PROVING NONINTERACTION:
AN OPTIMIZED APPROACH.

by

R. Gerth and E. Shrir.

Technical Report #388.

November 1985
Proving Noninteraction: 
An Optimized Approach

Rob Gerth
Computer Science Department, University of Utrecht, Utrecht, Holland.

Liuba Shira
Computer Science Department, Technion, Haifa, Israel.

ABSTRACT
A recent work of Elrad and Francez introduces a novel methodology of analyzing distributed programs: decomposition into communication closed layers i.e. no interaction occurs between layers.

This work proposes an optimization to the formal proof of layer non-interaction. The proof is localized to the bodies of component layers only. By this, the number of possible interaction cases to be considered in the proof is reduced substantially.

Our main result contains a definition of new localized communication closedness and a suggestion of a sound and complete proof rule for proving layer non-interaction. The proof rule is maximally efficient in the sense that it makes use only of the correctness proof of the layer itself.

The obtained result, in addition to program analysis, also sheds new light on the use of closed layer methodology in the construction of distributed programs.
1. Introduction

A recent work of Elrad and Francez [EFB2] introduces a novel methodology of analyzing distributed programs: decomposition into communication closed layers. According to this methodology, parts of processes are grouped together into layers so that there is interaction only between commands which are in the same layer. Intuitively, an execution of such decomposed programs is equivalent to synchronizing all the processes in the distributed program at layer boundaries. This approach is helpful in analysis of distributed programs and also useful in systematic construction of distributed programs.

An independent algorithmically oriented work of [SFRB3] uses automatically enforced communication closed layers in a transformational technique for recursive distributed algorithm synthesis.

A sequel work [SF85] also reasons with enforced communication closed layers while converting the program transformation in [SFRB3] into a transformation of proofs.

The present work suggests an optimization to the closed layer methodology both at the program analysis and program construction levels. We consider the following problem:

Assume given verified distributed programs designed for some fixed network. These programs serve as building blocks in construction of a larger program where they are combined in a layerwise manner, i.e. the corresponding processes are composed sequentially. Such component programs are called layers. To make the discussion concrete, we fix a specific notation. We adopt the notation and terminology of CSP [H78], to keep a natural link with [EF82] and [SF85]. However our results apply to other notations like ADA or OCCAM as well.

When automatically constructing a program with given layers, both [EF82] and [SF85] enforce communication closeness by including in the program a synchronizing section the role of which is to prevent the interactions between commands in distinct layers. This approach may lead to oversynchronization in the resulting program: It might be the case that component layers have the property that their constituent processes interact only with each other and do not interact with other parts of the program. We are interested in detecting efficiently this noninteraction property in order to omit the synchronizing part and thus to improve the overall message complexity of the resulting program.

The straightforward approach to the formal proof of layer non-interaction, is to consider the complete program without the synchronizing parts and, by verifying it as a whole, to make sure that no interlayer interactions occur. In general, the number of potential interactions one has to consider in such a proof is
significantly (combinatorically) larger than the number of interactions considered in the separate verification of all the component layers. [EF82] and [SF85] deal with proving layer communication closeness in a composed program in the above straightforward way.

In this work, we take an unexpected approach to the noninteraction proof. We localize this proof to the bodies of component layers only. i.e. instead of considering all possible interactions, we analyze the body of the layer and directly identify the sources of potential interlayer interaction. Towards this end, we strengthen the definition of a communication closed layer by defining "general" closeness. A generally closed layer is defined as a layer not having interlayer interactions in a sequential composition with an arbitrary other layer. This definition contrasts the original definition of closeness, which is relative to a specific context. With the new definition, we suggest a sound and complete proof rule for proving general closeness of a given layer. The proof rule is maximally efficient in the sense that it makes use only of the correctness proof of the layer itself.

In view of the obtained results, we consider the question of communication closeness enforcement and explore the conditions of performing this task automatically.

