APPRAISING FAIRNESS IN LANGUAGES FOR DISTRIBUTED PROGRAMMING

Krzysztof R. Apt, Nissim Francez, and Shmuel Katz

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-88-10 March 1988

APPRAISING FAIRNESS IN LANGUAGES FOR DISTRIBUTED PROGRAMMING†

by

Krzysztof R. Apt‡

Center for Mathematics and Computer Science, Kruislaan 413, 1098SJ Amsterdam, The Netherlands

Department of Computer Science, University of Texas at Austin, Austin, Texas, U.S.A.

Nissim Francez* and Shmuel Katz

Department of Computer Science, The Technion, Haifa, Israel

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. The notion of fairness most often mentioned for Ada is shown to be fully acceptable. For a model with nonblocking *send* operations, some variants of common fairness definitions are appraised, and two are shown to satisfy the suggested criteria.

---

† A preliminary version of this work appeared in [AFK].

‡ The work of the first author was partially supported by Office of Naval Research grant N00014-86-K-0763.

*The work of the second author was partially supported by the Fund for the Promotion of Research at the Technion.

1. Introduction

Fairness is an important concept which naturally arises in the study of nondeterministic systems, in particular when dealing with concurrent systems. A very general formulation is a statement of the form: if a certain choice is possible sufficiently often, then it is sufficiently often taken. Depending on the definitions of a ''choice'', ''possible'', and ''sufficiently often'', different notions of fairness arise. A variety of these fairness notions have been introduced in the literature and studied both from a proof theoretic and a semantic point of view. Semantics is usually introduced by means of a computational model which defines legal computations. A *two–leveled* approach is most often taken in which first the legal computations are described, and then a fairness notion is used to exclude some additional computations which otherwise would be legal. An overview, examples, and further references may be found in [Fr].

For nondeterministic programs some of the fairness notions 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.

In this paper, 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 given, and the theorems and proofs which lead to these results are presented.

Feasibility:

As noted above, any definition of fairness excludes some of the executions (the ''unfair'' ones) which otherwise would be legal executions of a program according to a 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 and initial state. That is, for every legal program and initial state some (finite or infinite) fair computation does exist. This restriction is closely related to the idea of implementing fairness by means of schedulers. Without it, 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 every partial computation to a fair one. This is the proposed *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 a fairness notion must incorporate the set of choices which are *jointly possible* at each stage, as in [GFK1].)

Equivalence Robustness:
For concurrent programs, the computational model used induces a *dependency* relation among actions. For example, an input action of a receiving process depends on a corresponding output action of a sending process. The computations of asynchronous, distributed systems are often modeled by 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 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 the equivalence induced by that model. 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. If this criterion is not satisfied, then fairness depends on the particular ratio of processor speeds or on the location of the observer, which is undesirable.

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. Any additional fairness requirement complicates the scheduling and may cause difficulties in defining a precise semantics or proving

---

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

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

Figure 1

correctness. Thus adding an additional liveness requirement of some sort of fairness is only justified if some benefit will accrue. That is, there must be some 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 properties will hold for some programs. As 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 here the impact of fairness assumptions on termination only. This is true because such assumptions are known not to affect partial correctness or, more generally, safety properties, and other liveness properties can be reduced to termination for derived programs (see [GFMdR]).

Plan of the paper

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 these criteria, or any variants and additions they prefer, to their favorite fairness definitions and computational models.

In the next section we introduce the formal definitions of the semantics and of the fairness criteria. Then in Section 3 the properties of six fairness notions for *CSP* are analyzed in detail. We conclude that only one of these common notions--*Strong Process Fairness*-- satisfies all three criteria. The joint action of *CSP* involves synchronous communication between a pair of processes. In Section 4, we study the case of N-way communication (for arbitrary $N > 2$), i.e., a joint action with synchronous communication among N processes. We show that none of the six common fairness definitions we consider satisfy all of the criteria. The difference between the 2-way and N-way cases lies in a greater possibility of ''conspiracies'' when $N>2$. That is, one group of processes may ensure that particular actions involving other processes are insufficiently often possible.

In Section 5 fairness for an abstraction of Ada is considered, while Section 6 defines and appraises fairness notions for a message-passing model with a nonblocking *send* operation. The Ada and the nonblocking *send* models have in common that the fairness notions relate to the receipt of a message or activation of a rendezvous within a single process. As is shown, for this reason all of the fairness notions considered will be equivalence robust for these models. In the Conclusions, some implications of our results are considered regarding proof rules for termination under a fairness assumption.

2. Formal definitions

2.1 Computational models

The models of computation considered here are assumed to have some common structural properties. By a *distributed* program we mean a fixed collection of *processes*. These processes have disjoint states and perform atomic *actions*. The model attributes each action either to one process, in which case we refer to it as a *local* action (of that process), or to two or more processes, in which case we refer to it as a *joint* action (of those processes). A *configuration* is a pair consisting of a global state and an atomic action to be taken.

Definition: A *computation* is a maximal sequence of configurations, where the action in a configuration transforms the state of that configuration to the state of the immediately following configuration.

We also assume that the state determines a predicate *enabled* over the possible actions which may appear in a configuration, as defined below.

