Event Fairness and Non-Interleaving Concurrency

Marta Z. Kwiatkowska
Department of Computing Studies, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom

Key words: Fairness; Concurrency fairness; Models for concurrency; Semantics; Non-interleaving semantics; Partial order semantics

Abstract. Event fairness suitable for non-interleaving concurrency is proposed. Fairness is viewed with respect to concurrency, rather than non-determinism, in the sense that no concurrent component of the system should be delayed indefinitely. Shields' asynchronous transition systems and Mazurkiewicz's traces have been used; the model gives rise to a partial order. A class of generalised notions of (weak, strong and unconditional) event fairness relative to progress requirements is derived. The weakest fairness notion in this class is shown to coincide with maximality with respect to the partial order over traces.

1. Introduction

Sequences of system states are increasingly often used to model the behaviour of discrete systems. For sequential systems, such a sequence forms an admissible execution sequence if it starts with an initial state, and each following state is obtained from its predecessor through an occurrence of some action. In the notation of labelled transition systems the state transition is denoted by \( q \xrightarrow{a} q' \), where \( q, q' \) are states and \( a \) is an action label; each action label determines a class of events that can occur in the system. We are only interested in complete execution sequences, that is those sequences which cannot be extended at the end; we call such sequences non-extendable. So a sequence is non-extendable if it is infinite or if it ends with a state in which no system action can be applied.

When concurrent systems are modelled using state sequences, the situation becomes more complex. The system is a collection of concurrent components,
possibly implemented on different physical processors. Each action occurrence can be local to some concurrent component, or performed under synchronisation with a number of partners. It is important that concurrent components are allowed to make progress wherever possible. In other words, apart from excluding extendable sequences, we would also like to exclude those sequences that ignore some concurrent component of the system. We say such execution sequences are not concurrency fair, and restrict the set of complete admissible executions to those that satisfy fairness constraints.

Fairness has originally been introduced in (sequential) labelled transition systems [LPS81, Plo82], although there have been some attempts to translate it into non-sequential models, for example Petri nets [Bes84a, Bes84b]. While interleaving identifies concurrency with non-deterministic choice, the non-interleaving approach provides means for a clear distinction between these two phenomena; it is precisely this feature that we would like to capitalise on while examining concurrency fairness. Our aim is to obtain a flexible notion of event fairness suitable for non-interleaving concurrency so that sufficient progress of concurrent components of the system is guaranteed. Since interleaving semantics is often considered inadequate when modelling, for example, real time systems and distributed systems, we hope our research will provide some insight into the relationship of fairness and semantic approaches to concurrency.

We emphasise we are interested in only one of the many facets of fairness, namely fairness with respect to concurrency, which guarantees that no concurrent component is delayed indefinitely. To make out intentions clear, let us consider the following, non-terminating, concurrent program:

\[ \texttt{(* (true \rightarrow x := x + 1) \parallel (true \rightarrow y := y - 1))} \tag{1.1} \]

where \( \parallel \) denotes parallel composition. An execution of this program consisting solely of the statement \( "x := x + 1" \) is not concurrency fair because it ignores one of its concurrent components (and neither is any execution comprising only a finite number of statements \( "x := x + 1" \)). The situation remains unchanged if \( "y := y - 1" \) is replaced with \( "x := x - 1" \); since \( x \) is a shared variable, the (atomic) statements \( "x := x + 1" \) and \( "x := x - 1" \) now require synchronisation, but nevertheless it is desirable that their progress be guaranteed. On the other hand, the sequential and non-deterministic program below:

\[ \texttt{(* ((true \rightarrow x := x + 1) \sqcap (true \rightarrow y := y - 1)))} \tag{1.2} \]

where \( \sqcap \) denotes the choice operator, has no concurrent components. Thus, an execution consisting of only a finite number of statements \( "x := x + 1" \) is concurrency fair.\(^1\)

The approach taken here is model-oriented, rather than proof-oriented as in e.g. [Fra86]. Since most models for concurrency have transition structure, we have chosen asynchronous transition systems [Shi85, Bed87, Shi88a, Kwi88a, Kwi88b, Kwi89a, Kwi89b] for our considerations. We have strong reasons to believe that many models determine asynchronous transition systems, which would suggest that the results presented here are not model dependent and can be translated into other settings.

\(^1\) It is possible to define fairness with respect to non-determinism, for example in the sense that every guard that evaluates to true infinitely often should be taken infinitely often (strong guard fairness), but this is not our major concern.