On proving communication closedness of distributed layers

86.07

chnology

On Proving Communication Closedness of Distributed Layers

Rob Gerth

Liuba Shira

August ,1986

86.07

On Proving Communication Closedness of Distributed Layers

Rob Gerth

Liuba Shira

August ,1986

86.07

to appear in the 6th Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
editor: F.A.J. van Neerven

On Proving Communication Closedness of Distributed Layers*

Rob Gerth[1,2,3] and Liuba Shrira[4,5]

August, 1986

Abstract. The notion of *communication closed layer* has been introduced as a way to define structured composition of distributed systems. An interesting question is how to verify the closedness of a layer. We formulate a proof rule proving closedness of a distributed layer. The rule is developed as an extension of the Apt, Francez and de Roever proof system for CSP. The extension is proved to be sound and relatively complete.

1. Introduction

A recent paper of Elrad and Francez [EF82] introduces a novel methodology of analysing 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

*This research was carried out during the first author's stays at the Department of Computer Science of the Technion in May 1985 and May 1986.

[1]Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, the Netherlands. Electronic mail: mcvax!eutrc3!wsinrobg.UUCP.

[2]Supported in part by the Foundation for Computer Science Research in the Netherlands (SION) with financial aid from the Netherlands Organization for Pure Scientific Research (ZWO).

[3]The author is working in and supported by ESPRIT project 937: "Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES)".

[4]Laboratory for Computer Science, MIT, Cambridge MA 02139, USA and Department of Computer Science, Technion, Haifa 32000, Israel. Electronic mail: liuba@mit-mc.ARPA.

[5]Supported in part by an HTI Postdoctoral Fellowship.

is helpful in analysis of distributed programs and also useful in systematic construction of distributed programs.

An independent algorithmically oriented paper of [SFR83] uses automatically enforced communication closed layers in a transformational technique for recursive distributed algorithm synthesis.

A sequel paper [SF85] also reasons with enforced communication closed layers while converting the program transformation in [SFR83] into a transformation of proofs.

The present paper 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 the construction of a larger program in which 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].

When automatically constructing a program with given layers, both [EF82] and [SF85] enforce communication closedness 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 formally verifying this noninteraction property in order to omit the synchronizing part and thus to improve the overall message complexity of the resulting program.

A straightforward approach to the formal proof of layer non-interaction, is to consider the complete program without the synchronizing parts and to show that no semantic matches occur between i/o actions belonging to different layers. This implies in fact the construction of cooperating proof outlines. [EF82] and [SF85] deal with proving layer communication closedness in a composed program in the above straightforward way.

The immediate problem with this approach is, that constructing such proof outlines is of the same order of complexity as verifying that the program as a whole behaves properly. So, it does not make sense to concentrate on communication closedness of the layers comprising this program. The second problem is that this approach only shows closedness with respect to a particular appended layer. This runs counter to the idea stated earlier of having such layers as "on-the-shelf" building blocks: This clearly supposes a notion of closedness that is independent of its environment.

In this paper, we take an alternative 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 analyse 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" closedness. 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 with the original definition of closedness, which is relative to a specific context. For this new definition, we suggest a sound and complete proof rule for proving general closedness 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. The only related work that we know of is [A85]. That paper introduces a static, syntactic analysis of CSP programs so as to restrict a priori the set of pairs of communication commands whose interaction have to be considered during program verification. No attempt is made, however, to connect this with a design methodology such as using closed layers.

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

We finish the introduction with a quick recapitulation of the version of the language CSP [H78] as it is used here, and of the CSP proof system [AFR80].

(1) The basic command of CSP is $[P_1 \| \cdots \| P_n]$ expressing concurrent execution of the sequential processes $P_1, \ldots, P_n$.

(2) Every P_i refers to a sequential statement S_i by $P_i::S_i$. No S_i contains variables subject to change in S_j $(i \neq j)$.

(3) Communication between P_i and P_j $(i \neq j)$ is expressed by the receive and send primitives, $P_j?x$ and $P_i!t$ (t a term), respectively. Execution of $P_j?x$ in $(P_i::S_i)$ and of $P_i!t$ (in $P_j::S_j$) is synchronized and results in the assignment of the value of t to the variable x. Such a pair of i/o commands is called a *syntactically matching* pair.