Definition:

i) An *action* is *enabled* in a configuration if it can serve as the next action executed (where the exact definition is model dependent).

ii) A *process* is *enabled* in a configuration if some (possibly joint) action attributed to it is enabled in the configuration.

iii) A process is *ready* for an action in a configuration if its local state is the projection of a state in which the action is enabled and the action is attributed to that process.

The second component of a configuration is always one of the actions enabled in that configuration and represents the one chosen to be executed at that point in the computation.

Similar approaches 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 for example [L1], [R], or [DM]) expressing only the essential causal relationships among the atomic actions (both local and joint). In this paper we will assume that the underlying partial orders are total over the local atomic actions of each individual process, so that two local actions of the same process are ordered. Clearly, every such partial order induces a dependency relation among actions, and a uniquely defined equivalence over interleaved computations of those satisfying the same partial order with the same actions.

Definition: Two atomic actions are *independent* if they are not related by the partial order.

Definition: if π and ρ are interleaved computations, then $\pi \equiv \rho$ iff π can be obtained from ρ by (possibly infinitely many) simultaneous transpositions of two independent atomic actions .

Thus we assume a combined semantics where both the collection of interleaved

computations and the equivalence relations defined by the underlying partial order are available. A temporal logic assuming this kind of semantics is defined and investigated in [KP].

In this paper, three additional assumptions are made about the syntax of the programs studied and the computational models considered:
(1) *Noninstantaneous readiness*: Every joint action is immediately followed by a configuration with a state in which each participant process is not ready for any joint action. This means that once a process executes a joint action it enters a local state in which *none* of the joint actions in which it can participate is enabled. The next local action could, of course, be a (possibly implicit) *skip* whose only effect is to make some joint action become a possible later choice.

This affects the definition of when a joint action is *continuously* enabled. The justification for the noninstantaneous readiness assumption is that joint (and other) actions take time at the implementation level, even though they are considered atomic on the program level. Thus if we wish to equate ''continuously'' with ''uninterruptedly'' (as we do), even the interruption caused by executing one action can be enough to make other (joint) actions temporarily disabled. As will be indicated in the proofs, this assumption influences the results we obtain regarding liveness enhancement. A more detailed examination of issues involved in deciding when a joint action should be considered enabled may be found in [FK]. Some other work in this area ([KdR]) assumes that only states where joint actions are possible choices need be considered as significant. In that case, it would be possible for a process which participates in a joint action A to nevertheless be ''continuously'' ready to participate in some other joint action B.

The noninstantaneous readiness assumption may be enforced either by assuming that local actions actually appear in the text after every joint action, or by positing a hidden local state and local *skip* action after every joint action.

(2) *Uniform choice*: A choice between a local and a joint action is never possible. This assumption is motivated by our desire to emphasize the influence of fairness assumptions on the execution of joint actions, and the fact that many fairness definitions do not relate at all to local actions. This and the previous assumption together 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. Again, this assumption is crucial to some of the results seen in later sections.

(3) *Minimal progress* [OL]: Every process in a state with enabled *local* actions will eventually execute some action. This minimal progress assumption is somewhat stronger than the fundamental liveness property mentioned in the introduction. According to this stronger assumption, a

process will not simply ''stop executing'' when it has local actions which may be chosen. In the sequel, all computations are assumed to satisfy the minimal progress property.

Note that this property could be itself considered to be a fairness assumption, and indeed has been in the literature. However, in [FdR] it is shown not to allow proving the termination of additional programs beyond those which terminated under the fundamental liveness assumption (that some atomic action is executed somewhere). In our terminology this means that minimal progress is not liveness enhancing in relation to the fundamental liveness property. We have chosen to ''build-in'' this assumption so that the focus of additional fairness definitions is on joint actions (e.g., interprocess communication). This assumption is significant for results on liveness enhancement, since the enhancement is relative to this minimal progress property.

2.2 Fairness and appraisal criteria

Now the possible definitions of fairness and the criteria for their appraisal may be expressed in terms of the computational models.

Definition: Given a (distributed) program P, **comp(P)** is the set of interleaved computations generated by P under the semantics of the model, assuming only the minimal progress property.

A *fairness notion* (or *fairness definition*) **F** is a rule for selecting, for any given program P, a subset of computations $\mathbf{F(P)} \subseteq \mathbf{comp(P)}$ such that **F(P)** contains all finite computations in **comp(P)**.

Note the indirect dependence of **F** on the model of computation, since **comp(P)** itself depends on the model. Actually, an arbitrary selection function would generally not be considered a fairness notion at all since the uniform predicate for deciding whether a computation is fair or not involves the choices made during the computation. A fairness definition would be expressed in terms of the predicates *enabled*, *ready*, and other predicates such as *executed* (true of an action if it has been executed in the previous configuration). However, such restrictions will not be imposed here formally, since in any case we do not intend to precisely characterize all possible fairness definitions, but rather to provide criteria for appraising specific examples of such definitions. Now we may state these criteria precisely.

A necessary condition for feasibility of **F** is that for all programs P, if $\mathbf{comp(P)} \neq \emptyset$, then $\mathbf{F(P)} \neq \emptyset$. As already explained, feasibility should also prevent a scheduler from "painting itself into a corner" with no possible continuation. Thus the definition is expanded to cover this difficulty.

