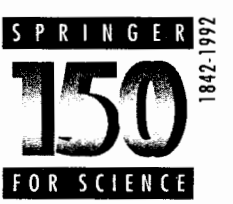
FORMAL ASPECTS OF COMPUTING

The International Journal of Formal Methods

Volume 4 Number 6 1992

Springer International



Formal Aspects of Computing (1992) 4: 582–591

Formal Aspects of Computing

On Equivalence-Completions of Fairness Assumptions*

Nissim Francez,[1] Ralph-J. J. Back[2] and Reino Kurki-Suonio[3]

[1] Computer Science Department, Technion, Haifa, Israel
[2] Computer Science Department, Abo Akademi, Finland
[3] Software Systems Laboratory, Tampere University of Technology, Finland

Keywords: Fairness; Completion; Partial-order semantics; Equivalence-robustness; Liveness properties

Abstract. The paper considers the treatment of fairness assumptions which are *not equivalence-robust*, a central issue in relating *interleaving semantics* to *partial order semantics*. A notion of *completion* is introduced and studied, and two specific completions are considered: *maximal completion*, which is easier to implement (shown by a broadcast bus implementation) but guarantees only weak liveness properties of programs using it; and *minimal completion*, which may be harder to implement but induces stronger liveness properties on programs using it. Some properties of completions are formulated. Finally, the impact of non-equivalence-robustness on compositionality with respect to separate fairness assumptions is considered.

1. Introduction

The notion of *equivalence-robustness* was introduced in [AFK88] as a desired property of fairness assumptions [Fra86] in programming languages for distributed programming. It was used to classify several of the existing fairness assumptions for such languages. As it turns out, several of the commonly used fairness assumptions are not equivalence-robust.

Basically, the equivalence (among computations) considered is that arising due

* Work started during a visit of the first author to the Computer Science Department, Abo Akademi, Finland, July–August 1988, and continued during the first author's stay at MCC in 1989/90.

Correspondence and offprint requests to: N. Francez, Computer Science Department, Technion, Haifa, Israel. e-mail: francez@cs.technion.ac.il

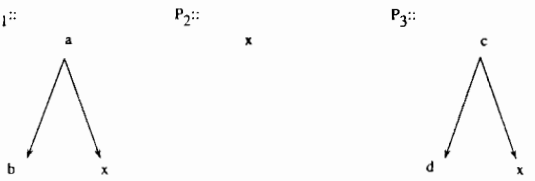


Fig. 1. An action-dependency graph.

to different interleavings of *independent* atomic actions. In the case of a *non-equivalence-robust* fairness assumption, a fair computation may be equivalent to an unfair one, which seems an undesirable situation. *Equivalence-robustness* excludes this situation. Thus, it serves as a natural bridge over the gap between *interleaving semantics* and *partial order semantics*, a central issue in current theory of concurrency and distributedness.

As a simple example consider the system of actions in Fig. 1, formulated informally in terms of *Interacting Processes* (*IP*). In this formalism, *multiparty interactions* serve as the interprocess synchronisation and communication mechanism. Comprehensive descriptions of the part of *IP* used in this example may be found in [Fra89] or [AFG90]. An extensive presentation of the underlying methodology for distributed program design based on such (and related) constructs will appear in [FrF93]. Here the following brief review suffices.

Each interaction[1] a has a (fixed) set of *participants*, $\mathscr{P}_a$, which are processes in a distributed program. An interaction a is *readied* by a process $P \in \mathscr{P}_a$ when P reaches a state where a is one of its next actions (i.e., processes have *choice* among actions). The interaction a is *enabled* when readied by *all* members of $\mathscr{P}_a$, and can then be executed. Enabled interactions a and b are in *conflict* if $\mathscr{P}_a \cap \mathscr{P}_b \neq \varnothing$, and are otherwise *independent*.

