School of
Computing Science

Region-based Memory Management

Advanced Operating Systems

Tutorial 3

Review of Lectured Material

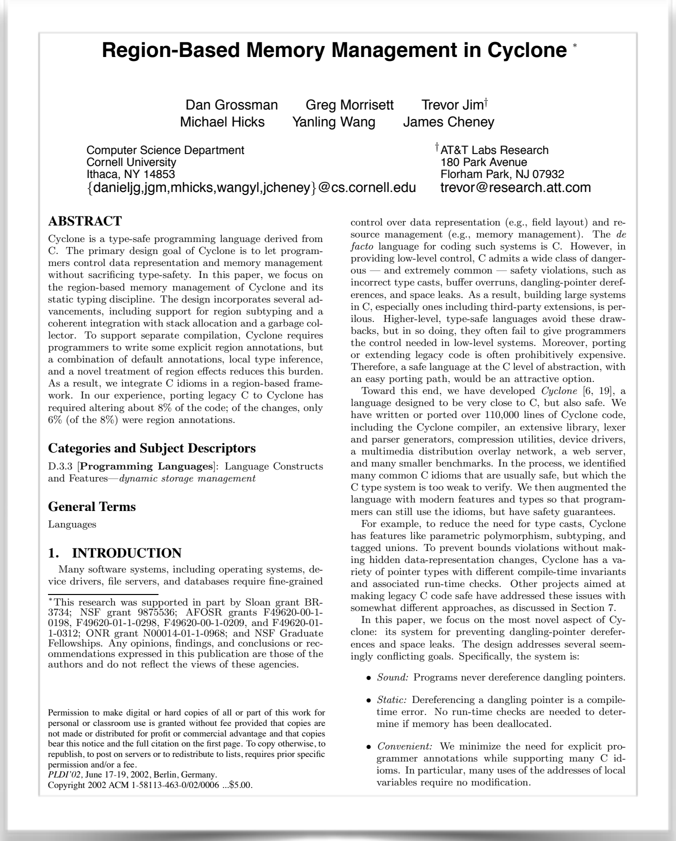
- Region-based memory management
  - Relation to stack-based management of local variables
  - Ownership and tracking changes to ownership
  - Borrow – shared references to immutable data vs. unique references to mutable data
  - Safety guarantees
  - Limitations – cyclic data structures and shared ownership

Key Learning Outcomes

- Understand how ownership rules allow automatic memory management
- Understand how borrowing rules enforce safety, and what safety guarantees are provided

Discussion

- Reading for this tutorial:
  - D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney, “Region-based memory management in Cyclone”, Proc. ACM PLDI, Berlin, June 2002. DOI:10.1145/512529.512563
- Discussion
  - What was Cyclone? Did the project’s goals make sense?
  - How does the region-based memory management system described differ from that outlined in the lecture?
    - Dynamic regions – compare to the lexically scoped regions and borrowing in Rust: how are regions defined for returned pointers?
    - Mechanisms to specify regions for function parameters/return values
    - Region polymorphism
  - Interactions with the garbage collector?
  - Other features added to C?
    - Variable sized arrays, not-null pointers, new string type
  - Ease of porting C code? Performance?
  - Does it make sense to try to extend C with region-based memory management?



Region-Based Memory Management in Cyclone [*]

Dan Grossman Greg Morrisett Trevor Jim[†]
Michael Hicks Yanling Wang James Cheney

Computer Science Department
Cornell University
Ithaca, NY 14853
{danieljg,jgm,mhicks,wangyl,jcheney}@cs.cornell.edu

[†]AT&T Labs Research
180 Park Avenue
Florham Park, NJ 07932
trevor@research.att.com

ABSTRACT

Cyclone is a type-safe programming language derived from C. The primary design goal of Cyclone is to let programmers control data representation and memory management without sacrificing type-safety. In this paper, we focus on the region-based memory management of Cyclone and its static typing discipline. The design incorporates several advancements, including support for region subtyping and a coherent integration with stack allocation and a garbage collector. To support separate compilation, Cyclone requires programmers to write some explicit region annotations, but a combination of default annotations, local type inference, and a novel treatment of region effects reduces this burden. As a result, we integrate C idioms in a region-based framework. In our experience, porting legacy C to Cyclone has required altering about 8% of the code; of the changes, only 6% (of the 8%) were region annotations.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*dynamic storage management*

General Terms

Languages

1. INTRODUCTION

Many software systems, including operating systems, device drivers, file servers, and databases require fine-grained control over data representation (e.g., field layout) and resource management (e.g., memory management). The *de facto* language for coding such systems is C. However, in providing low-level control, C admits a wide class of dangerous — and extremely common — safety violations, such as incorrect type casts, buffer overruns, dangling-pointer dereferences, and space leaks. As a result, building large systems in C, especially ones including third-party extensions, is perilous. Higher-level, type-safe languages avoid these drawbacks, but in so doing, they often fail to give programmers the control needed in low-level systems. Moreover, porting or extending legacy code is often prohibitively expensive. Therefore, a safe language at the C level of abstraction, with an easy porting path, would be an attractive option.

Toward this end, we have developed *Cyclone* [6, 19], a language designed to be very close to C, but also safe. We have written or ported over 110,000 lines of Cyclone code, including the Cyclone compiler, an extensive library, lexer and parser generators, compression utilities, device drivers, a multimedia distribution overlay network, a web server, and many smaller benchmarks. In the process, we identified many common C idioms that are usually safe, but which the C type system is too weak to verify. We then augmented the language with modern features and types so that programmers can still use the idioms, but have safety guarantees.

For example, to reduce the need for type casts, Cyclone has features like parametric polymorphism, subtyping, and tagged unions. To prevent bounds violations without making hidden data-representation changes, Cyclone has a variety of pointer types with different compile-time invariants and associated run-time checks. Other projects aimed at making legacy C code safe have addressed these issues with somewhat different approaches, as discussed in Section 7.

In this paper, we focus on the most novel aspect of Cyclone: its system for preventing dangling-pointer dereferences and space leaks. The design addresses several seemingly conflicting goals. Specifically, the system is:

- *Sound:* Programs never dereference dangling pointers.
- *Static:* Dereferencing a dangling pointer is a compile-time error. No run-time checks are needed to determine if memory has been deallocated.
- *Convenient:* We minimize the need for explicit programmer annotations while supporting many C idioms. In particular, many uses of the addresses of local variables require no modification.

[*] This research was supported in part by Sloan grant BR-3734; NSF grant 9875536; AFOSR grants F49620-00-1-0198, F49620-01-1-0298, F49620-00-1-0209, and F49620-01-1-0312; ONR grant N00014-01-1-0968; and NSF Graduate Fellowships. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not reflect the views of these agencies.

PLDI’02, June 17-19, 2002, Berlin, Germany.