Definition: **F** is *feasible* iff for every program **P** every finite initial segment of an interleaved computation in **comp(P)** can be extended to a computation in **F(P)**.

Definition: **F** is *equivalence robust* iff for every program **P** and every two computations π and ρ in **comp(P)**, $(\pi \in \mathbf{F(P)} \wedge \pi \equiv \rho) \Rightarrow \rho \in \mathbf{F(P)}$.

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

This definition means that P terminates under the assumption of **F**. 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 used only in p. Note that in general $[\pi]_p$ need not be a computation.

The following simple lemma will be useful in the sequel. It is a direct consequence of our assumption about the totality of the local dependence relation within a process.

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. Bougé (private communication) for *CSP* programs. We do not need this stronger version here.

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
(i) nested parallelism is disallowed,
(ii) the distributed termination convention is not adopted,
(iii) output commands may appear in guards,
(iv) the three additional assumptions given in the previous section are also imposed.

The semantics we consider is that of interleaved computation sequences as defined in [P]. According to this semantics the control of a process is identified with the part of the process text still to be executed. A configuration is then a vector of control points of the processes and a usual global state. This view can easily be converted into the configuration defined in Section 2.1 because the action taken can be extracted from the information available in successive control vectors, as may the predicate *enabled*.

In order to satisfy the noninstantaneous readiness assumption, we assume that each i/o command or i/o guard is immediately followed by a local action (which as mentioned might be

skip). To ensure the uniform choice assumption we postulate that in alternative and repetitive commands either all guards are boolean or all guards contain an i/o command. Finally, only computations satisfying the minimal progress assumption are considered. In the continuation, when the *CSP* model is referred to, all of the assumptions above are included.

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 satisfy a condition. That is, the ''choices'' for fairness could be among the processes, the pairs of processes which could communicate (i.e., the channels), or the individual communications.

Once it has been settled what is to be fair, the precise interpretation of ''sufficiently often'' must be determined. Two well-known possibilities for *CSP* are *weak* fairness, in which the choice is possible *continuously* from some point on, or *strong* fairness, in which the choice is possible *infinitely often*. Taking all of the combinations, six notions are obtained.
Strong Process (SP) fairness: an infinite computation is fair iff each process infinitely often ready to execute some joint atomic actions 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 i/o 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*: under our assumptions, no type of Weak fairness is liveness enhancing, and Strong Communication or Channel fairness are not equivalence robust. These results are summarized in Table I.

Proposition 1: The six notions of fairness defined above are all feasible for the *CSP* model.
Proof idea: For each fairness 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

	feasible	equivalence robust	liveness enhancing
SP	+	+	+
SCh	+	−	+
SC	+	−	+
WP	+	−	−
WCh	+	+	−
WC	+	+	−

Table I: Summary of appraisal for *CSP*

satisfies the condition for being in **D**, i.e., a computation satisfying the fairness notion under consideration. 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* seen in Figure 2, 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.

Versions of these schedulers could also be composed so that the conditions apply to *superimpose* (in the sense seen in [BF] and [K]) the scheduler on the program P, and so that the result would be a legal *CSP* program. Rather than using the shared variables in the schedulers described above, each process in P and the scheduler would be modified so that the values of the control locations and of the priority variables are sent as messages to the scheduler instead of

```
for each atomic action do
        if it is enabled then decrement its priority variable 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
```

Figure 2: *SELECT*

being read directly.

Because of the use of random assignments and possible nonuniqueness of the minimal priority variable, the scheduler itself is nondeterministic. The following *faithfulness theorem* holds, whose proof is a variant on abstract results in [OA].

Theorem: (Faithfulness)
1. Every computation of P generated by the scheduler is SCo fair.
2. Every SCo fair computation of P or any finite prefix of a computation can be generated by the scheduler.

Proof idea:
1. Consider a computation of P which is generated by the scheduler, and a pair of i/o commands which form a joint action. Each time this joint action is enabled in the sequence considered, its priority variable is decremented by 1. One can prove (see [OA]) that given n actions each priority variable is invariantly at least $-n+1$. This guarantees that every joint action infinitely often enabled is executed infinitely often.

Moreover, by the same argument, since local atomic actions also have associated priority variables which are decremented, every process with enabled local actions will eventually be activated so the minimal progress assumption will be met. The sequence generated by the scheduler is thus Strong Communication fair.

2. Consider a SCo fair computation of P or a prefix of a computation. To show that it can be generated by the scheduler, it is sufficient to define the appropriate values of the priority variables at the point where they are reset. We simply assign to each priority variable the number of times the associated action is enabled before it is taken (if at all). It is straightforward to see that this choice of values is consistent with the choices made by the scheduler. In fact, each action when taken will have its priority variable equal to zero. □

The above theorem immediately implies that Strong Communication fairness is feasible. For any finite prefix of a computation, by part 2 of the theorem it can be generated by the scheduler. The scheduler will then continue to choose events for execution. If it reaches a point at which no event can be chosen, this can only be because no event was enabled, and the same sequence of events define an execution which terminates from **comp(P)**, and thus is fair. Otherwise the scheduler will generate an infinite computation, which is also fair due to part 1 of the theorem. Thus every prefix of a computation has a fair extension, as required. Schedulers and faithfulness theorems may be obtained for the other fairness definitions merely by modifying the conditions for enabledness and for resetting the appropriate priority variables.

