COOPERATING-PROOFS FOR DISTRIBUTED PROGRAMS WITH MULTI-PARTY INTERACTIONS

by

Nissim Francez

Computer Science Department

Technion - Israel Institute of Technology

Haifa 32000, Israel

Abstract:

The paper presents a proof-system for partial-correctness assertions for a language for distributed programs based on *multi-party interactions* as its interprocess communication and synchronization primitive. The system is a natural generalization of the *cooperating proofs* introduced for partial-correctness proofs of CSP programs.

1. INTRODUCTION

The proof-theoretical notion of *cooperating-proofs* was introduced in [AFR80] as a tool for the verification of *CSP* distributed programs [Ho78], based on synchronous *send-receive* (also known as *handshaking* and *rendezvous*) interprocess communication. It follows the structure of two-leveled proof systems as introduced in [OG76] in the context of shared-variables concurrency. A closely-related proof system is presented in [LG81].

Since its introduction, the notion of cooperating-proofs was adapted to several contexts, capturing interprocess communication constructs of increasing complexity and of varieties of structure. It was applied to remote procedure calls in [GRR82] for the *DP* language [Bh78], to ADA's *rendezvous* in [GR84] and [BM82], to *monitors* in [GR86], and to *scripts* (a communication abstraction mechanism) in [FHT86]. Very good surveys of cooperating-proofs are [dR85] and [HR86]. Recently, the proof method was extended to *dynamic process creation* in [B87], and to *dynamic creation and destruction* in [FF87].

In all the above extensions, interprocess communication can be classified as *point-to-point* communication, involving two processes: a *sender* and a *receiver*. Multiple engagement *is* possible in some variants of a rendezvous only due to *nesting*. In recent developments of languages for distributed computing a new family of communication structures has evolved, referred to generically as *multi-party interactions*, involving a simultaneous activity of an arbitrary collection of processes, usually fixed in advance (not varying during computation). Several such proposals are: *scripts* [FHT86], *joint-actions* [BKs83], *teams* and *interactions* in *Raddle* [Fo87], *shared-actions* [RM87] and *compacts* [Ch87]. These primitives enjoy a higher level of abstraction, hiding a lot of low-level details and encourage modular programming and design.

In [EFK88] these proposals are classified into *communication primitives*, and *communication abstractions*.

In this paper, we focus on the former and propose an extension of *cooperating-proofs* to the language *IP* (*Interacting Processes*) which uses the multi-party (synchronous) interaction as its sole interprocess communication and synchronization primitive. This language was introduced in [AF87], where an adequate notion of *fairness* for multi-party interactions is proposed.

While we are witnessing an increasing use of this family of constructs, very little has been done on extending the verification techniques to apply to these constructs. The main point of this paper is showing that cooperating-proofs can be very smoothly extended to the multi-party interactions, with some natural generalization of the concepts involved, but with no need for any essentially different proof-theoretic machinery. It is hoped that these generalizations, which enable formal proofs of (partial-) correctness of programs formulated by means of multi-party

interactions, will encourage an even more extensive usage of such primitives.

Cooperating-proofs belong to the two-leveled proof-systems, in which local proofs for the distinct processes are designed at the first stage, employing some assumptions about the environment's behavior; then, at the second stage, the local proofs are confronted for mutual consistency of all the assumptions made. As is known by now, such two-leveled proofs, which are semi-compositional, are the best possible without extending the correctness properties and the specification language beyond the usual partial-correctness assertions $\{p\}S\{q\}$, where p and q are state predicates.

In Section 2 the IP language and its formal operational semantics are introduced. In Section 3 local proofs are described, while the global *cooperation test* is presented in Section 4.

2. THE LANGUAGE IP (Interacting Processes)

In this section we present a simple mini-programming language, called *IP (Interacting Processes)*, first presented in [AF87]. The language is an abstraction and simplification of languages having multi-party interaction as their interprocess communication and synchronization primitive, and is suitable for focusing on cooperating proofs for partial correctness. In particular, multi-party interactions can be used as *guards*, thereby generalizing both Dijkstra's original guarded commands [Dij 76], which have only boolean guards, and CSP [Ho 78], using synchronous binary communication as guards. This kind of a language was already mentioned in [AFK 87], without details and formal semantics.

