Fairness and Liveness

Thomas Wahl

Abstract

Fairness and Liveness are related notions of describing which of a collection of concurrent processes make progress in their execution. We present three common notions of fairness, and show their impact on the satisfaction of liveness properties by means of an example.

1 Fairness

A *fairness constraint* is imposed on (the scheduler of) the system that it fairly select the process to be executed next. Technically, a fairness constraint is a condition on executions (paths) of the system model. Although these constraints are not properties to be verified (rather, conditions assumed to be enforced by the implementation), they can be expressed in temporal logic. Here are three typical fairness constraints, formulated in LTL:

Absolute Fairness, Impartiality: every process should be executed infinitely often:

$$\forall i : \mathsf{GF}\,\mathrm{ex}_i \ .$$

However, this notion ignores that a process might at certain times not be ready to execute, it may not be *enabled*. The following two notions consider that.

Strong Fairness: every process that is infinitely often enabled should be executed infinitely often in a state where it is enabled:

$$\forall i : (\mathsf{GF}\,\mathrm{en}_i) \Rightarrow (\mathsf{GF}(\mathrm{en}_i \wedge \mathrm{ex}_i)) \, .$$

Weak Fairness: every process that is almost always enabled should be executed infinitely often:

$$\forall i : (\mathsf{FG}\,\mathrm{en}_i) \Rightarrow (\mathsf{GF}\,\mathrm{ex}_i) \, .$$

(Remark: For this notion, ex_i can equivalently be replaced by $(\mathrm{en}_i \wedge \mathrm{ex}_i)$.)

Remark.

$$\text{Absolute Fairness} \ \Rightarrow \ \text{Weak Fairness}, \ \text{Strong Fairness} \ \Rightarrow \ \text{Weak Fairness} \, .$$

This means that if a property is true under the assumption of Weak Fairness, then it is also true under Strong and under Absolute Fairness. However, Absolute Fairness does not imply Strong Fairness, nor vice versa.

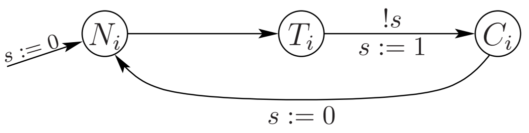
Fairness is not a property to be *checked* of a system. Consider the way asynchronous systems are modeled: as a graph in which each node represents a global system state, and each successor of a node corresponds to the new global state reached by a particular of the processes making a local transition. Such a model usually contains no information about how often a process is executed, or how the next process to be executed is chosen by the scheduler—in fact, the model is independent of the underlying operating system and hence of the scheduler: scheduling is treated nondeterministically.

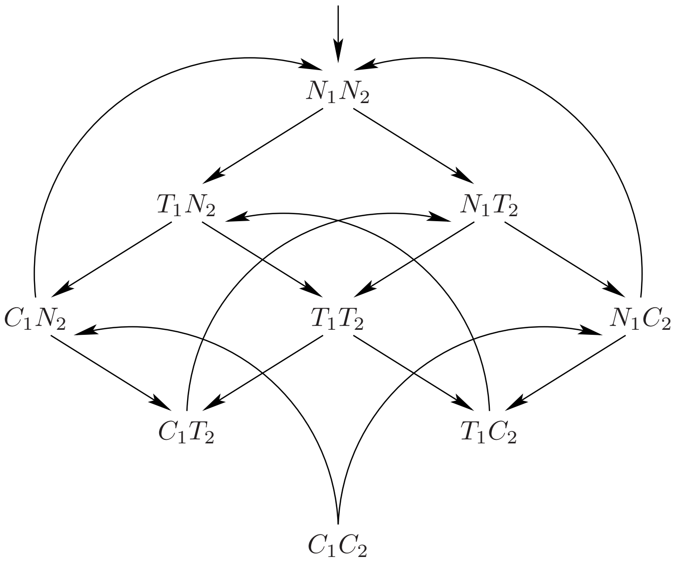
When verifying certain desirable properties, we might find that they are not satisfied by a system as it is described by the model. If, however, the system behaved fairly in its selection of processes to be executed, the property would hold. Since we don't know about the system's fairness, we model check such properties assuming the system behaves fairly.

2 Liveness properties

These properties describe the requirement that a process make progress toward a specific goal, the achievement of which depends on the fairness of the system. If a process never gets to execute, it usually cannot reach its goal. Therefore, these properties are often evaluated only along fair paths, i.e. paths that satisfy one of the above three fairness path requirements.

The following example shows a synchronization skeleton of a semaphore solution to the mutual exclusion problem of concurrent processes, and the induced 2-process Kripke structure (we omit s):





Other than safety, a typical property of interest is whether a process makes progress towards its goal—to enter its critical section—if it so desires, formulated as

$$\forall i : \mathsf{G}(T_i \Rightarrow \mathsf{F}\, C_i).$$

If the processes are selected randomly by the scheduler, this property does not hold. As a counter example, consider two processes and take the global state N_1T_2, which satisfies T_2. The future given by the infinite path $(N_1T_2, T_1T_2, C_1T_2)^\omega$, where always process 1 executes, does not satisfy $\mathsf{F}\, C_2$. The problem is that random scheduling allows unfair execution paths. A sensible implementation of the model will instead have a fair scheduler. It is therefore reasonable to check the property under the assumption of fairness.

If we require Weak Fairness, the weakest of the three fairness constraints (the other two imply it), then the formula still does not hold, with the same counter example. Process 2 is not almost always enabled along the above path (since it is periodically disabled at (C_1, T_2)), so Weak Fairness is vacuously true. — The property that a process is almost always enabled is too strong—it does not hold often in practice, such that the Weak Fairness constraint is often vacuously true and thus not useful.

If we require Absolute Fairness, then the property still does not hold, since the scheduler could choose to execute process 2 whenever in system state (C_1, T_2), in which process 2 is not enabled, so it makes no progress.

However, Strong Fairness says something about executing processes that are infinitely often enabled (as opposed to almost always), and this antecedent is not too strong. In fact, in the system above, each process is infinitely often enabled along *any* path: the states in which process i is not enabled are of the form (T_i, C_j), for some $j \neq i$. In this case, process j makes the next transition, leading to state (T_i, N_j) in which process i is enabled again. – Thus, the precondition of Strong Fairness is satisfied for process i, and the right hand side guarantees that i proceeds to C.