APPRAISING FAIRNESS IN LANGUAGES FOR DISTRIBUTED PROGRAMMING

by

Krzysztof R. Apt, Nissim Francez and Shmuel Katz

Received 10/22/86

Abstract: The relations among various languages and models for distributed computation and various possible definitions of fairness are considered. Natural semantic criteria are presented which an acceptable notion of fairness should satisfy. These are then used to demonstrate differences among the basic models, the added power of the fairness notion, and the sensitivity of the fairness notion to irrelevant semantic interleavings of independent operations. These results are used to show that from the considerable variety of commonly used possibilities, only strong process fairness is appropriate for *CSP* if these criteria are adopted. We also show that under these criteria, none of the commonly used notions of fairness are fully acceptable for a model with an n-way synchronization mechanism. Finally, the notion of fairness most often mentioned for Ada is shown to be fully acceptable.

Authors' addresses:

Krzysztof R. Apt

LITP, Universite Paris 7, 2, Place Jussieu, 75251 Paris, France

and

Laboratoire d'Informatique, Ecole Normale Superieure, 45, Rue d'Ulm, 75230 Paris, France

Nissim Francez, Shmuel Katz
Department of Computer Science
The Technion- Israel Institute of Technology
Haifa, Israel
francez@techsel.bitnet, katz@techsel.bitnet

1. Introduction

Fairness assumptions in selecting among a collection of possible operations in a nondeterministic or parallel program have received considerable attention in recent years. The semantic implications of a wide variety of possible definitions have been investigated, often by developing proof rules for showing termination assuming a particular definition (see [Fr] for an overview, examples, and further references). For nondeterministic programs some of the versions include weak fairness (also called justice), strong fairness, equifairness, and extreme fairness. For *CSP* [H] and other models for distributed computing, at least six reasonable variants have been defined and investigated. This wide variety of possibilities leads to a confusing situation: selection of a particular definition of fairness for any particular model or language relies almost exclusively on subjective, implicit criteria.

We suggest three simple semantic criteria which can aid in determining which notions are appropriate for which computational model. The criteria we propose are termed *feasibility, equivalence robustness*, and *liveness enhancement*. Below we informally explain the criteria and the results linking the criteria and the models. In subsequent sections the formal definitions are presented, and the theorems and proofs which lead to these results are presented.

Feasibility:
Any definition of fairness excludes some of the executions (the "unfair" ones) which otherwise would be legal executions of a program according to the basic semantics of the computational model. A necessary requirement of any definition of fairness for a computational model is to have some legal computation remain after this exclusion, for every possible program. That is, for every legal program some (finite or infinite) fair computation does exist. Without this requirement, no scheduler--which must produce one of the fair computations-- could correctly treat the fairness. Moreover, since any reasonable scheduler cannot 'predict' the possible continuations at each point of the computation, it should be possible to extend a partial computation to a fair one. This will be the feasibility criterion, and it subsumes the above necessary requirement.

As a simple example of an unfeasible definition of fairness for *guarded commands (GC)* [D], consider the following fairness definition:
all choices (referred to as directions) which are infinitely often possible must eventually be chosen equally often.

In Figure 1 a nonterminating program P is shown, for which there is no computation sequence satisfying the above definition, even though both directions are infinitely often possible. Thus no scheduler can be devised, and the fairness notion is not feasible for that model. In fact, feasible definitions of such equifairness must incorporate the set of choices which are jointly enabled at each stage, as in [GFK1].

Equivalence Robustness:
The computations of asynchronous, distributed systems are often modeled by means of interleaving the (atomic) actions of their component processes. However, it is clear that the order of execution of independent actions in such an interleaving is arbitrary. Thus two linear execution sequences which are identical up to the order of two independent actions, should be equivalent. This leads to the second criterion: a definition of fairness is *equivalence robust* (for a computational model) if it respects this equivalence. That is, for two infinite sequences which differ by a possibly infinite number of interchanges of independent actions (i.e., equivalent sequences), either both are fair according to the given definition, or both are unfair. Again, a more precise definition and an examination of the consequences of this criterion are given in the following sections.

