REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 26, 2007	Final Report Aug 31, 2001 to Dec 1, 2006

4. TITLE AND SUBTITLE
PECASE: Next Generation Systems Languages

5. FUNDING NUMBERS
F49620-01-1-0298

6. AUTHOR(S)
Morrisett, Greg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cornell University
Ithaca, New York 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
40078

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR
Suite 325, Room 3112
875 Randolph Street
Arlington, VA 22203-1768
Dr Robert Herklotz/NM

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-SR-AR-TR-07-0128

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; distribution is Unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT ***(Maximum 200 Words)***

The goal of this work is to explore techniques for making today's software, which is largely written in type-*unsafe*, low-level languages such as C, as reliable and trustworthy as code written in type-*safe*, high-level languages such as Java or ML. Type-safe languages automatically block or prevent common vulnerabilities such as buffer overruns, format string attacks, and overflow attacks which are all too common in today's critical software infrastructure. To this end, we have implemented a prototype compiler called *Cyclone*, which provides the benefits of type safety through a combination of static analysis, programmer annotations, and run-time checks. Particular emphasis has been placed on scalable, static analyses to ensure that programmers can retain good performance and high reliability.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Type-safety. Legacy code. Buffer overrun.	14
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Next Generation Systems Languages

AFOSR PECASE Grant F49620-01-1-0298

Final Report

31 August 2001 – 1 December 2006

Greg Morrisett
Computer Science Department
Harvard University
Cambridge, Massachusetts

(617) 495-9526 (phone)
greg@eecs.harvard.edu

Objectives

High-level programming languages, such as Java and C#, can increase the reliability of systems and drastically cut development costs. One reason is that the *type checkers* of these languages automatically identify common programming errors at compile time. Another reason is that the *type-safety* guarantee of such languages ensures that all primitive operations have a well-defined semantics, thus bounding the space of possible behaviors of a program with coding errors in it.

However, the operating systems, device drivers, network software, servers, and databases that power our computational infrastructure are all written in low-level, unsafe languages such as C or C++. Even the run-time systems, garbage collectors, JIT-compilers, and much of the libraries for Java and C# are coded in C. Consequently, these systems suffer from a variety of preventable flaws that can cost time, security, money, and even lives. As one example, over 50% of the successful attacks reported by the Computer Emergency Response Team (CERT) involve buffer overruns—an attack that

leverages the lack of type-safe arrays in C and C++. The Code Red and Blaster worms are examples of malicious code that used buffer overruns to cause widespread damage.

One reason for this state of affairs is that today's software systems started out in unsafe, low-level languages and have been evolved over many years. At this point, it is too expensive to throw out the existing code and replace it with code written in a high-level language. For example, Windows Vista contains over 60 million lines of code, much of which was inherited from Windows XP, which was in turn evolved from Windows NT. Microsoft simply cannot afford to pay the development costs needed to re-code an operating system that has been evolved over fifteen years.

Even if Microsoft could afford to re-code Vista in a type-safe language, there are substantial technical issues that prevent the adoption of today's high-level language technology for building production systems. In particular, the very mechanisms used to enhance safety prevent programmers from:

1. achieving high performance and low resource consumption,
2. interoperating with legacy systems,
3. directly accessing and controlling memory and devices, and
4. reasoning about the space or time requirements of programs.

Thus, even if we could start from scratch, we would have no effective, safe environment for re-coding our infrastructure.

Therefore, the goal of this project was to explore the design, implementation, and foundations of *safe, low-level programming languages.* The primary aim was to gain a deeper understanding of how programmers can gain control over the performance and interoperability of systems code without sacrificing type safety. In addition, the project aimed to introduce new language-level invariants and enforcement tools that go well beyond simple types and that can be used to build systems that are truly robust.

To evaluate the effectiveness of proposed mechanisms, the project built a prototype, next-generation systems language called Cyclone. Cyclone is based upon ANSI C so that the language can be used and evaluated in real systems contexts. In particular, the project ported and evaluated a number of medium sized programs from C to Cyclone. Basing the language upon C also ensured that systems programmers would be comfortable using the

language, and that interoperability with legacy C (and C++) code would be relatively simple.