A program $P :: [P_1 \| \cdots \| P_n]$ consist of a *concurrent composition* of $n \geq 1$ (fixed n) *processes*, having *disjoint* local states (i.e., no shared variables). A *process* P_i, $1 \leq i \leq n$ consists of a statement S, where S may take one of the following forms:

skip: A statement with no effect on the state.

assignment $x := e$: Here x is a variable local to P_i and e is an expression over P_i's local state. Assignments have their usual meaning of state transformation.

interaction $in\,[\bar{v} := \bar{e}]$: Here in is the *interaction name* and $[\bar{v} := \bar{e}]$ is an optional parallel assignment constituting a *local interaction-body* (where an empty body appears as $in\,[]$). All variables in $\bar{v}$ are local to P_i and different

from each other. The expressions $\bar{e}$ may involve variables *not local* to P_i (belonging to other parties of that interaction). The *participants* of an interaction in, denoted by PA_{in}, consists of all processes which syntactically refer to in in their program. When a process arrives (during execution) to a local interaction-box, it is said to *ready* the corresponding interaction. An interaction in is *enabled* only when *all* its participants have arrived to an interaction point involving in, at which point it can be executed. Its execution implies the execution of all the parallel assignments of all the local interaction-boxes of all participants. Every reference in the right hand side of an assignment in one local box to a variable belonging to another another participating process always means using the initial value, i.e. the one determined by the state at the time the interaction started. In [EFK88] some more complicated interaction-bodies are considered.

Thus, an interaction synchronizes all its participants, and all the bodies are executed in parallel. Upon termination of *all* bodies, each process resumes its local thread of control. Note that if the body $\bar{v}:=\bar{e}$ is empty, the effect is pure synchronization.

sequential composition $S_1; S_2$: First execute S_1; if and when it terminates, execute S_2. We freely use $S_1; \cdots ; S_k$ for any $k \geq 2$.

nondeterministic selection $[\underset{k=1,m}{[]} b_k; in_k[\bar{v}_k:=\bar{e}_k] \rightarrow S_k]$: Here $b_k; in_k[\bar{v}_k:=\bar{e}_k]$ is a *guard*, composed of two parts. The part b_k is a boolean expression over the local state of P_i. The part $in_k[\bar{v}_k:=\bar{e}_k]$ is an *interaction guard*. S_k is any statement. When such a statement is evaluated in some state, the k'th guard is *open* if b_k is true in that state and is readied at that stage. In general, several interactions may be readied by such a statement and are said to be in *conflict*. Executing this kind of statement involves the following steps: evaluate all boolean parts to determine the collection of open guards. If this collection is empty the statement *fails*. Otherwise a guard with an enabled interaction is passed (simultaneously with the execution of all the other bodies in the other parties) and S_k is executed.

nondeterministic iteration $*[\underset{k=1,m}{[]} b_k; in_k\ [\bar{v}_k:=\bar{e}_k] \rightarrow S_k]$: Similar to the choice, but execution terminates once no open guards exist, and the whole procedure is repeated after each execution of a guarded command.

In both the selection and iteration constructs, identically true guards may be omitted. Note that nested concurrency is excluded by this definition.

We now turn to formal definitions of the operational semantics, based on Plotkin's transition scheme [Pl 83]. In [AF87] two different semantics are defined: *serialized* and *overlapping* (compare with a similar distinction in [GFK 84] and [BKS 85]). The distinction between them is crucial to understanding the *hyperfairness* notion suggested in that paper. Since we are interested here only in partial correctness, only the serialized semantics is presented (they are equivalent in this respect, differing on liveness properties only). Throughout we assume some *interpretation* over which computations occur, and leave it implicit. The central characteristic of the semantics is that actions and interactions take place one at a time. A *configuration* $< [S_1 \| \cdots \| S_n], \sigma >$ consists of a concurrent program and a *global* state, assigning values to all variables. A configuration represents an intermediate stage in a computation where S_i is the rest of the program that process P_i has still to execute, while σ is the *current* state at that stage. We stipulate (for facilitating the definition) an *empty* program E (not in the IP language) satisfying the identities $S; E = E; S = S$ for every S. A configuration $< [E \| \cdots \| E], \sigma >$ is a *terminal* configuration. For a state σ, we use the usual notions of a *variant* $\sigma[a/x]$ and $\sigma[\bar{a}/\bar{x}]$.

