LiquidJava: Adding Lightweight Verification to Java

Catarina Gamboa[1], Paulo Alexandre Santos[1], Christopher S. Timperley[2], and Alcides Fonseca[1]

[1] LASIGE, Faculdade de Ciências da Universidade de Lisboa, Portugal
{cvgamboa,pacsantos,alcides}@ciencias.ulisboa.pt
[2] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
ctimperley@cmu.edu

1 Introduction

One of the major concerns in software development is the assurance of the quality of the produced products [1]. Developers want to detect potential bugs and vulnerabilities as soon as possible, and validation techniques that integrate with editors have been preferred and widely adopted. Type systems allow developers to specify the expected type of operations and verify, at compile time, if those types are respected. This behaviour is commonly integrated within editors to present localized and informative error messages. Refinement types are more expressive than the simple types in Java and similar languages [2], as their primary idea is to extend simple types with predicates that restrict the allowed values in variables or methods. Listing 1.1 shows an example of a method with the parameter and return types extended with refinements.

```
@Refinement("_ >= a && _ <= b")
public static int inRange(int a, @Refinement("b > a") int b){
    return a + 1;
}
...
inRange(10, 20) //correct
inRange(10, 2);//correct in Java, refinement type error
```

Listing 1.1: Methods Verification

Refinement Types can be used to detect simple out-of-bounds array access bugs [3], but also security issues [4] and protocol violations [5]. While such bugs are frequently encountered in Java, refinement types have not yet been added to Java's type system. In fact, refinement types have been introduced in ML [6], Haskell [7], C [8] and Typescript [9], but without widespread adoption by professional developers. One of the possible explanations is that the earliest languages with refinement type support were not widely popular in the industry. Another explanation could be a lack of experience of programmers in writing specifications.

In this work, we set out to explore how to promote the wide usage of refinement types in general and in a popular mainstream language like Java in particular.

2 LiquidJava

We determined the requirements of LiquidJava based on popular characteristics of successful static verification techniques (e.g., the NonNull annotation [10]) and developers' feedback [11]. These requirements guided the system design by restricting the refinements to Liquid Types [12] to ensure a decidable logic, and including the refinements in the code as optional parametric annotations with the refinements language as a string. Furthermore, the design of our refinements language was driven by an online survey with 50 participants that selected the preferable syntaxes for the different language features. Finally, to avoid an early deprecation of LiquidJava due to Java's fast release cycle, we implemented the refinement type checker on top of Spoon[13], which uses the Eclipse Compiler Kit, adopting Java features as they are released.

LiquidJava supports the refinement of variables, fields and methods (both parameters and return types). Moreover, LiquidJava extends classic refinements with specific features for Java, such as using ghost functions and states to model objects, and allowing legal state transitions to be defined. Finally, we implemented a prototype typechecker of LiquidJava and integrated it as an IDE plugin to improve the error reporting and usability of the framework.

3 Results

To evaluate LiquidJava, we conducted a research study where 30 participants were given tasks designed to assess the understandability and usability of our framework. The study showed that refinements are easy to understand and to introduce in the code after a small introduction to the framework and with access to multiple examples that can be seen on our project website[14]. When using LiquidJava, the participants were able to find and fix implementation bugs that were not fixed using plain Java and, in some cases, were able to do so faster. Overall, the participants showed interest in the framework and are open to using LiquidJava in their projects, which gives us confidence that participants find this tool accessible for its gains.

Acknowledgments

This work was supported by the Fundação para a Ciência e a Tecnologia (FCT) under LASIGE Research Unit, ref. (UIDB/00408/2020) and (UIDP/00408/2020) and the CMU—Portugal project CAMELOT (POCI-01-0247-FEDER-045915).

References

1. P., U.: The role of verification and validation in system developmentlife cycle. IOSR Journal of Computer Engineering (IOSRJCE) **5**(1) (2012) 17–20
2. Jhala, R., Vazou, N.: Refinement types: A tutorial. CoRR **abs/2010.07763** (2020)
3. Xi, H., Pfenning, F.: Eliminating array bound checking through dependent types. In: Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI), ACM (1998) 249–257
4. Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A.D., Maffeis, S.: Refinement types for secure implementations. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, IEEE Computer Society (2008) 17–32
5. Burnay, N., Lopes, A., Vasconcelos, V.T.: Statically checking REST API consumers. In de Boer, F.S., Cerone, A., eds.: Software Engineering and Formal Methods - 18th International Conference. Volume 12310. (2020) 265–283
6. Freeman, T.S., Pfenning, F.: Refinement types for ML. In Wise, D.S., ed.: Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), ACM (1991) 268–277
7. Vazou, N., Seidel, E.L., Jhala, R., Vytiniotis, D., Peyton-Jones, S.: Refinement types for haskell. In: ACM SIGPLAN Notices. Volume 49., ACM (2014) 269–282
8. Rondon P., Bakst A., Kawaguchi M., Jhala R.,: CSolve: Verifying C With Liquid Types. `http://goto.ucsd.edu/~rjhala/papers/csolve_verifying_c_with_liquid_types.pdf`
9. Vekris, P., Cosman, B., Jhala, R.: Refinement types for typescript. In Krintz, C., Berger, E., eds.: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, ACM (2016) 310–325
10. Fähndrich, M., Leino, K.R.M.: Declaring and checking non-null types in an object-oriented language. In Crocker, R., Jr., G.L.S., eds.: Proceedings of the 2003 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2003, ACM (2003) 302–312
11. Sadowski, C., Aftandilian, E., Eagle, A., Miller-Cushon, L., Jaspan, C.: Lessons from building static analysis tools at google. Commun. ACM **61**(4) (2018) 58–66
12. Rondon, P.M., Kawaguchi, M., Jhala, R.: Liquid types. In Gupta, R., Amarasinghe, S.P., eds.: Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, ACM (2008) 159–169
13. Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., Seinturier, L.: SPOON: A library for implementing analyses and transformations of java source code. Softw. Pract. Exp. **46**(9) (2016) 1155–1179
14. Gamboa, C.: Liquidjava - project website. `https://catarinagamboa.github.io/liquidjava.html` (2021)