Unlike C, Cyclone comes equipped with an advanced type and assertion system that (a) allows programmers to state and enforce crucial program invariants, (b) lets tools automatically check for safety and security properties, (c) gives the compiler sufficient information to produce high-performance code. All of the sources and documentation for Cyclone have been made freely available to the public (with an open-source license) and can be found on the Web at `http://cyclone.thelanguage.org`.

Status of Effort

Normal C programs can crash in any number of ways due to at least the following:

- unsafe casts
- unsafe pointer arithmetic
- dereferencing a pointer to deallocates storage
- failing to deallocate storage
- failure to allocate large enough buffers
- failure to test for NULL pointers
- unsafe unions

Cyclone prevents all of these problems (and more) through a combination of static and dynamic checks [7]. For instance, all pointer arithmetic in Cyclone is checked to ensure that pointer dereferences lie within the boundaries of a given object.

The Cyclone type checker rules out many simple errors, such as unsafe casts, at compile-time. The key challenge was ensuring that the the type-checker does not reject common idioms that are safe, but use potentially unsafe primitives. To this end, Cyclone supports various forms of *polymorphism* including parametric polymorphism, existential polymorphism, and physical

subtype polymorphism. Many subtelties arise in the interaction of polymorphism with the imperative features in C (e.g., the address-of-operator) [1], and thus great care had to be taken to ensure soundness.

The Cyclone type checker also rules out many simple memory management errors, such as dereferencing a pointer to a stack-allocated object which has been deallocated. To do so, Cyclone employs a sophisticated region-based, type-and-effect system [6] as well as a form of linear (or tracked) pointers [19,24]. In essence, the type checker conservatively tracks the set of locations that have not been deallocated at each program point. Programmers must provide annotations on procedure boundaries and data structures to give the type-checker enough information to be effective. However, in practice, the annotations are quite small and can be inferred with an external tool [19].

Some problems, such as an array index out of bounds, are prevented by run-time checks inserted by the compiler. To support these checks, the Cyclone compiler must also insert extra meta data on key objects so that the bounds can be determined dynamically. Constructing and maintaining the meta data is a serious source of ineffeciency and thus the Cyclone compiler performs a number of analyses to minimize the overheads.

Finally, the Cyclone type checker provides limited support for extended static checking, wherein programmers can specify pre- and post-conditions on procedures, and static assertions on loops. The compiler uses these pre- and post-conditions to refine its analysis so that most checks can be safely eliminated. For example, when bootstrapping the compiler (which is itself written in Cyclone), over 92% of the NULL-pointer and array-bounds checks are automatically eliminated.

The project ported both small benchmarks as well as complete applications to Cyclone to determine the overheads introduced by the compiler [20]. On small benchmarks, the Cyclone code is on average about 60% slower than unsafe C code. In contrast, Java is about 650% (over an order of magnitude) slower than C code. For more realistic benchmarks, the overheads of Cyclone do not seem to be a problem. For example, when ported from C to Cyclone, the Boa web server came within 2% of the C performance on throughput tests.

Accomplishments

The key technical accomplishments of this project are as follows:

- Adapted parametric and subtype polymorphism to low-level C code; constructed models and proofs of soundness for the resulting type system. In so doing, the project uncovered a number of subtle, previously unrecognized issues when parametric polymorphism is combined with pointer arithmetic.

- Designed, modeled, proved correct, and implemented a region-based, type-safe memory management system. This part of the type system ensures that code respects the lifetimes of data that are allocated on the stack or in lexically-scoped regions. In practice, this part of the type system was crucial for avoiding the overheads of heap-allocation when porting code from C to Cyclone and accounts for a relatively large part of the performance win of Cyclone over existing safe languages.