We now define the *(serialized) transition* relation $\rightarrow$ among configurations.

$$< [S_1 \| \cdots \| S_i \| \cdots \| S_n], \sigma > \rightarrow < [S_1 \| \cdots \| E \| \cdots \| S_n], \sigma > \quad (1)$$

for any $1 \le i \le n$ iff $S_i = skip$, or $S_i = *[\underset{j=1,n_i}{[]} b_j; in_j[\bar{v}_j := \bar{e}_j] \rightarrow T_j]$ and $\neg \underset{j=1,n_i}{\vee} b_j$ holds in σ.

$$< [S_1 \| \cdots \| S_i \| \cdots \| S_n], \sigma > \rightarrow < [S_1 \| \cdots \| E \| \cdots \| S_n], \sigma[\sigma[e]/x] > \quad (2)$$

for any $1 \le i \le n$ iff $S_i = (x := e)$.

$$\begin{aligned} &< [S_1 \| \cdots \| S_{i_1-1} \| S_{i_1} \| S_{i_1+1} \| \cdots \| S_{i_k-1} \| S_{i_k} \| S_{i_k+1} \| \cdots \| S_n], \sigma > \rightarrow \\ &< [S_1 \| \cdots \| S_{i_1-1} \| S'_{i_1} \| S_{i_1+1} \| \cdots \| S_{i_k-1} \| S'_{i_k} \| S_{i_k+1} \| \cdots \| S_n], \sigma' > \end{aligned} \quad (3)$$

iff the following holds: There is an interaction in with a set of participants $PA_{in} = \{P_{i_1}, \cdots, P_{i_k}\}$ (for some $1 \le k \le n$), and for every i s.t. $P_i \in PA_{in}$ one of the following conditions holds:

(a) $S_i = in[\bar{v}_i := \bar{e}_i]$ and $S'_i = E$

(b) $S_i = [\underset{j=1,n_i}{[]} b_j; in_j[\bar{v}_j := \bar{e}_j] \rightarrow T_j]$ and there exists some j, $1 \le j \le n_i$ s.t. b_j holds in σ, $in_j = in$ and $S'_i = T_j$

(c) $S_i = *[\underset{j=1,n_i}{[]} b_j; in_j[\bar{v}_j := \bar{e}_j] \rightarrow T_j]$ and there exists some j, $1 \le j \le n_i$ s.t. b_j holds in σ, $in_j = in$, $S'_i = T_j; S_i$.

Finally, for all these cases, $\sigma'=\sigma[\bar{e}]/\bar{v}]$, with $\bar{v}=\bigcup_{P_i \in PA_{in}} \bar{v}_i$ and $\bar{e}=\bigcup_{P_i \in PA_{in}} \bar{e}_i$.

For any $1 \leq i \leq n$, if

$$< [S_1 \| \cdots \| S_i \| \cdots \| S_n], \sigma > \rightarrow < [S_1 \| \cdots \| S'_i \| \cdots \| S_n], \sigma' > \qquad (4)$$

then

$$< [S_1 \| \cdots \| S_i; T_i \| \cdots \| S_n], \sigma > \rightarrow < [S_1 \| \cdots \| S'_i; T_i \| \cdots \| S_n], \sigma' >$$

For this semantics, we define the following notions.

Definition:

(1) A *(serialized) computation* π of P on an initial state σ is a maximal (finite or infinite) sequence of configurations C_i, $i \geq 0$, such that:

(a) $C_0 = < P, \sigma >$.

(b) For all $i \geq 0$, $C_i \rightarrow C_{i+1}$.

(2) The computation π *terminates* iff it is finite and its last configuration is terminal, and π *deadlocks* iff it is finite and its last configuration is *not* terminal.

(3) An interaction in is *enabled* in a configuration C iff C has one of the forms in clause (3) of the definition of ' $\rightarrow$ ' and all the conditions are satisfied for in.

(4) Two interactions in_1 and in_2 are in *conflict* in a configuration C iff both interactions are enabled in C and they have non-disjoint set of participants, i.e. $PA_{in_1} \cap PAin_2 \neq \emptyset$.

Based on this definitions it is now possible to interpret partial-correctness assertions in the usual way. In the next two sections we present the extension of cooperating proofs to the language IP.