2. The notion of general tail communication closeness

In this section we introduce the definition of general communication closeness and demonstrate the usefulness of the new notion.

Definition 1: A layer \{p\} S \{q\}, \( S = [S_1 \parallel \ldots \parallel S_n] \) is a General Tail Communication Closed (GTCC) layer iff for, arbitrary layer \( T = [T_1 \parallel \ldots \parallel T_n] \) and for all executions of the distributed program \{p\} P \{true\}, \( P = [S_1 ; T_1 \parallel \ldots \parallel S_n ; T_n] \), no synchronization occurs between a communication command in a process of \( S \) and a communication command in a process of \( T \).

Using the terminology of [AFR80], the definition requires that there are no semantic matches between communication commands in \( S \) and \( T \). We now discuss the new notion and state the additional assumptions we make about the programs the new definition concerns. To remind, a layer \( S \) is just any concurrent \( n \rightarrow \) process program \( S = [S_1 \parallel \ldots \parallel S_n] \). We assume that \( S \) is given together with a partial correctness proof for it. We are interested in exploring the precise conditions for the behaviour of \( S \) not to be affected by appending to it another layer \( T \). Obviously, a nonterminating layer \( S \) is amenable not to be affected by such appending. Also, a layer \( S \) making use of the Distributed Termination Convention of CSP is
trivially affected by appending to it another layer. (Distributed Termination Convention of CSP (DTC) is the mechanism that allows remote sensing of partner process termination by trying to communicate with such terminated process. DTC is used to disable communication guarded alternatives and to exit communication guarded loops due to partner process termination). Another trivial situation arises when some process in a layer $S$ addresses for communication a process nonexisting in $S$, but existing in layer $T$. We are not dealing with these anomalies here and assume throughout the discussion that layers are terminating, do not use the DTC convention of CSP, and that a processes of a layer address in communication commands only processes existing in that layer.

Before going on, we give examples for the definition of GTCC and also demonstrate that the above restrictions do not make this definition vacuous.

**Example 1:**

$$S = [P_1 :: P_2 !a || P_2 :: P_1 ?x]$$

Obviously, $S$ is a GTCC layer, i.e. for any layer $T$ appended to $S$, no communication will occur between commands in $S$ and $T$.

**Example 2:**

$$S' = [P_1 :: [P_2!a ->skip []true ->skip] || P_2 :: [P_1 ?x ->skip []true ->skip]]$$

$S'$ is a terminating layer when executed alone, does not use the DTC convention of CSP and each of it processes addresses only existing processes for communication. Still, $S'$ is non-GTCC. Consider for example $S'$ composed with the layer $S$ from example 1:

$$[P_1 :: [P_2! ->skip []true ->skip]; P_2!a || || P_2 :: [P_1 ?x ->skip []true ->skip]; P_1 ?x]$$

Whenever $P_2$ from layer $S'$ selects the local alternative, $P_1$ from layer $S'$ can communicate with $P_2$ via a communication command from layer $S$, the result being a deadlock.

We compare now the new notion of GTCC with the original definition of communication closeness (CC). The original notion of CC layer considers a specific distributed program $P$ and its given decomposition into layers \( \{ S^j \} \) \( j = 1 \cdots d \), s.t. \( P = [ S^1 ; \cdots S^d ] || \cdots || S^d ; \cdots ; S^1 ] \), and \( S^j = [ S^j_1 || \cdots || S^j_k ] \).

**Definition 2:** [EF92] A layer $S^j$ is a CC layer in $P$ if in all executions of $P$ no communication occurs between a communication command in a process of $S^j$ and a communication command in a process of some other layer $S^k$, \( k \neq j \).

The new notion strengthens the original definition by considering closeness with
respect to an arbitrary layer $T$, but also restricts the original notion by requiring 'tail' closeness only. We show that in spite of the restriction, the new definition is able to express the original notion of closeness.