- Designed, modeled, proved correct, and implemented extensions to the region-based type system to support linear (a.k.a. tracked) pointers. The linear pointers in Cyclone allow programmers to have a much finer degree of control over the lifetimes of heap-allocated data. They can be used to safely implement a wide range of memory management strategies, including standard malloc/free, arenas, and reference counting. With the addition of these facilities, Cyclone became the first type-safe language that could be used to build a garbage collector *within* the language, and without itself needing a meta-level garbage collector [14].

- Designed, modeled, proved correct, and implemented a static extended checker for Cyclone that ensures certain run-time checks, which are needed to ensure type safety, can be safely eliminated at compile time. The extended static checker runs as fast as a conventional type-checker, and yet statically verifies a range of properties that are outside the scope of traditional type-checkers including array bounds checks and NULL-pointer checks.

- The project produced four Ph.D.'s (J.Cheney, D.Grossman, M.Fluet, and S.Weirich.) Of these, three now have top faculty positions at leading univerities (Grossman is at Univ. of Washington, Fluet at Toyota

Technical Institute, and Weirich at Univ. of Pennsylvania) and one is a post doctoral candidate (Cheney at Univ. of Edinburgh.) In addition, the project supported two post doctoral candidates who now have faculty positions (M.Hicks at Univ. of Maryland, and A.Ahmed at Toyota Technical Institute.) Finally, the project supported two undergraduate students who are now in top Computer Science graduate programs (Francis Spaulding at Princeton, and Daniel Lee at Carnegie Mellon.)

Personnel Supported

Faculty: Greg Morrisett

Post Doctoral Students: Michael Hicks, Amal Ahmed

Graduate Students: James Cheney, Daniel Grossman, Matthew Fluet, Yanling Wang, Stephanie Weirich.

Undergraduate Students: Frances Spaulding, Daniel Lee.

Publications:

Copies of the publications can be found at the PI's web site: `http://www.eecs.harvard.edu/~greg`.

1. G. Morrisett, K. Crary, N. Glew, and D. Walker. Stack-based typed assembly language. *Journal of Functional Programming 12*, No. 1 (January 2002), University Press, Cambridge, England, 43–88.

2. Frederick Smith. *Certified Run-Time Code Generation.* Ph.D. Thesis, Cornell University, January 2002.

3. Dan Grossman. Existential Types for Imperative Languages. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 21–35.

4. Stephanie Weirich. Higher-order intensional type analysis. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 98–114.

5. G. Morrisett. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 1–5.

6. D. Grossman, G. Morrisett, T. Jim, M. Hicks, J. Cheney, and Y. Wang. Region-based memory management in Cyclone. *ACM Conference on Programming Language Design and Implementation* (Berlin, Germany, June 2002), 282–293.

7. T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. *Usenix Annual Technical Conference* (Monterey, CA, June 2002), 275–288.

8. Stephanie Weirich. *Programming With Types.* Ph.D. Thesis, Cornell University, July 2002.

9. Karl Crary, Stephanie Weirich, and Greg Morrisett. Intensional polymorphism in type erasure semantics. *Journal of Functional Programming*, 12(6):567-600, November 2002.

10. Daniel J. Grossman. Type-Safe Multithreading in Cyclone. *ACM Workshop on Types in Language Design and Implementation* (New Orleans, LA, January 2003).

11. Frederick Smith, Dan Grossman, Greg Morrisett, Luke Hornof, and Trevor Jim. Compiling for template-based run-time code generation. *Journal of Functional Programming*, 13(3):677-708, May 2003.

12. James Cheney and Christian Urban. System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. *Workshop on Unification* (Valencia, Spain, May, 2003).

13. Daniel J. Grossman. *Safe Programming at the C Level of Abstraction.* Ph.D. Thesis, Cornell University, August 2003.

14. Matthew Fluet and Daniel Wang. Implementation and Performance Evaluation of a Safe Runtime System in Cyclone. *Proceedings of the SPACE 2004 Workshop*, (Venice, Italy, January 2004).

15. James Cheney. The Complexity of Equivariant Unification. *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, (Turku, Finland, July 2004).

16. M. J. Gabbay and J. Cheney. A Proof Theory for Nominal Logic. *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, (Turku, Finland, July 2004), 139–148.