Proposition 2: Weak Communication, Weak Channel, and Strong Process fairness are equivalence robust for the *CSP* model.

Proof idea: It is easiest to show that *SP* fairness is equivalence robust for *CSP* by considering the unfair computations of an arbitrary program *P*. If π is Strong Process unfair, then from some point on there is a process P_i which is infinitely often enabled for at least one joint action but no joint action involving P_i is ever executed. Thus P_i is *continuously* ready for the communication, since there are no alternative local actions which it could execute. Here the Uniform Choice condition, i.e., 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, starting from some point in ρ, the process P_i is here also continuously ready for a joint action. Again, by the same lemma, there are infinitely many states in which the possible partner of P_i could have communicated with P_i, so the communication is enabled. Thus in this case also, ρ is *SP* unfair.

For the Weak Communication case, the assumption of being continuously enabled means that in an unfair computation neither participant process in a continuously enabled joint communication can do anything else. As before, this is also true in any equivalent computation sequence. Thus it too will be unfair, establishing the equivalence robustness. The *WCh* fairness is treated similarly.

Proposition 3: Strong Communication, Strong Channel, and Weak Process fairness are not equivalence robust for the *CSP* model.

Proof: We show that Weak Process fairness is not equivalence robust by exhibiting two equivalent 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. Even though the two computations are equivalent, one is Weak Process fair while 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 infinitely many 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.

2) P_0 executes its local action.

3) P_1 executes its local action.

$P :: [P_0 \,||\, \cdots \,||\, P_4]$

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: A conspiring program

4) P_3 and P_4 communicate.
5) P_3 executes its local action.
6) P_4 executes its local action.

This computation is clearly unfair to process P_2. 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. This computation is thus rendered Weak Process fair.

Similar examples may be constructed for *SCh* and *SCo* fairness. □

We have just shown that the Weak Process fairness condition can be satisfied vacuously in some computations by preventing the enabledness of the process involved, by having other processes (the possible partners for joint actions) execute other actions. However there exist equivalent computations in which some joint action is always possible for the process, rendering that computation unfair. For Weak Communication fairness this cannot occur because the only way to have a communication be continuously enabled is if both of the participants do not execute any other actions. If the communication is not continuously enabled because a participant did some other action, that action will also be performed in any equivalent computation.

In order to prove assertions about liveness enhancement, in a similar way to the approach in [FdR] and [KdR], we first compare the fairness notions in terms of ''strength'' in causing termination. However, the notions of fairness given there differ in that the channel level is replaced by a level dealing with a mixture of joint and local actions, the assumptions introduced in Section 2.1 are not considered, and weak fairness is defined differently. Nevertheless, using arguments similar to theirs, similar relations can be shown to hold. Below, $A \rightarrow B$ means that every *CSP* program which terminates under the fairness assumption A also terminates under the assumption B.

Theorem: (CSP-hierarchy) The relations below are the only ones which hold among the notions of fairness considered:

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

Proof (fragment): We show that $WP \rightarrow SP$ holds. Consider a *CSP* program P such that all of its Weak Process fair computations are finite. Then all Strong Process Fair computations of the same program are also finite, since every Strong Process Fair computation is also Weak Process fair. Other implications are equally straightforward to establish.

In order to see that $SP \rightarrow WP$ does not hold, consider the program shown in Figure 4. In every Strong Process Fair computation of the program, P_1 eventually communicates with P_2, and then termination is inevitable. However, the infinite computation in which P_1 never communicates is Weak Process Fair since the communication with P_2 is (infinitely often) disabled whenever P_2 communicates with P_3. Note that again the Noninstantaneous Readiness assumption is crucial, and in particular the fact that the *skip* on the right of the arrow is preceded by a local state in which no joint action involving P_2 is enabled.

Other cases of ''non-implications'' are left to the reader. □

Proposition 4: Strong Communication, Strong Channel, and Strong Process fairness are liveness enhancing for the *CSP* model.

Proof: To show that Strong Process fairness enhances liveness for *CSP*, we refer again to the

$P :: [\, P_1 \| P_2 \| P_3 \,]$,

where:

$P_1 :: b_1 := true$;

$*[\, b_1; P_2!0 \rightarrow b_1 := false \,]$

$P_2 :: b_2 := true$;

$*[\, b_2; P_1?x \rightarrow b_2 := false \;\; \Box b_2; P_3?x \rightarrow skip \,]$;

$P_3!0$

$P_3 :: b_3 := true$;

$*[\, b_3; P_2!0 \rightarrow skip \;\; \Box b_3; P_2?y \rightarrow b_3 := false \,]$

Figure 4. A program which terminates for Strong Process fairness.

program in Figure 4. 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. □

Proposition 5: Weak Communication, Weak Channel, and Weak Process fairness are not liveness enhancing for the *CSP* model.