Liveness Enhancement:
All distributed models assume a fundamental liveness property that an action will eventually be executed in some process if the system is not deadlocked. A justification for adding an additional liveness requirement of some sort of fairness -- which complicates the scheduling and may cause difficulties in defining a precise semantics or proving correctness-- is that there exists a program which has some liveness property which it would not have without the additional requirement. This criterion is termed *liveness enhancement* in order to emphasize that additional liveness assumptions will hold for some programs. As is shown in the sequel, this also depends on the particular model being considered, and is sensitive to fine details of the model. Some fairness assumptions cannot force a communication to occur in a model if it did not have to occur under the basic liveness property. These assumptions are not liveness enhancing for that model.

It is sufficient to consider the impact of fairness assumptions on termination only, because such assumptions are known not to affect partial correctness or safety properties, and other liveness properties can be reduced to termination for derived programs [GFMdR].

$P :: x := 1;\ *[\ true \rightarrow x := x+1$

$\quad [] \ x \ mod \ 3 = 0 \rightarrow x := x+1\ ].$

Figure 1

In the sequel, we appraise several fairness definitions and computational models under the criteria suggested above. These are only examples of the application of our approach. Readers are invited to apply the same, or their otherwise preferred, criteria to their favorite fairness definitions and computational models.

2. Formal definitions

As mentioned, an operational semantics of the models we consider may be expressed in terms of the possible interleaved computations of a program, i.e., as sequences of states (recording the values of all the variables) and the atomic actions leading from state to state. A *configuration* is a pair consisting of an action and a state. Each computation can be viewed as an initial state followed by a sequence of configurations. An atomic action is either a local action of one process, or a joint action when the communication/synchronization mechanism is involved (except when sending is a local action as in a nonblocking send model). An action is *enabled* (in a configuration) if it can serve as the next action executed (where the exact definition is model dependent).

We shall only consider models in which every joint action is followed by a local action (possibly a *skip*, having no effect on the state), and in which a choice between a local action and a joint action is never possible. These restrictions guarantee that the definitions of fairness considered here are immune to additions of local actions, like *skip*, in processes. In the terminology of [L2] we might say that these definitions are immune to *stuttering*, i.e., to repetitions of a configuration in a computation. The relevance of these assumptions is further discussed later.

Examples of the above-mentioned approach to defining semantics may be seen in [P] for *CSP*, and in [HLP] for a fragment of Ada. However, it is also reasonable, and even attractive to consider a *partial order* semantics (see [L1], [R], or [DM]) expressing only the essential causal relationships among the atomic actions (both local and joint). Clearly, for every such partial order, there is a uniquely defined equivalence over interleaved computations: if π and ρ are interleaved computations, then $\pi \equiv \rho$ is defined as: π can be obtained from ρ by (possibly infinitely many) simultaneous transpositions of two independent (i.e., not related by the partial order) atomic actions .

Our initial liveness assumption, referred to as the *minimal progress property* [OL], is somewhat stronger than the fundamental liveness property mentioned in the introduction. According to this assumption, every enabled local action is eventually executed. In the sequel, all computations are always assumed to satisfy the minimal progress property. Thus, the focus of additional fairness assumptions is on interprocess communication.

Given a (distributed) program P, comp(P) is the set of interleaved computations generated by P under the basic semantics of the model. A *fairness definition* D is a mapping from programs to sets of computations such that for each program P, D(P) is a subset of comp(P), and includes all finite computations in comp(P). A necessary condition for feasibility of D is that for all programs P, D(P) is nonempty. As already explained, in order to prevent a scheduler from "painting itself into a corner" with no possible continuation, we define
D is *feasible* iff every finite initial segment of an interleaved computation of a program P can be extended to a computation in D(P).

The other criteria are defined by:
D is *equivalence robust* iff for every two computations π and ρ in comp(P),
$(\pi \in D(P) \wedge \pi \equiv \rho) \rightarrow \rho \in D(P)$.

D is *liveness enhancing* if there is a program P such that comp(P) contains an infinite computation, but all computations in D(P) are finite.

This definition means that P terminates under the assumption of **D**. Because of the possible reduction of liveness properties to termination of a derived program, this is sufficient to express general liveness enhancement.

By a *projection* of a computation π on a process p, denoted by $[\pi]_p$, we mean the result of deleting from π all actions in which p is not involved and restricting the states to variables of p only.

The following simple lemma will be useful in the sequel.

Lemma: (projection equality)
if $\pi \equiv \rho$, then for each process p, $[\pi]_p = [\rho]_p$.
Note: the converse of this lemma was proved by L. Bouge (private communication) for *CSP* programs. We do not need this stronger version.