17. James Cheney. *Nominal Logic Programming.* Ph.D. Thesis, Cornell University, August 2004.

18. Matthew Fluet and Greg Morrisett. Monadic Regions. In *Proceedings of the ACM International Conference on Functional Programming* (ICFP'04), (Park City, Utah, September 2004), 103–114.

19. Michael Hicks, Greg Morrisett, Dan Grossman, and Trevor Jim. Experience with Safe Manual Memory-Management in Cyclone. In *Proceedings of the ACM International Symposium on Memory Management*, (ISMM'04), (Vancouver, British Columbia, October 2004), 73–84.

20. Dan Grossman, Michael Hicks, Trevor Jim, and Greg Morrisett. Cyclone: A Type-Safe Dialect of C. In *C/C++ User's Journal*, 23(1):6–13, January 2005.

21. Greg Morrisett, Matthew Fluet and Amal Ahmed. L^3: A Linear Language with Locations. *Seventh International Conference on Typed Lambda Calculi and Applications* (TLCA'05), (Nara, Japan, April 2005), 293–307.

22. Amal Ahmed, Matthew Fluet, and Greg Morrisett. A step-indexed model of substructural state. In *Proceedings of the ACM International Conference on Functional Programming* (ICFP'05), (Tallinn, Estonia, September 2005), 78–91.

23. Kevin Hamlen, Greg Morrisett and Fred B. Schneider. Computability Classes for Enforcement Mechansisms. *ACM Transactions on Programming Languages and Systems*, 28(1):175-205, January 2006.

24. Matthew Fluet, Greg Morrisett, and Amal Ahmed. Linear Regions Are All You Need. *European Symposium on Programming* (ESOP'06), (Vienna, Austria, March 2006).

25. Kevin Hamlen, Greg Morrisett, and Fred B. Schneider. Certified In-Lined Reference Monitoring for .NET. *ACM SIGPLAN Workshop on*

Programming Languages and Security, (PLAS), (Ottawa, Canada, June 2006).

26. Matthew Fluet and Greg Morrisett. Monadic regions. *Journal of Functional Programming*, 16(4-5):485-545, July 2006.

27. Stephen McCamant and Greg Morrisett. Evaluating SFI for a CISC Architecture. *15th Usenix Security Symposium*, (Vancouver, BC, August 2006).

28. Matthew Fluet. *Monadic and Substructural Type Systems for Region-Based Memory Management.* Ph.D. Thesis, Cornell University, January 2007.

Interactions/Transitions

- Trevor Jim of AT&T research worked with us to develop the Cyclone language, compiler, and tools. In addition, researchers at the University of Maryland, the University of Utah, Princeton, the University of Washington, and the University of Pennsylvania, and Cornell are all using Cyclone to develop research prototypes. Papers describing systems built with Cyclone have started to appear at major conferences, including IEEE OpenARCH, International Working Conference on Active Networks, and SOSP.

- Morrisett serves on the Microsoft Trustworthy Computing Academic Advisory Board. This group, which meets twice annually in Seattle, advises Microsoft regarding a range of security and privacy issues.

- Morrisett serves on the DARPA ISAT Advisory Board.

- Morrisett serves on the Fortify Technical Advisory Board. Fortify has adapted some of the ideas behind Cyclone's analysis in their product, Fortify Source Code Analysis.

- Morrisett serves on the editorial boards for: *Journal of Functional Programming* (chief editor), *Information Processing Letters*, *ACM Transactions on Programming Languages and Systems.* He also serves on

the Advisory Committee for the Semantics, Applications, and Implementations of Program Generation (SAIG) conference, and the ACM Conference on Types in Language Design and Implementation.

- Greg Morrisett spent nine months visiting Microsoft's Cambridge Research Laboratory, where he worked with researchers on programming language and security technology. In particular, Morrisett worked on the development of Microsoft's tools for automatically finding security flaws in production code, based on his experience with Cyclone. He also worked with student Kevin Hamlen and Microsoft researchers on the implementation of the .NET rewriting tool for inline reference monitors.