Proof: We show that Weak Process fairness does not enhance liveness for *CSP*. For this task we need to demonstrate that for every program *P*, if **comp(P)** contains any infinite interleaved computation π, then **comp(P)** also contains an infinite *WP* fair computation. Thus the *WP* fairness assumption does not cause 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 idea is to construct ρ so that the processes which were previously the cause of the unfairness will execute fairly, without affecting the processes which actively executed operations from some point on in the original infinite computation π. The construction will succeed because this can be done without forcing those active processes in π to participate in any new joint actions. 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, starting

at that point, for each configuration of π, a maximal subset of **A** with enabled actions not involving a process from outside **A** is identified. Configurations resulting from executing an action by each of those processes are then inserted, followed by the configuration resulting from executing the next action from π. Note that the part of the state involving the next action executed in π is not affected by the additions, so that the (modified) configurations can still include the original sequence of actions from π. 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 ρ in which all processes in a collection **B** are infinitely often activated. There exists an equivalent computation σ, in which no process is **B**-enabled.
Proof: For each process in turn defer its local actions in ρ maximally. In such a way, an equivalent computation σ 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 Noninstantaneous Readiness condition and by the definition of a mute process) none of the processes in **B** can communicate in the considered state. This establishes the claim. □

The above lemma concludes the proof that Weak Process fairness is not liveness enhancing, since **B** can be chosen to be the processes not in **A**. Similar but simpler reasoning shows that Weak Channel fairness and Weak Communication fairness are also not liveness enhancing. □

As a consequence of Propositions 4 and 5, 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.

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 ready in order for the action to be enabled. An attempt to participate in a joint action *delays* a process until all other parties are ready for that action. After the communication, a local action takes place in each participating process, guaranteeing the Noninstantaneous Readiness assumption. The Uniform Choice and Minimal Progress properties are again assumed.

Thus, we consider a language with a structure similar to *CSP*. Within each process, 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.

The definitions of fairness we consider are over the individual processes, over the N-way communications, and additionally (as a generalization of channel fairness from *CSP*) over the collection of joint actions possible among a group of participating processes. The definitions are:
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. A group of processes is called *enabled* if there is some joint action which is enabled with exactly that group of processes as participants.

The results for N-way communication which are implied by the propositions given below, are summarized in Table II. Note that the results are similar to the *CSP* case except for the equivalence robustness of Strong Process fairness.

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

	feasible	equivalence robust	liveness enhancing
SP	+	−	+
SG	+	−	+
SC	+	−	+
WP	+	−	−
WG	+	+	−
WC	+	+	−

Table II: Summary of appraisal for N-way communication

Theorem: (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 6: The six fairness definitions are feasible for the 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*) a distinct priority variable. In particular, for local actions the action group will consist of the single process to which the action is local. The program section *SELECTWG* seen in Figure 5 differs from the strong case given in Figure 2 for *CSP* in that the priority variable is reset whenever the associated action group is not enabled. The priority variables associated with single processes, which were defined because of local actions, ensure that the scheduler generates computations satisfying the Minimal Progress condition.

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

Proposition 7: Weak Communication and Weak Group fairness are equivalence robust for an N-way communication model.

Proof: Using arguments similar to those in the proof of Proposition 2 we will 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. □

---

for each action group *do*
    if it is enabled *then* decrement its priority variable by 1
        else reset the priority variable to an arbitrary nonnegative integer;
select an enabled action group with a minimal value for its priority variable;
reset the priority variable of the selected action group to an arbitrary nonnegative integer;
if a local action was selected *then* execute it
        else select and execute one of the enabled joint actions of the action group

---

Figure 5: *SELECTWG*

Proposition 8: Strong Process, Strong Group, Strong Communication, and Weak Process fairness are not equivalence robust for the N-way communication model.

Proof idea: In particular, unlike in the *CSP* model, Strong Process fairness is not equivalence robust. To see this, consider the following program (Figure 6). Here joint actions (a, b, c) are described by the set of participating processes and uninterpreted assignments (A, B, C), since the example depends only on multiple synchronization and is independent of the content of the communications. 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 action b is jointly executed by processes P_2 and P_3.
2) P_3 locally executes $L_{3,1}$.
3) P_2 locally executes $L_{2,2}$.
4) The action c is jointly executed by processes P_3 and P_4.

$P :: [P_1 \,||\, P_2 \,||\, P_3 \,||\, P_4]$

where

$a :: (P_1, P_2, P_4): A$

$b :: (P_2, P_3): B$

$c :: (P_3, P_4): C$

and

$P_1 :: *[a \rightarrow L_1]$

$P_2 :: *[a \rightarrow L_{2,1}$

$\quad []b \rightarrow L_{2,2}]$

$P_3 :: *[b \rightarrow L_{3,1}$

$\quad []c \rightarrow L_{3,2}]$

$P_4 :: *[a \rightarrow L_{4,1}$

$\quad []c \rightarrow L_{4,2}]$

Figure 6: A program with N-way communication

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 the joint action a (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), followed by the cycle on 2), 4), 3), 5), 1), 6) is Strong Process fair, because action a (and thus P_1) is never enabled in it. Specifically, in order to execute the joint action a, 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.