**Theorem 1.** Let \( \{p\} \ S \ {r} \) , \( \{r\} \ S' \ {v} \) and \( \{v\} \ S' \ {q} \) be GTCC layers. Let \( P \) be a composition of the layers \( S, S' \) and \( S'' \), \( P = [S_1;S_1';S_1'' \mid \cdots \mid S_n;S_n';S_n''] \). Then, layers \( S, S' \) and \( S'' \) are also communication closed (CC) layers in \( P \) (by Definition 2).

**Proof:** We need to show that no interlayer communications arise in \( P \). The proof is immediate by repeated use of the definitions: by the definition of \( S \) as a GTCC layer, there can be no communication between a command in layer \( S \) and a command in layer \( [S_1;S_1';S_1'' \mid \cdots \mid S_n;S_n';S_n''] \) in \( P \). This means that \( S' \) starts its execution in \( P \) is state-satisfying \( r \). Now, the possibility of a communication between a command in \( S' \) and a command in \( S'' \) in \( P \) contradicts the assumption that \( S' \) is a GTCC layer.

3. **Another definition of GTCC and the equivalence of the two definitions**

In this section we take a closer look into the notion of GTCC, aiming at verifying potential interlayer interactions inside a given layer. To be on firm ground, we construct a new definition of GTCC stated solely in terms of a given layer. The definition exposes more closely the "mechanics" of the interlayer interaction and gives the intuition behind the proof rule suggested subsequently.

Let us consider a non GTCC layer \( S \). By the definition of GTCC, there exist a layer \( T = [T_1;\cdots;T_n] \), s.t. when \( T \) is appended to \( S \), an interlayer communication occurs between a command in \( S' \) and a command in \( T \). To recall, by our assumptions, when executed alone, \( S \) is a perfectly terminating CSP layer. To characterize this situation solely in terms of \( S \), we need to consider some specific semantics for \( S \). We make use of the denotational linear history semantics of CSP following [FLP84] and, before going on, glance here briefly over its main concepts. The semantics uses the notion of history of communications. By history is meant a sequence of records of communications. A record stands for a triple \((a,i,j)\) which is associated with a communication between a process \( P_i \) and \( P_j \); \( a \) is the value sent by \( P_i \) to \( P_j \). \( h \) is used to denote a history. \( h_1,h_2 \) denotes concatenation of histories \( h_1 \) and \( h_2 \), \( h_{|i} \), the \( i \)-restriction of \( h \), denotes the communication sequence resulting from \( h \) by omission of all the communication records not involving process \( P_i \).

First the semantics of a single process is considered. Any computation defined by \( S_1 \) reaches a certain state \( \tau \) and produces a certain communication sequence \( h \) in order to get there. Whether this computation is realizable in a given environment
of other processes depends on the ability and readiness of these processes to cooperate in producing the corresponding communications on their side. Thus analysis of $P_1$ itself characterizes the correspondence between attainable states and the communication sequences required to achieve them. i.e. $\left[ [P_1] \right]$ is a mapping from initial state $\sigma$ into a set of pairs $<\tau, h>$ representing the existence of computations starting at state $\sigma$ which reach a state $\tau$ with communication history $h$. The meaning of a parallel program $\left[ [P_1][\cdots][P_n] \right]$ is obtained by merging the a priori meanings of single $P_i$. Only computations “compatible” with the other processes in the environment are included in the joint meaning.

Now we are ready to introduce the new definition of GTCC. For the sake of simplicity, we first handle the case $n=2$. The general case is discussed later.

**Definition 3:** A layer \{p\} $S \{q\}$, $S = [S_1 \parallel S_2]$ is a General Tail Communication Closed (GTCC) layer iff there are no $\sigma, \tau, h$, s.t. $<\tau, h> \in \left[ [S_1 || S_2] \right] (\sigma)$ and $<\ldots, h\cdot \alpha > \in \left[ [S_1 || (\sigma) \cup || S_2] \right] (\sigma)$, where $\alpha$ is a record of a communication between $S_1$ and $S_2$.