(4) In CSP conditional statements, a guard may be of the form $b;\alpha$, too, where α is an i/o command (see example 2). Such a *mixed* guard can be passed if the boolean expression, b, evaluates to true, α has a communication partner and the communication has been performed (i.e., a *semantic match* occurred). Apart from mixed guards, there are also pure boolean guards with obvious meaning. A guard evaluates to true if its boolean part does. Guards of the form *true* $;\alpha$ will often be written as α.

(5) Our version of CSP does not make use of the *distributed termination convention*[6]

(6) To simplify matters, loops can only have pure boolean guards. Since we do not make use of *DTC*, this does not restrict the generality of the language.

The CSP proof system of [AFR80] is based on the notion of *cooperation* of proof outlines. First the sequential CSP-processes are proved correct, using the following axiom: $\{p\}\alpha\{q\}$, where α is any i/o command. Such a proof associates with every sub-statement of a process, a pre and a post assertion; hence, associates a *proof outline* with a CSP-process. The post assertions of i/o commands can be anything and will contain assumptions about communications. These assumptions

[6]The distributed termination convention of CSP (*DTC*) is a mechanism that allows the remote sensing of partner processes' termination and is used to disable communication guarded alternatives and to exit communication guarded loops due to process termination.

have to be checked and this is the function of the cooperation test.

Basically, this test prescribes that for any two syntactically matching i/o commands and associated pre and post assertions, say $\{p_i\}P_i!e\{q_i\}$ and $\{p_j\}P_j?x\{q_j\}$, if there is in fact a semantic match, then after the communication the post assertions should hold: $\{p_i \wedge p_j\}x:=e\{q_i \wedge q_j\}$.

A problem is that $p_i \wedge p_j$ in general is too weak to express the non-occurrence of semantic matches. Hence, the introduction of a *global invariant*, *GI*, *auxiliary variables* and *bracketed sections*. Auxiliary variables are needed so that *GI* can express globally which communications occurred. Bracketed sections, $<S_1;\alpha;S_2>$, are uniquely associated with i/o commands and restrict the updatings, S_1 and S_2, of the (auxiliary) variables in *GI*.

The cooperation test now prescribes that for any two syntactically matching i/O commands with associated bracketed sections, say $\{p_i\}<S_i;P_i!e;\bar{S}_i>\{q_i\}$ and $\{p_j\}<S_j;P_j?x;\bar{S}_j>\{q_j\}$, the following formula holds:

$$\{p_i \wedge p_j \wedge GI\}S_i;S_j;x:=e;\bar{S}_i;\bar{S}_j\{q_i \wedge q_j \wedge GI\}$$

Given a set of proof outlines, $\{p_i\}P_i::S_i\{q_i\}$ i=1..n, that cooperate with respect to some *GI*, the parallel composition rule allows the conclusion that

$$\{\bigwedge_{i=1}^{n} p_i \wedge GI\}[P_1::S_1 \| \cdots \| P_n::S_n]\{\bigwedge_{i=1}^{n} q_i \wedge GI\}.$$

The reader is referred to [AFR80] for an exposition of the proof system and of the ideas on which it is based.

The above discussion only makes sense relative to a fixed interpretation for CSP and for the proof systems. Only then, is it possible to talk about semantics of CSP and validity of partial correctness specifications. Although mostly left implicit, we fix for the rest of the paper some interpretation I.

2. The notion of general tail communication closedness

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

Definition 1:

A layer $\mathbf{S}=[S_1\|\cdots\|S_n]$ is a *General Tail Communication Closed (GTCC)* layer w.r.t. a set of states Σ, $GTCC_I(\mathbf{S},\Sigma)$, *iff* for arbitrary layers $\mathbf{T}=[T_1\|\cdots\|T_n]$ and for all I-executions starting in states from Σ of the distributed program $\mathbf{P}=[S_1;T_1\|\cdots\|S_n;T_n]$, no synchronization occurs between a communication command in a process of **S** and a communication command in a process of **T**.