In the system in Fig. 1, there are three processes P_i, $1 \leqslant i \leqslant 3$, and five actions a, b, c, d and x. The participation relation is given by:

$$\mathscr{P}_a = \{P_1\}, \mathscr{P}_b = \{P_1\}, \mathscr{P}_c = \{P_3\}, \mathscr{P}_d = \{P_3\}, \mathscr{P}_x = \{P_1, P_2, P_3\}$$

Thus, a, b are local to P_1, and similarly c, d are local to P_3. However, x is joint to all of P_1, P_2, P_3. The arrows represent the choices among the *readiness* of actions, and each process iterates indefinitely. Thus, action b depends on action a and action d depends on action c. Action x depends on a, c *simultaneously*, etc. The computation (i.e., sequence of interaction executions) $\pi_1 = (abcd)^\omega$ is fair, in that each action which is infinitely-often enabled is infinitely-often executed. This kind of fairness is known as *strong interaction fairness* (*SIF*) [AFK88, AFG90]. Note that action x is *nowhere* enabled along π_1, since it is continuously readied by P_2, but only intermittently readied by P_1 and P_3, never simultaneously. However, the action b is independent of the action c. Hence, the following computation π_2, obtained from π_1 by permuting the order of these two independent actions, is equivalent to π_1:

$$\pi_2 = (ac \uparrow bd)^\omega$$

But π_2 is *unfair*, since action x *is* infinitely-often enabled along π_2 (at the positions marked with "$\uparrow$"). At this position, P_1 has just executed action a, facing a choice between both actions b and x (i.e., readying both). Similarly, P_3 has just executed action c, and readies both actions d and x. Finally, process P_2 readies action x

[1] We use action and interaction as synonyms here.

continuously. Thus, action x is readied by *all* its participants, i.e., it is enabled. Still, action x is never executed. This phenomenon is sometimes known as a *conspiracy*. See [AFG90] for a way of handling it (when it arises solely due to race conditions), as well as the explanation in the sequel. As a consequence of the equivalence between π_1 and π_2 (one being fair while the other not), one can deduce that *strong interaction fairness* is not equivalence robust [AFK88]. The absence of equivalence robustness may prevent the drawing of desired conclusions (about liveness properties of programs) from the presence of fairness assumptions. For example, no conclusion can be drawn about the eventual execution of action x in the above example.

Given a non-equivalence-robust fairness assumption F, one may be interested in its *completions*, resulting in other fairness assumptions which are derived from F and *are* equivalence-robust. As it turns out, there are two natural ways of completing an arbitrary fairness assumption F, to which we refer as the *maximal completion* F^+ and the *minimal completion* F^-. To understand the distinction, consider Fig. 2. In this figure, C represents the class of all the interleaved computations (of some program P), while F represents the subclass of fair computations, the only ones admissible by some given fairness notion. The dotted areas represent equivalence classes (within C), as induced by the equivalence relation considered above. There are three kinds of such classes. The first kind is that of *purely fair* classes (X_i, in Fig. 2), in which all members are fair. The second kind is that of *purely unfair* classes (Y_i in Fig. 2), in which all members are unfair. The third kind is that of *mixed* classes (Z_i in Fig. 2), containing both fair *and* unfair computations (e.g. $\pi_1, \pi_2 \in Z_1$ in Fig. 2).

A *conservative* completion (defined precisely below) has to resolve the fairness of such mixed classes. As it turns out, the two natural completions mentioned above differ in their treatment of mixed classes.

Maximal completion: make all computations in a mixed class *fair*. This approach is taken in [BaK88b] and [BaK88a] and its main advantage is its (relative) ease of implementation. Thus, if there is a legal way of avoiding commitment to execute some action, the scheduler of F^+ may avoid executing it.

Minimal completion: make all computations in a mixed class *unfair*. This approach is taken in [AFG90] (for a certain class of programs) and its main advantage is that it induces strong liveness properties on distributed programs (at the price of a more costly implementation than that of the maximal completion). Thus, if there is a legal way of committing to execute some action, the scheduler of F^- will commit.

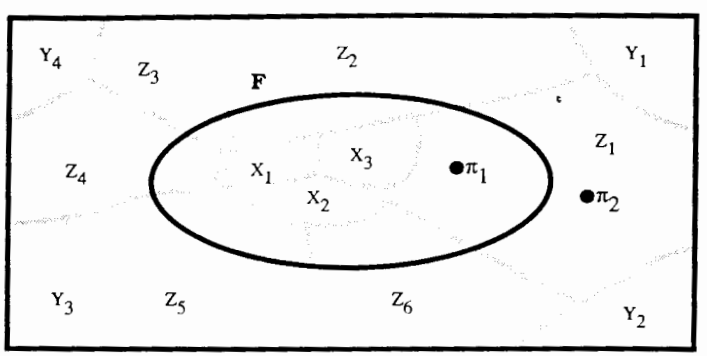


Fig. 2. The fairness-equivalence partitioning.

In terms of the example in Fig. 1, according to F^+ the action x need not ever be executed, while according to F^- it will be executed infinitely-often.