- Morrisett has also worked with researchers Martin Abadi and Ulfar Erlingsson at Microsoft's Silicon Valley Research Lab.

Invited Lectures: G. Morrisett

1. Explicit Regions in Cyclone. Invited lecture, New England Programming Languages Seminar. Boston, Massachussetts, October 2001.

2. Typed Assembly Language Background. Invited lecture, Intel Research Professor Forum, Santa Clara, California, Januray 2002.

3. Type Checking Systems Code. Invited lecture, Yale University, New Haven, Connecticut, February 2002.

4. Runtime Code Generation. IFIP Working Group 2.8 on Functional Programming, Las Vegas, Nevada, March 2002.

5. Type Checking Systems Code. Invited lecture, European Symosium on Programming, Grenoble, France, April 2002.

6. Type Checking Systems Code. Invited lecture, Cigital, Inc. Washington, D.C. April 2002.

7. Memory Management in Cyclone. Microsoft Research, Ltd. Cambridge, England. October 2002.

8. Analysis Issues for Cyclone. Keynote speaker, Conference on Program Analysis for Software Tools and Engineering. Charleston, South Carolina. October 2002.

9. Linearly Typed Assembly Language. IFIP Working Group 2.8 on Functional Programming. Crans-Montana, Switzerland. January 2003.

10. Cyclone: A Type-Safe Dialect of C. University of Kent. Canterbury, England. February 2003.

11. Cyclone: A Type-Safe Dialect of C. University of Birmingham. Birmhingham, England. February 2003.

12. Cyclone: A Type-Safe Dialect of C. University of London. London, England. March 2003.

13. Cyclone Memory Management. University of Edinburgh. Edinburgh, Scotland. April 2003.

14. An Introduction to Typed Assembly Language. Invited lecture, University of Cambridge. Cambridge, England. May 2003.

15. Cyclone Memory Management. Keynote speaker, UK Memory Management Workshop. University of Kent. Canturbury, England. May 2003.

16. Tutorial on Language-Based Security. ACM Conference on Programming Language Design and Implementation. San Diego, California. June 2003.

17. Regions and Beyond in Cyclone. Yale University. New Haven, Connecticut. June 2003.

18. Type-Safe Memory Management in Cyclone. Air Force Research Laboratory. Rome, New York. August 2003.

19. Security and Programming Languages. Harvard Industrial Partners. Cambridge, MA. October 2004.

20. What's Next for an Academic PL Researcher? University of Pennsylvania. Philadelphia, PA. November 2004.

21. Language Based Security. 10th European Winter School on Theoretical Computer Science. Tallinn, Estonia. February 2005.

22. A Type-Safe Dialect of C. NSA Workshop on High Confidence Computing. Baltimore, MD. March 2005.

23. Simplifying Regions. Carnegie Mellon University. Pittsburgh, PA. March 2005.

24. Open Problems for Certifying Compilation. Usenix Security Symposium. Baltimore, MD. August 2005.

25. The Next ML? ML Workshop. Tallinn, Estonia. September 2005.

26. Simplifying Regions. WG 2.8 Meeting. Tallinn, Estonia. September 2005.

27. Making C Type-Safe. Purdue University, West Laffayette, ID. October 2005.

28. Static Extended Checking for Cyclone. Verification, Model Checking, and Abstract Interpretation. Charleston, SC. January 2006.

29. Static Extended Checking for Cyclone. ITU University, Copenhagen, Denmark, May 2006.

30. Static Extended Checking for Cyclone. IBM Research, Westchester County, NY. September 2006.

31. Static Extended Checking for Cyclone. Intel Research Lab, Berkeley, CA. October 2006.

Honors and Awards

G. Morrisett:

- Sloan Fellow (1998).
- NSF Faculty Early Career Development (1999).
- Presidential Early Career Award for Scientists and Engineers (2000).

- Allen Newell Medal for Research Excellence, Carnegie Mellon University (2001).
- Ralph Watts Excellence in Teaching Award, Cornell University (2001).
- Best Paper, Usenix Security Symposium (2006).