3. Results for CSP

In this section the results concerning the *CSP* model are stated. We consider the language as defined in [H] except that nested parallelism and the distributed termination convention are disallowed, and output commands may appear in guards. Moreover, the restrictions mentioned in the previous section are also imposed. The semantics we consider is that of interleaved computation sequences as defined in [P]. Note that by this semantics, the execution of a communication in a guard of a repetition statement results in a configuration in which the control of the process is on the right hand side of the guard, and the

communications in the guards of the repetition are not enabled in the resulting state.

In a nondeterministic guarded commands program, it is usual to define fairness as some condition about the selection of the guards to be executed. In distributed models, more possibilities are open. Fairness could be defined over the processes, over the joint communication actions, or over some group of joint communication actions. In the context of CSP, it is reasonable to define fairness so as to guarantee that an action will be taken by each process which satisfies some condition, or that each communication satisfying a condition will occur, or that one communication will occur from each group of communications between two processes which also satisfy the condition.

Once it has been decided what is to be fair, the condition for demanding an eventual choice must be determined. Two well-known possibilities for CSP, (originating from the nondeterministic case [LPS]) are *weak* fairness in which the choice must be possible continuously from some point on, or *strong* fairness in which the choice is possible infinitely often. Thus taking all of the permutations, six notions are obtained.

Strong Process (SP) fairness: an infinite computation is fair iff each process infinitely often capable of executing an atomic action will infinitely often do so.
Strong Channel (SCh) fairness: an infinite computation is fair iff each pair of processes infinitely often capable of communication with each other do infinitely often communicate with each other (so that one of the possible communications between them is executed, possibly a different one every time).
Strong Communication (SCo) fairness: an infinite computation is fair iff each pair of input/output commands (i.e., each specific possibility of communication) which is infinitely often jointly enabled is executed infinitely often .

The weak versions, *WP, WCh, WCo,* respectively, are obtained by substituting "continuously from some point on" for the first occurrence of "infinitely often".
Furthermore, it is stipulated that all finite computations are fair w.r.t *all* fairness definitions.

The consequences of the following propositions are that although all six possibilities are feasible, only Strong Process fairness is both equivalence robust and liveness enhancing for CSP: all types of Weak fairness are not liveness enhancing, and Strong Communication or Channel fairness are not equivalence robust.

In [FdR] and [KdR] six related fairness definitions are considered and compared in terms of "strength" in causing termination. Those definitions differ in that the channel level is replaced by a level dealing with a mixture of joint and local actions, the restrictions we impose are not applicable, and weak fairness is defined differently. Nevertheless, using arguments similar to theirs, the following implications can be shown to hold (and the proof will not be given here):

Theorem: (CSP-hierarchy)

$$\begin{array}{ccc} WP & \rightarrow & SP \\ \downarrow & & \downarrow \\ WCh & \rightarrow & SCh \\ \downarrow & & \downarrow \\ WCo & \rightarrow & SCo \end{array}$$

We add the following results:
Proposition 1: the six definitions of fairness for CSP are all feasible for the model.
Proof idea: For each definition an explicit scheduler is exhibited and it is shown that any prefix of a legal computation can be generated by the scheduler. Moreover, if a prefix of a computation was generated by the scheduler, then the scheduler will generate a continuation which satisfies the condition for being in D. This idea has been used implicitly in [AO] and explicitly in [OA].

As an illustration of this technique, consider Strong Communication fairness. Given a CSP program P, associate with each of the atomic actions of P a distinct variable, called a *priority variable*. The scheduler can be viewed as a program executed in parallel to P, having access to all variables in P for inspection. It can also determine the control locations of all processes in P. The scheduler interacts with P by executing the program section $SELECT$, which determines the next action in the computation of P. After the execution of the selected action by P, the scheduler regains control, unless P has terminated or entered a deadlocked configuration.

All priority variables are initialized to arbitrary nonnegative integer values. The program $SELECT$ is displayed in Figure 2.

Because of the use of random assignment and possible nonuniqueness of the minimal priority variable, the scheduler itself is nondeterministic. The following *faithfulness theorems* hold, whose proofs are variants on those in [AO] and in [Fr, ch. 3], and of

---

for each atomic action *do*
    if it is enabled *then* decrement priority by 1;
select for execution an enabled action with a minimal value for its priority variable;
reset the priority variable of the selected action to an arbitrary nonnegative integer value

---

Figure 2: $SELECT$

more abstract results in [OA].

