Memory and type safety for embedded systems

Paul Soulier and Depeng Li

Deptartment of Information and Computer Science, University of Hawaii at Manoa, Honolulu, HI, 96822

Introduction

Security vulnerabilities that originate from memory-related errors have obviously negative consequences to systems programs and embedded systems. Many application domains have benefitted from advances in programming languages that have minimized or eliminated these types of errors while the vast majority of embedded systems are still written in the C programming language – an inherently unsafe language. This research explores the features necessary for an effective alternative to C.

Objective

Unify type and memory safety, anti-aliasing properties, and language-based memory containment into a type system that possess:

- Simplicity
- Expressive power
- Early error identification through static type checking

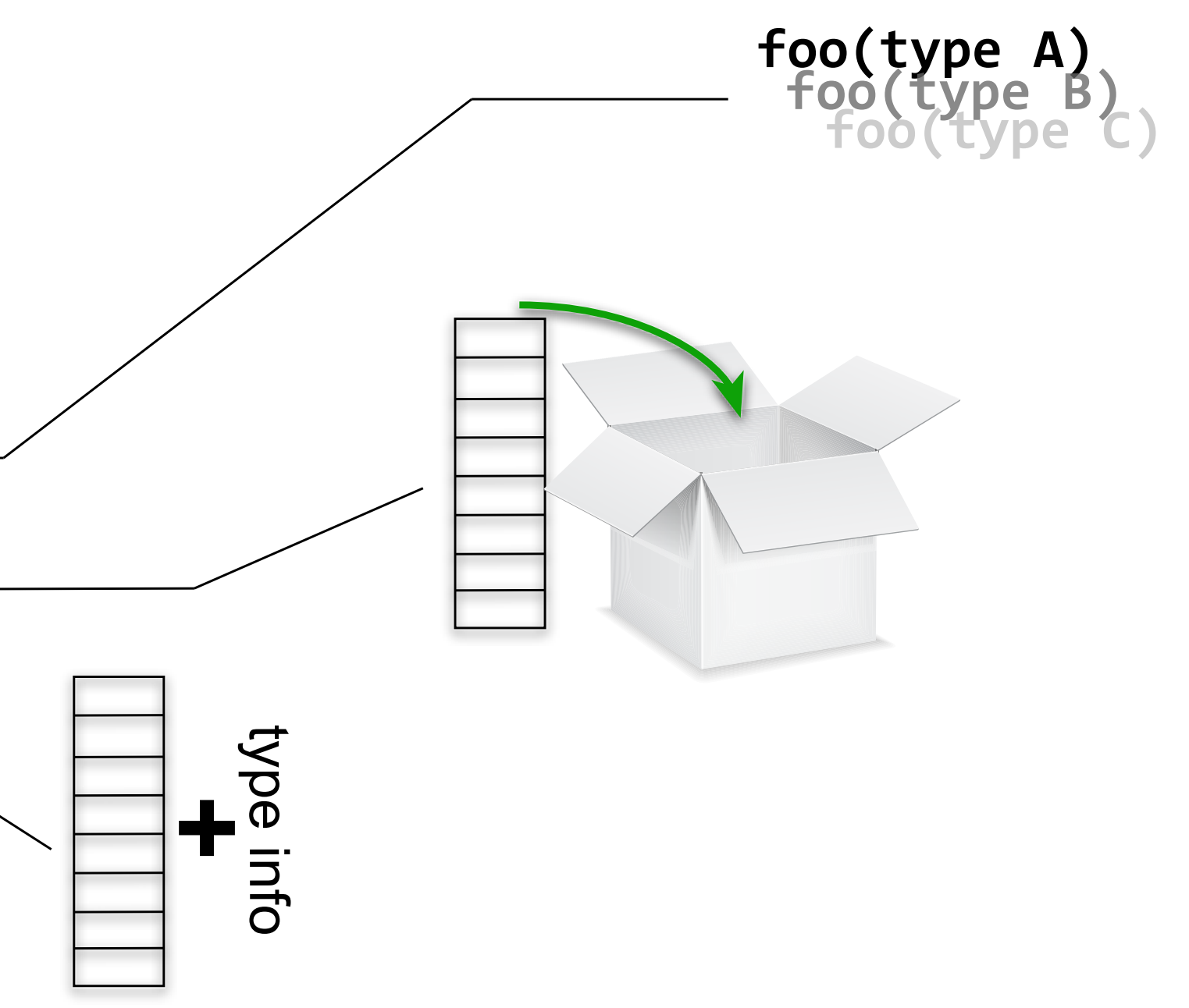
Type and Memory Safe

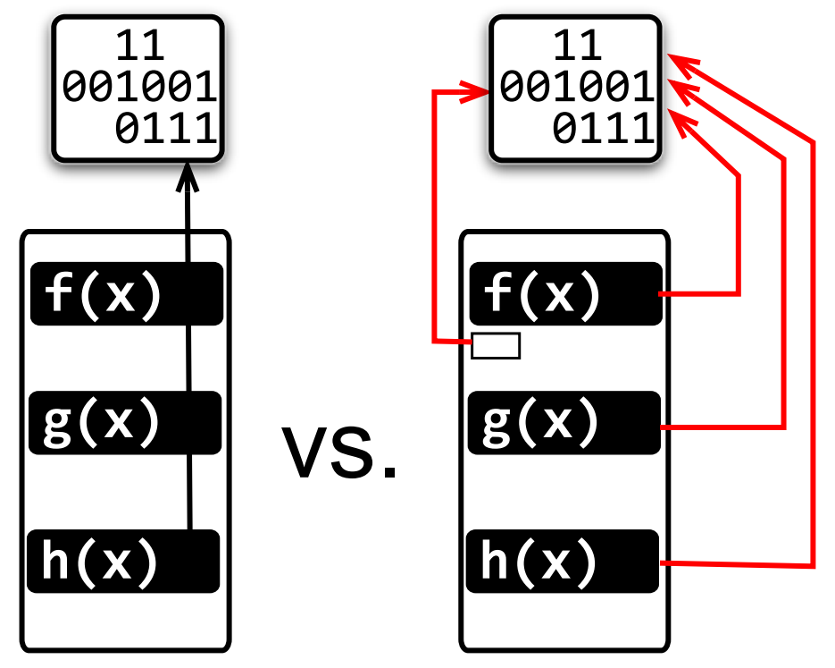
