CS842 Type Systems: The Cost of Safety

Jia Wu

Tuesday 25th July, 2017

1 Introduction

Type systems are a tradeoff between security and convenience. Dynamically typed systems are flexible enough to allow rapid implementation and deployment. However, these systems typically have some overhead for runtime checks versus their statically typed counterparts. This paper aims to investigate the performance overhead incurred in a few impartive programming languages via simple benchmarks. Additionally, the faults and merits of gradually typed languages with respect to performance will also be examined.

2 Related Work

Benchmarking the performance of gradually typed languages is not a novel idea. Groups such as Takikawa et al. [TFG+16] have built a framework for Typed Racket which examines the performance penalties of incrementally typing modules over multiple configurations. These configurations provide a view of how developers may gradually type their programs at a macro-level. This work suggests that incrementally typing a program has significant penalties incurring slowdowns of up to 105x relative to the untyped program. Takikawa et al. discovered that by fully typing all modules within a program, the resulting program has better performance than its untyped equivalent.

TypeScript is an unsound extension of Javascript. Groups such as Rastogi et al. [RSF+15] have developed extensions to compile safe TypeScript code. By conducting a two-phase compilation process which introduces some runtime type information, the authors are able to enforce type safety on a subset of TypeScript code. Again, the property of type safety comes at a cost of a slowdown factor of approximately 1.15x when porting a TypeScript Compiler.

3 Method

3.1 Environment

Benchmarks for the following experiments were executed in an Ubuntu Docker container running ontop of an Intel i7 4790k at 4.4GHz with 16GB RAM. Docker was selected as the

execution environment as it provides a consistent runtime environment regardless of deployment [Mer14]. The builtin linux time functionality was chosen to measure the elapsed real time betwen invocation and termination of each benchmark. All relevant code can be found at: `https://github.com/JRWu/spring2017_cs842typesystems_final`. Additionally, the docker image containing the execution environment can be sourced by using *docker pull jwu424/cs842*.

3.2 Languages

Three dynamically typed languages and their gradually typed counterparts were chosen for the benchmark purposes: PHP/HACK, Python/Cython, Javascript/Typescript.

Hack [AEM+14]

3.3 Benchmarks

3.3.1 Arithmetic

A set of arithmetic operations were implemented as functions.

3.3.2 Strings

3.3.3 IO

3.3.4 Recursion

4 Results

The performance numbers represent an average of 10 executions.

5 Discussion

5.1 Threats to Validity

5.2 Future Work

6 Conclusion

References

[AEM+14] Keith Adams, Jason Evans, Bertrand Maher, Guilherme Ottoni, Andrew Paroski, Brett Simmers, Edwin Smith, and Owen Yamauchi. The hiphop virtual machine. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 777–790, New York, NY, USA, 2014. ACM.

[Mer14] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[RSF+15] Aseem Rastogi, Nikhil Swamy, Cédric Fournet, Gavin Bierman, and Panagiotis Vekris. Safe & efficient gradual typing for typescript. *SIGPLAN Not.*, 50(1):167–180, January 2015.

[TFG+16] Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, and Matthias Felleisen. Is sound gradual typing dead? In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 456–468, New York, NY, USA, 2016. ACM.