Closed Schedulers: Constructions and Applications to Consensus Protocols

Ronit Lubitch & Shlomo Moran
Dept. of Computer Science, Technion, Haifa 32000, Israel

Abstract. Analyzing distributed protocols in various models often involves a careful analysis of the set of admissible runs, for which the protocols should behave correctly. In particular, the admissible runs assumed by a t-resilient protocol are runs which are fair for all but at most t processors. In this paper we define closed sets of runs, and suggest a technique to prove impossibility results for t-resilient protocols, by restricting the corresponding sets of admissible runs to smaller sets, which are closed, as follows:

For each protocol PR and for each initial configuration c, the set of admissible runs of PR which start from c defines a tree in a natural way: the root of the tree is the empty run, and each vertex in it denotes a finite prefix of an admissible run; a vertex u in the tree has a son v if v is also a prefix of an admissible run, which extends u by one atomic step.

The tree of admissible runs described above may contain infinite paths which are not admissible runs. A set of admissible runs is closed if for every possible initial configuration c, each path in the tree of admissible runs starting from c is also an admissible run. Closed sets of runs have the simple combinatorial structure of the set of paths of an infinite tree, which makes them easier to analyze.

We introduce a unified method for constructing closed sets of admissible runs by using a model-independent construction of closed schedulers. We use this construction to provide unified proofs of impossibility results in various models of asynchronous computations. One of our results generalizes a known impossibility result in a non-trivial way.

1 Introduction

A distributed decision task is a distributed task in which every processor eventually makes an irreversible decision step. One of the more challenging problems in distributed computing is the characterization of the decision tasks that can be solved in the presence of crash (fail stop) failures, under which a processor may stop participating in the protocol prematurely. A protocol that solves such a task in the presence of at most t crash failures is called t-resilient. A general characterization of tasks that can be solved in the presence of t crash failures is known only for the case t = 1 [2]. In spite of the large number of papers published in this area, our understanding of t-resilient protocols for t > 1 is still
quite limited. For instance, for each $t \geq 2$, it is not yet known whether there are $t$-resilient protocols for the renaming task with $n + t - 1$ new names [1], or for the $k$-set consensus task with $k = t$ [3].

The difficulty of this problem does not seem to depend on the specific model of computation studied (i.e., shared memory or message passing), but more on the inherent difficulty of coordination between processors in a totally asynchronous environment, and in particular on the impossibility to distinguish between faulty processors and processors which are very slow, but in working order. Consequently, it is possible to have a $t$-resilient protocol for a given task, with the following unpleasant property: The number of steps that may be executed by the protocol, when started from a certain initial configuration, before it fulfills its task, is unbounded.

In this paper we propose an approach for analyzing asynchronous protocols which avoids the difficulty mentioned above. In this approach, we restrict the set of runs for which the protocol is required to behave correctly to a set of a simple structure, which we call "closed". A closed set of runs has the property that if a protocol is guaranteed to fulfill some task in each run in it, then it is guaranteed to fulfill that task within a fixed number of steps. We use this approach to provide alternative proofs for the impossibility of $t$-resilient consensus protocols in various models. One of these proofs generalizes the result of [6] in an interesting way.

1.1 Protocols and Runs

A distributed system consists of a set of $n$ ($n \geq 2$) asynchronous processors $\{p_1, \ldots, p_n\}$, modeled as (not necessarily finite) state machines, and of some means of communication among the processors (e.g., shared memory or message passing).

Each processor $p$ acts according to a deterministic transition function $t_p$. The transition function is described by the set of atomic steps which can be taken by the processor. An atomic step consists of a possible change of the processor's state, and of reading and/or writing from the communication means. A protocol for a given distributed system is a set of $n$ transition functions, one per processor.

A configuration of the system is a description of the system at some moment. It consists of the internal state of each processor and of the contents of the communication means. An initial configuration is one in which each processor is in an initial state, and the communication means contains some default value.

For each processor $p$ and for each configuration $c$, there is a (finite) set of atomic steps that can be taken by $p$ from the configuration $c$. A run of a protocol is an infinite sequence of atomic steps that can be taken in turn starting from some (initial) configuration $c$. Each atomic step is performed by one of the processors, and brings the system to a subsequent configuration. We say that a run $r$ is applicable to a configuration $c$ if it is a run that may start from the configuration $c$. If $r$ is applicable to $c$, then for every (finite) prefix $r'$ of $r$, the configuration resulted from applying $r'$ to $c$ is denoted by $\sigma(c, r')$. 