Formal Justification of a Proof System for Communicating Sequential Processes

KRZYSZTOF R. APT

Université Paris 7, France

Abstract. In a previous paper a proof system dealing with partial correctness of communicating sequential processes was introduced. Soundness and relative completeness of this system are proved here. It is also indicated in what way the semantics and the proof system can be extended to deal with the total correctness of the programs.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs*

General Terms: Languages, Theory

Additional Key Words and Phrases: Partial correctness, CSP, operational semantics, cooperating proofs, soundness, relative completeness, total correctness

1. *Introduction*

In Apt et al. [2] we introduced a proof system in which partial correctness and deadlock freedom of communicating sequential processes (see Hoare [8]) can be proved. This system is an appropriate extension of the usual Hoare proof system which takes care of the special features of CSP programs. The system is a bit unusual in that some of the axioms, visibly the input and output axioms, allow us to deduce *any* postassertion after an I/O command. Even though the soundness of this system should be intuitively clear once one has grasped the main ideas behind the system, it is instructive to provide a formal proof of it.

One of the aims of this paper, which can be viewed as a formal justification of [2], is to present such a proof. We also prove the completeness of the system relative to the set of all sentences true in the standard model of Peano arithmetic. These results are counterparts of the corresponding results concerning proof systems for parallel programs and proved in Owicki [11, 12]. In fact, both the soundness and completeness results follow the same line of reasoning. The completeness proof is structured in a way similar to the corresponding proof given in [1], concerning Owicki's proof system for parallel programs.

Even though the results we prove here concern partial correctness only, it should be clear how to modify them to deal with deadlock freedom. The results of this paper can be extended in two ways. First, by an easy modification of the argument in [1] we can show that in the correctness proofs we can restrict ourselves to recursive (i.e., effectively computable) assertions. Second, by "parameterizing" the loop invariants

Author's address: LITP, Université Paris 7, 2 Place Jussieu, 75251 Paris, France.

with an integer counter we obtain a proof system which can be used to prove total correctness of communicating sequential processes and formally studied. We discuss this issue in the last section of the paper.

The paper is organized as follows. In Section 2 we introduce a fragment of CSP which we subsequently consider. In Section 3 we define an operational semantics for the language, and in Section 4 we discuss the proof system considered for CSP. The soundness of the system is proved in Section 5, and relative completeness in Section 6. Finally, in Section 7 we extend the notions considered here to the case of total correctness.

Knowledge of [2] is not required to follow the proofs presented here. However, it is strongly suggested that the reader become acquainted with the intuitive ideas standing behind the system, which are presented in [2, Sec. 2]. Also, for a proper understanding of the definitions introduced in the last section, knowledge of [2, Sec. 4] is needed.

One important aspect of the fragment of CSP considered has been left out of the considerations. It is the loop exit convention according to which an I/O guarded repetition command can be also exited when all processes addressed in I/O guards whose Boolean part is true have terminated (see [2, 8]). This omission was strongly urged by both referees of the paper. In the original version of the paper we proved the soundness of the proof system in the case in which this aspect was incorporated. We also indicated there that contrary to the claim stated in [2], the proof system there considered is not complete, as there is no way to prove the valid formula,

$$\{\mathbf{true}\}*[\mathbf{true},\ P_2?x \to \mathbf{skip}]\,\|\,*[\mathbf{true},\ P_1?y \to \mathbf{skip}]\{\mathbf{false}\}.$$

An omission of the loop exit convention results in simpler proofs dealing with one proof system only.

2. *Preliminaries*

The aim of this section is to introduce a fragment of CSP that is a subset of the one originally defined in [2]. We also introduce semantical notions and fix notation.

Throughout the paper we fix an arbitrary first-order language L with equality. Its formulas are called *assertions* and are denoted by the letters p, q, r. Simple variables are denoted by the letters x, y, z, and expressions by the letter t; $p[t/x]$ stands for a *substitution* of t for all free occurrences of x in p.

We consider here programs written in a subset of CSP whose expressions are built up from nonlogical symbols of L. By a *parallel program* $P_1\|\cdots\|P_n$ we mean a parallel composition of *component programs* or *processes*. Each process is a sequential program built up from atomic statements using statement constructors. By an atomic statement we mean here assignment $x := t$, the **skip** statement, or an *I/O command*. An I/O command can be either an *input statement* $P_j?x$ or an *output statement* $P_j!t$. The letters α, β are used to denote I/O commands.

Processes or statements are denoted by the letters P, R, S. I/O commands are usually considered in the context of a parallel composition. If this is the case we say that an I/O command $P_j?x$ (or $P_j!t$) *refers* to P_j or to the jth process (or *addresses* P_j) *even* if the jth process in the parallel composition is denoted by another letter. We say that the I/O commands α and β *match* if α is taken from the ith process and β from the jth process, α refers to the jth process, β refers to the ith process, and one of α, β is an input and the other an output statement.

Boolean expressions are denoted by the letters b, c. We allow Boolean constants **true** and **false**. Statements are built up using the composition operation ";" and

allowing *alternative command* $[\Box(j = 1, \ldots, m)\ b_j \to R_j]$ and *repetitive command* $*[\Box(j = 1, \ldots, m)\ b_j \to R_j]$.

A variable x occurring in a program S is *subject to change* if it occurs on a left-hand side of an assignment within S or in an input statement from S. By definition, CSP processes are *disjoint*, which means that a parallel composition $P_1 \| \cdots \| P_n$ is syntactically correct if for $i, j = 1, \ldots, n$ no variable subject to change in P_i occurs in P_j $(i \neq j)$. We also assume that in the context of parallel composition only other component programs are referred to in I/O commands. All programs considered are assumed to be syntactically correct. A semantics of these programs is defined formally in the next section. For an informal definition of their semantics the reader is referred to [8].

By a *correctness formula* we mean a construct of the form $\{p\}\, S\, \{q\}$ where p, q are formulas of L and S is a CSP program or a fragment of it. We denote correctness formulas by the letters ϕ, ψ.

An *interpretation* of L consists of a nonempty domain and assigns to each nonlogical symbol of L a relation or function over its domain of appropriate arity and kind. The letter J stands for an interpretation.

Given an interpretation J, by a *state* we mean a function assigning to all variables of L values from the domain of the interpretation. States are denoted by the letters σ, τ. The notions of a value of an expression t in a state σ and truth of a formula p in a state σ (written as $\models_J p(\sigma)$) are defined in the usual way. A formula is *true under J*, written as $\models_J p$, if $\models_J p(\sigma)$ holds for all states σ. By Tr_J we mean the set of all formulas of L true under J. For a state σ and program S we define $\sigma \restriction S$ to be the restriction of σ to the domain $\mathrm{Var}(S)$, which is the set of all variables occurring in S.

The I/O guarded selection command $[\Box(j = 1, \ldots, m)\ b_j, \alpha_j \to R_j]$ and the *I/O guarded repetition command* $*[\Box(j = 1, \ldots, m)\ b_j, \alpha_j \to R_j]$ are not included in the subset considered here. These constructs were allowed in [2] and play an important role in the original definition of CSP given in [8]. They are, however, omitted here for the following reasons. They can be defined in the fragment considered here as $[\Box(j = 1, \ldots, m)\ b_j, \alpha_j \to R_j]$ is semantically equivalent to $[\Box(j = 1, \ldots, m)\ b_j \to \alpha_j; R_j]$ and $*[\Box(j = 1, \ldots, m)\ b_j, \alpha_j \to R_j]$ is semantically equivalent to $*[\Box(j = 1, \ldots, m)\ b_j \to \alpha_j; R_j]$ (the latter due to the fact that the loop exit convention is not considered here). A consequence of these equivalences is that the proof rules dealing with the I/O guarded commands can be straightforwardly derived from the proofs rules dealing with the equivalent constructs. As a result, the soundness and completeness proofs presented here hold equally well for an appropriate extension of the proof system dealing with the I/O guarded commands.

It should be stressed that the equivalence of $[\Box(j = 1, \ldots, m)\ b_j, \alpha_j \to R_j]$ and $[\Box(j = 1, \ldots, m)\ b_j \to \alpha_j, R_j]$ holds in the case of partial correctness only as the latter construct introduces an additional possibility of deadlock.

It should be also noted that the loop exit convention is also omitted in [10], where a proof system similar to this of [2] is presented.

3. *Semantics*

In this section we define an operational semantics for the CSP programs we consider. For each interpretation J and program R we want to define a binary relation on states $M_J(R)$ which gives the input–output semantics of program R under the interpretation J.

This semantics makes use of the Hennessy–Plotkin [7] idea of considering the "$\to$" relation between pairs consisting of a program and state.

Assume a given interpretation J. We first define the relation

$$\langle S, \sigma\rangle \to \langle S', \tau\rangle$$

for S and S' subprograms of a process. The intuitive meaning of this relation is: executing S alone for one step in a state σ can nondeterministically lead to a state τ with S' being the remainder of S still to be executed. It is convenient to allow the empty program E. Then S' is E if S terminates in τ. We assume that for any S, E; $S = S$; $E = S$. We also assume the meaning of assignment to be known.