In the sequel, we will often ignore the I-subscript and write $GTCC(\mathbf{S},\Sigma)$

Using the terminology of [AFR80], the definition requires that there are no semantic matches between communication commands in **S** and **T**. Observe that any layer **S** is trivially $GTCC(\mathbf{S},\emptyset)$. We now state the additional assumptions we make about programs. Remember that a *layer* S is just any concurrent n-process program $\mathbf{S}=[S_1\|\cdots\|S_n]$. We assume that **S** is given together with a (partial) correctness proof of a specification for it. We are interested in exploring the precise conditions for the behaviour of **S** not to be affected by appending another layer

T to it. In order to concentrate on this aspect, *we assume that layers do not admit computations other than (properly) terminating ones.* If he so wishes, the reader can remove this restriction by substituting total correctness for partial correctness throughout the paper and by adding a deadlock freedom proof as a premis of the rule developed below.

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

Example 1:

$$\mathbf{S}=[P_1::P_2!a \,\|\, P_2::P_1?x\,]$$

Obviously, **S** is a *GTCC* layer with respect to any set of initial states. I.e., for no layer **T** appended to **S**, communication will occur between commands in **S** and **T**.

Example 2:

$$\mathbf{S}'=[P_1::[P_2!a \rightarrow skip \,\Box\, true \rightarrow skip\,] \,\|\, P_2::[P_1?x \rightarrow skip \,\Box\, true \rightarrow skip\,]].$$

S$'$ is a terminating layer when executed alone and does not use the *DTC* convention of *CSP*. Still, *GTCC*$(\mathbf{S}',\Sigma)$ does not hold for any non empty Σ. Consider for example **S**$'$ composed with the layer S from example 1:

$$[P_1::[P_2!a \rightarrow skip \,\Box\, true \rightarrow skip\,];P_2!a \,\|\, P_2::[P_1?x \rightarrow skip \,\Box\, true \rightarrow skip\,];P_1?x\,].$$

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 a deadlock.

We now compare the new notion of *GTCC* with the original definition from [EF82] of communication closedness (*CC*). The original notion of *CC* layer considers a specific distributed program P with a specification $\{p\}S\{q\}$ and a given decomposition into layers $\mathbf{S}^j$ $(j=1 \cdots d)$, s.t. $\mathbf{S}^j = [S_1^j \| \cdots \| S_n^j]$, and $P = [S_1^1; \cdots ;S_1^d \| \cdots \| S_n^1; \cdots ;S_n^d]$.

Definition 2 [EF82]:

A layer $\mathbf{S}^j$ is a *CC* layer in $\{p\}P\{q\}$ iff there is no execution of **P** starting in a state satisfying p in which a communication occurs between a communication command in a process of a layer $\mathbf{S}^j$ and a communication command in a process of some other layer $\mathbf{S}^k$, $k \neq j$.

Our new notion strengthens the original definition by considering closedness with respect to an arbitrary layer **T**, but also restricts the original notion by requiring tail closedness only. In spite of the restriction, the new definition is able to express the original notion of closedness:

Theorem 1:

Let $\mathbf{S}^1$,..., $\mathbf{S}^d$ be layers such that $\{p^{i-1}\}S^i\{p^i\}$ $i=1,..,d$. Let **P** be the composition of the layers $\mathbf{S}^1$,..., $\mathbf{S}^d$, $\mathbf{P}=[S_1^1; \cdots ;S_1^d \| \cdots \| S_n^1; \cdots ;S_n^d]$. If *GTCC*$(\mathbf{S}^i,p^{i-1})$ holds for $i=1,..,d$ then the layers $\mathbf{S}^1$,..., $\mathbf{S}^d$ are also communication closed (*CC*) layers in $\{p^0\}\mathbf{P}\{p^d\}$ (according to Definition 2).

The proof of this theorem is postponed to the end of section 3.

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

In this section we take a closer look at the notion of *GTCC*, aiming at verifying potential interlayer interactions inside a given layer. To be on firm ground, we formulate 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.