Theorem: (Faithfulness)
1. Every computation of P generated by the scheduler is SCF.
2. Every SCF communication of P can be generated by the scheduler.

Proposition 2: Only Weak Communication, Weak Channel, and Strong Process fairness (from the six possibilities) are equivalence robust for CSP.
Proof idea: We show that Weak Process fairness is not equivalence robust by exhibiting two interleaving computations for a program (Figure 3), a variant of the Dining Philosophers, with five cyclically arranged processes, each able to communicate with its immediate neighbors. The two computations are equivalent but one is Weak Process fair and the other is not. This occurs because in one computation the middle process (i.e., p_2) could communicate in every state with at least one of its neighbors, but does not, leading to an unfair computation, while in the other, there are an infinite number of states in which the middle process cannot communicate or otherwise advance at all, because both partners are communicating elsewhere. Thus in the second computation the middle process' noncommunication does not violate the weak fairness condition.

The first computation consists of an indefinite repetition of the following finite segment:
1) p_0 and p_1 communicate.

$P :: [p_0 || \cdots || p_4]$

where

$p_i :: l_i := true\ ;\ r_i := false\ ;$

$*[p_{i-1}?l_i \rightarrow$

$\quad [l_i \wedge r_i \rightarrow eat [] \neg(l_i \wedge r_i) \rightarrow skip]$

$[] p_{i+1}?r_i \rightarrow$

$\quad [l_i \wedge r_i \rightarrow eat [] \neg(l_i \wedge r_i) \rightarrow skip]$

$[] l_i ; p_{i-1}!true \rightarrow l_i := false$

$[] r_i ; p_{i+1}!true \rightarrow r_i := false$

$]$.

Figure 3

2) p_0 executes its local action.
3) p_1 executes its local action.
4) p_3 and p_4 communicate.
5) p_3 executes its local action.
6) p_4 executes its local action.

The second computation consists of the indefinite repetition of the finite segment in which the same events take place in the order 1), 4), 2), 3), 5), 6). Here, p_2 is not enabled after step 4), where all its partners "passed the arrow" and are unavailable for communication. The whole computation is thus rendered Weak Process fair.

Similar examples may be constructed for SCh and SCo fairness.

It is easiest to show that SP fairness is equivalence robust for CSP by considering the unfair computations. If π is Strong Process unfair, then from some point on there is a process p which is infinitely often enabled for a joint action but is never executed. Thus p is *continuously* available for the communication, since it does nothing else. Here the restriction to a model where local actions are not nondeterministic alternatives to communications is essential. Now consider any equivalent computation ρ. By the projection equality lemma, in this computation as well, from some point on p is continuously available for a joint action. Again, by the same lemma, there are infinitely many states in which the possible partner of p could have communicated with p. Thus in this case also, ρ is SP unfair. WCo and WCh fairness may be treated similarly.

Proposition 3: Only Strong Communication, Strong Channel, and Strong Process fairness are liveness enhancing for CSP.
Proof (fragment): We show that Weak Process fairness does not enhance liveness for CSP. By similar (but simpler) reasoning it may be shown that WCh and WCo also do not. For this task we need to demonstrate that if there is any infinite interleaved computation π for a program P, there is also an infinite WP fair computation of P, so that the fairness assumption does not allow proving termination of additional programs. Obviously, if π is WP fair, we are done. Otherwise, let A be the set of processes which are activated in π only finitely often. Now a new computation ρ will be constructed from π. The computation ρ will be identical to π up to the point where all the processes in A have executed all of their actions. Then we insert between every two configurations of computation π the configuration resulting from an activation from each process of A in an action not involving a process from outside A, whenever possible. The resulting computation can still be WP unfair as some process p from A can, from some point onwards, continuously be ready to communicate only with processes not in A. To

handle this situation we first introduce a number of notions.

Given a computation and a collection B of processes, call a process p B-enabled if, from some point onward, it can continuously communicate with a process in B. By a *chunk* of a computation we mean a fragment consisting of an execution of a sequence of local actions belonging to a pair of processes, together with a communication between these two processes. A process is *mute* in a configuration c in a computation if it does not participate in any communication after c. A state is *good* (in some computation) if it either is an initial state of a chunk, or it results from an action in a mute process.