We define the above relation by the following clauses:

(i) $\langle \mathbf{skip}, \sigma\rangle \to \langle E, \sigma\rangle$;
(ii) $\langle x := t, \sigma\rangle \to \langle E, \mathscr{M}_J(x := t)(\sigma)\rangle$;
(iii) $\langle [\Box(j = 1, \ldots, m)\ b_j \to R_j], \sigma\rangle \to \langle R_k, \sigma\rangle$ if $\models_J b_k(\sigma)$ $(1 \le k \le m)$;
(iv) $\langle *[\Box(j = 1, \ldots, m)\ b_j \to R_j], \sigma\rangle \to \langle R_k; *[\Box(j = 1, \ldots, m)\ b_j \to R_j], \sigma\rangle$ if $\models_J b_k(\sigma)$ $(1 \le k \le m)$;
(v) $\langle *[\Box(j = 1, \ldots, m)\ b_j \to R_j], \sigma\rangle \to \langle E, \sigma\rangle$ if $\models_J \neg b_j(\sigma)$ for $j = 1, \ldots, m$.

By a *history* we mean a sequence of *records of communications* (r.o.c.). A record of communication is a term introduced in [5]. It stands for a triple (a, i, j) which is associated with a communication between processes P_i and P_j: a is the value sent by P_i to P_j. The empty history is denoted by ϵ. We use the letter h with a possible subscript to denote a history. $h_1 \circ h_2$ denotes concatenation of histories h_1 and h_2.

We now define the relation

$$\langle S_1 \| \cdots \| S_n, \sigma\rangle \to_k^h \langle S_1' \| \cdots \| S_n', \tau\rangle$$

for parallel programs $S_1 \| \cdots \| S_n$ and $S_1' \| \cdots \| S_n'$, history h, and a natural number $k \ge 0$. The intuitive meaning of this relation is: executing $S_1 \| \cdots \| S_n$ in a state σ can lead in k steps to a state τ with h recording all communications that took place and $S_1' \| \cdots \| S_n'$ being the remainder of $S_1 \| \cdots \| S_n$ still to be executed.

The relation is defined by the following clauses:

(1) $\langle S_1 \| \cdots \| S_n, \sigma\rangle \to_0^\epsilon \langle S_1 \| \cdots \| S_n, \sigma\rangle$.
(2) If $\langle S_i, \sigma\rangle \to \langle S_i', \tau\rangle$ $(1 \le i \le n)$, then

$$\langle S_1 \| \cdots \| S_n, \sigma\rangle \to_1^\epsilon \langle S_1 \| \cdots \| S_{i-1} \| S_i' \| S_{i+1} \| \cdots \| S_n, \tau\rangle.$$

In this case the progress in execution took place due to the execution of S_i alone.

(3) If $S_i \equiv P_j!t$ and $S_j \equiv P_i?x$ $(1 \le i, j \le n)$, then

$$\langle S_1 \| \cdots \| S_n, \sigma\rangle \to_1^{(a,i,j)} \langle S_1' \| \cdots \| S_n', \mathscr{M}_J(x := t)(\sigma)\rangle,$$

where a is the value of t in the state σ, $S_k' \equiv S_k$ for $k \neq i, j$, and $S_i' \equiv S_j' \equiv E$.

In this case the progress in execution took place by performing the communication between the ith and jth process. The effect of this communication is equal to the execution of the corresponding assignment statement.

(4) If $\langle S_1 \| \cdots \| S_n, \sigma\rangle \to_k^h \langle S_1' \| \cdots \| S_n', \tau\rangle$, then for any S,

$$\langle S_1 \| \cdots \| S_i; S \| \cdots \| S_n, \sigma\rangle \to_k^h \langle S_1' \| \cdots \| S_i'; S \| \cdots \| S_n', \tau\rangle.$$

This clause handles the case of the composition of programs.

(5) If $\langle \bar{S}, \sigma\rangle \to_{k_1}^{h_1} \langle \bar{S}_1, \sigma_0\rangle$ and $\langle \bar{S}_1, \sigma_0\rangle \to_{k_2}^{h_2} \langle \bar{S}_2, \tau\rangle$, then

$$\langle \bar{S}, \sigma\rangle \to_{k_1+k_2}^{h_1 \circ h_2} \langle \bar{S}_2, \tau\rangle.$$

Finally we define the meaning of a parallel program $S_1 \| \cdots \| S_n$ by putting

$$\mathcal{M}_J(S_1 \| \cdots \| S_n)(\sigma) = \{\tau: \text{for some history } h \text{ and } k \geq 0, \langle S_1 \| \cdots \| S_n, \sigma\rangle \to_k^h \langle \underbrace{E \| \cdots \| E}_{n \text{ times}}, \tau\rangle\}.$$

Note that we did not provide a separate clause for the case of I/O commands. Their semantics is obtained as a special case of the last definition. We have $\mathcal{M}_J(\alpha)(\sigma) = \emptyset$, which should be interpreted as a statement that an execution of an I/O command alone does not terminate properly.

The use of histories in the above definition is needed only for the completeness proof. The explicit counting of steps is not needed in the definition either. It is however useful in both the soundness and completeness proofs.

For a parallel program P and assertions p, q we say that $\{p\}P\{q\}$ is *true under* J ($\models_J \{p\}P\{q\}$) if

$$\forall \sigma, \tau\ [(\models_J p(\sigma) \wedge \tau \in \mathcal{M}_J(P)(\sigma)) \to \models_J q(\tau)].$$

In the subsequent considerations we shall sometimes use the notion of *computation.* By a computation we mean here a sequence of elementary steps. Each step is associated with an execution of a **skip** or assignment statement, a loop exit, evaluation of a Boolean expression, or execution of a communication.

4. *Proof System*

We now present a proof system for the fragment of CSP considered. This system was introduced in [2], to which the reader is referred for additional information about the need and motivation for various axioms and proof rules. Also, example proofs can be found there. A different but similar proof system was independently introduced in [10]. Both proof systems derive from related work on proof systems for parallel programs by Lamport [9] and Owicki and Gries [13, 14].

To reason within the system about CSP programs, one has first to provide proofs for component processes and then to deduce properties of a parallel program by analyzing the proofs for components. First we give axioms and proof rules needed to generate proofs for component processes. Let C stand for the proof system consisting of the following axioms and proof rules:

A1. *Input*

$$\{p\}P_i?x\{q\}.$$

A2. *Output*

$$\{p\}P_i!t\{q\}.$$

A3. *Assignment*

$$\{p[t/x]\}x := t\{p\}.$$

A4. *Skip*

$$\{p\}\textbf{skip}\{p\}.$$

R1. *Alternative command*

$$\frac{\{p \wedge b_j\}R_j\{q\}_{j=1,\ldots,m}}{\{p\}[\Box(j = 1, \ldots, m)\ b_j \to R_j]\{q\}}.$$

R2. *Repetitive command*

$$\frac{\{p \wedge b_j\}R_j\{p\}_{j=1,\ldots,m}}{\{p\}*[\Box(j = 1, \ldots, m)\ b_j \to R_j]\{p \wedge \bigwedge_{j=1}^{m} \neg b_j\}}.$$

R3. *Composition*

$$\frac{\{p\}S_1\{q\},\ \{q\}S_2\{r\}}{\{p\}S_1;\ S_2\{r\}}.$$

R4. *Consequence*

$$\frac{p \to p_1,\ \{p_1\}S\{q_1\},\ q_1 \to q}{\{p\}S\{q\}}.$$

If $\mathscr{A}$ is a set of assertions and G a proof system, we write $\mathscr{A} \vdash_G \phi$ to indicate that there exists a proof in G of ϕ which uses as assumptions for the consequence rule assertions from $\mathscr{A}$.

The subsequent metarule for comparing the proofs for component processes refers to a special form of these proofs. This special form, called a *proof outline* (see [14]), is characterized by the fact that each substatement S of the component program is preceded and succeeded by an assertion, pre(S) and post(S), respectively. These pre- and postassertions satisfy certain properties which are listed in the following lemma.

LEMMA 1. *Let J be an interpretation, and let S be a component program. Then $Tr_J \vdash_C \{p\}S\{q\}$ iff there exist assertions pre(R) and post(R) for all subprograms R of S such that the following formulas are true under J:*

(*i*) $p \to pre(S), post(S) \to q$;
(*ii*) $pre(R) \to post(R)[t/x]$ *if* R *is* $x := t$;
(*iii*) $pre(\mathbf{skip}) \to post(\mathbf{skip})$;
(*iv*) $pre(R) \to pre(R_1), post(R_1) \to pre(R_2), post(R_2) \to post(R)$ *if* R *is* $R_1; R_2$;
(*v*) $pre(R) \wedge b_j \to pre(R_j), post(R_j) \to post(R)$, *for* $j = 1, \ldots, m$, *if* R *is* $[\Box(j = 1, \ldots, m)\ b_j \to R_j]$;
(*vi*) $pre(R) \wedge b_j \to pre(R_j), post(R_j) \to pre(R), pre(R) \wedge \bigwedge_{j=1}^{m} \neg b_j \to post(R)$, *for* $j = 1, \ldots, m$, *if* R *is* $*[\Box(j = 1, \ldots, m)\ b_j \to R_j]$.

