Separation-Based Reasoning for Deterministic Channel-Passing Concurrent Programs

Aimee Borda and Adrian Francalanza

University of Malta

Tractable proof systems for concurrent processes are hard to achieve because of the non-deterministic interleaving of statements. This is particularly true when the scheduling of statements have an effect on the final state of the program, referred to as racy conditions. As a result, when we try to reason about concurrent programs, we have to consider all possible thread interleaving to accurately approximate the final result. Since, scalability is important we limit our reasoning to a subset of the concurrent programs that are race-free. By doing so we gain serialized reasoning, where examining one possible path will automatically encapsulate the other interleaving.
Fundamentally, races are caused when two concurrent processes try to access shared resources simultaneously (for example two concurrent processes trying to alter a location in memory at the same time) and hence the final result depends on the last statement that updates the location.
A popular approach for delimiting the class of race-free processes is through resource separation. [3] Through the resources assigned to each process we can approximate the interference that a process can create to its concurrent processes. Therefore, if the resources available are split disjointly among the concurrent processes, we guarantee that no interference between the processes is created. More concretely, if all concurrent processes work within different parts of the memory, each process will not create race conditions for other processes. Moreover, this guarantee allows us to reason about each process independently, as a sequential process.
Nonetheless, the formalism must also handle dynamic resource separation and resource sharing, where a subset of the resources' access rights are shared amongst the processes. A popular example is the reader-writer problem. Here, race-conditions are created only, if one of the concurrent processes modifies the content of a shared location, since trying to read that location will depend on whether the statement is scheduled before or after the write has been committed. Hence, resource sharing can be allowed if the actions performed by each process on that resource do not create races to values, which in this example imply that we have concurrent reads but exclusive writes.
Separation logic is a possible formalism which handles this quite elegantly.[3,2] In [1], a proof system inspired from separation logic to reason about concurrent processes for the message passing model is described. However, it can only reason in terms of the complete disjointness of resources. In this work, we are trying to push the boundaries of non-racy processes by examining some form of resource sharing and how these structures preserve determinism.

References

1. Adrian Francalanza, Julian Rathke, and Vladimiro Sassone. Permission-based separation logic for message-passing concurrency. *Logical Methods in Computer Science*, 7(3), 2011.
2. Peter W. O'Hearn. Resources, concurrency, and local reasoning (abstract). In David A. Schmidt, editor, *ESOP*, volume 2986 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2004.
3. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.