Stephen, I see substantial improvement in this critique--you understand the genre and the assignment now. Great work. You identify Eraser, explain the core component and give your assessment. I'd like to see you give an overview of the system in terms of its modules and interactions. I see that you propose that it isn't really a system, but I think it would help to try this anyway. Even software has modules. Your analysis defines each design goal in this particular context, then explains which system features support or enable it. Your tone is appropriate and your focus is on target. Overall, nice work.

Grade: A-

1 Overview.

The *Eraser* software discussed in ***Eraser: A Dynamic Data Race Detector for Multithreaded Programs,* Stefan Savage** et al. is the sole focus of this paper.

The primary goal of the software is to provide an effective testing suite for detecting data races in multi-threaded programs using lock-based synchronization. Rather than discuss fault tolerance, it would be more appropriate to analyze how effective it is at detecting race conditions. In one word, efficacy is the main design goal of Eraser, as the motivation behind it was to fix all the shortcomings of its predecessors. Past attempts failed to detect race conditions in a number of contexts. Simplicity and performance are considered secondarily, as the creators aim to bring their software to as many users and use-cases as possible, but performance past basic functionality is not investigated in this paper. Security and fault-tolerance are not applicable design principles to Eraser, but again, the notion of efficacy is important as prior attempts at race-condition detection software frequently failed to catch all data races. The author introduces Eraser and its underlying algorithm, *Lockset*, and further details their performance on a range of threaded code.

The introduction identifies the purpose of Eraser and names the key criteria in comparison to previous systems, ok.

2 Analysis.

The author explains the underlying algorithm, *Lockset*, along with two modifications to fix certain edge cases in which it fails. He does a good job of explaining this algorithm, as it simply watches all accesses to protected variables during execution. It records which locks a thread possesses when it accesses a variable, so that it can deduce which lock consistently protects which variable. After execution, if there is no lock that has been used in every access, then there is no consistent locking. This approach is known as *lockset refinement* (2.0). The author goes forward to improve the initial algorithm by introducing the notions of virgin, shared, and exclusive states (2.2).

This section explains the lockset algorithm, good. You don't need to comment on how the author writes the paper; give your own understanding & assessment (I see more of this towards the end of the paragraph)

2.1 Efficacy

Defining this desgin goal in context, great

Efficacy is the central design goal of the Eraser software, where efficacy is how effective the software is at finding race conditions in different contexts. Past attempts at race-condition detection like the *happens-before* and *lock covers* techniques failed to consistently find all data races in certain contexts (1.2). While the Eraser software can not guarantee that a program is free from races, it vastly improves on other algorithms based on *happens-before*, and is a vast improvement to manual debugging (4.5). Eraser tends to throw false positives rather than fail to find races. Given that Eraser aims improve on the number of race condition scenarios recognized compared to past algorithms like *happens-before*, and be more efficient than manual debugging, Eraser does succeed in its goal of efficacy.

Again, you compare to previous systems, appropriate because this is the main area of improvement.

2.2 Performance

Not sure what this means?

Performance to the functional level is important to maintain the chief goal of efficacy. Instead of building Eraser to handle scaling to any size, it was engineered to perform consistently under standard conditions. Though, slow performance could drastically alter the run time of the program being tested, possibly masking or creating new bugs and altering the behavior of the program. While the author explicitly states that ”Performance was not a major goal in our implementation of Eraser” (3.2), this is slightly contradictory to their overall goal. Consequently, programs experienced a slowdown of %1000-%3000 when ran with Eraser compared to without. The paper fails to address the needs of time-sensitive applications, or applications that use networked machines; though, the author does speculate at the performance bottlenecks, as to invite others to improve Eraser.

Again, you specifically define the goal in this context. You also consider how well Eraser meets the goal in design and in practice. Great.

2.3 Simplicity

Simplicity is another shortcoming of the Eraser software. Its underlying algorithm commonly reports false positives, and an interface had to be introduced to remedy this. Annotations have to be inserted into the tested code base to suppress known false alarms, using the interface of 'EraserIgnoreOn()' and 'EraserIgnoreOff()' (3.3). The 'EraserReuse(addr,size)' annotation aims at preventing false positives from re-used memory, since Eraser is watching the low-level execution. Additionally, Eraser is only designed to work with the C **pthreads** lock library. A set of 4 more annotations were added for declaring private implementations of locks. Only having native support for pthreads based locks is a serious drawback of Eraser in terms of portability, and the list of annotations that have to be added into the source code are pitfalls of its simplicity of use. There are many cases where Eraser would just work as desired, but there are also cases where many of these exceptions that would greatly increase the complexity of its use. Hence, the simplicity of Eraser fails again to extend past basic functionality.

Direct connection between the goal (simplicity) and the system feature that enables or supports it. Good.

3 Conclusion.

Given that Eraser is more of a software than a system, efficacy, performance, and simplicity were analyzed rather than fault-tolerance, security, simplicity, and scalability. In summary, Eraser succeeds in meeting its central goal of providing a data race detection software better than its predecessors, but come up short on its secondary design considerations. More specifically, Eraser's performance has many areas which need improved. However, the authors speculate that performance *can* be improved and discuss ways to improve it. Also, there are certain scenarios in which Eraser would be a dependable testing software, which could be easily applied to an existing code base and work flawlessly. But, this is not guaranteed. Frequently, existing code needs to be modified to add annotations to suppress false positives, or explicit declarations of locks to Eraser if the pthreads C library is not used. Overall, the Eraser software is an effective and needed tool for testing the validity of multi-threaded code, but leaves room for improvement.

You explicitly state which goals you consider and why, then give your assessment. Great. (I'll leave it to the TAs to decide whether those are the right goals.)

Final word count ≈ 920