PROOF. The proof proceeds by induction on the struction of S. It is a counterpart of a corresponding lemma from [11]. The details are straightforward and are left to the reader. □

This lemma shows that when we are discussing the proofs for component programs, it will be sufficient to restrict attention to assertions pre(R) and post(R) satisfying the conditions listed above.

The next concept we need is that of *bracketing.*

Definition. A process P_i is *bracketed* if the brackets "⟨" and "⟩" are interspersed in its text so that

(i) for each program section ⟨S⟩ (called a *bracketed section*), S is of the form $S_1; \alpha; S_2$ where S_1 and S_2 do not contain any I/O statements, and
(ii) all I/O statements are bracketed.

S_1 and S_2 do not need to appear in the definition of bracketing. The completeness proof shows that it is sufficient to consider bracketed sections of the form ⟨$\alpha; S$⟩, where S is an assignment statement. The reason for introducing brackets is to delimit program sections within which the global invariant I need not necessarily hold.

With each proof of $\{p\}P_1 \| \cdots \| P_n\{q\}$ we now associate a global invariant I and appropriate bracketing. The proof rule concerning parallel composition has the following form.

R5. *Parallel composition*

$$\frac{\text{proofs of } \{p_i\}P_i\{q_i\},\ i = 1, \ldots, n, \text{ cooperate}}{\{p_1 \wedge \cdots \wedge p_n \wedge I\}P_1 \| \cdots \| P_n\{q_1 \wedge \cdots \wedge q_n \wedge I\}},$$

provided no variable free in I is subject to change outside a bracketed section.

We now define when proofs cooperate. Assume a given bracketing of $P_1 \| \cdots \| P_n$ (to which we referred in the clause concerning the free variables of I). We say that two bracketed sections $\langle S_1 \rangle$ and $\langle S_2 \rangle$ *match* if they contain matching I/O commands.

Definition. The proofs of the $\{p_i\}P_i\{q_i\}$ $(i = 1, \ldots, n)$ *cooperate* if

(i) the assertions used in the proof of $\{p_i\}P_i\{q_i\}$ have no free variables subject to change in P_j $(i \neq j)$, and
(ii) $\{\text{pre}(S_1) \wedge \text{pre}(S_2) \wedge I\}S_1 \| S_2\{\text{post}(S_1) \wedge \text{post}(S_2) \wedge I\}$ holds for all matching pairs of bracketed sections $\langle S_1 \rangle$ and $\langle S_2 \rangle$.

To establish the second clause of cooperation, we use the following additional axioms and proof rules.

A5. *Communication*

$$\{\textbf{true}\}P_i?x \| P_j!t\{x = t\},$$

provided $P_i?x$ and $P_j!t$ are taken from P_j and P_i, respectively.

A6. *Preservation*

$$\{p\}S\{p\},$$

provided no free variable of p is subject to change in S.

R6. *Formation*

$$\frac{\{p\}S_1;\ S_3\{p_1\},\ \{p_1\}\alpha \| \bar{\alpha}\{p_2\},\ \{p_2\}S_2;\ S_4\{q\}}{\{p\}(S_1;\ \alpha;\ S_2) \| (S_3;\ \bar{\alpha};\ S_4)\{q\}},$$

provided α and $\bar{\alpha}$ match, S_1, S_2, S_3, and S_4 do not contain any I/O commands, and no variable in S_1; S_2 is subject to change in S_3; S_4, and vice versa.

Finally, we need the following three rules which are used not only in the cooperation proofs.

R7. *Conjunction*

$$\frac{\{p\}S\{q\},\ \{p\}S\{r\}}{\{p\}S\{q \wedge r\}}.$$

R8. *Substitution*

$$\frac{\{p\}S\{q\}}{\{p[t/z]\}S\{q\}},$$

provided z does not appear free in S and q.

R9. *Auxiliary variables.* Let AV be a set of variables such that $x \in \text{AV}$ implies that x appears in S' only in assignments $y := t$, where $y \in \text{AV}$. Then if q does not

contain free any variables from AV and S is obtained from S' by deleting all assignments to variables in AV,

$$\frac{\{p\}S'\{q\}}{\{p\}S\{q\}}.$$

This concludes the presentation of our proof system. In the subsequent sections we call this system T.

5. *Soundness*

In this section we prove that the proof system T is sound in the sense of the following theorem.

SOUNDNESS THEOREM. *For any interpretation J and correctness formula ϕ, if $Tr_J \vdash_T \phi$, then $\models_J \phi$.*

PROOF. It is difficult to prove the soundness of T directly, because the rule of parallel composition is in fact a metarule. To resolve this difficulty, we transform T into an equivalent system T' which uses the usual notion of proof and is therefore easier to study.

T' is obtained from T by replacing proof rule R5 by another rule.

Let $\mathrm{VC}(\{p\}P\{q\})$ (verification conditions for $\{p\}P\{q\}$), where P is a process, stand for the list of all assertions listed in conditions (i)–(vi) of Lemma 1. Let $P_1 \| \cdots \| P_n$ be a parallel program. Assume a given bracketing of $P_1 \| \cdots \| P_n$. Consider $\mathrm{VC}(\{p_i\}P_i\{q_i\})$ for $i = 1, \ldots, n$ which satisfy the disjointness property (i.e., assertions from $\mathrm{VC}(\{p_i\}P_i\{q_i\})$ have no free variables subject to change in P_j for $i \neq j$). Let $\mathrm{Coop}(\{p_i\}P_i\{q_i\}_{i=1,\ldots,n}, I)$ stand for the list of all correctness formulas of the form

$$\{\mathrm{pre}(S_1) \wedge \mathrm{pre}(S_2) \wedge I\}S_1 \| S_2\{\mathrm{post}(S_1) \wedge \mathrm{post}(S_2) \wedge I\},$$

where $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are some matching bracketed sections. We assume that the global invariant I satisfies the restriction mentioned in rule R5.

The rule in question has the form

$$\frac{\mathrm{VC}(\{p_i\}P_i\{q_i\})_{i=1,\ldots,n},\ \mathrm{Coop}(\{p_i\}P_i\{q_i\}_{i=1,\ldots,n}, I)}{\{p_1 \wedge \cdots \wedge p_n \wedge I\}P_1 \| \cdots \| P_n\{q_1 \wedge \cdots \wedge q_n \wedge I\}}.$$

We call it the combined rule.

Using Lemma 1, it is easy to prove that T and T' are indeed equivalent. Instead of proving soundness of T we prove soundness of T'.

An axiom which is true under all interpretations will be called *valid*, and a proof rule which preserves truth under all interpretations will be called *sound*.

To prove soundness of T', it is enough to show that all axioms of T' are valid and all proof rules are sound. It follows from the fact that T', in contrast to T, uses the usual notion of proof.

As the next step in preparation of the proof of the Soundness Theorem we introduce the following notion. Let S be a subprogram of a process P. By induction on the structure of P we define a program *after*(S, P). Informally speaking, *after*(S, P) is a remainder of P still to be executed just after the execution of its subprogram S has terminated, and *before*(S, P), defined by

$$\textit{before}(S, P) \equiv S; \textit{after}(S, P),$$

is a remainder of P still to be executed just before the execution of the subprogram S has started.

If $S \equiv P$, then $after(S, P) \equiv E$. Otherwise,

(i) if P is $[\Box(j = 1, \ldots, m)\ b_j \to R_j]$ and S is a subprogram of R_j $(1 \leq j \leq m)$, then

$$after(S, P) \equiv after(S, R_j);$$

(ii) if P is $*[\Box(j = 1, \ldots, m)\ b_j \to R_j]$ and S is a subprogram of R_j $(1 \leq j \leq m)$, then

$$after(S, P) \equiv after(S, R_j);\ P;$$

(iii) if P is $S_1; S_2$, then if S is a subprogram of S_1, then

$$after(S, P) \equiv after(S, S_1);\ S_2,$$

and otherwise,

$$after(S, P) \equiv after(S, S_2).$$

Assume now a given bracketing of a process P, and let R be a substatement of P. We say that R is a *normal* substatement if each bracketed section of P lies either outside or inside of R. In other words, neither the beginning nor the end of R lies within a bracketed section of P.

Assuming now a given bracketing of processes $P_1, \ldots, P_n$, we say that the program $S_1 \| \cdots \| S_n$ is *admissible* if for each $i = 1, \ldots, n$, S_i is either $before(R, P_i)$ or $after(R, P_i)$ for some normal subprogram R of P_i.

For the rest of this section we fix an arbitrary interpretation J. To avoid excessive notation, we shall write $\models \phi$ instead of $\models_J \phi$ and $\mathcal{M}(\cdots)$ instead of $\mathcal{M}_J(\cdots)$.

We now prove the soundness of the combined rule, which is the most complicated case in the proof of the Soundness Theorem. Assume that all formulas occurring in the premise of the rule are true under J. We now wish to prove that the conclusion of the rule is true under J. To this purpose we first prove the following lemma.

