Revisiting Abstraction Functions For Reasoning About Concurrency

Jeannette M. Wing
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract

Hoare introduced abstraction functions for reasoning about abstract data types in 1972 [1]. What happens in the presence of concurrency? Reasoning about objects in a concurrent system necessitates extending the notion of abstraction functions in order to model the system's inherent nondeterminisitic behavior. My talk presented in detail the extensions required to reason about lock-free concurrent objects, used to build linearizable systems. It also discussed briefly a different extension required to reason about atomic objects, used to build fault-tolerant distributed systems.

Much of this work is joint with Maurice Herlihy and previously published in two papers [2, 3].

1 Background

Hoare introduced abstraction functions for reasoning about abstract data types in 1972 [1]. In the sequential domain, an implementation consists of an *abstract type* ABS, the type being implemented, and a *representation type* REP, the type used to implement ABS. The subset of REP values that are *legal* representations is characterized by a predicate called the *representation invariant*, I: REP → BOOL. The abstraction function,

A: REP → ABS

maps each representation value that satisfies the invariant to a single abstract value.

2 What happens in the presence of concurrency?

A *concurrent object* is a data object shared by concurrent processes. *Linearizability* is a correctness condition for concurrent objects defined in terms of the semantics of abstract data types. Intuitively, linearizability requires that each operation executed appear to "take effect" instantaneously and that the order of nonconcurrent operations be preserved. These two requirements allow us to describe acceptable concurrent behavior directly in terms of acceptable sequential behavior, an approach that simplifies both formal and informal reasoning about concurrent programs.

In order to prove that an implementation of a concurrent object is correct, i.e., linearizable, it is necessary to extend the notion of an abstraction function. The abstraction function must map a single representation value (that satisfies the invariant) to a *set* of abstract values:

A: REP → 2^{ABS}

The change to the range of the abstraction function results from inherent nondeterminism as defined by linearizability. Intuitively, the nondeterminism arises because at any point in time operations concurrently performed on an object may or may not have "taken effect." For each operation we want to permit the possibility that it has or has not. The paper by Herlihy and Wing [2] motivates linearizability, discusses the change to the notion of abstraction function, and walks through an example of a FIFO queue in depth.

3 What happens in the presence of concurrency and faults?

An *atomic object* is a data object shared by concurrent transactions. A transaction is a process that executes a sequence of operations where that sequence (in contrast to each individual operation) is considered an atomic unit, i.e., "all-or-nothing." A transaction may succeed, in which case all its operations take effect; or, it may fail, in which case none of its operations take effect. Faults such as lost messages and site crashes are masked as aborted (failed) transactions.

Atomicity is a correctness condition for atomic objects defined in terms of the semantics of abstract data types. It requires that all transactions that perform operations on an object be "all-or-nothing," serializable (the concurrent execution of a set of transactions must be equivalent to some sequential execution), and permanent (effects of committed transactions persist).

In order to prove that an implementation of an atomic object is correct, i.e., atomic, it is necessary to extend the notion of an abstraction function. The abstraction function must map a single representation value (that satisfies the invariant) to a *set* of *sequences* of abstract operations:

A: REP → 2^{OPS*}

where OPS is the set of operations on the abstract type, ABS. Nondeterminism (set of sequences) is inherent because any serialization should be allowed. We need to keep track of history information (sequences of operations) because the future successful completion of a transaction may require that its operations be "inserted in the middle" of a history, where the resulting history is a serialization. The paper by Wing [3], based on earlier work by Herlihy and Wing, explains in detail the model of computation for atomic objects and the extended notion of abstraction functions for reasoning about the correctness of their implementations.

References

[1] C.A.R. Hoare, "Proof of Correctness of Data Representations," *Acta Informatica*, Vol. 1, 1972, pp. 271-281.

[2] M.P. Herlihy and J.M. Wing, "Linearizability: A Correctness Condition for Concurrent Objects," *ACM TOPLAS*, Vol. 12, No. 3, July 1990, pp. 463-492.

[3] J.M. Wing, "Verifying Atomic Data Types," *International Journal of Parallel Programming*, Vol. 18, No. 5, 1989, pp. 315-357.