Lemma: (disabling)
Consider a computation π in which all processes in a collection B are infinitely often activated. There exists an equivalent computation ρ, in which no process is B-enabled.
Proof: For each process in turn defer its local actions in π maximally. In such a way, an equivalent computation ρ is obtained, which consists of a sequence of chunks, possibly interleaved with actions from mute processes. This computation has infinitely many good states. Consider any good state in which each process from B was activated at least once. In such a state, the control in each process in B is either just after the communication belonging to its most recently executed chunk, or just after a local action in case it is mute. In both cases (by the restrictions imposed above and by the definition of a mute process) none of the processes in B can communicate in the considered state. This establishes the claim and thereby the proposition.

As a consequence of the proposition, the classes of terminating programs for all three weak levels coincide, in contrast to the proper inclusion shown in [KdR]. The difference seems to be due to the fact that their notion of "Weak" still involves an element of "infinitely often" enabled. Ours stresses that "continuously" enabled really means that nothing else is done by the process involved.

Note: (L. Bouge) the restriction that every joint action is immediately followed by a local action is crucial here. In order to see its role, consider the program in Figure 4, and its computation in which p_1 and p_2 communicate only with p_4. Then, p_3 is continuously capable of communication with p_1 or with p_2, because according to the CSP semantics in [P], passing from the end of the body of a loop to the beginning of the loop is instantaneous. This computation is equivalent only to itself, so the disabling lemma no longer holds. In fact, proposition 3 itself does not hold any more either. In order to obtain a program which terminates under the assumption of WP fairness, it suffices to modify the above program, so that a communication between p_3 and p_1 or p_2 triggers the termination of all processes.

$$P :: [p_1 \,||\, p_2 \,||\, p_3 \,||\, p_4]$$

where

$$p_1 :: *[p_3!0 \rightarrow skip \;[]\; p_4!0 \rightarrow p_4!0]$$

$$p_2 :: *[p_3!0 \rightarrow skip \;[]\; p_4!0 \rightarrow p_4!0]$$

$$p_3 :: *[p_1?x \rightarrow skip \;[]\; p_2?x \rightarrow skip\,]$$

$$p_4 :: *[true \rightarrow p_1?y\,;p_1?y\,;p_2?y\,;p_2?y\,]$$

Figure 4

Finally, to show that Strong Process fairness enhances liveness for CSP, we refer to the program in [Fr, Figure 5.1]. In that program, two processes are engaged in an indefinite "chattering", terminated only by the intervention of a third process, which is necessarily activated if SP fairness is assumed. The program does not terminate without a fairness assumption. SCh and SCo are then also liveness enhancing for CSP due to the hierarchy theorem.

4. Results for N-way Communication

An N-way communication (considered in [BK-S1], [RS] or [Fo]) is a *joint action* executed simultaneously by a number of processes (possibly more than two), each of which must be available in order for the communication to take place. The attempt to participate in a joint action *delays* a process until all other parties are available. After the communication, a local action takes place in each participating process.

Thus, we consider a language with a structure similar to CSP. The guards constitute a reference to a joint action, possibly preceded with a local boolean condition. The guarded statement is a multiple assignment, specifying the local change of state in each participating process. A computation is an interleaving of atomic (either local or joint) actions.

The definitions of fairness we consider are over the individual processes, over the communications, and additionally (as a generalization of channel fairness from CSP) over the collection of actions possible among a group of processes. The definition is:
Strong Group (SG) fairness: an infinite computation is fair iff each set of processes infinitely often capable of communication will infinitely often communicate.
Weak Group (WG) fairness is defined analogously.

The following theorem has been (essentially) established in [BK-S2].

Theorem 2: (**N-way hierarchy**) the implications of the CSP hierarchy theorem hold for the N-way synchronization model, when SG and WG are substituted for SCh and WCh, respectively.

Proposition 4: the six fairness definitions are feasible for an N-way communication model.

Proof idea: analogous to the proof of proposition 1. As an example, we consider a scheduler for WG fairness. Given a distributed program P in this model, associate with each group of processes that (syntactically) can all participate in some joint action (referred to as an *action group*) and with each local action a distinct priority variable. The program section $SELECT$ is obtained in a similar way to the CSP case. We skip the presentation of its code.