LEMMA 2. *Assume that all formulas from* $VC(\{p_i\}P_i\{q_i\})_{i=1,\ldots,n}$, $\mathrm{Coop}(\{p_i\}P_i\{q_i\}_{i=1,\ldots,n}, I)$ *are true under J. Assume that for some states* σ, τ, $k \geq 0$, *history h, and admissible program* $S_1 \| \cdots \| S_n$,

$$\langle P_1 \| \cdots \| P_n, \sigma\rangle \to_k^h \langle S_1 \| \cdots \| S_n, \tau\rangle$$

and

$$\models (p_1 \wedge \cdots \wedge p_n \wedge I)(\sigma).$$

Then for $i = 1, \ldots, n$,

(*i*) *if for some normal R,* S_i *is* $before(R, P_i)$, *then* $\models pre(R)(\tau)$;
(*ii*) *if for some normal R,* S_i *is* $after(R, P_i)$, *then* $\models post(R)(\tau)$;
(*iii*) $\models I(\tau)$ *holds.*

The lemma states that whenever control in each process is outside a bracketed section (which is implied by the admissibility of $S_1 \| \cdots \| S_n$), then the appropriate pre- and postassertions and the global invariant I hold.

PROOF. The proof proceeds by induction on k.

If $k = 0$, then $\sigma = \tau$, and each S_i is $before(P_i, P_i)$. The formulas $p_i \to \mathrm{pre}(P_i)$ for $i = 1, \ldots, n$ are all among premises of the combined rule and so are assumed to be true under J. Hence

$$\models (\mathrm{pre}(P_1) \wedge \cdots \wedge \mathrm{pre}(P_n) \wedge I)(\tau)$$

holds.

Now suppose the assumptions of the lemma hold for some $k > 0$, and that by the induction hypothesis the claim of the lemma holds for all $k' < k$. For some $S_1', \ldots, S_n'$, h_1, h_2, and σ_1,

$$\langle P_1 \| \cdots \| P_n, \sigma \rangle \to_{k-1}^{h_1} \langle S_1' \| \cdots \| S_n', \sigma_1 \rangle \qquad (*)$$

and

$$\langle S_1' \| \cdots \| S_n', \sigma_1 \rangle \to_1^{h_2} \langle S_1 \| \cdots \| S_n, \tau \rangle. \qquad (**)$$

There are two cases to consider.

Case I. The program $S_1' \| \cdots \| S_n'$ is admissible.

The progress in computation $(**)$ took place either

(1) by executing one step of some S_i' or
(2) by executing a communication between some S_i' and S_j'.

We consider these two possibilities in turn.

(1) As a representative case consider the situation when S_i' is *before*(R, P_i), where R is $[\square(j = 1, \ldots, m)\ b_j \to R_j]$. The progress had to take place here by evaluating some b_l $(1 \leq l \leq m)$ to **true**. Clause (iii) from Section 3 applies here, and by the definition of *before*, S_i is *before*(R_l, P_i) and τ is σ_1. By the induction hypothesis, $\vDash \text{pre}(R)(\sigma_1)$. Also $\vDash b_l(\sigma_1)$. The formula $\text{pre}(R) \wedge b_l \to \text{pre}(R_l)$ is one of the premises of the combined rule and so assumed to be true under J. Hence $\vDash \text{pre}(R_l)(\sigma_1)$, and since $\sigma_1 = \tau$, we get $\vDash \text{pre}(R_l)(\tau)$ as desired. The other possibilities for S_i' require similar reasoning.

By the induction hypothesis, $\vDash I(\sigma_1)$. Since both $S_1 \| \cdots \| S_n$ and $S_1' \| \cdots \| S_n'$ are admissible, the progress in computation within S_i took place outside a bracketed section. By the assumption concerning I, none of its free variables has changed its value in the computation $(**)$. So $\vDash I(\tau)$ holds.

(2) Suppose that α and $\bar{\alpha}$ are the matching I/O commands which were executed. Then

$$\begin{array}{lll} S_i' & \text{is} & \textit{before}(\alpha, P_i), \\ S_j' & \text{is} & \textit{before}(\bar{\alpha}, P_j), \\ S_i & \text{is} & \textit{after}(\alpha, P_i), \\ S_j & \text{is} & \textit{after}(\bar{\alpha}, P_j). \end{array}$$

Since both $S_1' \| \cdots \| S_n'$ and $S_1 \| \cdots \| S_n$ are admissible, α and $\bar{\alpha}$ must both constitute a bracketed section. By the induction hypothesis,

$$\vDash (\text{pre}(\alpha) \wedge \text{pre}(\bar{\alpha}) \wedge I)(\sigma_1),$$

and by the definition of $\mathcal{M}$, $\tau \in \mathcal{M}(\alpha \| \bar{\alpha})(\sigma_1)$. The correctness formula

$$\{\text{pre}(\alpha) \wedge \text{pre}(\bar{\alpha}) \wedge I\}\alpha \| \bar{\alpha}\{\text{post}(\alpha) \wedge \text{post}(\bar{\alpha}) \wedge I\},$$

being a premise of the rule, is assumed to be true under J. Hence

$$\vDash (\text{post}(\alpha) \wedge \text{post}(\bar{\alpha}) \wedge I)(\tau),$$

as desired.

Suppose now that $S_l' \equiv S_l$ for some l. Thus $l \neq i$ in case (1) and $l \neq i, j$ in case (2). For some normal R, $\vDash \text{pre}(R)(\sigma_1)$ (or $\vDash \text{post}(R)(\sigma_1)$). By the disjointness property of $\text{VC}(\{p_i\}P_i\{q_i\})$ for $i = 1, \ldots, n$, a progress in computation of one or two processes

cannot affect the corresponding pre- and postconditions related to other processes. Thus $\models \text{pre}(R)(\tau)$ (or $\models \text{post}(R)(\tau)$). This settles case I.

Case II. For all $S'_1, \ldots, S'_n$ satisfying (*) and (**), $S'_1 \| \cdots \| S'_n$ is not admissible.

It is easy to see that this assumption and the admissibility of $S_1 \| \cdots \| S_n$ implies that in state τ some process has just terminated an execution of a bracketed section. Hence at least one communication took place in the computation mentioned in the formulation of the lemma, that is, h is not empty. Let (a, i, j) be the last element of h. For some h_1 we have $h = h_1 \circ (a, i, j)$. It follows that ith and jth processes are in state τ just after an execution of a bracketed section, say R_1 and R_2, respectively. In other words, S_i is *after*(R_1, P_i) and S_j is *after*(R_2, P_j).

We now claim that for some σ_1 and $k_1 \leq k$,

$$\langle P_1 \| \cdots \| P_n, \sigma \rangle \to^{h_1}_{k-k_1} \langle S'_1 \| \cdots \| S'_n, \sigma_1 \rangle$$

and

$$\langle S'_1 \| \cdots \| S'_n, \sigma_1 \rangle \to^{(a,i,j)}_{k_1} \langle S_1 \| \cdots \| S_n, \tau \rangle$$

hold, where

$$\begin{aligned} S'_l &\equiv S_l \quad \text{for} \quad l \neq i, j, \\ S'_i &\equiv \textit{before}(R_1, P_i), \\ S'_j &\equiv \textit{before}(R_2, P_j). \end{aligned}$$

Intuitively, the claim states that there exists a computation with history h which started in state σ and reached state τ in which the last k_1 steps consisted exclusively of executing the above mentioned bracketed sections of P_i and P_j.

The desired computation can be obtained from the original one by deleting from it all (k_1) steps performed by P_i and P_j after reaching the beginning of the corresponding bracketed sections and then appending them while preserving their order at the end of the resulting computation. The computation obtained clearly has history h and also leads to state τ. This is a consequence of the fact that all processes are disjoint and P_i and P_j do not communicate with any P_l $(l \neq i, j)$ after having reached the beginning of the corresponding bracketed sections in the original computation. Thus changing the order of the computation cannot affect its outcome.

σ_1 is the state reached in the constructed computation after performing $k - k_1$ steps. By the induction hypothesis we now have

$$\models (\text{pre}(R_1) \wedge \text{pre}(R_2) \wedge I)(\sigma_1).$$

The rest of the proof is now the same as in case I(2). This concludes the proof of Lemma 2. □

The soundness of the combined rule is now an immediate consequence of Lemma 2. Suppose that for some states σ and τ, $\models (p_1 \wedge \cdots \wedge p_n \wedge I)(\sigma)$ and $\tau \in \mathcal{M}(P_1 \| \cdots \| P_n)(\sigma)$. By Lemma 2, $\models (\text{post}(P_1) \wedge \cdots \wedge \text{post}(P_n) \wedge I)(\tau)$ holds, as *after*(P_i, P_i) is E for $i = 1, \ldots, n$. Now, $\text{post}(P_i) \to q_i$ for $i = 1, \ldots, n$ are all among premises of the combined rule and are assumed to be true under J. Hence $\models (q_1 \wedge \cdots \wedge q_n \wedge I)(\tau)$ holds. Thus we showed that

$$\models (p_1 \wedge \cdots \wedge p_n \wedge I) P_1 \| \cdots \| P_n \{q_1 \wedge \cdots q_n \wedge I\}$$

holds as desired.

