6.033 System Critique: Eraser

Zachary Holbrook

March 15, 2018

1 Introduction

Eraser is a tool for dynamically detecting data races in multithreaded programs. Multithreaded programs are notoriously hard to debug because race conditions can occur randomly depending on the interleaving of threads [1]. Better tools are needed to help developers ensure correctness of multithreaded programs, and Eraser attempts to address this problem. To dynamically detect data races, Eraser uses the Lockset algorithm to detect if write-shared variables are protected by locks [1]. Additionally, a simple state machine is used to control when accesses of a shared variable need to be lock protected [2.2]. However, Eraser still emits false positives, so it provides annotations for suppressing specific race condition warnings [3.3].

Eraser's primary goals are to detect data races, to be simple in its implementation, and to be easy to use. The system largely meets its goals of detecting data races, as shown by numerous examples on real world programs [4], and being simple to implement. However it fell short on being easy to use, primarily due to the large number of false positives. Also, while not primary design goals, performance and portability are lacking in the initial design for some use cases, but could be improved upon.

1.1 Simplicity

At its core, Eraser relies on a relatively simple algorithm to detect data races: the Lockset algorithm. Lockset works by maintaining a set of locks that could be protecting a shared variable and intersecting this set with the set of locks in use when a variable is accessed [2]. When the set of locks protecting a variable becomes empty, there is a potential data race because the single lock protection discipline is violated. Eraser also maintains a simple state machine for each variable that indicates if it is being initialized, is read shared, or is write shared [2.1]. Then, Eraser only requires that accesses to variables in the write shared state are lock protected [2.2]. Eraser implements this algorithm by adding instruction to a program binary using ATOM, a tool for modifying program binaries on the Alpha processor [3]. This makes Eraser simpler to implement because it can simply add Eraser instructions around each load and store without needing to parse and understand high level languages. However, the simplicity gained through using only dynamic testing has tradeoffs with ease of use and performance. Because not even simple static checks are made, more false positives are dynamically detected, and Eraser adds more dynamic checks than might be necessary.

1.2 Ease Of Use

It's easy to run a program with Eraser and get a list of potential data races. However, it's not easy to weed out the false alarms from this list. The Lockset algorithm produces many false alarms because it enforces a simple locking discipline: that a write shared variable is always protected by at least one lock whenever an access occurs [2]. However, other strategies exist to protect shared variables. For example, a program might use message queues or thread safe objects. Therefore, developers must create many annotations to suppress false alarms [3.3]. The authors seem to imply that this isn't that bad, but for very large software systems, it could be a potentially long and error prone process. Developers might need to suppress thousands of false alarms, and errors could easily be made where a true data race is accidently suppressed because developers have to manually inspect source code to determine if a data race warning is legitimate. This makes Eraser harder to use for large software projects without significant investment time to first ensure that false alarms are correctly suppressed.

1.3 Performance

While Performance was not a major design goal of Eraser [3.2], the authors did include an important performance enhancement that made storing the locksets feasible. This enhancement was to store unique locksets in a table. Then each variable could simply store an index into this table, thereby greatly reducing the memory requirements of Eraser [3.1]. Despite this, Eraser still had a 10-30 times slowdown compared to running the original program. For many use cases this slowdown is fine [4]. However, this could make it difficult to use Eraser for some use cases, like those involving time sensitive programs or programs that already take a long time to run [3.2]. Eraser could be extended to these use cases if some performance improvements were made, like making better use of the features of ATOM, or doing a small amount of static analysis beforehand to remove some dynamic checks [3.2].

1.4 Portability

Eraser was originally designed to be used on program binaries for the Alpha processor using ATOM [3]. However, it could be extended to other platforms like x86 or ARM by using program binary modification tools for those platforms. This would be reasonable because both x86 and ARM use load and store instructions, so Eraser could insert similar instructions around these that it does for load and stores for the Alpha processor. However, it would be more difficult to port Eraser to interpreted programs like Python or Java. These programs don't produce binaries compiled to machine code, so Eraser would need to take a different approach to modifying them. While potentially difficult, its at least in principle possible because Java is compiled to an intermediate Java bytecode that could be potentially edited, and Python bytecode can be modified during runtime.

1.5 Conclusion

Overall, Eraser achieves its goals of dynamically detecting race conditions, as evidenced by several tests on real world programs [4], and it achieves its goal of being simple to implement. However, it isn't very

easy to use because of the high number of false alarms that take significant work to suppress using annotations. If modifications were made to reduce the number of false alarms, perhaps by some preliminary static checking, albeit at the cost of some simplicity, Eraser would be more useful. Detecting data races in concurrent programs remains a difficult problem today, and tools that could reliably detect data races without many false alarms would be very useful.

1.6 Works Cited

S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, “Eraser: a dynamic data race detector for multithreaded programs,” ACM Transactions on Computer Systems, vol. 15, no. 4, pp. 391–411, 1997.

Word Count: 989