In the next section we formalise these notions and consider some of their consequences.

2. Equivalence Completions of Fairness

In order not to complicate the presentation, we assume that some underlying semantics determines, for each distributed program

$$P = [\underset{i=1,n}{\|} P_i]$$

a collection C_P of all its *computations*, assumed here to have the form $C_P \subseteq (A^* \cup A^\omega)$, a subset of the finite or infinite sequences over some set A. For example, A might be thought of as *global states*, *configuration*, *events*, etc. Here we will typically consider A to be a set of multiparty interactions, though the results do not depend on this interpretation. Furthermore, the underlying semantics equips A with a *dependency* relation, which is a partial order, and is respected by C_P. A typical partial order is the one used by Lamport [Lam78], where members of A are either *local* to some process P_i, $1 \leqslant i \leqslant n$, and totally ordered, or correspond to *communications* between different processes, and a *reception* of a message depends on its *sending*. For multiparty interactions, the dependency relation presented above is typical.

Definition. A *fairness* assumption determines for each program P a subset $F_P \subseteq C_P$, s.t.

1. For every P, $F_P \neq \varnothing$
2. For every P, $A^* \cap C_P \subseteq F_P$

The first requirement was called *feasibility* in [AFK88] and proposed there as a classification criterion, similar to equivalence-robustness. Here we assume it is always satisfied. The second requirement means that every finite computation is always (i.e., for any fairness assumption) fair.

In general, fairness assumptions have additional structure. The underlying semantics usually provides predicates (on A) such as *readied* (by some P_i), *enabled*, *executed*, etc., and F is formulated using these predicates (see [Fra86] for a comprehensive survey). Since the specific way of formulating F is immaterial to what follows, it suffices to consider F at the level of generality present above.

Definition. For $\pi_1, \pi_2 \in C_P$, π_1 and π_2 are *equivalent*, denoted by $\pi_1 \approx \pi_2$, iff each is obtained from the other by (possibly an infinite number of) order-preserving transpositions (between consecutive elements).

For example, if a and b are independent, then

$$(ab)^\omega \approx (ba)^\omega \approx ab(ba)^\omega$$

As is well-known (e.g. [Rei84]), equivalent computations can be interpreted as different observations of the *same* partial-order computation. In general, one would expect that properties of a system will be uniformly observed by all its possible observations.

Definition [AFK88]. A fairness assumption F is *equivalence-robust* iff for every program P, F_P is a union of " $\approx$ " equivalence-classes in C_P.

In terms of the picture in Fig. 2, a fairness assumption F is equivalence-robust if C_P has *no* mixed classes, i.e., two equivalent computations are either both fair or are both unfair. We refer the reader to [AFK88] for a classification of various fairness assumptions regarding their equivalence-robustness.

Suppose now that (some given) F is *not* equivalence-robust, but is appealing from some other point of view. One would like to have some derived fairness assumptions, which both preserve the main properties of F and, in addition, are equivalence-robust. When can we say that $\hat{F}$ is derived from F? Since the derivation is driven by the equivalence " $\approx$ " on computations, the idea is that $\hat{F}$ should agree with F on non-mixed equivalence-classes of C_P, but may redefine the fairness of computations in mixed classes. This motivates the following definitions.

Definition.

1. An " $\approx$ " equivalence-class X of C_P is *purely-fair* iff $X \subseteq F_P$, it is *purely-unfair* iff $X \cap F_P = \varnothing$, and is *mixed* otherwise.
2. The *fair core* $FC_P(F)$ of F_P is the collection of all the purely F-fair equivalence-classes of C_P, the *unfair core* $UC_P(F)$ is the collection of all the purely F-unfair equivalence-classes of C_P, and $MC_P(F)$ is the collection of all F-mixed equivalence-classes.
3. A fairness assumption $\hat{F}$ is *F-conservative* iff for every P, both $FC_P(F) \subseteq FC_P(\hat{F})$ and $UC_P(F) \subseteq UC_P(\hat{F})$ hold.

Thus, if $\hat{F}$ is a conservative modification of F it makes some of the mixed equivalence-classes in $MC_P(F)$ purely-fair, while it makes some other mixed classes in $MC_P(F)$ purely-unfair. A typical F-conservative $\hat{F}$ will be uniform in P, since typical definitions of fairness assumptions are uniform in P.

Clearly, an arbitrary F-conservative fairness assumption $\hat{F}$ need not be equivalence-robust. However, we are interested in those conservative modifications $\hat{F}$ of F which are.

