We present a new approach to check for Commutativity in concurrent programs from their run-time state-chart graphs. Two operations are said to be commutative if changing the order of their execution do not change the resultant effect on the working object. Commutative property is capable of boosting performance in concurrent transactions such that transactional concurrency is comparable to a non-blocking linearizable version of a similar data structure type. Transactional concurrency is a technique that analyses object semantics, as object states, to determine conflicts and recovery between conflicting operations. Processes that commute at object level can be executed concurrently at transaction level without conflicting with one another. In our approach we generate graphs by tracking run-time execution of concurrent program and representing object states in all possible thread interleavings as states and transitions. Using state-chart notations, we capture the object states at each execution step and compare their states before and after transitions as processed by a known set of operations reordered in different ways. Considering the non-deterministic nature of concurrent programs, we exhaustively capture program states during method invocation, method response, atomic instruction execution, etc., for determining commutativity. This leverages to examine state transitions at a thread level granularity, across all possible interleavings. With this methodology user can not only verify for commutativity, but also can visually check ways in which functions commute at object level, which is an edge over current state-of-the art tools. The object level commutative information helps in identifying faulty implementations and performance improving considerations. We use a graph database to represent state nodes that further assists to check for other concurrency properties using cypher queries.

Approved for distribution by Damian Dechev, Committee Chair, on June 20, 2016.

The public is welcome to attend.