Let us consider a layer **S** that is not *GTCC* with respect to some Σ. By definition there exist a layer $\mathbf{T} = [T_1 \| \cdots \| T_n]$, such that when **T** is appended to **S**, an interlayer communication occurs between a command in **S** and a command in **T**. Remember that from our assumptions, **S** is a perfectly terminating *CSP* layer when executed alone. To characterize this situation solely in terms of **S**, we have to formally give a semantics to **S**. We make use of the denotational linear history semantics of *CSP* following [FLP80][7] and bring briefly its main concepts into remembrance. Since layers always terminate by assumption, we ignore those aspects of the semantics that deal with non-termination and deadlock.

The semantics uses the notion of history of communications. By history is meant a sequence of records of communications. A record is a triple (a,i,j) which is associated with a communication between process P_i and P_j: a is the value sent by P_i to P_j. h is used to denote a history. $h_1\hat{}h_2$ denotes concatenation of histories h_1 and h_2. The i-restriction of h, $h\mid_i$, denotes the communication sequence resulting from h by omitting all the communication records not involving process P_i.

First the semantics of a single process is considered. Any computation defined by S_i reaches a certain state τ (which may equal $\perp$ denoting an incomplete or partial computation) and produces a certain communication sequence h in order to get there. Whether this (partial) 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_i itself characterizes the correspondence between attainable states and the communication sequences required to achieve them. I.e., $M(P_i)$ is a mapping from initial state σ into a set of pairs $<\tau,h>$ representing the existence of computations starting at state σ which reach a state τ with communication history h. (For technical reasons that do not concern us, M is extended in [FLP80] to an automorphism on sets of such pairs.) The meaning of a parallel program $M([P_1\| \cdots \| P_n])$ is obtained by merging the a priori meanings of the P_i's. Only a priori computations that are *compatible* with the other processes in the environment will be merged. Computations (τ_i,h_i) respectively (τ_j,h_j) from P_i respectively P_j are compatible if $h_i\mid_j = h_j\mid_i$.

Our semantics assigns to each statement and starting state the set of partial and complete computations that the statement can perform. It does not include any deadlock information.

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 treated afterwards.

[7]Note that the semantics as described in the final version ([FLP84]), slightly differs from the one used here.

Definition 3:

A layer $\mathbf{S} = [S_1 \| S_2]$ is $GTCC(\mathbf{S},\Sigma)$ iff there are no σ, $\tau \neq \perp$, h, i such that $\sigma \in \Sigma$, $< \cdots , h> \in M(\mathbf{S})(\sigma)$, $< \cdots , h\hat{}\alpha> \in M(S_i)(\sigma)$ and $<\tau,h> \in M(S_{i \bmod 2 + 1})(\sigma)$, where α is a record of a communication between S_1 and S_2.

Paraphrasing this, if a layer violates the closedness condition, definition 3 postulates a (partial) layer computation ($< \cdots ,h>$) that leaves control in S_i in front of an i/o command ($< \cdots ,h\hat{}\alpha>$), but allows $S_{i \bmod 2 + 1}$ to terminate ($<\tau,h>$). The correspondence with definition 1 is obvious.

The extension of definition 3 to the general case ($n > 2$) is not completely straightforward. We consider partial communication histories that are prefixes of terminating ones, and whose corresponding i-restrictions relate the computation in the joint meaning with a computation in the a priori meaning of process P_i.

Definition 4:

A layer $\mathbf{S} = [S_1 \| \cdots \| S_n]$ is $GTCC(\mathbf{S},\Sigma)$ iff there are no σ, $\tau \neq \perp$, h, i and j, such that $\sigma \in \Sigma$, $< \cdots ,h> \in M(\mathbf{S})(\sigma)$, $< \cdots ,h\mid_i\hat{}\alpha> \in M(S_i)(\sigma)$ and $<\tau,h\mid_j> \in M(S_j)(\sigma)$, where α is a record of a communication between S_i and S_j.