Let us denote the computations in the definition by $c$ and $c'$, where $c = <\tau, h>$ and $c' = <\ldots, h\cdot \alpha >$. $c$ represents a communication history from the joint meaning of $S_1$ and $S_2$, while $c'$ stands for a matching history from the a priori meaning of one of the processes, extended with an additional record. Note, that by our assumptions, a communication command may address only an existing process and while computation $c$ is terminating, $c'$ need not be such. Note also, that $c'$ may also be a computation of $\left[ S \right]$ for some $\sigma'$, but, intuitively, what matters is that $c$ with the shorter history is a computation of $\left[ S \right] (\sigma)$ and thus $\alpha$ represents the communication command that is “responsible” for the potential interlayer interaction when $S$ is tail composed with some other layer. We now prove that though the above constructed definition is stated solely in terms of the layer $S$ itself, the notion of GTCC defined by it is identical to the notion defined by definition 1.

**Theorem 2:** A layer $S$ is GTCC by definition 1 iff it is GTCC by definition 3.

**Proof:** Suppose a layer $S$ is not GTCC by definition 1. Then, $\left[ [S_1 || S_2] \right]$ has a computation starting in state $\sigma$ with history $h$ in which w.l.o.g. $S_2$ has terminated and $S_1$ is suspended at a communication command with $S_2$. Thus, $<\tau, h> \in \left[ [S_1 || S_2] \right] (\sigma)$, and because $S_1$ is suspended at a communication $<\ldots, h\cdot \alpha > \in \left[ S_1 \right] (\sigma)$, so $S$ is not a GTCC also by definition 3.

Now assume that a layer $S$ is not GTCC by definition 3, i.e. there are $\tau, h, \alpha$ s.t. a terminating $<\tau, h> \in \left[ [S_1 || S_2] \right] (\sigma)$, and w.l.o.g. $<\ldots, h\cdot \alpha > \in \left[ S_1 \right] (\sigma)$. Now $S_2$
can terminate without performing \( \alpha \); whenever \( T_2 \) in some layer \( T \) offers a matching communication capability for \( \alpha \), an interaction may occur. Thus, \( S \) is not \( GTCC \) by definition 1.

The extension of definition 3 and Theorem 2 to the general case (\( n > 2 \)) are straightforward by the use of \( i \)-restricted communication histories in relating the computation in the joint meaning with a computation in the a priori meaning of process \( P_i \).

**Definition 4:** A layer \( \{p\} S \{q\} \), \( S = [S_1 || \cdots || S_n] \) is a **General Tail Communication Closed (\( GTCC \))** layer iff there are no \( \sigma, \tau \), \( h \) and \( i \), s.t.

\[
< \tau, h > \in \ll S \rr (\sigma) \text{ and } \\
< \cdots, h_i \rho \alpha > \in \ll S_i \rr (\alpha), \text{ where } \alpha \text{ is a record of a communication between } S_1 \text{ and } S_2.
\]

4. The proof rule for \( GTCC \).

In this section we present a proof rule for proving the \( GTCC \) of a given layer. The rule operates solely on the partial correctness proof of the layer itself. We assume the proof is carried out in the \( CSP \) proof system proposed in [AFR80] i.e. each process is given a proof outline (or alternatively a proof from assumptions see [AB85]), and the proofs cooperate with respect to some global invariant \( GI \). For the sake of simplicity, we first state the rule for the case of two processes and extend it to the general case later.

Assume a layer \( \{p\} [S_1 || S_2] \{q\} \) is given together with its partial correctness proof, i.e. proof outlines for \( \{p\} S_1 \{q_1\} \) were constructed cooperating w.r.t some global invariant \( GI \). The proof rule for proving \( S \) to be \( GTCC \) is given in Figure 1 below.