Validity of axioms of T' and soundness of other proof rules of T' are straightforward to prove. For example, to show the soundness of the formation rule, it is sufficient to use the reasoning concerning changing the order of the computation steps which was applied in case II in the proof of Lemma 2.

Note also that the somewhat unusual input and output axioms are valid according to the semantics of the I/O commands.

This concludes the proof of the soundness theorem. □

6. *Relative Completeness*

Having proved soundness of T, we now concentrate on the issue of completeness of T. Assume that the underlying language L is a countable first-order extension of the language L_P of Peano arithmetic. Fix an interpretation J_0 of L in the domain of natural numbers which is an extension of the standard interpretation of L_P in natural numbers. We now show that the proof system T is relatively complete with respect to J_0. More precisely, we prove the following theorem.

COMPLETENESS THEOREM. *For any correctness formula ϕ, if $\models_{J_0}\phi$, then $Tr_{J_0} \vdash_T \phi$. Recall that Tr_{J_0} is the set of all formulas of L true under J_0.*

PROOF. In the subsequent considerations we shall need to code finite sequences of natural numbers by natural numbers; $\langle a_1, \ldots, a_n\rangle$ will stand for a code of the sequence $a_1, \ldots, a_n$. If $a = \langle a_1, \ldots, a_n\rangle$, then by definition, $a \circ c = \langle a_1, \ldots, a_n, c\rangle$; $\langle\ \rangle$ denotes the code of the empty sequence. We shall implicitly assume various properties of the functions "$\langle \cdots \rangle$" and "$\circ$," like their definability by expressions of L and injectivity. The proofs of these properties can be found in [15].

As in Section 5, in order to avoid excessive notation we write $\models\phi$, Tr, and $\mathcal{M}(\cdots)$ instead of $\models_{J_0}\phi$, Tr_{J_0}, and $\mathcal{M}_{J_0}(\cdots)$, respectively.

Assume now that

$$\models\{p\}P_1\| \cdots \|P_n\{q\} \tag{1}$$

holds for some program $P_1\| \cdots \|P_n$ and assertions p and q. Let $h_1, \ldots, h_n$ be some fresh variables not occuring in p, $P_1, \ldots, P_n$, or q. We now transform each P_i into another program P_i^* by replacing each command of the form $P_j?x$ $(P_j!t)$ by $\langle P_j?x;\ h_i := h_i \circ \langle x, j, i\rangle\rangle$ $(\langle P_j!t;\ h_i := h_i \circ \langle t, i, j\rangle\rangle)$. This transformation defines a bracketing on $P_1^*\| \cdots \|P_n^*$.

Let $\bar{x}$ be the list of all variables of $P_1\| \cdots \|P_n$, and let $\bar{z}$ be a list of some fresh variables of the same length as $\bar{x}$. Let p' be defined by

$$p' \equiv p \wedge h_1 = \langle\ \rangle \wedge \cdots \wedge h_n = \langle\ \rangle \wedge \bar{x} = \bar{z}.$$

We now prove

$$\mathrm{Tr} \vdash_{T'} \{p'\}P_1^*\| \cdots \|P_n^*\{q\}. \tag{2}$$

Given a history h, let $[h]_i$ denote the subsequence of h consisting of all triples (a, k, l) such that $k = i$ or $l = i$. Intuitively, $[h]_i$ is the part of h which relates to communications involving process P_i. We shall call $[h]_i$ a projection of h on process P_i.

We say that a number y *codes* a history $h = (a_1, i_1, j_1) \circ \cdots \circ (a_k, i_k, j_k)$ if $y = \langle\langle a_1, i_1, j_1\rangle, \ldots, \langle a_k, i_k, j_k\rangle\rangle$.

Let I be a formula of L, and let $\mathcal{F} = \{h_1, \ldots, h_n, \bar{z}\}$ be the set of its free variables

and such that for all states σ the following holds:

$$\models I(\sigma) \leftrightarrow \exists \sigma', \tau, S_1', \ldots, S_n', k, h[\models p'(\tau), \langle P_1^* \| \cdots \| P_n^*, \tau\rangle \rightarrow_k^h \langle S_1' \| \cdots \| S_n', \sigma'\rangle, \sigma'(u) = \sigma(u) \text{ for } u \in \mathscr{F}, S_1' \| \cdots \| S_n' \text{ is admissible, and for } i = 1, \ldots, n, \sigma(h_i) \text{ codes } [h]_i].$$

Now let R be a subprogram of P_i^* $(1 \leq i \leq n)$. Let pre(R) and post(R) be the assertions whose free variables are those of P_i^* and such that for all states σ the following hold:

$$\models \text{pre}(R)(\sigma) \leftrightarrow \exists \sigma', \tau, S_1', \ldots, S_n', k, h[\models p'(\tau), \langle P_1^* \| \cdots \| P_n^*, \tau\rangle \rightarrow_k^h \langle S_1' \| \cdots \| S_n', \sigma'\rangle, \sigma' \upharpoonright P_i^* = \sigma \upharpoonright P_i^*, S_i' \text{ is } \textit{before}(R, P_i^*), \text{ and } \sigma(h_i) \text{ codes } [h]_i],$$

$$\models \text{post}(R)(\sigma) \leftrightarrow \exists \sigma', \tau, S_1', \ldots, S_n', k, h[\models p'(\tau), \langle P_1^* \| \cdots \| P_n^*, \tau\rangle \rightarrow_k^h \langle S_1' \| \cdots \| S_n', \sigma'\rangle, \sigma' \upharpoonright P_i^* = \sigma \upharpoonright P_i^*, S_i' \text{ is } \textit{after}(R, P_i^*), \text{ and } \sigma(h_i) \text{ codes } [h]_i]$$

Let us call a computation of $P_1^* \| \cdots \| P_n^*$ *good* if it starts in a state satisfying p'.

Informally speaking, $I(\sigma)$ holds if σ' can be reached by a good computation at the end of which each process is outside a bracketed section. Here σ' is a state which agrees with σ on all auxiliary variables.

Informally speaking, pre(R)(σ) (or post(R)(σ)) holds if σ' can be reached by a good computation at the end of which process P_i^* is about to execute R (or has just terminated an execution of R). Here σ' is a state which agrees with σ on all variables of P_i^*.

From now on the predicates I, pre(R), and post(R) are always meant to be the ones defined above.

It can be shown that the above defined global invariant I and pre- and postassertions can be defined in L. The proof is similar to the one given in Section 4 of [1] and is omitted here.

We now show that the above global invariant I and the pre- and postassertions satisfy the conditions listed in the premise of the combined rule defined in the previous section, where for all i, p_i is pre(P_i^*) and q_i is post(P_i^*).

To check the verification conditions is a straightforward matter and we leave it to the reader. The proof of cooperation clauses is much less trivial, and the rest of this section is devoted to their proof. We shall first need the following definition.

Let σ be a state, and let $\{i_1, \ldots, i_l\}$ be a subset of $\{1, \ldots, n\}$. We call a list of component programs $R_{i_1}, \ldots, R_{i_l}$ ($\sigma, i_1, \ldots, i_l$)-*reachable* if for some programs R_j for $j \in \{1, \ldots, n\} - \{i_1, \ldots, i_l\}$ and σ' such that $\sigma'(u) = \sigma(u)$ for $u \in \text{Var}(P_{i_1}^*, \ldots, P_{i_l}^*)$, $\langle P_1^* \| \cdots \| P_n^*, \tau\rangle \rightarrow_k^h \langle S_1 \| \cdots \| S_n, \sigma'\rangle$, where $S_i \equiv \textit{before}(R_i, P_i^*)$ for some history h, $k \geq 0$, and state τ such that $\models p'(\tau)$. If, additionally, $\sigma'(u) = \sigma(u)$ for $u \in \mathscr{F}$, we say that $R_{i_1}, \ldots, R_{i_l}$ is ($I, \sigma, i_1, \ldots, i_l$)-*reachable*. Note that for R a subprogram of P_i^*, $\models \text{pre}(R)(\sigma)$ holds iff R is (σ, i)-reachable. Also, if $R_{i_1}, \ldots, R_{i_l}$ is ($\sigma, i_1, \ldots, i_l$)-reachable, then for $j \in \{i_1, \ldots, i_l\}$, $\models \text{pre}(R_j)(\sigma)$ holds.

The following lemma is crucial for the proof of the completeness theorem.

MERGING LEMMA. *Suppose that for $j = 1, \ldots, l$, R_{i_j} is either an I/O statement or E. If each R_{i_j} is (σ, i_j)-reachable for $j = 1, \ldots, l$ and $\models I(\sigma)$, then $R_{i_1}, \ldots, R_{i_l}$ is ($I, \sigma, i_1, \ldots, i_l$)-reachable.*

The proof of the lemma is given in the appendix.