Type and memory safe languages are well understood with many existing implementations. Embedded systems pose unique challenges to type systems that are not addressed by most languages where the necessity to interact with hardware or protocols necessitates the ability to precisely define and manipulate unboxed data types and structures without compromising performance.

Polymorphic parameterization retains performance but can produce code bloat.

Boxing provides flexibility at the cost of performance.

Static boxing combines benefits from other methods while minimizing code bloat and maintaining performance.





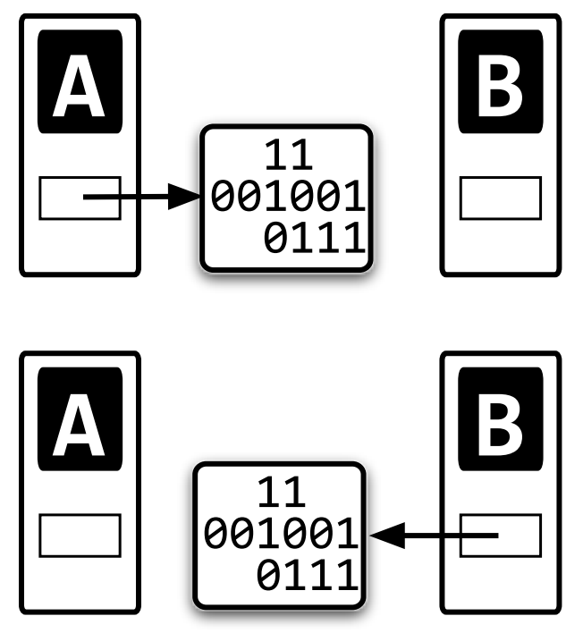
Anti-Aliasing

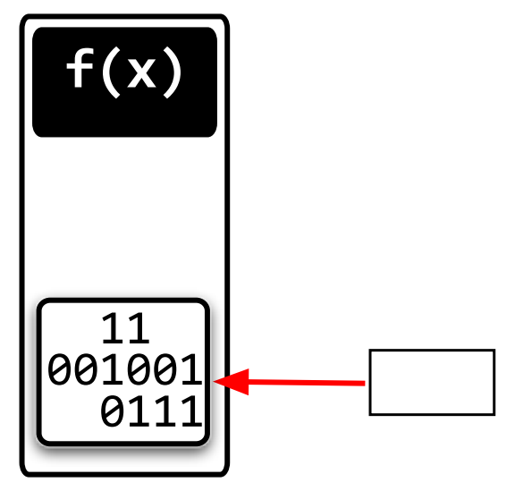
Aliasing in a software system occurs when a single object in memory is addressed by more than one reference. Aliasing has the unwanted potential to reduce code comprehensibility and complicate automated methods of code verification and analysis. This, in turn, can lead to unanticipated state changes when objects are manipulated in unexpected ways. Using value semantics, linear types, and specific pointer binding rules, aliasing can be minimized.

Value semantics can minimize the unnecessary use of aliases (e.g.: function parameters, local variables, etc.)