**Discussion:** Intuitively, the rule characterizes what happens before the first interlayer interaction occurs. The computation leading to this interaction starts as a valid computation of \( S \). For (say) process \( P_1 \) to interact interlayer, its companion process \( P_2 \) has to be terminated by then. The termination of \( P_2 \) was in no way affected by the appending of a layer \( T \), as \( P_2 \) has not communicated with \( T \). Thus, \( P_2 \) can terminate at that point also in a valid computation of \( S \). As all the computations of \( S \) are terminating, whenever \( P_2 \) has terminated, \( P_1 \) must be able to terminate without any additional communication.

Let us now apply the rule to layers \( S \) and \( S' \) from example 1: \( S \) satisfies the requirements the proof rule vacuously, while \( S' \) fails to satisfy the rule and is proven non-\( GTCC \).
for \( p = 1, 2 \) do
for any command \([[] g_i \rightarrow L_i \] \) in \( S_p \)
with a pre-assertion \( t \) and at least
one i/o guarded alternative and one local alternative
do
show \( |= - (g_{3-p} \land t \land GI \land \bigvee_{k=1}^{t} g_k) \)
where \( g_{i_1} \ldots g_{i_t} \) is the set of boolean parts of the mixed guards
od
od

\( \{r\}[S_1 \parallel S_2 \} \{q\} \) is a GTCC layer w.r.t. \( r \)

---

The extension of the above rule to the general case of \( n > 2 \) processes is straightforward. Consider a layer \( \{r\}[S_1 || \ldots || S_n \} \{q\} \), given together with its partial correctness proof. The rule for proving \( S \) to be GTCC is presented in Figure 2 below.

A natural question to consider next is whether the proposed GTCC rule takes care of all the "sources" of potential interlayer interaction. The answer is given in the next section where we prove soundness and completeness of the proposed rule.

5. The soundness and completeness of the proof rule for GTCC.

---

for \( p = 1 \) to \( n \) do
for any command \([[] g_i \rightarrow L_i \] \) in \( S_p \)
with a pre-assertion \( t \) and at least one i/o guarded alternative
and one local alternative
where \( g_{i_1} \ldots g_{i_t} \) is the set of mixed guards,
referring to processes \( S_{j_1} \ldots S_{j_t} \) in their i/o parts,
do
show \( |= - \bigvee_{k=1}^{t} (t \land g_{i_k} \land GI \land g'_{i_k}) \)
where \( g'_{i_1} \ldots g'_{i_t} \) is the set of boolean parts of the mixed guards
od
od

\( \{r\}[S_1 || \ldots || S_n \} \{q\} \) is a GTCC layer w.r.t. \( r \)

---

Figure 2. The rule GTCC
We prove that the proposed \( GTCC \) rule is sound in the sense of the following theorem.

**Theorem 3:** For any layer \( \{r\}S \{q\} \), if \( \vdash_{gtcc} S \) is \( GTCC \) then \( \models S \) is \( GTCC \)

**Proof:** Assume \( S = [S_1 \mid S_2] \) is not \( GTCC \). First note, that the CSP proof system used for the layers \( S \) partial correctness proof, is sound [A83]. If it would be possible to find proof outlines satisfying the premises of the rule non vacuously, that would contradict the soundness of the CSP proof system. To derive the contradiction, let \( q_2, t \) and \( G_L \) be the assertions from the presumed proof outlines showing \( GTCC \), and \( q'_2, t' \) and \( G_L' \), be the corresponding assertions used in the CSP proof system completeness proof; By the definition of the above assertions:

\[
\models q'_2 \land t' \land G_L' \rightarrow q_2 \land t \land G_L
\]

As \( \tau \models -(q_2 \land t \land G_L \land V \lor g') \),

also \( \tau \models (q'_2 \land t' \land G_L' \land V \lor g') \),

leading to the required contradiction because

\[
(q'_2 \land t' \land G_L' \land V \lor g')
\]

precisely claims the existence of a computation witnessing non-closeness in state \( \tau \).