Definition. A conservative modification $\hat{F}$ of F is a *completion* of F iff for every P, $MC_P(\hat{F}) = \varnothing$.

In other words, a completion of F decides for *every* mixed class whether it becomes purely fair or purely unfair. An immediate consequence of these definitions is the following.

Proposition (completion robustness). For every fairness assumption F and every completion $\hat{F}$ of F, $\hat{F}$ is equivalence-robust.

We now define two particularly useful completions, which treat all the mixed equivalence classes uniformly.

Definition. The maximal completion F^+ of F is the fairness assumption, s.t. for every P:

1. $FC_P(F^+) = FC_P(F) \cup MC_P(F)$
2. $UC_P(F^+) = UC_P(F)$

Thus, every mixed equivalence class of F is made purely fair by F^+.
The dual construction is similarly defined.

Definition. The *minimal completion* F^- of F is the fairness assumption, s.t. for every P:

1. $FC_P(F^-) = FC_P(F)$
2. $UC_P(F^-) = UC_P(F) \cup MC_P(F)$

Thus, every mixed equivalence class of F is made purely unfair by F^-. An immediate consequence of these definitions is that for every F and every P, $MC_P(F^+) = MC_P(F^-) = \emptyset$, so both F^+ and F^- are indeed completions, and hence also equivalence-robust. Furthermore, these are the extreme completions, in the sense of the following proposition which follows directly from the definitions.

Proposition (extremality).

1. For every F, every conservative completion $\hat{F}$ of F, and for every P: $F_P^- \subseteq \hat{F}_P \subseteq F_P^+$.
2. If F is equivalence-robust, then $F^+ = F^- = F$.

Note that the two extreme completions are *dual* to each other, in that $F_P^- = ((F_P^c)^+)^c$ and $F_P^+ = ((F_P^c)^-)^c$, where F^c is the (set theoretic) complement of F. However, lacking a general characterisation of fairness assumptions, it is unclear whether the complement of a fairness assumption is itself a fairness assumption. So, we do not rely heavily on duality arguments here.

3. Comparing Completions

How can the two extreme completions be compared? As mentioned in the introduction, each serves a different purpose. The maximal completion F^+ is the easier to implement. On the other hand, fewer liveness properties of programs *using* F^+ can be assumed. The minimal completion has exactly the opposite characteristics. In general, it would be the harder to implement. On the other hand, more liveness properties of programs using F^- can be guaranteed.

3.1. Ease of Implementation

In order to compare the two completions in terms of implementation cost, we consider the case where F is *strong interaction fairness* (*SIF*) in an *overlapping* semantics, where execution of (independent) actions may overlap in time. Formal definitions of this semantics can be found in [BaK88a] or [AFG90]. For the implementation model we consider a shared-bus broadcast network, which is assumed to have the following two properties:

Serialisation: the network serialises all messages broadcast by the various processes.

Fair access: if a process attempts to broadcast some message infinitely-often along the bus, it will eventually succeed.

The full implementation details may be found in [BaK88a]. Here we sketch some of the details so that the comparison can be made. A process P_i may broadcast two kinds of messages:

Willingness message w_S^i: here $S = \{a \in A \mid P_i \in \mathscr{P}_a\}$ is the set of interactions *readied* by P_i (possibly after evaluation of local boolean guards).

Selection message s_a^i: here $a \in A$ is an interaction for which process P_i has received willingness messages from all $P_j \in \mathscr{P}_a$, and P_i *commits* to executing a.

A correct implementation of SIF^+, requiring $2 * |\mathscr{P}_a| + 1$ broadcast messages (per execution of a), based on the above implementation model, is presented in [BaK88a]. More messages (and message-types) would be required in any attempt to implement SIF itself, since equivalent prefixes generate the same local sequences of broadcast messages. The additional messages would be needed to eliminate unfair interleavings that are equivalent to fair ones. To see this, consider again the example in Fig. 1. Suppose we have the following sequence of broadcasts:

$$P_1 :: \ldots w^1_{\{a\}} s^1_{\{a\}} w^1_{\{b,x\}} s^1_{\{b\}} \ldots$$

$$P_2 :: \ldots w^2_{\{x\}} \ldots$$

$$P_3 :: \ldots w^3_{\{c\}} s^3_{\{c\}} w^3_{\{d,x\}} s^3_{\{d\}} \ldots$$

These sequences of broadcast messages will generate either π_1 or π_2 (as in Section 1), depending on the way the bus serialises messages broadcast by different processes. Both are acceptable for SIF^+, but only π_1 is acceptable under SIF. Hence, more messages are needed to enforce the required interleaving. In the original presentation in [BaK88a], this difference was cast in terms of the *simulation mapping* in the proof of correctness of the implementation.