3. LOCAL PROOF-OUTLINES

The basic components of a proof are represented as *proof-outlines*, which were introduced in [OG76]. Our use of them is similar to that in [AFR80]. proof-outlines constitute the local part of a proof, in which separate reasoning about each process is carried out. proof-outlines are to be contrasted with each other at the second stage of a

correctness proof. (A formulation in terms of *proofs from assumptions* appears in [Ap84].) The proof-outlines for IP satisfy the usual partial-correctness axioms and rules, and in addition the following *interaction-box axiom (iba).*

(*iba*) $\{p\}\ in\ [\overline{v}:=\overline{e}]\{q\}$, for *arbitrary* local assertions p and q.

This axiom is the natural generalization of the i/o axioms in [AFR80, LG81] and their counter-parts in the more complicated applications of cooperative proofs mentioned in the introduction.

In so much as the i/o axioms constitute an *assumption* about the state of the communication partner (in some matching communication), the interaction-box axiom may be considered to constitute a *joint-assumption* about the preinteraction states of *all* the participants of some collection of matching local interaction-boxes. The collection of all such joint-assumptions made by all members of PA_{in} are confronted simultaneously in the *cooperation-test* described in the next section. This is the proof-theoretical counterpart of the synchronized nature of a multi-party interaction.

Example: As a simple example, we consider the following applications of the iba axiom.

$$P :: [P_1 :: in\ [x:=x+y+z]\ ||\ P_2 :: in\ [y:=x+y+z]\ ||\ P_3 :: in\ [z:=x+y+z]].$$

Each of the following is a valid proof-outline:

$$\{x \geq 0\}\ in\ [x:=x+y+z]\{x \geq 0\}\ \text{ for } P_1,$$
$$\{y \geq 0\}\ in\ [y:=x+y+z]\{y \geq 0\}\ \text{ for } P_2,$$
$$\{z \geq 0\}\ in\ [z:=x+y+z]\{z \geq 0\}\ \text{ for } P_3.$$

Thus, in the first one, we conclude locally for P_1 a postcondition $x \geq 0$ based on its own precondition and an assumption about the preconditions of P_2 and P_3, about the values of y and z. As it happens, the conjunction of the respective preconditions $y \geq 0$ and $z \geq 0$ indeed satisfies the assumption, as would be revealed during the cooperation test.

As is usual for partial correctness proofs in distributed programs, another kind of an assumption made by a process is that a certain interaction will *never occur* (compare with [AFR80] and [Ap84] for the same phenomenon in *send-receive* communication). This can be seen in the following simple example.

Example: Consider the program Q in Figure 1. Note that $PA_{in_2} = \{P_1, P_2, P_3\}$, *in spite* of the identically $false$ guards (introduced for simplicity, to avoid local computation) of in_2 in P_2 and P_3.

As a part of a proof of $\{true\}Q\{x=y=0\}$, we might have a proof-outline for P_1 with a postcondition $\{x=0\}$. However, since x is modified both in the body of in_1 and in the body of in_2, to preserve the sequential

$Q :: [P_1 :: [in_1[x := y] \rightarrow skip \ \square \ in_2[x := z] \rightarrow skip]$
$\quad \| $
$\quad P_2 :: y := 0; [true \rightarrow in_1[] \ \square \ false \rightarrow in_2[]]$
$\quad \| $
$\quad P_3 :: z := 1; [true \rightarrow in_1[] \ \square \ false \rightarrow in_2[]]$
$\quad].$

Figure 1: Example program Q

nondeterministic branching rule an assumption is needed also about the local interaction-box $in_2[x := z]$, even though this interaction will never occur. We obtain the following (Figure 2) proof-outline for P_1. The interaction in_2 will pass the cooperation-test *vacuously*, by having a *false* precondition in any matching collection of local interaction-boxes.

4. GLOBAL INVARIANTS AND THE COOPERATION TEST

Let P be an IP program and in an interaction in P. Our most basic aim, after having separate proof-outlines for each P_i in P, is to establish

$$(*) \ \{ \underset{P_j \in PA_{in}}{\wedge} pre_j \} \left[in[\bar{v}_{i_1} := \bar{e}_{i_1}] \| \cdots \| in[\bar{v}_{i_k} := \bar{e}_{i_k}] \right] \{ \underset{P_j \in PA_{in}}{\wedge} post_j \}.$$

Here $\{pre_j\}\ in[\bar{v}_j := \bar{e}_j]\{post_j\}$ is taken from the local proof-outline of P_j, $P_j \in PA_{in}$, for a syntactically-matching collection of local interaction-boxes. The correctness assertion (*) captures the operational synchronous nature of an interaction.

