Finitary Fairness in Event-B

Emil Sekerinski and Tian Zhang

Department of Computing and Software, McMaster University, Canada
{emil, zhangt26}@mcmaster.ca

1 Introduction

In the design of concurrent systems, fairness allows to abstract from scheduling policies of in multi-process systems and from processor speeds in multi-processor systems. In Event-B, like in action systems, the choice among events is nondeterministic [1, 2]. Fairness restricts this nondeterministic. In this paper we propose a way to express fairness in Event-B. Finitary fairness has been proposed as a way of further restricting standard fairness [3]. It is a "more realistic" notion of fairness, it allows some systems to be modelled for which standard fairness is not sufficient, and it is more easily used for proving properties in Event-B than standard fairness. We give a general transformation from an Event-B model, in which some events are marked as fair, into an equivalent plain Event-B model. A theoretical justification is given. A similar transformation was proposed in [3], but does not lead to "equivalent" computations. The contribution of this paper is this new transformation.

2 Motivation

Consider the event system in Fig. 1, which is taken from [3]. Both events L and R have no guards and are thus always enabled. Both events are specified to be fair. A *schedule* of an event system is a sequence of names of events that can occur in an execution (which is going to be made precise shortly). A schedule can be a finite or an infinite sequence; in the example all possible schedules are infinite. For example, a schedule could start with:

$$LRRLRLLLRR\ldots$$

Fairness of L implies that a schedule cannot contain an infinite sequence of R's. A schedule is *bounded* if for some natural number k, no fair event is neglected more than k times consecutively. Finitary fairness of an event system means that all schedules are bounded. For the example, a schedule in which the number of consecutive R's continues to increase is not bounded:

$$LRLRRLRRRLRRRRL\ldots$$

Suppose the events belong to different processes. A *scheduler* is an automaton with event names as the alphabet. For above schedule to be generated by an automaton, the automaton would need to count the number of R's and would need an unboundedly large state. Conversely, if the schedule is bounded, only finite state is needed. Thus the

In Abrial, J.-R., Butler, M., Joshi, R., Troubitsyna, E., and Woodcock, J. C. P., Eds. Refinement Based Methods for the Construction of Dependable Systems (2010), no. 09381 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

invariants
$x \in BOOL$
$y \in \mathbb{N}$
initialisation
$x, y := TRUE, 0$
fair event L
$x := NOT\ x$
fair event R
$y := y + 1$

Fig. 1. Event system with two fair events.

bounded schedules are exactly the languages of finite state schedulers. Since any practical scheduler uses a fixed amount of memory, finitary fairness is not only an adequate, but a more precise abstraction from scheduling policies than standard fairness.

Suppose that the events are executed on different processors; the speeds of the processors may differ and may vary. Finitary fairness implies that the speeds of the processors may not drift apart unboundedly. Alur and Henzinger formalize this claim in terms of timed transition systems [3]. Again, finitary fairness allows a more precise abstraction of multiprocessor systems.

Since finitary fairness is more restrictive than standard fairness, one can expect more properties to hold under finitary fairness. For example, the event system of Fig. 1 will eventually reach a state in which $x = TRUE \wedge \neg powerOf2(y)$ holds: if this property would always be false, then L must be scheduled only when $powerOf2(y)$ holds, for increasing values of y, but that is impossible in a bounded schedule.

The finitary restriction can be used for modelling *unknown delays* of timed systems. In a distributed consensus, processes have to agree on a common output value, but each process may fail and not deliver a value. This can be solved using finitary fairness, as shown in [3], but cannot be solved using standard fairness only [4].

Proof rules for the termination of events in presence of fairness can get involved: not all events must decrease the variant. It is sufficient if events that don't decrease the variant keep those fair events that do decrease the variant enabled, as by fairness these will eventually be taken. The proof rule requires that an invariant is specified for each event, e.g. as used in [5] for the refinement of action systems. This would require the proof rules of Event-B to be significantly expanded.

The alternative that we follow is to transform an event system by replacing fair events with regular events and introducing an explicit scheduler [6, 2]. The standard proof rules of Event-B can then be applied. Figure 2 illustrates this. The event system of (a) is supposed to eventually terminate as x is set initially some natural number and fair event R decrements x. In the transformed event system fairness is achieved by introducing a counter c that is decremented each time the (regular) event L is taken. This eventually forces R to be taken as L becomes disabled when c reaches zero. When R is taken, c is set again to a new positive value. In (b) the counter can does not have an upper bound, but still event R will eventually be taken; this ensures standard fairness. In

