On Verification of Parallel Message-Passing Processes

Stein Krogdahl and Olav Lysne

1Department of Informatics, University of Oslo, Norway
2Simula Research Laboratory, Oslo, Norway

Abstract. One way to reason about parallel processes is to assume that the execution of each process is subdivided into ‘small enough’ steps, and that these are executed in an interleaved fashion, thus obtaining a sequential program. The steps should be so small that for any parallel execution there will, in a suitable sense, exist a corresponding interleaved execution ending in the same state. The usual way to ensure this is to require that each step should contain at most one global access. However, if the global entities are communication channels, then larger steps may in some cases be allowed, and this may make reasoning about the programs easier. This paper explores these cases, and discusses consequences for verification and deadlock avoidance.

Keywords: Interleaved execution; Parallel processes; Program verification

1. Introduction

To reason, intuitively or formally, about parallel programs is usually a demanding task. One approach for obtaining a firmer grasp on this problem is to ‘reduce’ the parallel execution to a sequential one by subdividing each process of the parallel execution into a sequence of small enough ‘elementary steps’, and simulate a parallel execution by allowing the steps to be executed in an interleaved fashion. What ‘small enough steps’ should mean will heavily depend upon how often the processes access global entities. An obviously safe requirement is simply to say that each interleaved step should make at most one global access, and that this access should act as an atomic action relative to the global entity in question.

By this technique, we can to a large extent reason about parallel executions as if they were sequential ones. The underlying assumption is then that for each parallel execution (PE) of the real system, there is a corresponding interleaved execution (IE) where each individual process experiences exactly the same sequence of actions, and where the final state after IE is the same as after PE. There exist several extensions of Hoare logic into parallel settings based on this idea (and on steps with at most one global access), both for shared variables [Lam80, Owe92] and for the message-passing paradigm [StS95].

Reasoning about interleaved, sequential executions can be done in more or less formal ways (and the choice here is not a topic for this paper). However, one natural way to approach such a task is to find an
invariant that will hold between the steps of the interleaved execution, and from which we can conclude what we want to know about the system. The verification then simply amounts to showing that the invariant is true initially, and that it is preserved by each step (when executed alone).

Obviously, the larger (and thus often ‘more natural’) the interleaved steps are, the more we can hope that a simple and ‘intuitive’ invariant will be strong enough to prove what we want. However, making the steps larger very soon gets in conflict with the requirement that each parallel execution should have a corresponding interleaved one. Thus, it is interesting to ask whether there are situations where the interleaved steps may contain more than one global access, without destroying this property. If the global entities are traditional variables, this seems not to be the case. However, if the global entities are communication lines transporting messages, then such cases exist.

One such case can be sketched as follows (see Fig. 1): Assume that two parallel processes $A$ and $B$ are connected by two FIFO-lines $AB$ and $BA$, carrying messages from $A$ to $B$ and from $B$ to $A$ respectively. Furthermore, assume that process $A$ consists of execution steps that (in the general case) start with one input from $BA$ and end with one output to $AB$ (but where one or both may be missing), and that $B$ is subdivided into similar steps that may input from $AB$ at the start and output to $BA$ at the end. We currently appeal to the reader’s intuition when claiming that, in a suitable sense, each parallel execution of such a system will have a corresponding interleaved one.

However, under many other conditions, we can easily construct parallel executions for which no corresponding interleaved execution exists. Assume, e.g., that we allow more than one process to issue messages to the same line, and that we have a setup with one FIFO-line to which two processes $A$ and $B$ may issue messages, and from which a process $C$ receives messages. Further, assume that both $A$ and $B$ have just one execution step, and that the one of $A$, $s_A$, issues two messages $m_A1$ and $m_A2$ to the line, and that the one of $B$, $s_B$, issues one message $m_B$. If $A$ and $B$ run in parallel, process $C$ may obviously receive the message sequence $m_A1, m_B, m_A2$. However, no interleaved execution of the steps $s_A$ and $s_B$ will make $C$ receive the same sequence.

In reported verification efforts, similar steps to those of the first example above (Fig. 1) are used as units for interleaved executions and thereby for verification, see e.g. [Kro78, KrL97, LyK97, Ver87, GaT90]. However, none of these papers contain a more systematic discussion of what types of steps can be allowed, and how large they may be.

Another area where the size of the interleaved steps has an impact is within debugging of distributed systems. Chandy and Lamport presented a method by which a consistent snapshot of a distributed system can be taken [ChL85]. This method suffers from supporting the test of only a very limited set of global state predicates. In [MaN91] Marzullo and Neiger propose an algorithm that from a terminated (or halted) execution allows a server to reason on whether a given predicate is definitely true or possibly true at some state in the execution.

Although both of the above results have been refined and improved since they were first published (see e.g. [GaW96, VeD95, Mal92, Lee99]), they are still severely limited by the number of states they need to consider. The reason is that they implicitly rely on a notion of atomic steps, where each step is allowed to perform at most one input or output of a message from/to a line.

The theme of this paper is to start with the simple case described in Fig. 1, and try to generalise it as far as possible. We will look at questions like: What will happen if the lines are not FIFO? What if more than one process can send to or receive from one given line (as in the second example above)? What if the capacity of the lines to store messages is limited? What about deadlock, which can easily occur in such systems? What type of properties can be proven by this type of reasoning?

In the discussion of these questions we shall refer to the communication lines as ‘channels’, and to the