$P_1 :: \{true\}$
$\quad [in_1[x := y]\{x=0\} \rightarrow skip \ \{x=0\}$
$\quad \square$
$\quad in_2[x := z]\{x=0\} \rightarrow skip \ \{x=0\}]$
$\quad] \{x=0\}$

Figure 2: A proof-outline for P_1

The first step is to introduce the *(global) interaction axiom (gia)*, the natural generalization of the *communication axiom* of [AFR80] and [LG81].

$(gia)\ \{p^{\overline{v}}_{\overline{e}}\}[\ \|_{P_j \in PA_{in}}\ in[\overline{v}_j := \overline{e}_j]]\{p\}$, where $\overline{v} = \bigcup_{P_j \in PA_{in}} \overline{v}_j$ and $\overline{e} = \bigcup_{P_j \in PA_{in}} \overline{e}_j$.

This axiom captures the operational semantics of an interaction as a parallel (atomic) execution of all the parallel assignments is the local interaction-boxes. This is the usual axiom for multiple assignments [LG81], with two levels of multitude being collapsed: One within a local interaction-box and another among concurrent local interaction-boxes. The syntactic constraint of all variables in $\overline{v}$ being different follows from the formation rules of local interaction-boxes in IP and from process state disjointness.

Example: Returning to the program P in a previous example, an application of the interaction-axiom and the rule of consequence immediately yield

$$(**)\ \{x \geq 0 \wedge y \geq 0 \wedge z \geq 0\}\Big[\ in[x := x+y+z\,] \,\|\, in[y := x+y+z\,] \,\|\, in[z := x+y+z\,]\Big]\ \{x \geq 0 \wedge y \geq 0 \wedge z \geq 0\}.$$

We might also prove a stronger partial-correctness assertion, about P, with the same precondition and the postcondition $x=y=z \geq 0$. To obtain this proof, we have to appeal also to the usual *substitution rule* of [Go75] (see also [Ap84] for its use for cooperation proofs). The problem is that we need to "freeze" initial values of variables, to which local interaction-boxes have access, but local assertions do not. We modify the proof outlines for that example.

The stronger proof-outlines for the program P above are:

$$\{x = a \geq 0\}\ in[x := x+y+z\,]\{x = a+b+c \geq 0\}\ \ for\ P_1,$$
$$\{y = b \geq 0\}\ in[y := x+y+z\,]\{y = a+b+c \geq 0\}\ \ for\ P_2,$$
$$\{z = c \geq 0\}\ in[z := x+y+z\,]\{z = a+b+c \geq 0\}\ \ for\ P_3.$$

Since

$$x = a \geq 0 \wedge y = b \geq 0 \wedge z = c \geq 0 \ \Rightarrow\ x+y+z = a+b+c \geq 0$$

An application of the global interaction-axiom and the consequence rule yields

$$\{x = a \geq 0 \wedge y = b \geq 0 \wedge z = c \geq 0\}\ P\ \{x = a+b+c = y = z \geq 0\}.$$

By weakening the postcondition, we finally get

$$\{x = a \geq 0 \wedge y = b \geq 0 \wedge z = c \geq 0\}\ P\ \{x = y = z \geq 0\}.$$

Since none of a, b and c is free either in P or in the postcondition, we can substitute $a \mid x$, $b \mid y$, $c \mid z$, to obtain

$$\{x \geq 0 \wedge y \geq 0 \wedge z \geq 0\}\ P\ \{x = y = z \geq 0\}.$$

Without substitution, we could obtain an even stronger postcondition, "remembering" the summation effect of the program.

□

However, as in the case of communicating processes [AFR80], things are a little more complicated. The main problem is the identification of semantically-matching interactions. So, similarly to the communicating processes case, we introduce *auxiliary variables* [OG76, AFR80], that carry the additional information needed to express the conditions for such a semantic matching, which happen not to be present in the state. In addition, we introduce a *global invariant* and *bracketed sections*, where the invariant has to hold before and after every execution of a semantically-matching tuple of bracketed sections. The global invariant may refer to the variable in *all* processes. While its main role is the one stated above, in actual proofs it can be used to propagate *any* global information.

We now slightly extend the bracketing notion of communicating processes.

Definition:

(1) A process P_i is *bracketed* iff the brackets ' < ' and ' > ' is interspersed in its body in such a way that each *bracketed section* $< S >$ is of the form $S_1; in[\overline{v}:=\overline{e}]; S_2$, for some local S_1 and S_2 (i.e., without interaction boxes, possibly empty).