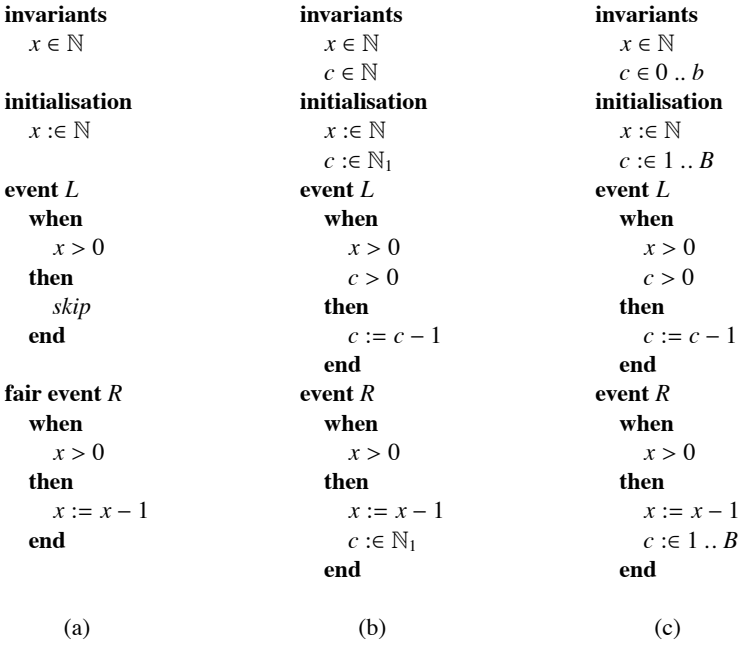

Fig. 2. (a) Event system with fair event R. (b) Counter c is used to ensure finitary fairness of R. (c) Counter c is used to ensure standard fairness of R

(c) this counter be at most B, hence B gives an upper bound of how many times event R can be ignored before it must be taken; this ensures finitary fairness.

A further reason for preferring finitary fairness is that it can simplify proofs of termination. For a set of events to terminate, there must exist a variant, a function from the state to a well-founded domain, and all events have to decrease the variant. For proving the termination of the event system in Fig. 2 (c), following variant with natural numbers as the well-founded domain is sufficient:

$$\textbf{variant}$$
$$x * (B + 1) + c$$

Event L decreases the variant by decreasing c. Event R decreases the variant by decreasing x; while c may increase, as c is at most B, the variant is still decreased. A similar variant cannot be given for the event system in (b). Natural numbers as the well-founded domain are not sufficient with standard fairness.

3 Fair Event Systems

A *fair event system* P is a structure (Q, E, T, I, F) where

- Q is a set of states,

- E is a set of events,
- T is a set of transitions, relations over $Q \times Q$ indexed by E,
- I is the set of initial states, $I \subseteq Q$,
- F is a set of fair events, $F \subseteq E$

We write $T(e)$ for the transition relation of event e. A *computation* p of P is a finite or infinite maximal sequence of states and events alternating, written

$$p = \sigma_0 \xrightarrow{e_0} \sigma_1 \xrightarrow{e_1} \sigma_2 \xrightarrow{e_2} \cdots$$

such that $\sigma_i \in Q, e_i \in E, \sigma_0 \in I$, and $\sigma_i \mapsto \sigma_{i+1} \in T(e_i)$. That is, states σ_i and σ_{i+1} must be in relation $T(e_i)$. A computation is a finite sequence, or is *terminating*, if it ends with a state s_n that is not in the domain of any transition relation, $\forall e \in E \cdot s_n \notin dom(T(e))$. Otherwise it is an infinite sequence, or is *nonterminating*.

The *schedule* of a computation p is the projection of the sequence p to only the events; the *trace* of p is the projection of p to only the states, i.e. for p as above:

$$schedule(p) = e_0 e_1 e_2 \ldots$$
$$trace(p) = \sigma_0 \sigma_1 \sigma_2 \ldots$$

We write $schedule_i(p)$ for e_i, the i-th event of computation p and $trace_i(p)$ for σ_i, the i-th state of computation p. The *guard* of an event is the domain of its relation, $grd(e) = dom(T(e))$; an event is *enabled* in a state if the state is in its guard, otherwise *disabled*. A computation p is *bounded* if it is finite or if for some $k \in \mathbb{N}$, for all fair events $e \in F$, event e cannot be enabled for more than k consecutive states without being taken, formally:

$$\forall i \in \mathbb{N} \cdot \exists j \in i \mathinner{..} i + k \cdot schedule_j(p) = e \vee trace_j(p) \notin grd(e)$$

When considering finitary fairness, we are interested only in the bounded computations. This definition of fair event systems generalizes that of transitions systems in [3] by indexing the transitions with the events and by allowing only some events to be fair.

An Event-B model defines the set of states through the variables and invariants, the transition relations through guards and generalized substitutions, and the initial states through the initialization. Thus fair event systems are an abstract representation of Event-B models, in which we additionally allow some events to be specified as fair.

