Language-Based Security Final Project Report
FeatherEvaluator

Erick Bauman
exb131030@utdallas.edu
Tristan Duckworth
txd123130@utdallas.edu
Shamila Wickramasuriya
scw130030@utdallas.edu

December 15, 2016

1 Project Motivation

Popular languages today have many vulnerabilities, especially higher-level languages that intend to make life easier for developers and prevent them from making common mistakes. Java, which provides a safer environment to program in than C or C++, nevertheless has a large and complex runtime program with its own share of vulnerabilities. The high complexity of a modern language can benefit from formalization so that properties of both the core semantics and a runtime implementation can be proved to behave as intended. Unfortunately, Java is complex, and we are far from having a fully validated JRE.

However, it is possible to take a first step in this direction. A formal definition of the core syntax of Java exists in the form of Featherweight Java [2], which has allowed proofs to be made demonstrating certain properties of Java. This omits many of the features of Java, such as assignment, base types, and access control. However, this simplification makes proofs much more feasible.

Several Coq implementations of Featherweight Java have been written, allowing for proofs to be machine-checked [1, 3, 4]. Unfortunately, the implementations available only provide definitions of evaluating Featherweight Java expressions in a propositional form. While practical for use in proofs, these implementations do not provide a functional implementation, which would be useful for eventually proving the correctness of an actual Java runtime. While eventually it would be practical to prove the correct behavior of a functional implementation that evaluates Java bytecode instead of source code, we wanted to focus on demonstrating and proving the soundness of a functional implementation of the Featherweight Java semantics in Coq.

2 Accomplishments

We implemented a function to evaluate the small-step semantics of Featherweight Java, based on a cast-free implementation of Featherweight Java [4].

We wrote part of a proof of soundness of our function, which stated that for any expression, if our implementation produced a resulting expression, then the propositional evaluation would hold:

```
forall (e1 e2:fexp) (fct:fctable),
feval e1 fct = Some e2 -> eval (fexp2exp e1) (fexp2exp e2).
```

3 Project Summary

3.1 Overview

Since we could start with an existing Featherweight Java implementation, the process of proving that the functional implementation was "correct" meant proving that for any expression, if our function produces a resulting expression, then the original propositional implementation holds for that expression. Since the propositional implementation has already been proved correct by the original authors, this is sufficient to prove soundness (but not completeness) of the function.

We discovered that the original representation of expressions was not well-suited to computation. In particular, the congruence rules that specify how expressions used as arguments are reduced allow any argument to be reduced at a given time. Since the original implementation of expressions, `exp`, made use of lists to represent collections of arguments, we were unable to write proofs when they depended on the reduction of these arguments. In light of this, we implemented a new representation, `fexp`, that can be readily translated to `exp` (for the sake of writing proofs with respect to the propositional definitions of evaluation) and does not represent arguments as a list (allowing us to reason about the computations we perform).

Our work currently consists of three major source files:

- `FEV_Definitions.v` contains the definition of `fexp` and `fexp2exp`, redefines some of the original definitions (such as that for class tables) in terms of `fexp` rather than `exp`, adds some supporting functions for evaluation, and finally contains `feval` itself. As a bonus, it contains a function `teval` that evaluates any `fexp` with `feval` until no more progress can be made.
- `FEV_Properties.v` contains auxiliary lemmas for reasoning about lists and our supporting functions as well as the (incomplete) proof of soundness.
- `FEV_Example.v` contains an example of a computation being performed on a simple `fexp` as well as a great number of old examples meant for a version of our evaluator that targeted `exp` instead.

3.2 Difficulties

4 Team Coordination

Task	**Person**
Slide Creation	Tristan, Shamila
Presentation	Tristan
Coding	Erick, Tristan, Shamila
Proofs	Tristan, Shamila
Report—Project Motivation	Erick
Report—Accomplishments	Erick
Report—Project Summary	Tristan
Report—Future Work	Shamila

5 Future Work

We hope to prove that our evaluator only produces states that is valid under the propositional definition of single-step evaluation for Featherweight Java. Our proof is yet to be completed and proving the in reverse direction for total correctness is another possible future work.

Further, we believe that our work can be extended to prove the correctness of Java byte code instead of the source code. The result will be useful in proving the correctness of Java Runtime Environment including the safety properties.

References

[1] Benjamin Delaware, William Cook, and Don Batory. Product lines of theorems. In *ACM SIGPLAN Notices*, volume 46, pages 595–608. ACM, 2011.

[2] Atsushi Igarashi, Benjamin C Pierce, and Philip Wadler. Featherweight java: a minimal core calculus for java and gj. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(3):396–450, 2001.

[3] Julian Mackay, Hannes Mehnert, Alex Potanin, Lindsay Groves, and Nicholas Cameron. Encoding featherweight java with assignment and immutability using the coq proof assistant. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, pages 11–19. ACM, 2012.

[4] Bruno De Fraine with help from Erik Ernst and Mario Sudholt. Cast-free featherweight java, 2008.