We now prove that although this definition is stated solely in terms of the layer $\mathbf{S}$ itself, the notion of *GTCC* defined by it is identical to the notion defined by definition 1. Strictly speaking, we do not so much prove the equivalence of two alternative definitions, as show that definition 4 indeed captures the intuition of definition 1.

Theorem 2:

Fix any interpretation I. A layer $\mathbf{S}$ is $GTCC_I$ by definition 1 iff it is $GTCC_I$ by definition 4.

For its proof, we need some notation and 2 auxiliary lemma's.

Definition 5:

Given layers $\mathbf{S}_i \equiv [S_i^1 \| \cdots \| S_n^i]$ $i=1,2$, a history h and state σ such that $< \cdots ,h> \in M([S_1^1;S_1^2 \| \cdots \| S_n^1;S_n^2])(\sigma)$. Let $h=h'\hat{}\alpha\hat{}h''$ and $\alpha=(i,j,a)$.

- α *involves* S_k^1 iff $< \cdots ,h'\mid_k\hat{}\alpha> \in M(S_k^1)(\sigma)$ (hence i or j equals k).
- α *involves* S_k^2 iff $<\tau,\bar{h}'\mid_k> \in M(S_k^1)(\sigma)$ and $\alpha\mid_k \neq \lambda$[8] for some $\tau \neq \perp$ and prefix $\bar{h}'$ of h'.

Lemma 1:

Given layers $\mathbf{S}_i \equiv [S_1^i \| \cdots \| S_n^i]$ $i=1,2$, its composition $\mathbf{S} \equiv [S_1^1;S_1^2 \| \cdots \| S_n^1;S_n^2]$, a state σ and two histories $h=h'\hat{}\alpha\hat{}\beta\hat{}h''$ and $\bar{h}=h'\hat{}\beta\hat{}\alpha\hat{}h''$.
If α does not involve any S_j^1 and β does not involve any S_j^2, then $< \cdots ,h> \in M(\mathbf{S})(\sigma)$ implies $< \cdots ,\bar{h}> \in M(\mathbf{S})(\sigma)$ and $h\mid_i = \bar{h}\mid_i$ for $i=1..n$.

Proof:

Since α does not involve any S_j^1, α witnesses in fact a communication within $\mathbf{S}^2$. Likewise, β witnesses a communication within $\mathbf{S}^1$. Let $\alpha=(i,j,a)$ and $\beta=(k,l,b)$. Then this implies that $\{i,j\} \cap \{k,l\} = \emptyset$, since both S_i^1 and S_j^1 must

[8]the empty sequence

have terminated for α to occur, whereas neither S_k^1 nor S_l^1 can terminate before β occurs. Consequently, $h \mid_i = \tilde{h} \mid_i$ for all $i = 1..n$ and hence $< \cdots, \tilde{h} > \in M(\mathbf{S})(\sigma)$.

Lemma 2:

Let $\mathbf{S}^i \equiv [S_1^i \| \cdots \| S_n^i]$ $i = 1,2$ and $<\tau, h> \in M([S_1^1; S_1^2 \| \cdots \| S_n^1; S_n^2])(\sigma)$ for some states σ, τ and history h. Let $\{i_1, \ldots, i_k\}$ be a set of indices of processes S_i^2 for which there is an α in h involving S_i^2.
Then, if there is no record α in h involving both some S_i^1 and some S_j^2 (i.e., if there is no intra-layer communication), there is a history $h' = \tilde{h}' \hat{\ } \bar{h}'$ such that $h' \mid_i = h \mid_i$ for $i = 1..n$, $< \cdots, \tilde{h}' > \in M(\mathbf{S}^1)(\sigma)$ and for every $i \in \{i_1, \ldots, i_k\}$ there is a $\tau_i \neq \perp$ such that $<\tau_i, \tilde{h}' \mid_i > \in M(S_i^1)(\sigma)$ and $< \cdots, \bar{h}' \mid_i > \in M(S_i^2)(\tau_i)$.

This lemma has two interesting corollaries. Its full generality is needed in the proofs of theorem 2 and theorem 1.