3.2. Liveness Enhancement

This property was introduced in [AFK88] as an absolute property of a fairness notion. Here we consider a *relativised* version of it. We say that a fairness notion F_2 is *liveness enhancing w.r.t.* fairness notion F_1 if there exists a program P having some liveness property under F_2 and not having the same property under F_1.

In order to see that F^- induces stronger liveness properties than F, we again consider SIF. For example, in the situation depicted in Fig. 1, a minimal completion will guarantee that action x *is* infinitely often executed in any computation. The originally fair computation π_1 (along which x was not infinitely-often executed), due to its equivalence to the unfair π_2, is excluded by F^-. One implementation strategy for minimal completions, used in [AFG90], is to *delay* operations, letting independent ones overtake. This needs, however, some global information about the program, and was presented there for a certain class of so called *conspiracy resistant* programs. In Fig. 1, delaying P_1 after action a or delaying P_3 after c sufficiently often, will guarantee that x is enabled sufficiently often.

3.3. Compositionality

Another aspect of comparison is the amenability for compositional application, when two fairness assumptions F_1 and F_2 are imposed. Can they be treated separately?

From general properties of boolean operations and equivalence relations, we get the following consequence:

Proposition (conjunctive distributivity). For any P,

1. $FC_P(F_1 \cap F_2) = FC_P(F_1) \cap FC_P(F_2)$
2. $UC_P(F_1 \cap F_2) \supseteq UC_P(F_1) \cup UC_P(F_2)$

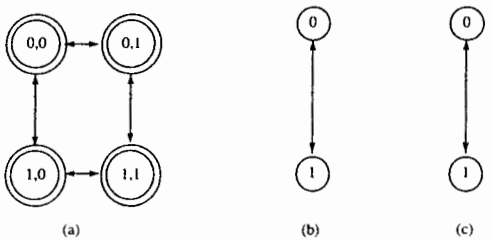


Fig. 3. Combining fairness assumptions.

However, $MC_P(F_1 \cap F_2)$ need not be a simple combination of $MC_P(F_1)$ and $MC_P(F_2)$. As a consequence, neither of the corresponding closure-equalities holds, namely $(F_1 \cap F_2)^+ \neq F_1^+ \cap F_2^+$ and $(F_1 \cap F_2)^- \neq F_1^- \cap F_2^-$. Still, there is one difference between the two closures, in that $F_1^- \cap F_2^-$ is a conservative modification of both F_1 and F_2, since

$$FC_P((F_1 \cap F_2)^-) = FC_P(F_1 \cap F_2) = FC_P(F_1) \cap FC_P(F_2)$$

and

$$UC_P((F_1 \cap F_2)^-) = UC_P(F_1 \cap F_2) \cup MC_P(F_1 \cap F_2) \supseteq UC_P(F_1) \cup UC_P(F_2)$$

The analogous result does not hold for $F_1^+ \cap F_2^+$. Consequently, as observed in [BaK88b], one needs equivalence robustness or joint minimal completions to guarantee compositional treatment of different fairness assumptions.

A dual distributivity lemma holds for $F_1 \cup F_2$, and maximal completion can successfully deal with weakening.

To see the impact of the absence of compositionality, consider the transition system in Fig. 3(a), which is the global system obtained by combining the two local systems (b) and (c). The global state $s = (s_1, s_2)$ is an ordered pair of local states.

Consider two local computations π_1 and π_2, each changing infinitely-often the value of one bit-component of the global state. Suppose that an observer of the first computation defines a fairness assumption $F_1 \stackrel{df.}{=}$ infinitely-often $s_1 = 0$, which is indeed observed in π_1. Similarly, an observer of the second computation defines a fairness assumption $F_2 \stackrel{df.}{=}$ infinitely-often $s_2 = 1$, which is indeed observed in π_2.

When these two assumptions are combined to $F = F_1 \wedge F_2$, i.e., infinitely-often *both* $s_1 = 0$ and $s_2 = 1$ occur, no global view of the resulting computation needs to observe F. For example, the interleaving may be so arranged that only the global states in $\{s_1 = 0 \wedge s_2 = 0, s_1 = 0 \wedge s_2 = 1, s_1 = 1 \wedge s_2 = 0\}$ ever occur.

