Object-oriented Languages: Types and Semantics" [?]. The semantics defines a translation from a simple object-oriented language to a variant of the lambda calculus which is itself type-safe. A user can experiment with language extensions by defining additional syntax and translation rules and can rely on Gallimaufry to verify the type-safety of the modified language.

The core of Gallimaufry is a translator from $\mathcal{SOOL}$, the simple object-oriented language of [?] to a variant of the lambda calculus. The translator is written in O'Caml. We have chosen as a target for the translation a variant of the lambda calculus described by Pierce in [?] and for which an interpreter has been developed [?]. Thus Gallimaufry provides a "reality check"; a program written in the extended language can be translated to a lambda calculus and executed using an existing interpreter.

2 Contributions

Gallimaufry is a work in progress. However, we expect that this project will result in the following contributions.

- Most importantly, Gallimaufry is, to our knowledge, the only language extension development framework with an integrated type-safety component and as such is a significant step forward in the area of language design.
- We believe that the technique we are using to develop the proof of correctness of the translation in Gallimaufry is novel. The first step is the development of the translator. The translator is a working program translating programs written in an object-oriented language to semantically equivalent programs written in the lambda calculus. The translator is then itself used as model for definitions suitable for the Coq Proof Assistant [?]. These definitions are the structures used by Coq in the proof. In the first place, the generation direction is unusual [?]; first the translator is built and then the proof is derived from the source code of the translator. This is in the opposite direction to the operation of the Coq program extraction facility. Unlike Krakatoa [?] and Why [?], the proof is extracted directly from the program, rather than specifications of the program's behavior. Recollect that the goal of Gallimaufry is a proof of the type-safety of a language and that the translator translates programs written in that language. The construction of the translator is just a step on the way to the proof. In contrast, the purpose of Krakatoa and Why is to add confidence to an existing program. Having a working translator for a model gives us greater confidence in the Coq definitions that we derive from it. During development

of the translator we have detected a few errors in the syntax and typechecking rules of $\mathcal{SOOL}$ as defined in [?]; this emphasizes the importance of the translator development in constructing a correct proof.

- We believe that Gallimaufry is the first tool to automate a proof of type-safety using a *translational* semantics, rather than an operational semantics, a denotational semantics or an axiomatic semantics. Translation is one of the most familiar concepts in programming languages; compilers translate from a higher-level language to a lower-level language and optimizing compilers transform code while maintaining semantics. We chose a translational semantics for Gallimaufry because we believe that familiarity with the concept of translation will make its use more intuitive for the intended user who may not have a strong background in programming language semantics. Proving type safety of a language using a translational semantics requires developing a translation from the source language, the language for which type safety is to be proved, to the target language, the language for which type safety is proved, and demonstrating that the translation is correct, i.e., sound and well defined. The translation is sound if the translation of every well-typed source language expression is well-typed in the target language and if the translation of the type of the source expression is the type of the translated expression. The translation is well defined if there exists a unique translation for every expression. By implementing the $\mathcal{SOOL}$ translator, and by automating the proof of type-safety in Coq we hope to demonstrate the soundness of the translational semantics developed in [?]. And by demonstrating a technique for automating such a proof we will have made a significant contribution to the area of type theory.
- The ultimate purpose of Gallimaufry is to allow a language designer to experiment with language features. Gallimaufry will allow the user to introduce language features by describing the syntax and the translation. From these, Gallimaufry will construct the necessary additional Coq definitions and replay the proof using pre-existing tactics and hints as well as some that are suggested by the new language feature. It will then either complete the proof, so that the user is assured of the type safety of the new feature, or indicate that the proof fails and where it runs into trouble. Our tool is aimed at a user who has some familiarity with type theory and the basics of compiler design but who does not necessarily understand the underlying logic of the proof. A significant challenge will be to make this tool useful to such a user. Thus another contribution of our work will be the techniques we develop for extracting meaningful messages from the failure of the proof that will allow the user to correct the syntax or modify the translation of the language feature.

References

[1] Jim Alves-Foss, editor. *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*. Springer, 1999.

[2] Egon Börger and Wolfram Schulte. A programmer friendly modular definition of the semantics of Java. In *Formal Syntax and Semantics of Java*, pages 353–404. Springer-Verlag, 1999.

[3] Kim B. Bruce. *Foundations of Object-oriented Languages: Types and Semantics*. MIT Press,

2002.

[4] Robert Cartwright and John L. McCarthy. Recursive programs as functions in a first order theory. In *Proceedings of the International Conference on Mathematical Studies of Information Processing*, pages 576–629. Springer-Verlag, 1979.

[5] The Coq Proof Assistant. http://coq.inria.fr/.

[6] Sophia Drossopoulou and Susan Eisenbach. Java is type safe — probably. *Lecture Notes in Computer Science*, 1241, 1997.

[7] Sophia Drossopoulou, Susan Eisenbach, and Sarfraz Khurshid. Is the Java type system sound? *Theor. Pract. Object Syst.*, 5(1):3–24, 1999.

[8] GJ : A Generic Java Language Extension. http://www.cis.unisa.edu.au/˜ pizza/gj/.

[9] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.*, 23(3):396–450, 2001.

[10] J2SE 5.0. http://java.sun.com/j2se/1.5.0/.

[11] Krakatoa. http://krakatoa.lri.fr/.

[12] Anne Mulhern. Gallimaufry. http://www.cs.wisc.edu/˜ mulhern/gallimaufry.

[13] Tobias Nipkow and David von Oheimb. Javalight is type-safe—definitely. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 161–170. ACM Press, 1998.

[14] Benjamin C. Pierce. Types and programming languages. http://www.cis.upenn.edu/˜ bcpierce/tapl/.

[15] Benjamin C. Pierce. *Types and programming languages.* MIT Press, 2002.

[16] R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine—Definition, Verification, Validation.* Springer-Verlag, 2001.

[17] Don Syme. Proving Java type soundness. In *Formal Syntax and Semantics of Java*, pages 83–118. Springer-Verlag, 1999.

[18] Why: A software verification tool. http://why.lri.fr/.