Also, a similar *faithfulness theorem* is provable, expressing the fact that all and only WG fair computations are generated by this scheduler.

Proposition 5: Only WCo and WG fairness are equivalence robust for an N-way communication model.

Proof idea: in particular, unlike the CSP model, Strong Process fairness is not equivalence robust. To see this, consider the following program (Figure 5). Here joint actions are denoted by the set of participating processes and "abstract" assignments (M_i), as the example is independent of the actual communications. All boolean guards are identically true and omitted. Subscripted occurrences of L denote local actions. Again, the example is independent of the details of all these actions.

Consider the infinite computation of P which repeats the following cycle:

1) The joint action a_2 is executed.
2) p_3 locally executes $L_{3,1}$.
3) p_2 locally executes $L_{2,2}$.
4) The joint action a_3 is executed.
5) p_3 locally executes $L_{3,2}$.
6) p_4 locally executes $L_{4,2}$.

In this computation, P_1 is infinitely often enabled to participate in a joint action (after steps 3) and 6)), but never does so. Thus, this computation is not Strong Process fair.

On the other hand, an equivalent computation in which the above steps are executed in the order 1), and the cycle on 2), 4), 3), 5), 1), 6)
is Strong Process fair, because p_1 is never enabled in it. Specifically, in order to execute the joint action a_1, the processes p_1, p_2 and p_4 must all be jointly available. However, in no state in this computation are both p_2 and p_4 available.

$P :: [p_1 \,||\, p_2 \,||\, p_3 \,||\, p_4]$

where

$a_1 :: (p_4, p_1, p_2) : M_1$

$a_2 :: (p_2, p_3) : M_2$

$a_3 :: (p_3, p_4) : M_3$

and

$p_1 :: *[a_1 \rightarrow L_1]$

$p_2 :: *[a_2 \rightarrow L_{2,1}$
$\quad []a_2 \rightarrow L_{2,2}]$

$p_3 :: *[a_2 \rightarrow L_{3,1}$
$\quad []a_3 \rightarrow L_{3,2}]$

$p_4 :: *[a_1 \rightarrow L_{4,1}$
$\quad []a_3 \rightarrow L_{4,2}]$

Figure 5

The desired effect is obtained by delaying local actions, preventing process availability and thereby disabling joint actions.

Using arguments similar to those in the proof of Proposition 2 we now show that WG is equivalence robust. The proof for WCo is analogous. Consider a computation π which is WG unfair. Then, from some point on an action group can continuously execute a joint action. Thus, from some point on all processes in that group are never activated. If ρ is an equivalent computation, then by the projection equality lemma the same holds for ρ. By the same lemma, all processes in the above-mentioned action group can continuously participate in that same joint action. So, ρ is WG unfair as well.

Proposition 6: Only SCo, SG, and SP are liveness enhancing for an N-way communication model.

Proof : Since CSP programs are special cases of programs with N-way communications, by Proposition 3, the three methods above are liveness enhancing. The argument for the negative results is also similar to the one in Proposition 3. In fact, it is enough to redefine the notions of *chunk* and B-enabled for the N-way model, and the proof goes through. We omit the details.

From the above results, it follows that none of the six definitions of fairness satisfy all three of the criteria for this model.

5. Results for generalized Ada

In this section we consider the generalization of the process queues from the Ada definition to a fairness notion defined in [PdR]. They show that the generalization has equivalent power to the queueing strategy, but is less restrictive. We demonstrate that it is an acceptable notion of fairness for the Ada model, according to all three criteria. The propositions and proofs have a general structure analogous to the previous sections.

The sublanguage considered, *ACF* (*Ada* communication fragment), contains the essentials of the tasking together with a minimal sequential structure within tasks. An *ACF* program contains a fixed number of disjoint processes without *any* sharing of variables. Each process has a number of declared *entries*. A process may execute assignment and use usual branching and repetition. In addition, it may *call* an entry in another process, *accept* an entry-call from another process or *select* one of several alternative entry-call acceptances.

According to the operational semantics of *ACF* presented in [PdR], the joint actions are the engagement in a rendezvous and the termination of a rendezvous, both involving parameter copying. A *computation* is once again an interleaving of atomic actions. The local actions are assumed to satisfy the minimal progress property mentioned before.