4. Conclusions

The paper considers the treatment of non equivalence-robust fairness assumptions and proposes two ways of obtaining *completions* of a given fairness assumption which *are* equivalence-robust. Both ways have advantages and disadvantages and are useful from different perspectives: one from the programming point of view and the other from the implementation point of view.

It is not our aim here to develop a whole theory regarding the notions

introduced. Rather, we want to focus on the *possibility* of defining two completions with opposite characteristics. Neither in [AFG90] nor in [BaK88b] and [BaK88a], each using one completion, is the possibility of the dual one considered. Thus, this paper should clarify the situation regarding completions and allow for the right decision in the appropriate circumstances.

Certainly, more theoretical study has to be carried out to obtain a better understanding of these issues. Some questions, not dealt with so far, which deserve further study, are as follows.

- For specific formalisms for expressing fairness properties, given an expression for F, can expressions be derived for F^+, F^-?
- How does completion affect other properties of programs, different from fairness?
- Given *proof-rules* for F-fair termination (see [Fra86]), can one derive proof-rules for F^+-fair termination and for F^--fair termination?
- Similarly, given an *explicit-scheduler* for F (see [ApO83] or [Fra86, Ch. 3]), can schedulers be derived for F^+, F^-?

Many other questions can, of course, be raised, suggesting an interesting research topic. Actually, these completions are not necessarily confined to fairness assumptions. They are applicable to any property of partial-order computations expressible in terms of its interleavings. These properties are of importance in the study of distributed programs because they reflect properties observable by sequential observers, the most common type of observers of such programs. Recently, a new temporal logic based on partial orders, namely *ISTL* [KaP90, KaP92], was proposed. It might be the right tool with which some of these questions can be studied.

Acknowledgements

The first author wishes to thank Shmuel Katz for drawing attention to the equivalence of closures and direct definitions in terms of schedulers, and to Orna Grumberg for useful discussions. The part of the first author was partially funded by a grant from the Israeli Academy for Sciences (basic research) and by the Fund for the Promotion of Research in the Technion.

References

[ApO83] Apt, K. R. and Olderog, E.-R.: Proof Rules and Transformations Dealing with Fairness. *SCP*, 3, 65–100 (1983).

[AFG90] Attie, P. C., Francez, N. and Grumberg, O.: Fairness and Hyperfairness in Multi-Party Interactions. In: *17th ACM-POPL*, San-Francisco, CA, 17–19 January, pp. 292–305, 1990. (To appear in *Distributed Computing*.)

[AFK88] Apt, K. R., Francez, N. and Katz, S. M.: Appraising Fairness in Distributed Languages. *Distributed Computing*, **2**, 226–241 (1988).

[BaK88a] Back, R.-J. J. and Kurki-Suonio, R.: Distributed Cooperation with Action Systems. *TOPLAS*, 10(4): 513–554, October 1988.

[BaK88b] Back, R.-J. J. and Kurki-Suonio, R.: Serilizability in Distributed Systems with Handshaking. In: T. Lepisto and A. Salomaa, (eds), *LNCS 172*, pp. 52–56. Springer-Verlag, July 1988.

[Fra86] Francez, N.: *Fairness*. Springer-Verlag, New York, 1986.

[Fra89] Francez, N.: Cooperative Proofs for Distributed Programs with Multi-party Interactions. *IPL*, **32**, 235–242 (1989).

[FrF93] Francez, N. and Forman, I. R.: *Interacting Processes: a Multiparty Approach to Coordinated Distributed Programming* (Forthcoming).

[KaP90] Katz, S. M. and Peled, D.: Interleaving Set Temporal Logic. *TCS*, 75, 263–287 (1990).

[KaP92] Katz, S. M. and Peled, D.: Verification of Distributed Programs using Representative Interleaving Sequences. (To appear in *Distributed Computing*, 1992.)

[Lam78] Lamport, L.: Time, Clocks and Ordering Events in Distributed Systems. *Communications of the ACM*, **21**(7), 558–565 (1978).

[Rei84] Reisig, W.: Partial Order Semantics vs. Interleaving Semantics for csp-like Languages and its Impact on Fairness. In: *LNCS 172*, J. Paredaens (ed), Springer-Verlag, July 1984.

Received April 1990
Accepted in revised form August 1992 by C. B. Jones