Linear types provide a degree of flexibility found in pointers while disallowing aliasing

Pointer binding rules are used to manage unwanted aliasing

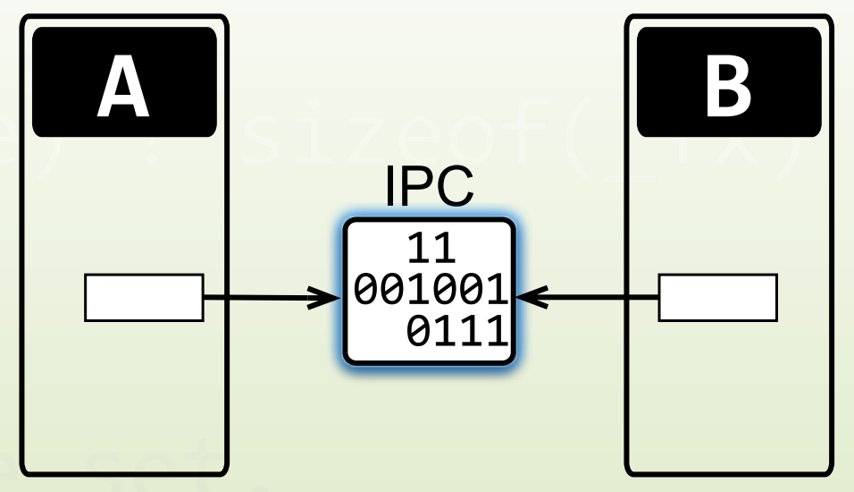




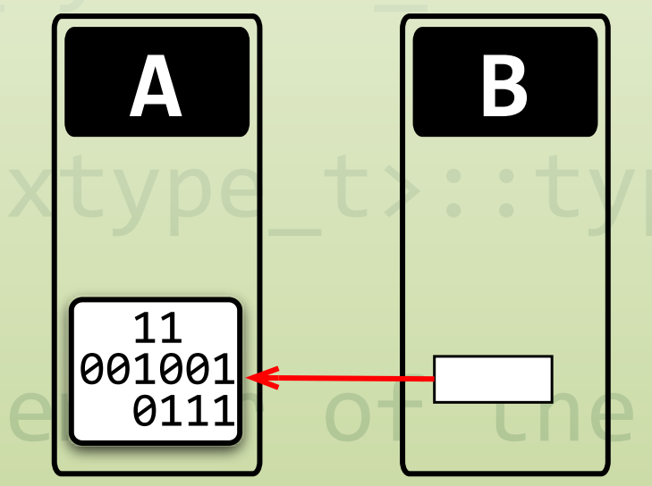
Memory Containment

Multi-processor devices have become commonplace and with the proliferation of such architectures, the complexity of maintaining memory integrity also increases. Language-level support for concurrency offers many advantages in providing memory-safe operations. Software-based isolation can confine memory access to within a given process and ensure accesses outside of the process employ proper IPC mechanisms.

Proper inter-thread memory access is enforced through language mechanisms.



Thread-local data is confined to the originating thread; external references are disallowed.



Future Direction

- Formalization of the type system
- Addressing semantic issues necessary for a practical language
- This research is part of a larger project to design a complete language with a focus on secure application development

Conclusion

With more computational power and wireless connectivity being incorporated into smaller packages, embedded systems are finding their way into many critical applications such as insulin pumps, sensor networks, and industrial control devices. Given the potential consequences of security flaws in such devices the need for new type systems and programming languages to help prevent software flaws is a critical component to ensuring the security of future embedded software systems.

Refrences:

- E. Brewer et al, Thirty Years is Long Enough: Getting Beyond C., Proceedings of the 10th conference on Hot Topics in Operating Systems-Volume 10. USENIX Association, 2005.
- J. Shapiro, Programming Language Challenges in Sys- tems Codes., Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE, 2008.
- M. Tofte, J. Talpin. Region-based memory manage- ment., Information and Computation 132.2 (1997): 109-176.
- D. Grossman, et al. Region-based memory management in Cyclone., ACM SIGPLAN Notices. Vol. 37. No. 5. ACM, 2002.
- J.T. Morrisett, et al. Cyclone: A Safe Dialect of C., USENIX Annual Technical Conference, General Track. 2002.
- H. J. Boehm. Threads cannot be implemented as a library., Technical Report HPL-2004-209, Hewlett Packard, 2004.
- J.R. von Behren, J. Condit, E.A. Brewer. Why Events Are a Bad Idea (for High-Concurrency Servers)., Ho- tOS. 2003.

Contacts:
Paul Soulier: psoulier@hawaii.edu
Depeng Li: depengli@hawaii.edu