Corollary 1:

Let $\mathbf{S}^i \equiv [S_1^i \| \cdots \| S_n^i]$ $i = 1,2$. If $GTCC(\mathbf{S}^1, \Sigma)$, then for all states σ and τ $\exists h$ $<\tau, h> \in M([S_1^1; S_1^2 \| \cdots \| S_n^1; S_n^2])(\sigma)$ iff $\exists h'$ $<\tau, h'> \in M(\mathbf{S}^1; \mathbf{S}^2)(\sigma)$, and the histories are related by $h \mid_i = h' \mid_i$ $i = 1,..,n$.

Proof:

Evident.

This corollary makes precise the idea that communication closedness allows layer composition to be treated as sequential composition. In fact, we have the following evident

Corollary 2:

The following rule is sound (same notation as above):

$$\frac{\{p^i\} \mathbf{S}^i \{q^i\},\ q^1 \rightarrow p^2 \qquad GTCC(\mathbf{S}^1, p^1)}{\{p^1\}[S_1^1; S_1^2 \| \cdots \| S_n^1; S_n^2]\{q^2\}}$$

Proof of Lemma 2:

Repeatedly using lemma 1 yields a history h' for which $h' \mid_i = h \mid_i$ $i = 1..n$ and on which no communication involving some S_k^2 precedes any communication involving some S_l^1. Let $h' = \tilde{h}' \hat{\ } \bar{h}'$, where $\tilde{h}'$ is the longest prefix of h' containing only communications involving some S_l^1. Then $< \cdots, \tilde{h}' > \in M(\mathbf{S}^1)(\sigma)$. Moreover for every $i \in \{i_1, \ldots, i_k\}$, S_i^1 must terminate on $\tilde{h}'$ since otherwise the associated communication could not occur: $<\tau_i, \tilde{h}' \mid_i > \in M(S_i^1)(\sigma)$ and $< \cdots, \bar{h}' \mid_i > \in M(S_i^2)(\tau_i)$ (by definition $\tau_i \neq \perp$).

Proof of Theorem 2:

Suppose the layer $\mathbf{S} \equiv [S_1 \| \cdots \| S_n]$ is not $GTCC_T(\mathbf{S}, \Sigma)$ according to definition 1. Then we can find another layer $\mathbf{T} \equiv [T_1 \| \cdots \| T_n]$ and indices i and j, $i \neq j$, such that synchronization occurs between i/o commands in S_i and T_i during some execution of the program $\mathbf{P} \equiv [S_1; T_1 \| \cdots \| S_n; T_n]$ that starts in a state $\sigma \in \Sigma$. Hence, there is a history $\tilde{h}$ and a record α such that $< \cdots, \tilde{h} \hat{\ } \alpha > \in M_I(\mathbf{P})(\sigma)$, $< \cdots, \tilde{h} \hat{\ } \alpha > \notin M_I(\mathbf{S})(\sigma)$ and $< \cdots, \tilde{h} \mid_i \hat{\ } \alpha > \in M_I(S_i)(\sigma)$ (α witnesses a communication between S_i and T_j). W.l.o.g., assume that α is the only such record in $\tilde{h} \hat{\ } \alpha$ witnessing non-closedness. This implies that there is no record in $\tilde{h}$ that involves both some S_l and some T_k. Hence, lemma 2 can be used. This gives a history $\bar{h}=\bar{h}'\hat{}\bar{h}''$ such that $<\cdots,\bar{h}'>\in M_I(\mathbf{S})(\sigma)$ and $<\tau,\bar{h}'|_j>\in M_I(S_j)(\sigma)$ for some $\tau\neq\perp$. Since, $\bar{h}|_i=\tilde{h}|_i$, we also have $<\cdots,\bar{h}'|_i\hat{}\alpha>\in M_I(S_i)(\sigma)$. Hence, $\mathbf{S}$ is not $GTCC_I(\mathbf{S},\Sigma)$ according to definition 4.