Now let $\langle R_1\rangle$ and $\langle R_2\rangle$ be matching bracketed sections of P_i^* and P_j^*, respectively. Without loss of generality assume that R_1 contains an input statement and R_2 contains an output statement. We prove

$$\models\{\mathrm{pre}(R_1) \wedge \mathrm{pre}(R_2) \wedge I\}R_1 \| R_2\{\mathrm{post}(R_1) \wedge \mathrm{post}(R_2) \wedge I\}. \tag{3}$$

Assume

$$\models(\mathrm{pre}(R_1) \wedge \mathrm{pre}(R_2) \wedge I)(\sigma) \tag{4}$$

for some state σ. By the definition of the pre-assertions and the merging lemma there exist states σ', τ, history h, $k \geq 0$, and programs $S_1', \ldots, S_n'$ such that $\sigma'(u) = \sigma(u)$ for $u \in \mathscr{F} \cup \mathrm{Var}(P_i^*, P_j^*)$ and

$$\models p'(\tau), \tag{5}$$

$$\langle P_1^* \| \cdots \| P_n^*, \tau\rangle \to_k^h \langle S_1' \| \cdots \| S_n', \sigma'\rangle, \tag{6}$$

where S_i' is *before*(R_1, P_i^*) and S_j' is *before*(R_2, P_j^*). Suppose now that $\sigma_1 \in \mathscr{M}(R_1 \| R_2)(\sigma)$. Let σ_1' be such that $\sigma_1'(u) = \sigma_1(u)$ for $u \in \mathscr{F} \cup \mathrm{Var}(P_i^*, P_j^*)$ and $\sigma_1'(u) = \sigma'(u)$ for other variables. Then $\sigma_1' \in \mathscr{M}(R_1 \| R_2)(\sigma')$. Thus for some value a and $k_1 > 0$,

$$\langle S_1' \| \cdots \| S_n', \sigma'\rangle \to_{k_1}^{(a,j,i)} \langle S_1'' \| \cdots \| S_n'', \sigma_1'\rangle, \tag{7}$$

where S_i'' is *after*(R_1, P_i^*), S_j'' is *after*(R_2, P_j^*), and for $k \neq i, j$, S_k'' is S_k'. Together, (6) and (7) imply

$$\langle P_1^* \| \cdots \| P_n^*, \tau\rangle \to_{k+k_1}^{h^\circ(a,j,i)} \langle S_1'' \| \cdots \| S_n'', \sigma_1'\rangle.$$

By the definition of the pre- and postassertions and I we thus get

$$\models(\mathrm{post}(R_1) \wedge \mathrm{post}(R_2) \wedge I)(\sigma_1'),$$

and we can replace here σ_1' by σ_1. This proves (3).

The next step in the proof consists of showing that

$$\mathrm{Tr} \vdash_{T'} \{\mathrm{pre}(R_1) \wedge \mathrm{pre}(R_2) \wedge I\}R_1 \| R_2\{\mathrm{post}(R_1) \wedge \mathrm{post}(R_2) \wedge I\}. \tag{8}$$

Expression (8) is a consequence of (3) and the following lemma, on whose proof we now concentrate.

LEMMA 3. *For any matching bracketed sections* $\langle R_1\rangle$ *and* $\langle R_2\rangle$, *if* $\models\{p\}R_1 \| R_2\{q\}$, *then Tr* $\vdash_{T'} \{p\}R_1 \| R_2\{q\}$.

PROOF. Suppose that $\models\{p\}R_1 \| R_2\{q\}$ holds. Assume that R_1 is of the form $S_1; \alpha; S_2$ and R_2 is of the form $S_3; \bar{\alpha}; S_4$, where α and $\bar{\alpha}$ are matching I/O commands.

Now let p_1 and p_2 be assertions such that for any state σ,

$$\models p_1(\sigma) \leftrightarrow \exists\tau[\mathscr{M}(S_1; S_3)(\tau) = \sigma \wedge \models p(\tau)]$$

and

$$\models p_2(\sigma) \leftrightarrow \forall\tau[\mathscr{M}(S_2; S_4)(\sigma) = \tau \to \models q(\tau)].$$

One can show that p_1 and p_2 exist.

By the definition of p_1 and p_2 both $\models\{p\}S_1; S_3\{p_1\}$ and $\models\{p_2\}S_2; S_4\{q\}$ hold. By a slightly refined version of the completeness results proved in [3] and [12] (adapted

to our syntax) both

$$\mathrm{Tr} \vdash_{T'} \{p\}S_1;\ S_3\{p_1\} \qquad \text{and} \qquad \mathrm{Tr} \vdash_{T'} \{p_2\}S_2;\ S_4\{q\} \tag{9}$$

hold.

An observant reader will note that we intend to apply the formation rule. A quick look at the list of premises of the formation rule reveals that we should now prove

$$\mathrm{Tr} \vdash_{T'} \{p_1\}\alpha \| \bar{\alpha}\{p_2\}. \tag{10}$$

We shall first prove

$$\models \{p_1\}\alpha \| \bar{\alpha}\{p_2\}. \tag{11}$$

Assume the contrary. For some states σ and τ $\models p_1(\sigma)$, $\tau \in \mathcal{M}(\alpha \| \bar{\alpha})(\sigma)$ and $\not\models p_2(\tau)$. By the definition of p_1, for some state τ_1, $\sigma \in \mathcal{M}(S_1;\ S_3)(\tau_1)$ and $\models p(\tau_1)$. By the definition of p_2, for some state τ_2, $\tau_2 \in \mathcal{M}(S_2;\ S_4)(\tau)$ and $\not\models q(\tau_2)$.

By the definition of $\mathcal{M}$ we now have $\tau_2 \in \mathcal{M}(R_1 \| R_2)(\tau_1)$. Since also $\models p(\tau_1)$ and we assumed that $\models \{p\}R_1 \| R_2\{q\}$ holds, we get $\models q(\tau_2)$, which gives the contradiction. This proves (11).

We now prove (10). Assume α is of the form $P_j?x$ and $\bar{\alpha}$ is of the form $P_i!t$. According to our semantics the effect of executing $\alpha \| \bar{\alpha}$ is the same as performing the assignment $x := t$. Thus (11) states that $\models \{p_1\}x := t\{p_2\}$.

It is now useful to recall Floyd's forward assignment axiom (see [4]),

$$\{p_1\}x := t\{\exists y[p_1[y/x] \wedge x = t[y/x]]\}.$$

It is a well-known fact that Floyd's axiom is valid and together with the consequence rule forms a complete reasoning system for the assignment statement. This implies

$$\models \exists y[p_1[y/x] \wedge x = t[y/x]] \rightarrow p_2.$$

By the definition of (the fragment of) CSP, processes are disjoint, so x does not occur in t. Hence we have

$$\models \exists y\, p_1[y/x] \wedge x = t \rightarrow p_2. \tag{12}$$

By the preservation axiom,

$$\mathrm{Tr} \vdash_{T'} \{\exists y\, p_1[y/x]\}\alpha \| \bar{\alpha}\{\exists y\, p_1[y/x]\}.$$

By the communication axiom,

$$\mathrm{Tr} \vdash_{T'} \{\mathbf{true}\}\alpha \| \bar{\alpha}\{x = t\};$$

so by the conjunction rule,

$$\mathrm{Tr} \vdash_{T'} \{\exists y\, p_1[y/x]\}\alpha \| \bar{\alpha}\{\exists y\, p_1[y/x] \wedge x = t\}. \tag{13}$$

Obviously $p_1 \rightarrow \exists y\, p_1[y/x]$; so (12) and (13) imply (10).

Applying now the formation rule, we get by (9) and (10), $\mathrm{Tr} \vdash_{T'} \{p\}R_1 \| R_2\{q\}$. This concludes the proof of Lemma 3. □

Remark. One might wonder why in [2] we did not adopt in the proof system a communication axiom à la Floyd's assignment axiom and used the communication axiom A5 and the preservation axiom instead. The answer is that we felt that communication axiom A5 captures the meaning of the communication in a more intuitive way than any of the assignment axioms. A communication has two (proof-theoretic) aspects: first, it is a special case (thanks to the disjointness proviso) of the

assignment which is exactly captured by the communication axiom; second, it enjoys the preservation property (of course satisfied by the other programs, as well) which is exactly captured by the preservation axiom.

Having proved Lemma 3, we justified (8). We can thus apply the combined rule and we get

$$\mathrm{Tr} \vdash_{T'} \{\mathrm{pre}(P_1^*) \wedge \cdots \wedge \mathrm{pre}(P_n^*) \wedge I\} P_1^* \| \cdots \| P_n^* \{\mathrm{post}(P_1^*) \wedge \cdots \wedge \mathrm{post}(P_n^*) \wedge I\}.$$

To prove (2), it is now sufficient to show

$$\models p' \rightarrow \mathrm{pre}(P_1^*) \wedge \cdots \wedge \mathrm{pre}(P_n^*) \wedge I \tag{14}$$

and

$$\models \mathrm{post}(P_1^*) \wedge \cdots \wedge \mathrm{post}(P_n^*) \wedge I \rightarrow q. \tag{15}$$

The proof of (14) is a trivial consequence of the fact that by the definition of the "$\rightarrow$" relation we have

$$\langle P_1^* \| \cdots \| P_n^*, \sigma \rangle \rightarrow_0^{\epsilon} \langle P_1^* \| \cdots \| P_n^*, \sigma \rangle.$$

To prove (15), suppose now that for some state σ,

$$\models (\mathrm{post}(P_1^*) \wedge \cdots \wedge \mathrm{post}(P_n^*) \wedge I)(\sigma).$$

By the definition of the postassertions and the Merging Lemma there exists a state τ such that $\models p(\tau)$ and for some history h and $k \geq 0$,

$$\langle P_1^* \| \cdots \| P_n^*, \tau \rangle \rightarrow_k^h \langle \underbrace{E \| \cdots \| E}_{n \text{ times}}, \sigma \rangle.$$

It is easy to see that (1) implies

$$\models \{p\} P_1^* \| \cdots \| P_n^* \{q\};$$

so by the above, $\models q(\sigma)$, which proves (15).

