Report 1: Eraser: A Dynamic Data Race Detector for Multithreaded Programs

Dalia Camacho García-Formentí,
Instituto Tecnolgico Autnomo de México (ITAM)
Mexico City, Mexico

***Abstract*—Here I present a report on the article "Eraser: A Dynamic Data Race Detector for Multithreaded Programs" by Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. This report includes a summary of the main contributions in the article and an analysis.**

I. Summary

Eraser [1] is a tool that detects data races on dynamically defined variables. A data race occurs when two threads access a global variable simultaneously, none of the threads locks the variable, and at least one of the threads has an intent to write.

Previous approaches either identified data races from static variables or used the principle of *happens before*. The *happens before* principle states that events within a thread occur in order. And events from different threads can be ordered in terms of the order in which they access global variables. The problem with the *happens before* principle is that it depends on the scheduler. This means that the order in which events happen may not be consistent between different trials of the same program, and many trials would be necessary to track down all possible data races.

The idea of Eraser is to define a consistent set of locks, $C(v)$ for every global variable v, that are always present whenever a thread has access to v. If at the end of the program the set has at least one lock, then there is no risk of data races on variable v. However, if $C(v) = \emptyset$ then there is a risk of a data race and a warning is thrown. The set $C(v)$ is defined by the Lockset algorithm[1]:

```
Let $locks\_held(t)$ be the set of locks held by thread $t$
For each $v$ initialise $C(v)$ to the set of all locks
On each access to $v$ by thread $t$,
    set $C(v) := C(v) \cap locks\_held(t)$;
    if $C(v) = \emptyset$, issue a warning.
```

To define a data race detecting algorithm that takes advantage of the access dynamics to the variables, they define the different states that a global variable can be in. These states are *Virgin, Exclusive, Shared*, and *Shared-Modified*. The virgin state refers to initialisation of the variable v, exclusive means that only one thread has accessed v with either read or write. A variable v goes from an exclusive state to a shared state, when another thread performs a read action of v. And a variable v goes from exclusive or shared to shared modified when a different thread performs a write action or when one of the threads performs a write action. This is taken into account in the lockset algorithm as follows [1]:

```
Let $locks\_held(t)$ be the set of all locks held by thread $t$
Let $write\_locks(t)$ be the set of locks held in write mode by $t$
For each $v$ initialise $C(v)$ to the set of all locks
On each read of $v$ by thread $t$,
    set $C(v) := C(v) \cap locks\_held(t)$;
    if $C(v) = \emptyset$, issue a warning.
On each write of $v$ by thread $t$,
    set $C(v) := C(v) \cap write\_locks(t)$;
    if $C(v) = \emptyset$, issue a warning.
```

In the latter algorithm a distinction is done between read and write accesses, if a variable v is modified, then the only possible locks in set $C(v)$ are those held in write mode. This increases the probability of $C(v) = \emptyset$ and a warning to be raised. This helps guarantee that data races are detected more often when a write action is performed.

After defining the latter algorithm they implemented Eraser in the SPIN operating system. In this particular case Eraser did not perform that well, due to the interruptions received by the Kernel and timeouts. Eraser was also tested in Alta Vista and Vesta Cache server, where most warnings were false alarms, some of which were intentional data races. But, they did find a serious data race in Vesta. They also tried Eraser on previous versions of the programs that contained data races, and it could find the problems easily. To evaluate the performance of Eraser on programs made by non-experts, they tested it on students' programs, where Eraser performed quite well.

At the end they propose a further modification to the algorithm, which is as follows:

```
On each read of $v$ by thread $t$,
    if $C(v) = \emptyset$, issue a warning.
On each write of $v$ by thread $t$,
    set $C(v) := C(v) \cap write\_locks(t)$;
    if $C(v) = \emptyset$, issue a warning.
```

II. Analysis

Overall Eraser seems a good tool to detect data races, however, we can infer from their tests that a lot of false alarms are raised. In most cases they propose to avoid these warnings by annotating the code where false alarms occur. Annotating the code may be a tedious task, and is also error prone.

It would be best if the algorithm was modified, if possible, in a way in which false alarms do not appear as often. I believe the third algorithm is best, since it does not change $C(v)$ when a variable is read. If we consider the case of a read only variable, the first two algorithms may show warnings, which won't happen with the third algorithm. Moreover, the third algorithm is as strict with write locks as the second.

The third algorithm takes advantage of the access dynamics of global variables in the cases of shared and shared-modified states. However, a write action in the exclusive and in the shared-modified states is considered in the same fashion. This may produce warnings even when a variable is in an exclusive state throughout the program. Probably, adding a flag to the algorithm that defines the state of the variable and constructing the sets $C(v)$ jointly for all threads may help issue only the relevant warnings.

However, in terms of avoiding dangerous data races, Eraser seems to be performing quite well.

References

[1] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson, "Eraser: A dynamic data race detector for multithreaded programs," *ACM Trans. Comput. Syst.*, vol. 15, pp. 391–411, Nov. 1997.