We now show that the rule can not also be satisfied vacuously (if it could, that would mean that there are other "sources" of interlayer interaction in addition to the alternative command of the special form considered by the rule). It suffices to find in some process, say \( S_1 \), a command \( [[[q_1 \rightarrow L_4]]] \) of the required form and construct a computation in which \( S_2 \) has terminated and \( S_1 \) is in front of this command (in some state \( \tau \)). We use the definition 1 of \( GTCC \). This claims the existence of a computation in which \( S_1 \) stands in front of a communication command while, on its partners side, \( S_2 \) has already terminated as \( T_2 \) is in front of a communication command as well. See figure 3 below.

![Figure 3](image-url)
Let us denote this configuration by $e$. From state $e S_1$ can terminate. Moreover $S$ has no nonterminating computations (satisfying $\tau$). Thus, $e$ must offer a choice between either doing or not doing a communication. The only statement offering such a choice is a conditional statement with an i/o guarded and local alternatives i.e. a command of the form $[[]] g_1 \rightarrow L_4$.

**Completeness of the rule.**

The proposed GTCC rule is complete in the sense of the following theorem.

**Theorem 4**: For any layer $\{r\} S\{q\}$,

if $\vdash S$ is GTCC then $\vdash_{\text{GTCC}} S$ is GTCC

**Proof**: The proof of the completeness of the rule follows directly from the CSP proof system completeness proof [AB83], by the use of the so called completeness assertions.

6. **Automatic closeness enforcement, revisited.**

To facilitate-distributed program synthesis with communication closed layers, [EF82] and [SF83] advocate the use of an automatic layer communication closeness enforcement facility. We discuss here the feasibility of such a facility in view of our better understanding of the mechanism of communication closeness. The proposed closeness enforcement facility should be used as follows: the programmer supplies code for component layers and specifies their required composition using a special syntactic construct. The compiler generates code to be inserted between the supplied component layers and composes the resulting program from the supplied and generated code, so that in the composed program communications occur only between commands that belong to the same layer. Obviously, to make the automation of the compiler task feasible, the generated synchronizing code should not depend on the contents of specific layers. Let us denote the synchronizing code generated for a $n-$process layer by $Z^n$.

We show that, for any $Z^n$, it is in general impossible to prevent interaction between commands in a programmer supplied layer and commands in the synchronizing code. Let us consider a specific non GTCC layer $L$ and some synchronizing code $Z^2$ as in example 3 below.

If the first communication suggested by $Z_1$ is an input, then whenever $P_1$ chooses a boolean alternative, this input command matches semantically the output guard in $P_2$. Similarly, if the first suggested communication of $Z_2$ is an output, it may semantically match the input guard of $P_2$. The above example should be sufficient to convince the reader that an automatic generation of a general synchronizing layer is impossible, unless an extra level of synchronization is introduced. The tag
Example 3:

\[ Z^2 = \{ Z_1 \mid Z_2 \}. \]

\[ L = [P_1 \cdot [P_2 \cdot \text{z} \rightarrow \text{skip} \mid P_2 \cdot \alpha \rightarrow \text{skip} \mid \text{true} \rightarrow \text{skip} ]] \]

\[ \mid P_2 \cdot [P_1 \cdot \text{z} \rightarrow \text{skip} \mid P_2 \cdot \alpha \rightarrow \text{skip} \mid \text{true} \rightarrow \text{skip} ] \]

facility of the CSP notation provides this required level. Tagging the communications ensures that no command in \( S \) will syntactically match a command in \( Z^n \). At runtime, the use of tags results in inefficiency as additional checks have to be performed.

On the other hand, if the user supplies layers together with their proofs, the task of detecting that a layer is non GTCC can be performed automatically, and thus, the compiler can in certain cases tailor the synchronizing code according to the detected closeness violation.
Acknowledgements

We would like to thank Nissim Francez and Orna Grumberg for helpful discussions.

References