This proves (2). Applying the rule of auxiliary variables, we now obtain $\mathrm{Tr} \vdash_{T'} \{p'\} P_1 \| \cdots \| P_n \{q\}$. Finally, applying the substitution rule, we get $\mathrm{Tr} \vdash_{T'} \{p\} P_1 \| \cdots \| P_n \{q\}$. But the proof systems T and T' are equivalent; so we proved $\mathrm{Tr} \vdash_T \{p\} P_1 \| \cdots \| P_n \{q\}$, as desired.

The other cases in the proof of the Completeness Theorem are all dealt with in the standard way. This concludes the proof of the Completeness Theorem. □

7. *Total Correctness*

In this section we concentrate on the issue of total correctness of the CSP programs we consider. Informally speaking, we say that a program P is *totally correct under J with respect to assertions p and q* ($\|\models_J \{p\} P \{q\}$) if any computation of P starting in a state satisfying p successfully terminates and its terminating state satisfies q. To express it formally, we shall rather need a formulation by contraposition, which can be phrased as follows: $\|\models_J \{p\} P \{q\}$ holds if

(1) whenever a computation of P starting in a state satisfying p terminates, then the state in which it terminates satisfies q (i.e., $\models_J \{p\} P \{q\}$ holds);
(2) no computation of P starting in a state satisfying p diverges;
(3) no computation of P starting in a state satisfying p becomes blocked.

We now define formally when a computation diverges or becomes blocked.

Definition. Let σ be a state and P a parallel program.

(i) We say that P *can diverge from* σ iff there exists an infinite sequence $\langle P_i, \sigma_i\rangle$ $(i = 0, 1, \ldots)$ such that for some histories h_i $(i = 0, 1, \ldots)$,

$$\langle P, \sigma\rangle = \langle P_0, \sigma_0\rangle \to_1^{h_0} \langle P_1, \sigma_1\rangle \to_1^{h_1} \langle P_2, \sigma_2\rangle \to_1^{h_2} \cdots$$

(ii) We say that P *can become blocked from* σ if there exists a finite sequence $\langle P_i, \sigma_i\rangle$ $(i = 0, 1, \ldots, k)$ such that

(a) for some histories h_i $(i = 0, 1, \ldots, k-1)$,

$$\langle P, \sigma\rangle = \langle P_0, \sigma_0\rangle \to_1^{h_0} \langle P_1, \sigma_1\rangle \to_1^{h_1} \cdots \to_1^{h_{k-1}} \langle P_k, \sigma_k\rangle,$$

(b) $P_k \not\equiv \underbrace{E \| \cdots \| E}_{n \text{ times}}$

(c) for no $\langle P_{k+1}, \sigma_{k+1}\rangle$ and history h_k, $\langle P_k, \sigma_k\rangle \to_1^{h_k} \langle P_{k+1}, \sigma_{k+1}\rangle$.

We can introduce the following definition.

Definition. $\models_J \{p\}P\{q\}$ iff

(1) $\vdash_J \{p\}P\{q\}$,
(2) $\forall\sigma[\vdash_J p(\sigma) \to P$ cannot diverge from $\sigma]$,
(3) $\forall\sigma[\vdash_J p(\sigma) \to P$ cannot become blocked from $\sigma]$.

We now modify our proof system so that it can be used to prove total correctness of the CSP programs. The first two properties listed above can be proved together, provided one modifies appropriately the proof rule dealing with the repetitive command. We replace the proof rule R2 by the following rule.

R2.′ *Repetitive command*

$$\frac{p(0) \to \bigwedge_{j=1}^{m} \neg b_j,\ p(n+1) \to \bigvee_{j=1}^{m} b_j,\ \{p(n+1) \wedge b_j\} R_j \{p(n)\}_{j=1,\ldots,m}}{\{\exists n\, p(n)\} * [\Box(j = 1, \ldots, m)\ b_j \to R_j] \{p(0)\}}.$$

Here $p(n)$ is an assertion with a free variable n which does not appear in the programs considered and ranges over natural numbers.

This proof rule is motivated by a corresponding proof rule dealing with **while**-loops, introduced in [6].

To prove the third property (freedom of blocking), we first modify the proof rule dealing with the alternative command. We replace the proof rule R1 by the following proof rule.

R1.′ *Alternative command*

$$\frac{p \to \bigvee_{j=1}^{m} b_j,\ \{p \wedge b_j\} R_j \{q\}_{j=1,\ldots,m}}{\{p\}[\Box(j = 1, \ldots, m)\ b_j \to R_j]\{q\}}.$$

The additional premise rules out the possibility of *abortion*—a situation in which all Boolean guards of an alternative construct evaluate to **false**.

Another possibility of blocking is by *deadlock*. By a deadlock we mean here a situation in which at least one process did not terminate its execution, each process which did not terminate waits for a communication, and no process can proceed. This issue was dealt with in [2, Sec. 4], and we can readily adopt the same approach here. The only difference is that we disallow here the convention of exiting an I/O

guarded loop owing to a termination of other processes. Consequently, in contrast to [2], no further changes in the proof system are needed.

However, to deal properly with the I/O guarded selection command omitted here, one can no longer rely on the equivalence mentioned in Section 2, as it does not hold for total correctness. A correct solution would be to modify appropriately the semantics and reintroduce the I/O guarded selection rule defined in [2].

Theorem 1 from [2, Sec. 4] provides a sufficient condition for proving deadlock freedom. We refer the reader there for further details.

It should be stressed that the notion of blocking used in this paper differs from the one used in [2] in that here we consider abortion as a special case of blocking.

We can summarize the situation as follows. We presented here a proof system in which total correctness of the CSP programs can be proved. We also provided a semantic definition of total correctness. The next step would be to prove soundness and completeness in an appropriate sense of this system. The appropriate notions would be those of arithmetical soundness and completeness introduced in [6]. We believe that the desired proofs can be obtained by an appropriate modification of the proofs provided in this paper.

Appendix

MERGING LEMMA. *Suppose that for* $j = 1, \ldots, l$, R_{i_j} *is either an I/O statement or E. If each* R_{i_j} *is* (σ, i_j)*-reachable for* $j = 1, \ldots, l$ *and* $\models I(\sigma)$ *holds, then* $R_{i_1}, \ldots, R_{i_l}$ *is* $(I, \sigma, i_1, \ldots, i_l)$*-reachable.*

PROOF. Throughout the proof the predicate I refers to the definition from Section 6.

To understand better the proof of the Merging Lemma, it is useful to provide an informal interpretation of the claim. The state σ fixes the values of the histories h_i $(i = 1, \ldots, n)$. The assumption $\models I(\sigma)$ implies that the h_i's are projections of a history h of a single computation. For each $j = 1, \ldots, l$, h_{i_j} is also a projection of a history of a computation leading to R_{i_j}. We now wish to show the existence of a single computation c with history h which leads to $R_{i_1}, \ldots, R_{i_l}$ simultaneously. This computation c will be an appropriate interleaving of the considered computations. We also need to ensure that c leads to a state satisfying I. The last claim holds due to the fact that the R_i's are either E or just before a bracketed section, so c leads to an admissible program.

We must first formalize a few notions. By a computation we mean a finite or infinite sequence of pairs $\langle \bar{S}_i, \sigma_i \rangle_{i=1,2,\ldots}$ such that for each i, $\langle \bar{S}_i, \sigma_i \rangle \to_1^h \langle \bar{S}_{i+1}, \sigma_{i+1} \rangle$ for some h. Each elementary step

$$d = \langle S_1 \| \cdots \| S_n, \sigma_i \rangle \to_1^h \langle S_1' \| \cdots \| S_n', \sigma_{i+1} \rangle$$

in such a computation is associated with one or two (in the case of communication) processes which progressed in execution. We say that this elementary step was *performed* by any such process. If it was performed by the jth process, then define $[\langle \bar{S}_{i+1}, \sigma_{i+1} \rangle]_j = \langle S_j, \sigma_{i+1} \restriction P_j^* \rangle$, where $\bar{S}_{i+1} \equiv S_1 \| \cdots \| S_n$ and otherwise $[\langle \bar{S}_{i+1}, \sigma_{i+1} \rangle]_j$ is the empty sequence. Given now a computation $c = \langle \bar{S}_i, \sigma_i \rangle_{i=1,2,\ldots}$, we define $[c]_j$ to be the sequence $[\langle \bar{S}_i, \sigma_i \rangle]_j$ for $i = 2, 3, \ldots$.

