Chapter 5: The Relative Power of Primitive Synchronization Operations

This chapter shifts focus from building correct objects to understanding the **relative power of different synchronization primitives**. It introduces the concept of **Consensus Numbers**, a key metric for measuring the power of these primitives.

Consensus Numbers

A **consensus protocol** is a way for threads to agree on a single outcome from a set of proposed inputs. For a given object, its
consensus number is the maximum number of threads that can solve the consensus problem using only that object and atomic registers[11]. The consensus number provides a formal way to classify synchronization primitives.

- **Atomic Registers (Consensus Number 1):** Atomic registers, even though they are the building blocks of other objects, have a consensus number of only 1[22]. This means they cannot be used to solve consensus for more than one thread.

- **Compare-and-Swap (Consensus Number ∞):** The compareAndSet() operation is an example of a primitive with an **infinite consensus number**[333]. This means it can be used to solve consensus for any number of threads, making it one of the most powerful primitives.

Wait-Free and Lock-Free Objects

A key concept in this chapter is
wait-freedom, which is a type of progress condition[44].

- **Wait-free** operations guarantee that every thread can make progress in a finite number of steps, regardless of the behavior of other threads[55]. This means no thread can be delayed indefinitely. The problem with wait-free protocols is that they can be very complex.

- **Lock-free** protocols guarantee that at least one thread will make progress, even if others are delayed[66].

Key Synchronization Primitives and Their Consensus Numbers

The chapter analyzes the power of various synchronization primitives:

- **FIFO Queues:** A simple FIFO (First-In, First-Out) queue has a consensus number of 2[77]. This means you can use a FIFO queue to solve consensus for two threads. However, if the queue is augmented with a

  peek() method (which allows you to look at the first item without removing it), it has an **infinite consensus number**[88]. This shows how a small change can dramatically increase the power of an object.

- **Read-Modify-Write (RMW) Operations:** A general getAndSet() RMW operation has a consensus number of 2[99]. An example is a simple

  swap() operation.
- **Multiple-Assignment Objects:** An (n, n(n+1)/2)-assignment object has a consensus number of at least n[1010]. These objects are a class of read-modify-write operations where threads can update a specific set of fields atomically.

The Compare-and-Swap (compareAndSet()) Operation

The compareAndSet() operation is a pivotal primitive in this chapter.

- It atomically compares the value of a memory location with an expected value, and if they are the same, it updates the location with a new value.
- It's a powerful primitive that can be used to implement many other synchronization operations, including those with infinite consensus numbers[111111]. Its widespread use in modern hardware makes it a practical and powerful tool for building concurrent data structures.

- The
  compareAndSet() operation is not a classic Read-Modify-Write (RMW) operation because it returns a Boolean value (true or false) instead of the register's prior value[12]. However, you can use it along with a

  get() method to create a new object that does behave like a classic RMW primitive[13].