(2) A collection of local interaction-boxes $\{in_i[\overline{v}_i:=\overline{e}_i] \mid i \in A\}$ *syntactically-match* iff for some interaction in, $\bigwedge_{i \in A} in_i = in$ holds, where $in_i[\overline{v}_i:=\overline{e}_i]$ occurs in process P_i. Furthermore, $A = PA_{in}$.

(3) A collection $<S_i>$, $i \in A$, of bracketed sections *syntactically match* iff the collection of respective local interaction -boxes occurring within them syntactically match.

(4) An *outline section* $\{p\}<S>\{q\}$ is a bracketed section with its precondition and postcondition taken from a local proof-outline of some process. Outline sections syntactically-match when their respective bracketed sections match.

□

As for communicating processes, the bracketing suggest a slightly different operational semantics, where the grain of atomicity is a semantically matching pair of bracketed sections. As far as partial correctness is concerned, this is equivalent to the original semantics, and enables updating auxiliary variables together with visible effects in an interaction. For an explanation of this equivalence in term of temporal logic see [KP87]. This approach extends to IP as well, taking as atomic steps semantically matching collections of local interaction-boxes.

We now introduce a global invariant I, to be preserved by a program P when executed with this coarser grain of interleaving, induced by the bracketed sections. The invariant has free variables whose values are modifiable only *within* bracketed sections.

Definition:

(1) A syntactically matching collection of outline sections $\{pre_i\} <S_i> \{post_i\}$, $1 \le i \le n$ *cooperates* w.r.t. the invariant I iff $\{ \underset{i=1,m}{\wedge} pre_i \wedge I \} <S_1 \| \cdots \| S_n> \{ \underset{i=1,m}{\wedge} post_i \wedge I \}$ can be proved.

(2) For an IP program $P :: [\underset{i=1,n}{\|} P_i]$, proof-outlines $\{p_i\} P_i \{q_i\}$, $1 \le i \le n$ *cooperate* w.r.t. I iff every syntactically matching collection of outline sections cooperate w.r.t. I.

□

To establish the condition in (1), one can use the global interaction-axiom and *formation rules* similar to the ones in [AFR80] or [Ap84].

We now can introduce the usual *concurrent composition (cc)* meta-rule, which is the *same* as for communicating processes., but with the extended definition of cooperation among proofs.

Let $P :: [P_1 \| \cdots \| P_n]$ be an IP program and let $\{p_i\}P_i\{q_i\}$, $1 \le i \le n$, be valid local proof-outlines.

$$(cc) \quad \frac{\{p_i\}P_i\{q_i\},\ 1 \le i \le n,\ cooperate\ w.r.t.\ I}{\{ \underset{i=1,n}{\wedge} p_i \wedge I \}\ P\ \{ \underset{i=1,n}{\wedge} q_i \wedge I \}}.$$

It is interesting to note that the concurrent composition rule remains intact, and is valid for many situations of cooperation of local outline. It might be interesting to find an abstract characterization of cooperation that will have (cc) as a valid rule. No such characterization exists to date.

Finally, one uses the usual *auxiliary variables* rule ([OG76], [AFR80]) to get rid of the auxiliary variables and assingments introduced for the sake of the proof only.

Acknowledgements

I wish to thank Shmuel Katz for useful discussions. This work was initiated during a summer visit to MCC, July-August 1987 and then continued during consulting with MCC. The continuation at the technion was also partially supported by the Foundation for Research in Electronics, Computers and Communications administered by the Israeli Academy of Sciences and Humanities and by the Fund for the Promotion of Research in the Technion.

References:

[AF 87] P.C. Attie, N. Francez: "Fairness and hyperfairness in multiparty interactions", MCC-STP TR 356-87, August 1987. A revised and corrected version, with Orna Grumberg as an additional coauthor, was submitted for publication.

[AFK 87] K.R. Apt, N. Francez, S. Katz: "Appraising Fairness in Distributed Languages", proc. 14th ACM-POPL symposium, Munich, Germany, Jan. 1987.

[AFR 80] K.R. Apt, N. Francez, W.P. de Roever: "A proof system for communicating sequential processes", ACM-TOPLAS 2, 3: 359-380, July 1980.