The fairness notion suggested in [PdR] for *ACF* is the following: a computation π is *fair* if no process may wait forever on an entry-call to an entry e while infinitely many entry-calls for e are accepted in π. This notion does not exactly fall into any of the categories of fairness previously mentioned. We refer to it as *Entry fairness*.

The main theorem in [PdR] states, that for programs which do not refer to attributes of the explicit entry queues (present in the original *Ada*), the class of fair computations coincides with the class of admissible computations by the original queueing requirements.

As the usage of the entry queues can serve as a scheduler for the entry-calls, we immediately obtain
Proposition 7: *Entry fairness* is feasible for the *ACF* model.

In order to show the equivalence robustness, note that the above definition of fairness relates only to processes which are waiting continuously on an entry-call. That is, the continuous availability of the calling process p for a rendezvous is built into the definition. Thus the restriction that local actions cannot be alternatives to communication actions (used in Proposition 2 to establish the continuous availability of one side of a *CSP* communication) is not imposed here. In the case of conditional entry-calls, the fairness assumption applies only after a commitment to the entry-call is made and the local action is not taken. (In terms of the operational semantics of [PdR], this commitment is made when a rendezvous transition occurs.)

Proposition 8: *Entry fairness* is equivalence robust for the *ACF* model.
The proof uses the same argument as that for SP fairness in Proposition 2, since the persistence of entry-calls is now given.

Proposition 9: *Entry fairness* is liveness enhancing for the *ACF* model.
Proof: Consider the program seen in Figure 6. Without fairness, the rendezvous between p_1 and p_2 need never occur, and the program will not terminate. With Entry fairness, termination is guaranteed (b will become false, and the second *accept* will only be possible with p_3, causing c to also become false).

In passing, we note that *ACF* already has *unbounded nondeterminism* without additional fairness assumptions. Thus, merely exhibiting a program that implements random assignments using fairness does not suffice to prove Proposition 9.

6. Conclusions

Specific instances of results similar to the ones here have been pointed out elsewhere, as disturbing anomalies. The fact that Weak Process fairness is not equivalence robust for the *CCS* model was indicated to us by Gerardo Costa. In [BK-S2] the lack of equivalence robustness for fairness in the N-way communication model is noted (of course using different terminology). As a solution, they suggest semantic assertions about the computations which are sufficient to guarantee equivalence robustness for the subclass of programs which satisfy the assertions. A similar approach concerning Strong Communication fairness for *CSP* is undertaken in [GFK2]. Unfortunately, it is difficult both to prove whether a program satisfies the assertions, and to understand the implications of a program with such a semantic definition.

We have shown that for the *CSP* and *Ada* models, an alternative approach is viable: to evaluate the fairness notions more carefully to find one which is feasible, inherently equivalence robust, and yet liveness enhancing. We also found that for a model with a non-blocking *send* a fairness notion exists, satisfying all three criteria. As the details are very similar to the *Ada* case, we did not present them here. It is not clear whether such a fairness definition can be devised for the N-way communication model. In general, the idea of defining criteria, and then systematically evaluating the potential definitions of fairness for the

$P :: [p_1 \,||\, p_2 \,||\, p_3]$

where

$p_1 :: p_2.e(false, y)$.

$p_2 :: x := true$;

while x *do*

accept $e(z, x) \rightarrow x := z$;

if $\neg x$ *then accept* $e(z, x) \rightarrow x := false$.

$p_3 :: w := true$;

while w *do* $p_2.e(true, w)$.

Figure 6

computational model according to those criteria, clarifies the advantages and drawbacks of the alternatives, and should be useful in language design.

While working on these results, we have noted that yet another natural equivalence relation among *CSP*-like programs, underlying the transformation to *normal form* of such programs [AC], is also not respected by fairness. The original program and its normal form differ, for example, w.r.t the restriction of a local action immediately following every communication. One can not play some of the tricks we did here, if communication need not be confined to (top level) guard positions. A more extended version of this paper will elaborate on this issue.

It would be interesting to obtain characterization theorems, that for each notion of fairness characterize the equivalences respecting that fairness, and vice versa, for each equivalence relation, characterize the fairness notions respecting it.

Acknowledgements

We thank Luc Bouge for helpful comments and discussions on the subject of this paper. The work reported was carried out during a visit of the first author in the CS dept., Technion. The part of the second author was partially supported by the fund for the promotion of research, the Technion.

