

ON THE ABSENCE OF LIVELOCKS
IN PARALLEL PROGRAMS

Y.S. Kwong
Unit for Computer Science
McMaster University
Hamilton, Ontario, Canada

Abstract

We explore in this paper the subtle correctness criterion of the absence of livelocks in parallel programs. The basic concepts underlying livelocks are formalized. A classification of livelocks into two types according to their causes of formation is introduced. Two techniques for proving the absence of livelocks are also presented. One is based on the notion of problem reduction; the other is an extension of the well-founded set method for proving termination in sequential programs.

1. Introduction

In this paper we investigate what is probably one of the least explored correctness criteria in asynchronous parallel computation, namely, the absence of livelocks. Deadlock-freedom has long been recognized as an important correctness property in concurrent systems. Informally speaking, a process is said to have a deadlock if there exists a reachable state in which no matter what the system does, it is impossible for the process to be executed any further. However, livelock is a very different notion. It refers to the phenomenon in which a process is never executed along an infinite computation even though it is not deadlocked. Until recently, livelocks have always been discussed informally in the literature as "starvation" or "permanent blocking." As we shall show in later sections, these notions are only special cases of livelocks. A formal investigation is obviously desirable.

First, we review what had appeared in the literature. In [3], Dijkstra proposed a solution to the critical section problem, which guarantees mutual exclusion and if several processes are competing for entry, at least one of them succeeds. The solution is not deadlocked in the sense that it is possible for all processes to gain entry. However, as observed by Knuth [11], some process may have to wait until eternity while others are executing their critical sections. Holt [6] called it "permanent blocking" in his investigation of resource allocation. A very elegant example of livelocks is Dijkstra's Five Dining Philosophers in [4], where he pointed out the danger of

"individual starvation" due to the conspiracy of neighbors. In [1], Ashcroft observed that in an airline reservation system, a booking process may be permanently stopped by a "continuously changing pattern of constraints," and he coined the term "livelock" which is adopted in this paper. He also noted that even with the finite delay property and in the absence of all deadlocks, livelocks may still occur. More recent works include, for example, Lamport's extension of Floyd's assertion method to verifying "liveness" properties [15], and also van Lamsweerde and Sintzoff's treatment of the absence of starvation through fixed point theory [16].

We believe that an important first step towards a better understanding of the topic of livelocks is to put it on a precise setting to facilitate investigations. In this paper we attempt to present a formal treatment. Section 2 describes our model for representing parallel programs. We then address the question: What constitutes a livelock? To start with, we first examine in Section 3 a number of simple examples that lead very naturally to the concepts of the finite delay property, computations, and their validity. Hopefully, they will provide an intuitive understanding of the livelock concept which will then be formalized in Section 4. Also presented is a classification of livelocks in accordance with their causes of formation. In Section 5 we discuss techniques for proving the absence of livelocks.

In a subsequent paper we shall investigate how scheduling disciplines for process queuing would affect livelocks in parallel programs. A parallel program is considered to have two system representations: an abstraction and an implementation. The latter has queuing disciplines being represented explicitly, whereas the former has not. We may envision that abstractions of parallel programs are obtained from their corresponding implementations by reductions or by masking out all details of scheduling. We shall show that the behaviour of livelocks in abstractions and implementations of parallel programs is not always the same, even if some fair scheduling disciplines are employed. The interactions between the two types of livelocks and the crucial role of the so-called "finite-delay property" in livelock investigations will be examined. The implications of our results on verification of livelock-freedom will also be discussed.

2. Model of Parallel Programs

An asynchronous parallel program, or simply a parallel program, is represented graphically by a labeled Petri net, where tokens denote instruction pointers of processes, place nodes (or simply places)