[Ap 84] K.R. Apt: "Proving correctness of CSP programs - a tutorial", Proc. int. summer school "Control flow and data flow: concepts in distributed programming", Springer-Verlag (M. Broy - ed.), NATO ASI series F, 1984.

[dB 86] F.S. deBoer: "A proof rule for process-creation", proc. 3rd working conf. on Formal description of programming concepts, Ebberup, Denmark, August 1986.

[Bh 78] P. Brinch-Hansen: "Distributed Processes - a concurrent programming concept", CACM 21, 1978, pp. 934-941.

[BKS 83] R.J. Back, R. Kurki-Suonio: "Decentralization of Process Nets With Centralized Control", proc. 2nd ACM-PODC, Montreal, Canada, August 1983.

[BM 82] H. Barringer, I. Mearns: "Axioms and proof rules for Ada tasks", IEE PROC.m vol. 129, Pt. E, no. 2, March 1982.

[BKs 85] R.J. Back, R. Kurki-Suonio: "Serializability in Distributed Systems With Handshaking", TR 85-109, CMU, 1985.

[Ch 87] A. Charlesworth: "The Multiway Rendezvous", ACM-TOPLAS 9,2:350-366, July 1987.

[Dij 76] E.W. Dijkstra: **A Discipline of Programming**, Prentice-Hall, 1976.

[EFK 88] M. Evangelist, N. Francez, S. Katz: "Multi-party interactions for interprocess communication and synchronization", MCC/STP TR, 1988. To appear in IEEE-TES, 1989.

[FHT 86] N. Francez, B.T. Hailpern, G. Taubenfeld: "SCRIPT - A Communication Abstraction Mechanism and Its Verification", Science of Computer Programming 6,1, pp. 35-88, Jan. 1986.

[Fo 87] I.R. Forman: "On the Design of Large Distributed Systems", TR STP-098-86 (Rev. 1.0), MCC, Austin, TX, Jan. 1987. A preliminary version presented at the First International Conf. on Computer Languages, Miami, FL, October 1986.

[FF 88] L. Fix, N. Francez: "Proof rules for dynamic process creation and destruction", Technion, April 1988, submitted for publication.

[GFK 84] O. Grumberg, N. Francez, S. Katz: "Fair Termination of Communicating Processes", 3rd ACM-PODC Conference, Vancouver, BC, Canada, August 1984.

[Go 75] G.A. Gorelick: "A complete axiomatization system for proving assertions about recursive and nonrecursive programs", TR 76, Dept. of Computer Science, University of Toronto, 1975.

[GR 84] R. Gerth, W.P. deRoever: "A proof system for concurrent Ada programs", SCP 4, 1984, 159-204.

[GR 86] R. Gerth, W.P. deRoever: "Proving monitors revisited - a first step towards verifying object oriented systems", Fundamenta Informatica IX, 1986.

[GRR 82] R. Gerth, W.P. deRoever, M. Roncken: "Procedures and concurrency- a study in proof", proc. 5th ISOP, 1982. LNCS 137, Springer-Verlag, 1982.

[Ho 78] C.A.R. Hoare: "Communicating Sequential Processes", CACM 21,8, pp. 666-678, August 1978.

[HR 86] J. Hooman, W. P. deRoever: "The quest goes on - a survey of proofsystems for partial correctness for CSP", EUT TR 86-WSK-01, Eindhoven U., January 1986.

[KP88] S. Katz, D. Peled: "Interleaving sets temporal logic", proc. 6th ACM-PODC, Vancouver, BC, August 1987. To appear in TCS.

[LG 81] G. Levin, D. Gries: "Proof techniques for communicating sequential processes", ACTA INFORMATICA 15: 281-302, 1981.

[OG 76] S. Owicki, D. Gries: "Verifying properties of parallel programs: An axiomatic approach", CACM 19, 5: 279-286, August 1976.

[Pl 83] G.D. Plotkin: "An Operational Semantics for CSP", TC.2 working group conference on the formal description of programming concepts, Garmisch Partenkierchen, (D. Biorner, ed.), North Holland, 1983.

[dR85] W.P. deRoever: "The cooperation test: a syntax-directed verification method", in: **Logics and models of concurrent systems,** (K.R. Apt - ed.), Nato ASI series F vol. 13, Springer Verlag, 1985.

[RM 87] S. Ramesh, S.L. Mehndiratta: "A methodology for developing distributed programs", IEEE-TSE vol. SE-13,8: 967-976, Aug. 1987.