References

(1)
[AO] K.R. Apt, E.-R. Olderog, Proof rules and transformations dealing with fairness, SCP 3, pp. 65-100, 1983.

(2)
[AC] K.R. Apt, Ph. Clermont, Two normal form theorems for CSP programs. IBM TJ Watson research Center RC 10975, July 1985

(3)
[BK-S1] R.J. Back and K. Kurki-Suonio, Decentralization of process nets with centralized control, Proceedings of 2nd ACM PODC, Montreal, August 1983.

(4)
[BK-S2] R.J. Back and K. Kurki-Suonio, Serializability in distributed systems with handshaking, CMU TR 85-109, 1985.

(5)
[DM] P. Degano and U. Montanari, Concurrent histories, a basis for observing distributed systems, to appear in JCSS.

(6)
[Fo] I. Forman, On the design of large distributed systems, Proceedings of International Conference on Computer Languages, Miami Beach, Florida, October, 1986.

(7)
[Fr] N. Francez, *Fairness*, Texts and monographs in computer science series (D. Gries, ed.), Springer-Verlag, New York, 1986.

(8)
[FdR] N. Francez and W. P. de Roever, Fairness in communicating processes, unpublished memo, Computer Science Dept., Utrecht University, July 1980.

(9)
[GFK1] O. Grumberg, N. Francez, and S. Katz, A complete proof rule for strong equifairness, in Proceedings of 2nd Workshop on Logics of Programs, CMU, in LNCS 164 (E. Clarke and D. Kozen, eds.), 1983; also to appear in JCSS, 1986.

(10)
[GFK2] O. Grumberg, N. Francez, and S. Katz, Fair termination of communicating processes, Proceedings of 3rd ACM PODC, Vancouver, August 1984.

(11)
[GFMdR] O. Grumberg, N. Francez, J. Makowsky, and W.P. de Roever, A proof rule for fair termination of guarded commands, Information and Control 66, 1/2: 83-102, July/August, 1985.

(12)
[H] C.A.R. Hoare, Communicating sequential processes, CACM 21, 8, August 1978.

(13)
[HLP] W. Hennessey, Wei-Li, G. Plotkin, Semantics for Ada tasks, proceedings of TC.2 Working conference on the formal description of programming concepts, Garmisch Partenkirchen (D. Biorner, ed.), North Holland, 1983.

(14)
[KdR] R. Kuiper and W.P. de Roever, Fairness assumptions for CSP in a temporal logic framework, proceedings of TC.2 Working conference on the formal description of programming concepts, Garmisch Partenkirchen (D. Biorner, ed.), North Holland, 1983.

(15)
[L1] L. Lamport, Time, clocks, and the ordering of events, CACM 21, 1978, pp. 558-566.

(16)
[L2] L. Lamport, What good is temporal logic?, proceedings of IFIP 9th world congress, Paris, France, September 1983.

(17)
[LPS] D. Lehmann, A. Pnueli, and J. Stavi, Impartiality, justice, and fairness: the ethics of concurrent termination, Proceedings of 8th ICALP, Acco, Israel, July 1981, in LNCS 115 (O. Kariv and S. Even, eds.), Springer-Verlag, 1981.

(18)
[OA] E.-R. Olderog, K.R. Apt, Fairness in parallel programs, the transformational approach, TR 86-11, Univ. of Kiel, 1986 (submitted for publication).

(19)
[OL] S.S. Owicki, L. Lamport, Proving liveness properties of concurrent programs, ACM-TOPLAS 4, 3, July 1982: 455-495.

(20)
[P] G.D. Plotkin, An operational semantics for CSP, proceedings of TC.2 Working conference on the formal description of programming concepts, Garmisch Partenkirchen (D. Biorner, ed.), North Holland, 1983.

(21)
[PdR] A. Pnueli and W.P. de Roever, Rendezvous with Ada: a proof-theoretic view, RUU-CS-82-12, University of Utrecht, July 1982. Also in: proceedings of the AdaTec conference, Crystal City, 1982.

(22)
[R] W. Reisig, Partial order semantics versus interleaving semantics and its impact on fairness, Proceedings of 11th ICALP, Antwerp, 1984.

(23)
[RS] J. Reif, P. Spirakis, Probabilistic bidding gives optimal distributed resource allocation, TR, Aiken Computation Lab, July 1983.