Roadrunner Strengths:

* Flexible
* Robust
* Reduces overhead on implementing dynamic analyses
* Manages messy low-level details of dynamic analysis and provides a clean API for communicating an event stream to back-end analysis tools
* Each event describes some operation of interest performed by the target program, such as accessing memory, acquiring a lock, forking a new thread, etc.
* Separating of events allow the developers to focus on essential algorithmic issues of a particular analysis, rather than on orthogonal infrastructures complexities
* Offers comparable performance to traditional, monolithic analysis prototypes, while being up to an order of magnitude smaller in code size.
* Writing a ROADRUNNER back-end analysis tool only requires defining methods to appropriately handle various events of interest.

Detections:

Race conditions : [5,6,8,10,21,24,27,31]

Race detection tool:

* Lockset: 24
* Eraser with Barrier: 24
* Fast Track: 10
* Happens Before: 19 (analysis only shared, mutable memory locations)

Deadlocks: [1,16]

Violations of atomicity [9,13,28,29]

Atomizer: 9 (weakness may ignores free-race location)

Determinism properties [4,23]

* Eraser [3] is a dynamic data race detector that targets lock-based synchronization used in multithreaded programs. Eraser performs a lockset analysis to ensure that the program strictly follows a locking discipline. A simple locking discipline is that each variable shared between threads is protected by a particular lock.
* When the program being monitored does not follow the locking discipline, Eraser flags a data race. While this approach identifies data races that may occur in schedules other than the one that is examined they extend their detector to support thread start and join operations by introducing an ownership model. Though it reduces the number of false positives considerably compared to Eraser, it is still not precise. The advantage of this technique is that it reduces the overhead of race detection to under 50% by applying static and dynamic optimization techniques.

Roadrunner Weaknesses:

* a drawback is that it may have false positives. This is because there could be regions in the program where accesses to a variable are unprotected by locks, like initialization, and also the lock protecting a variable could change overtime.
* Another major drawback of this approach is that it only handles lock-based synchronization. It does not handle other forms of synchronizations like fork-join, barriers, post-wait, etc., which are used extensively in multithreaded programs.
* Scalability
* Relies heavily on JVM Jit’s Compiler