The desired effect is obtained here by delaying local actions, preventing process availability and thereby disabling joint actions. Note that at least three participants in a joint action are necessary to generate such an example, and thus the reasoning does not apply to the *CSP* model with binary joint actions.

Proposition 9: Strong Communication, Strong Group, and Strong Process fairness are liveness enhancing for an N-way communication model.
Proof: Since *CSP* programs are special cases of programs with N-way communications, by Proposition 4, the three methods above are liveness enhancing. □

Proposition 10: Weak Communication, Weak Group, and Weak Process fairness are not liveness enhancing for the N-way communication model.
Proof idea: The argument is similar to the one in Proposition 5. 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. However, it should be realized that with other assumptions about the model of computation, and other definitions of fairness, it is possible to satisfy all three criteria. In fact, in [AF] a new notion of fairness called *hyperfairness* is proposed for an N-way model, and this notion was specifically designed to be feasible, equivalence robust, and liveness enhancing for the model.

5. Results for an Ada-like communication fragment

In this section we consider a generalization of the process queues from the Ada definition to a fairness notion suggested 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 constructs such as **while** or **if-then**. In addition, it may *call* an entry in another process, using the syntax *<process-name>.<entry-name>(<actual-parameter-list>)*. This suspends execution of the calling process until a corresponding *accept* statement in the called process has completed executing due to that call. The *accept* statement has the form *accept <entry-name>(<formal-parameter-list>) → <statement>*. It can execute (by passing parameters, executing the statement, and passing back the **out** parameters) when it is reached in the process containing it and a call from another process has been made with that *entry-name*. There also is a *select* statement which has *accept* statements as nondeterministic alternatives.

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 of *Ada*.

The usage of the entry queues can serve as a scheduler for the entry-calls, where the queues play a role analogous to the priority variables of the other schedulers. We immediately obtain
Proposition 11: *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 Uniform Choice assumption 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 needed here.

Proposition 12: *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 13: *Entry fairness* is liveness enhancing for the *ACF* model.
Proof: Consider the program given in Figure 7. 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 (z and then x will become false, and the second *accept* will only be possible with P_3, causing w to also become false).

In passing, we note (as mentioned in [GdR]) 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 13.

$P :: [P_1 \parallel P_2 \parallel P_3]$

where

$P_1 :: P_2.e(false, y)$.

$P_2 ::$ $x := true$;

while x **do**

accept e (**in** z, **out** v) $\rightarrow$ **begin** $x := z$; $v := z$ **end**;

accept e (**in** z, **out** v) $\rightarrow$ $v := false$.

$P_3 ::$ $w := true$;

while w **do** $P_2.e(true, w)$.

Figure 7: A fairly terminating *Ada* program

6. Results for models with nonblocking send

In traditional message-passing models on a network, there are *send* and *receive* operations for communication, but, unlike *CSP*, the *send* operation terminates independently of message arrival. That is, it cannot be blocked and is a purely local action. A *receive* operation can then be executed only if a "corresponding" *send* operation has been previously executed on the other end of the appropriate channel, and in some sense (which needs to be precisely defined) the message has "arrived" at the process containing the *receive*. Again, we wish to abstract away from an operational consideration of explicit queues of messages, and to consider fairness in terms of the *receive* operations which must occur. For this reason, we will consider a message to be available at a receiving process as soon as it has been sent. Since a process can "pause" arbitrarily long before executing a local operation, this is sufficient to represent possible delays in the delivery of a message. Note that here a *receive* operation is treated as a joint action even though only one process (directly) participates in it.

As an example, in the sequel we consider a language syntactically identical to *CSP*, but with the send operation ($P\,!e$) interpreted as nonblocking. In such a context, since *send* is a local operation, it will not be used in guards as an alternative to *receive* operations ($P?v$) in order to maintain the Uniform Choice assumption. A *receive* action is *enabled* if the process containing it is at a control point where the action can be chosen for execution and moreover some matching *send* operation has been executed and the message sent has not yet been received. As previously, a *process* is enabled in a state if it contains enabled *receive* operations in that state. Three versions of fairness will be considered, analogous to the Process, Channel, or Communication fairness seen for other models, each in a Weak and a Strong version.

Process fairness is defined as in the other models we have considered: if the process is sufficiently often enabled, then one of the *receive* actions in it (which are the only "joint" actions) will be executed. On the other hand, it is reasonable to define a version of Channel fairness in terms of the *receive* operations, to be called *Receive fairness*:
Each *receive* operation which is sufficiently often enabled, is infinitely often executed. This is analogous to the Channel case because the enabledness condition means that a matching *send* operation was executed earlier in the process identified by the *receive*, and that two processes must therefore communicate.

Finally, a fairness called *Message fairness* is defined by: Each message which is sufficiently often capable of being received, is indeed received. That is, if a *receive* operation is enabled sufficiently often after a message has been sent by a matching *send*, that particular message will eventually be the one received. This is analogous to Communication fairness because an individual communication is considered.

Since once it is sent, a message will not be retracted (and we are not considering faulty message links), the only difference between the weak and the strong versions is the control location of the receiving process. For Weak fairness, the desired action (executing a *receive* operation or receiving a particular message) must occur if the enabling condition is continuously true from some point on and this is equivalent to being at a control point where a *receive* operation is enabled, from some point on. For the Strong versions, it is sufficient for the enabling condition to be true repeatedly (infinitely often).