Now let $A = \{i_1, \ldots, i_l\}$. Let c denote a computation whose existence is guaranteed by the fact that $\models I(\sigma)$, and let c_j, for all $j \in A$, denote the corresponding computation whose existence is guaranteed by the assumption of the (σ, j)-reachability of R_j. All

these computations start in the same state τ_0 defined by

$$\begin{aligned} \tau_0(x_i) = \tau_0(z_i) &= \sigma(z_i) && \text{for } i = 1, \ldots, k, \\ \tau_0(h_i) &= \langle\ \rangle && \text{for } i = 1, \ldots, n, \\ \tau_0(y) &= \sigma(y) && \text{for all other variables.} \end{aligned}$$

Note that $\models p'(\tau_0)$.

We now prove that there exists a computation c' starting in the state τ_0 such that for $j \in A [c']_j = [c_j]_j$ and for $j \notin A$, $[c']_j = [c]_j$. These properties of c' clearly imply that c' is a computation ensuring the $(I, \sigma, i_1, \ldots, i_l)$-reachability of $R_{i_1}, \ldots, R_{i_l}$.

The proof of the claim proceeds by induction on the length $|h|$ of the history h of the computation c.

If $|h| = 0$, then by the construction of $P_1^* \| \cdots \| P_n^*$ and the definition of I we have $\sigma(h_i) = \langle\ \rangle$ for $i = 1, \ldots, n$; so in computations c_j for $j \in A$ and c, no communication between processes took place. The desired computation can now be easily constructed, thanks to the assumption of disjointness of the processes. It is obtained from a concatenation of $[c_j]_j$ for $j \in A$ and $[c]$ for $j \notin A$, appropriately extended to a computation sequence.

Assume now that $|h| > 0$. For some history h_1, value a, and $l_1, l_2 \in \{1, \ldots, n\}$, we have $h = h_1 \circ (a, l_1, l_2)$. For some computations d_1 and d_2, $c = d_1 \circ d_2$, where d_1 has history h_1 and ends in an admissible program and d_2 has history (a, l_1, l_2). Let σ_0 be the state to which d_1 leads. By the definition of I we have $\models I(\sigma_0)$. We can assume that all steps in d_2 were performed by the l_1st and l_2nd process. This implies that for $j \neq l_1, l_2$ we have $\sigma_0 \restriction P_j^* = \sigma \restriction P_j^*$. There are now four cases to consider.

Case I. $l_1 \notin A$, $l_2 \notin A$. For $j \in A$ we have $j \neq l_1, l_2$; so by the definition, R_j is (σ_0, j)-reachable with the computation c_j. Also, d_1 ensures that $\models I(\sigma_0)$. By the inductive assumption there exists a computation d' which starts in τ_0 and such that

(i) for $j \in A$, $[d']_j = [c_j]_j$,
(ii) for $j \notin A$, $[d']_j = [d_1]_j$.

In particular, $[d']_{l_1} = [d_1]_{l_1}$ and $[d']_{l_2} = [d_1]_{l_2}$. Thus both d' and d_1 reach the same point and state in the l_1st and l_2nd processes. It implies that $c' = d' \circ d_2$ is a computation, and it is clear that it is the required one.

Case II. $l_1 \in A$, $l_2 \notin A$. By the construction of $P_1^* \| \cdots \| P_n^*$ we have $\sigma(h_{l_1}) = \sigma_0(h_{l_1}) \circ (a, l_1, l_2)$, and also $\sigma(h_{l_1})$ codes $[h']_{l_1}$, where h' is the history of the computation c_{l_1}. This means that in the computation c_{l_1}, the last communication performed by the l_1st process was with the l_2nd process.

For some computations d_3 and d_4 we have $c_{l_1} = d_3 \circ d_4$, where d_4 is the computation which starts with the execution of the abovementioned communication. Let σ_1 be the state to which d_3 leads, and let R' be the I/O statement of the l_1st process to which d_3 leads. By definition, R' is (σ_1, l_1)-reachable. We can assume that all steps in d_4 were executed by the l_1st and l_2nd process.

Now let σ'' be such that $\sigma'' \restriction P_i^* = \sigma_1 \restriction P_i^*$ for $i = l_1, l_2$ and $\sigma''(x) = \sigma(x)$ for other variables. Then for $j \in A - \{l_1\}$, R_j is (σ'', j)-reachable with the computation c_j, and R' is (σ'', l_1)-reachable with the computation d_3. Also, d_1 ensures that $\models I(\sigma'')$ holds.

To establish the last claim, observe first that $\sigma''(h_i) = \sigma(h_i) = \sigma_0(h_i)$ for $i \neq l_1, l_2$, since in the computation d_2 (leading from σ_0 to σ), all steps were performed by $P_{l_1}^*$ and $P_{l_2}^*$. By definition, $\sigma''(h_i) = \sigma_1(h_i)$ for $i = l_1, l_2$. But $\sigma_1(h_i) = \sigma_0(h_i)$ for $i = l_1, l_2$, since $\sigma(h_i) = \sigma_0(h_i) \circ (a, l_1, l_2)$ and $\sigma(h_i) = \sigma_1(h_i) \circ (a, l_1, l_2)$. So $\sigma''(u) = \sigma_0(u)$ for $u \in \mathscr{F} = \{h_1, \ldots, h_n, \bar{z}\}$, as the values of the variables from $\bar{z}$ do not change in the

considered computations. But $\models I(\sigma_0)$ and all free variables of I are in $\mathscr{F}$, so $\models I(\sigma'')$ as desired.

Now by the inductive hypothesis concerning the computations c_j for $j \in A - \{l_1\}$, d_3, and d_1, there exists a computation d' which starts in τ_0 and such that

(i) for $j \in A - \{l_1\}$, $[d']_j = [c_j]_j$;
(ii) $[d']_{l_1} = [d_3]_{l_1}$;
(iii) for $j \notin A$, $[d']_j = [d_1]_j$.

Now let c' be a concatenation of d' with the elementary step consisting of the considered communication between $P^*_{l_1}$ and $P^*_{l_2}$ and $[d_2]_{l_2}$ and $[d_4]_{l_1}$, appropriately extended to a computation sequence. It is clear that c' is the desired computation.

The case when $l_1 \notin A$ and $l_2 \notin A$ is analogous, and the case when $l_1, l_2 \in A$ is treated in a similar way and left to the reader. □

ACKNOWLEDGMENTS. I would like to thank N. Francez and W. P. de Roever for helpful and stimulating discussions which took place while working on [2]. These discussions avoided deadlock and were at the source of this paper. I am very grateful to both referees of the paper for their remarkably precise and useful reports, which significantly influenced the final version of the paper.

REFERENCES

1. APT, K.R. Recursive assertions and parallel programs. *Acta Inf. 15* (1981), 219–232.
2. APT, K.R., FRANCEZ, N., AND DE ROEVER, W.P. A proof system for communicating sequential processes. *Trans. Prog. Lang. Syst. 2,* 3 (July 1980), 359–385.
3. COOK, S.A. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7* (1978), 70–90.
4. FLOYD, R.W. Assigning meanings to programs. In *Proc. Symp. in Applied Mathematics 19,* American Mathematical Society, Providence, R.I., 1967, pp. 19–31.
5. FRANCEZ, N., HOARE, C.A.R., LEHMANN, D.J., AND DE ROEVER, W.P. Semantics for nondeterminism, concurrency and communication. *J. Comput. Syst. Sci. 19* (1979), 290–308.
6. HAREL, D. First-order dynamic logic. *Lecture Notes in Computer Science 68,* Springer, New York, 1979.
7. HENNESSY, M.C.B., AND PLOTKIN, G.D. Full abstraction for a simple programming language. In *Proc. 8th Symp. on Mathematical Foundations of Computer Science,* Lecture Notes in Computer Science 74, Springer, New York, 1979, pp. 108–120.
8. HOARE, C.A.R. Communicating sequential processes. *Commun. ACM 21,* 8 (Aug. 1978), 666–677.
9. LAMPORT, L. Proving the correctness of multiprocess programs. IEEE *Trans. Softw. Eng. 3* (1977), 125–143.
10. LEVIN, G.M. Proof rules for communicating sequential processes. Ph.D. Dissertation, Computer Science Dep., Cornell Univ., Ithaca, N.Y., 1980.
11. OWICKI, S. Axiomatic proof techniques for parallel programs. Ph.D. Dissertation, Computer Science Dep., Cornell Univ., Ithaca, N.Y., 1975.
12. OWICKI, S. A consistent and complete deductive system for verification of parallel programs. Proc. 8th Ann. ACM Symp. on Theory of Computing, Hershey, Pa., 1976, pp. 73–86.
13. OWICKI, S., AND GRIES, D. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM 19,* 5 (May 1976), 279–285.
14. OWICKI, S., AND GRIES, D. An axiomatic proof technique for parallel programs. *Acta Inf. 6* (1976), 319–340.
15. SHOENFIELD, J.R. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967.

RECEIVED APRIL 1980; REVISED FEBRUARY 1982; ACCEPTED MARCH 1982