4 The Finitary Weakly Fair Transformation

Let $P = (Q, E, T, I, F)$ be a fair event system. We assume that $E = \{e_1, \ldots, e_n\}$ and that F is the subset $\{e_1, \ldots, e_m\}$ with $m \leq n$. The *finitary weakly fair transformation* $FWF(P)$ ensures finitary fairness by introducing counter variables $c_1, \ldots, c_m$, one for each fair event. The counters indicate the priority of events. Once the counter of an event reaches zero, that event must be tested: if it is enabled, it must be taken, otherwise it is skipped. The counters are kept distinct, therefore only one counter can be zero. The counters are initialized to values between 1 and B. On every transition, the guards of all fair events must be tested: if an event is enabled, its counter must be decreased, otherwise its counter is reset to a value between 1 and B. Formally, $FWF(P) = (Q', E, T', I', \varnothing)$ where for some $B \geq m$:

- $Q' = Q \times \mathbb{N}^m$
- For every event $e_i \in E, (\sigma, c_1, \ldots, c_m) \mapsto (\sigma', c'_1, \ldots, c'_m) \in T'(e_i)$ if:
  1. if e_i is a regular event, $e_i \in E - F$, then
     $\sigma \mapsto \sigma' \in T(e_i) \wedge$
     $(\wedge j \in 1 \,..\, m \cdot c_j > 0 \wedge ((\sigma \in grd(e_j) \wedge c'_j = c_j - 1) \vee (\sigma \notin grd(e_j) \wedge c'_j \in 1 \,..\, B)))$
  2. if e_i is a fair event, $e_i \in F$, then
     $\sigma \mapsto \sigma' \in T(e_i) \wedge$
     $(\wedge j \in 1..m - \{i\} \cdot c_j > 0 \wedge ((\sigma \in grd(e_j) \wedge c'_j = c_j - 1) \vee (\sigma \notin grd(e_j) \wedge c'_j \in 1..B)))$
     $\vee$
     $(c_i = 0 \wedge \sigma \notin grd(e_i) \wedge \sigma' = \sigma \wedge c'_i \in 1 \,..\, B)$
  3. $distinct(c'_1, \ldots, c'_n)$
- I' is such that $(\sigma, c_1, \ldots, c_n) \in I'$ if
  1. $\sigma \in I \wedge (\wedge j \in 1 \,..\, m \cdot c_j \in 1 \,..\, B)$
  2. $distinct(c_1, \ldots, c_n)$

All counters of the finitary fair transformation are between 0 and B and are distinct, i.e. for all computations p of $FWF(P)$ and for all natural numbers i with $0 \leq i < |trace(p)|$:

$$trace_i(p) = (\sigma, c_1, \ldots, c_n) \Rightarrow c_1 \in 0 \,..\, B \wedge \ldots \wedge c_n \in 0 \,..\, B \wedge distinct(c_1, \ldots, c_n) \quad (1)$$

This property follows by induction over i: with $FWF(P) = (Q', E, T', I', \varnothing)$ the initial states I' satisfy (1) and transitions T' preserve (1). In the transformation of fair events a case analysis is needed: when the counter of an event reaches zero, the event's transition is take if enabled, otherwise not. This case analysis leads to splitting a fair event E into E and E' in the transformed system. For a fair event system P, we call the schedules of the computations of P simply the schedules of P and the traces of computations of P simply the traces of P. The schedules of P and $FWF(P)$ are necessarily different, as $FWF(P)$ contains the auxiliary primed events. The *restriction* of a sequence s onto a set S is the subsequence of s containing only elements of S. Following theorem justifies the finitary fair transformation.

Theorem 1. *For a fair event system P, the schedules of $FWF(P)$ restricted to the events of P are exactly the bounded schedules of P.*

References

1. Métayer, C., Abrial, J.R., Voisin, L.: Event-B Language, in RODIN Project Deliverable 3.2. (2005)
2. Back, R.J.R.: Refinement calculus, part ii: Parallel and reactive programs. In deBakker, J.W., deRoever, W.P., Rozenberg, G., eds.: REX Workshop on Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness. Lecture Notes in Computer Science 430, Mook, The Netherlands, Springer Verlag (1989) 67–93
3. Alur, R., Henzinger, T.A.: Finitary fairness. ACM Trans. Program. Lang. Syst. **20**(6) (1998) 1171–1194
4. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. Journal of the ACM **32** (1985) 374–382
5. Back, R., Xu, Q.: Refinement of fair action systems. Acta Informatica **35**(2) (1998) 131–165
6. Apt, K.R., Olderog, E.R.: Proof rules and transformations dealing with fairness. Sci. Comput. Program. **3**(1) (1983) 65–100