In Table III the results of the appraisal for this model are summarized. As previously, the justifications are found in the propositions below.

The locality of *send* as seen here is similar to the local nature of the *call* of the version of *Ada* seen in the previous section, even though the *call* is blocking. In fact, a standard implementation of the message channels using queues can be used here also to show the feasibility of all six of these definitions of fairness, just as was done for the abstraction of the Ada queues.

Proposition 14: The six notions of fairness defined above are feasible for the nonblocking *send* model.

Proposition 15: All six notions of fairness defined above are equivalence robust for the nonblocking *send* model.

Proof: We show that Strong Message fairness is equivalence robust. In order to do this, consider a *SM* unfair computation π and any equivalent computation ρ. By definition, π includes a *send* action of some message, but not the corresponding *receive* action for that message, even though corresponding *receive* actions are infinitely often enabled. By the Projection Equality lemma, the *send* action will also eventually occur in ρ and from that moment on the enabledness in ρ of all corresponding *receive* actions is only dependent on the control location of the process

	feasible	equivalence robust	liveness enhancing
SP	+	+	−
SR	+	+	+
SM	+	+	+
WP	+	+	−
WR	+	+	−
WM	+	+	−

Table III: Summary of appraisal for nonblocking send *CSP*

containing them.

Again by the Projection Equality lemma, these *receive* actions will be infinitely often enabled but none of them will be executed with this message. Thus ρ is also *SM* unfair.

An analogous argument holds for other fairness notions. All of them depend on the fact that a *send* action will occur in all equivalent computations if it occurs in one and that the enabledness of the corresponding *receive* action is only dependent on the control location of the process containing the *receive*. Thus, there is no possibility of conspiracies. That is, we cannot produce a computation equivalent to an unfair one, but which is made fair by preventing eventual enabledness of actions which were enabled in the unfair computation. □

This result shows a connection between equivalence robustness and the degree of synchronization in joint actions. At least for these definitions of fairness, when there is no synchronization all are equivalence robust, when there is handshaking between two, three of six notions are equivalence robust, and when there are N-way communications only two out of six are still equivalence robust.

Proposition 16: Strong Receive and Strong Message fairness are liveness enhancing for the nonblocking send model.

Proof: As in the programs of Figures 4 and 7, it is easy to design a program in this model in which two processes exchange messages, while a single message sent to one of them from a third process causes all three to terminate if it is ever received. The nonterminating computations, in which the message causing termination is simply ignored in favor of messages from another process, are ruled out by either Strong Receive or Strong Message fairness. Since only one message is sent from the third process, there is no difference between the two fairness notions for this example. Under either type of fairness the program always terminates, and by definition this shows liveness enhancement. □

Proposition 17: Strong Process, Weak Process, Weak Receive, and Weak Message fairness are not liveness enhancing for the nonblocking *send* model.

Proof: As in previous proofs, it is most natural to consider an infinite unfair computation, and to show that there must also be an infinite fair one. For the types of fairness given above, there is no way to force the processes which are infinitely often activated in the unfair infinite computation to receive a message, even if other processes intermittently are made to receive or send messages. For all of the Weak forms, it is clear that the fairness notion only influences the selection of a *receive* operation for processes which from some point on do no other operation. Strong Process fairness also cannot affect the operation of the processes which are participating in the infinite computation, because they are indeed executing *receive* operations, and any changes in

the other processes are irrelevant. Unlike the *CSP* model, here Strong Process fairness is also not liveness enhancing because in the nonblocking send model the sending of a message is a local action not related to fairness, and a process with a matching *receive* (which might be participating in an infinite computation) need not receive the message. For *CSP*, the demand that a process participate in a joint action (for example, by sending a message) forced particular messages to be received by another process (the one with the matching *receive*). □

7. 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 a notion of fairness in the N-way communication model is noted (of course using different terminology).

As seen in the consideration of liveness enhancement, one way to express the difference between a model with a fairness assumption and one without is to consider the implications for termination of programs. In [BK-S2] and in [GFK] the termination properties of various models and fairness definitions are considered. Those works must deal with the problem that equivalence robustness is not maintained by many of the models and fairness definitions. 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. For example, in [GFK] an incomplete two-level proof system is suggested for the *CSP* model with Strong Communication fairness. Rules are given which allow showing that for a particular program the fairness definition does respect the equivalence classes of computations generated for that program. Then, separately, it is shown that the program terminates for all the so-called *serialized* computations. Unfortunately, the rules for the first part are complex, not easy to apply, and only treat some obvious cases.

We have shown that for a variety of models and notions of fairness an alternative approach is viable: to evaluate the fairness notions more carefully to find those which are feasible, inherently equivalence robust, and yet liveness enhancing. By establishing once and for all that a fairness definition is equivalence robust for a model, and furthermore is feasible and liveness enhancing, it becomes possible to state simple proof rules for termination of programs. In other words, we need not worry about possible "conspiracies" of some processes against others as was seen in the program of Figure 6.

