Semantic Analysis of Advanced Programming Languages

Harley Eades III
Computer Science
The University of Iowa

1 Introduction

There are two major problems growing in two areas. The first is in Computer Science, in particular software engineering. Software is becoming more and more complex, and hence more susceptible to software defects. Software bugs have two critical repercussions: they cost companies lots of money and time to fix, and they have the potential to cause harm.

The National Institute of Standards and Technology estimated that software errors cost the United State's economy approximately sixty billion dollars annually, while the Federal Bureau of Investigations estimated in a 2005 report that software bugs cost U.S. companies approximately sixty-seven billion a year [5, 8].

Software bugs have the potential to cause harm. In 2010 there were a approximately a hundred reports made to the National Highway Traffic Safety Administration of potential problems with the braking system of the 2010 Toyota Prius [1]. The problem was that the anti-lock braking system would experience a "short delay" when the brakes where pressed by the driver of the vehicle [7]. This actually caused some crashes. Toyota found that this short delay was the result of a software bug, and was able to repair the the vehicles using a software update [6]. Another incident where substantial harm was caused was in 2002 where two planes collided over Überlingen in Germany. A cargo plane operated by DHL collided with a passenger flight holding fifty-one passengers. Air-traffic control did not notice the intersecting traffic until less than a minute before the collision occurred. Furthermore, the on-board collision detection system did not alert the pilots until seconds before the collision. It was officially ruled by the German Federal Bureau of Aircraft Accidents Investigation that the on-board collision detection was indeed faulty [3].

The second major problem affects all of science. Scientific publications are riddled with errors. A portion of these errors are mathematical. In 2012 Casey Klein et al. used specialized computer software to verify the correctness of nine papers published in the proceedings of the International Conference on Functional Programming (ICFP). Two of the papers where used as a control which where known to have been formally verified before. In their paper [2] they show that all nine papers contained mathematical errors. This is disconcerting especially since most researchers trust published work and base their own work off of these papers. Kline's work shows that trusting published work might result in wasted time for the researchers basing their work off of these error prone publications. Faulty research hinders scientific progress.

Both problems outlined above have been the focus of a large body of research over the course of the last forty years. These challenges have yet to be completed successfully. The work I present here makes up the foundations of one side of the programs leading the initiative to build theory and tools which can be used to verify the correctness of software and mathematics. This program is called program verification using dependent type theories. The second program is automated theorem proving. In this program researchers build tools called model checkers and satisfiability modulo-theories solvers. These tools can be used to model and prove properties of large complex systems carrying out proofs of the satisfiability of certain constraints on the system nearly automatically, and in some cases fully automatically. As an example André Platzer and Edmund Clarke in 2009 used automated theorem proving to verify the correctness of the in flight collision detection systems used in airplanes. They actually found that there were cases where two plans could collide, and gave a way to fix the problem resulting in a fully verified algorithm for collision detection. That is he mathematically proved that there is no possible way for two plans to collide if the systems are

operational [4]. Automated theorem provers, however, are tools used to verify the correctness of software externally to the programming language and compiler one uses to write the software. In contrast with verification using dependent types we wish to include the ability to verify software within the programming language being used to write the software. Both programs have their merits and are very fruitful and interesting.

This report summarizes my dissertation by chapter. Each section will be given the name of a chapter, and then the contents of the section will consist of a summery of that chapter. I make sure to include my already published work as well as on going work that needs to be done before my defense.

2 Chapter 0: History and Background

3 Chapter I: Design

3.1 Freedom of Speech

3.2 Separation of Proof from Program (Sep^3)

3.3 Dualized Type Theory (DTT)

4 Chapter II: Analysis

4.1 Basic Analysis

4.1.1 Free Speech

4.1.2 Dualized Type Theory

4.2 Normalization by Hereditary Substitution

4.2.1 Stratified System F (SSF) and its Extensions

4.2.2 The $\lambda\Delta$-Calculus

4.3 Categorical Semantics

4.3.1 Semi-Bilinear Logic

4.3.2 Split Bi-Intuitionistic Logic

4.3.3 Dualized Type Theory

4.3.4 Nested Bi-Intuitionistic Logic

5 Conclusion

References

[1] blogs.consumerreports.org. Consumer reports cars blog: Japan investigates reports of prius brake problem, 2010.

[2] C. Klein, J. Clements, C. Dimoulas, C. Eastlund, M. Felleisen, M. Flatt, J. McCarthy, J. Rafkind, S. Tobin-Hochstadt, and R. Findler. Run your research: on the effectiveness of lightweight mechanization. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 285–296, New York, NY, USA, 2012. ACM.

[3] The German Federal Bureau of Aircraft Accidents. Investigation report, 2004.

[4] A. Platzer and C. Edmund. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009.

[5] Research Triangle Institute. The Economic Impacts of Inadequate Infrastructure for Software Testing, 2002. Sponsored by the Department of Commerce's National Institute of Standards and Technology.

[6] Reuters. Toyota to recall 436,00 hybrids globally-document, February 2010.

[7] thedetroitbureau.com. Nhtsa memo on regenerative braking, April 2011.

[8] United States Federal Bureau of Investigation. 2005 FBI Computer Crime Survey.