In general, the idea of defining criteria, and then systematically evaluating the potential definitions of fairness for the 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 [ABC], is 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 cannot employ some of the techniques we have used here, if communication need not be confined to (top level) guard positions. 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 Bougé for valuable comments and discussions on the subject of this paper, and in particular for pointing out the importance of the Noninstantaneous Readiness assumption. The work reported was carried out during a visit of the first author in the Computer Science Department of the Technion.

References

(1)
[ABC] K.R. Apt, L. Bougé, Ph. Clermont, Two normal form theorems for CSP programs, *Information Processing Letters* 26, pp. 165-171, 1987/88.

(2)
[AFK] K.R. Apt, N. Francez, and S. Katz, Appraising fairness in languages for distributed programming, Proceedings of 14th ACM-POPL Symposium, Munich, West Germany, January 1987.

(3)
[AO] K.R. Apt and E.-R. Olderog, Proof rules and transformations dealing with fairness, *Science of Computer Programming* 3, pp. 65-100, 1983.

(4)
[AF] P. Attie and N. Francez, Fairness and hyperfairness in multiparty interactions, MCC-STP Technical Report, July 1987. Submitted for publication.

(5)
[BK-S1] R.J. Back and K. Kurki-Suonio, Decentralization of process nets with centralized control, Proceedings of 2nd ACM-PODC Symposium, Montreal, August 1983.

(6)
[BK-S2] R.J. Back and K. Kurki-Suonio, Serializability in distributed systems with handshaking, CMU Technical Report 85-109, 1985.

(7)
[BF] L. Bougé and N. Francez, A compositional approach to superimposition, Proceedings of 15th ACM-POPL Symposium, San Diego, Calif., January 1988.

(8)
[DM] P. Degano and U. Montanari, Concurrent histories, a basis for observing distributed systems, to appear in

Journal on Computer and System Sciences.

(9)
[Fo] I. Forman, On the design of large distributed systems, Proceedings of International Conference on Computer Languages, Miami Beach, Florida, October, 1986.

(10)
[Fr] N. Francez, *Fairness*, Texts and monographs in computer science series (D. Gries, ed.), Springer-Verlag, New York, 1986.

(11)
[FdR] N. Francez and W. P. de Roever, Fairness in communicating processes, unpublished memo, Computer Science Dept., Utrecht University, July 1980.

(12)
[FK] N. Francez and S. Katz, Fairness and the axioms of control predicates, to appear in *International Journal of Parallel Programming*.

(13)
[GdR] R.T. Gerth and W.P. deRoever, A proof system for concurrent Ada programs, *Science of Computer Programming*, vol. 4, pp 159-204, 1984.

(14)
[GFK1] O. Grumberg, N. Francez, and S. Katz, A complete rule for equifair termination, *Journal on Computer and System Sciences*, vol. 33, no. 3, pp. 313-332, December, 1986.

(15)
[GFK2] O. Grumberg, N. Francez, and S. Katz, Fair termination of communicating processes, Proceedings of 3rd ACM-PODC Symposium, Vancouver, August 1984.

(16)
[GFMdR] O. Grumberg, N. Francez, J. Makowsky, and W.P. de Roever, A proof rule for fair termination of guarded commands, *Information and Control* vol. 66, no. 1/2, pp. 83-102, July/August, 1985.

(17)
[H] C.A.R. Hoare, Communicating sequential processes, *Communications of the ACM* 21, 8, pp. 666-677, August 1978.

(18)
[HLP] W. Hennessey, Wei-Li, G. D. Plotkin, Semantics for Ada tasks, Proceedings of TC.2 Working Conference On the Formal Description of Programming Concepts, Garmisch Partenkirchen (D. Bjorner, ed.), North Holland, 1983.

(19)
[K] S. Katz, A superimposition control construct for distributed systems, MCC-STP Technical Report STP-268-87, August 1987.

(20)
[KP] S. Katz and D. Peled, Interleaving set temporal logic, Proceedings of 6th ACM-PODC Symposium, Vancouver, Canada, August, 1987.

(21)
[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. Bjorner, ed.), North Holland, 1983.

(22)
[L1] L. Lamport, Time, clocks, and the ordering of events, *Communications of the ACM* 21, pp. 558-566, 1978.

(23)
[L2] L. Lamport, What good is temporal logic?, Proceedings of 9th IFIP Congress, Paris, France, September 1983.

(24)
[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.

(25)
[OA] E.-R. Olderog and K.R. Apt, Fairness in parallel programs, the transformational approach, to appear in ACM

Transactions on Programming Languages and Systems.

(26)
[OL] S.S. Owicki, L. Lamport, Proving liveness properties of concurrent programs, ACM *Transactions on Programming Languages and Systems* 4, 3, pp. 455-495, July 1982.

(27)
[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.

(28)
[PdR] A. Pnueli and W.P. de Roever, Rendezvous with Ada: a proof-theoretic view, Proceedings of the AdaTec conference, Crystal City, 1982.

(29)
[R] W. Reisig, Partial order semantics versus interleaving semantics and its impact on fairness, Proceedings of 11th ICALP, Antwerp, 1984.

(30)
[RS] J. Reif, P. Spirakis, Probabilistic bidding gives optimal distributed resource allocation, Aiken Computation Lab Technical Report, Harvard University, July 1983.