Finally, we can give the promised

Proof of Theorem 1:

Suppose we have a history $h\hat{}\alpha$ on which communication closedness (CC) is violated for the the first time by a communication occuring between layers $\mathbf{S}^j$ and $\mathbf{S}^k$, where $j<k$. Say, between S_u^j and S_v^k; $\alpha=(u,v,a)$ witnesses this communication and is in fact the first such witness. Define $\mathbf{S}^{\leqslant i}\equiv[S_1^1;...;S_1^i\,\|\cdots\|S_n^1;...;S_n^i]$ and define $\mathbf{S}^{>i}$ and $\mathbf{S}^{<i}$ analogously ($\mathbf{S}^{<1}\equiv\Lambda$). Let $\Sigma^i\equiv\{\sigma\mid\sigma\models p^i\}, i=1..n$.

Violation of CC implies the existence of states $\tau\neq\perp$ and $\sigma\in\Sigma^0$ and a splitting of h, $h=h_1\hat{}h_2$, such that $<\cdots,h\hat{}\alpha>\in M(\mathbf{S}^{\leqslant k})(\sigma)$, $<\tau_v,h_1|_v>\in M(S_v^{\leqslant j})(\sigma)$ and $<\cdots,h|_u\hat{}\alpha>\in M(S_u^{\leqslant j})(\sigma)$. Since closedness is not violated on h, lemma 2 gives a history $h'=\tilde{h}'\hat{}\tilde{h}'$ such that $h'|_i=h|_i$ $i=1..n$, $<\cdots,\tilde{h}'>\in M(\mathbf{S}^{\leqslant j})(\sigma)$, $\tilde{h}'|_v=h_1|_v$ and $\tilde{h}'|_u=h|_u$. Hence, $GTCC(\mathbf{S}^{\leqslant j},\Sigma^0)$ does not hold.

By definition $GTCC(\mathbf{S}^{<j},\Sigma^0)$ does hold. Hence, corollary 1 implies that there is a h'_1 for which $h'_1|_i=h_1|_i$ $i=1,..,n$ and $<\cdots,h'_1>\in M(\mathbf{S}^{<j};\mathbf{S}^j)(\sigma)$. Split h'_1 as $\bar{h}'_1\hat{}\bar{h}'_1$ such that $<\tau,\bar{h}'_1>\in M(\mathbf{S}^{<j})(\sigma)$ for some $\tau\neq\perp$. By definition this is possible. Then, from the assumptions of theorem 1, $\tau\in\Sigma^{j-1}$. From this we may conclude that $GTCC(\mathbf{S}^j,\Sigma^{j-1})$ is violated, which proves the theorem.

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 [A85]) 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. The proof rule for proving **S** to be *GTCC* is given in Figure 1.

Discussion. First observe that an assertion attached to some point in a program characterizes the states in which execution can arrive at this point. Hence, for $p=1$, the formula $q_2\wedge t\wedge GI\wedge\bigvee_{k=1}^{l}g'_{j_k}\wedge\bigvee_{k=1}^{m}g''_{i_k}$ characterizes those states in which execution is *simultaneously* (*GI*) at the front of an i/o guarded alternative in S_1 (t) and at the end of S_2 (q_2) while S_1 has the option of doing an additional communication ($\bigvee_{k=1}^{l}g'_{j_k}$). Since the i/o guarded alternative has an open local branch, too ($\bigvee_{k=1}^{m}g''_{i_k}$), S_1 need not engage in a communication at this point. Clearly satisfaction of this formula is a sufficient condition for failure of closedness of a layer. We claim (and prove later on) that it is a sufficient condition, too.

Let us now apply the rule to layers **S** and **S**′ from example 1: **S** satisfies the requirements of the proof rule vacuously and can be shown *GTCC* (**S**,*true*), while **S**′ can only be shown *GTCC* (**S**,*false*).

The extension of the above rule to the general case of $n > 2$ processes is straightforward. The rule for proving **S** to be *GTCC* is presented in Figure 2.

A natural question to consider next is wether 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*

Soundness and completeness of the *GTCC* -rule as formalized in theorems 3 and 4, is based on soundness and completeness of the [AFR80] proof system as proved in [A83]. That paper assigns semantics to CSP-statements (actually to a superset of our dialect), using a transition relation, $\overset{h}{\underset{n}{\rightarrow}}{}^*$: $(S,\sigma)\overset{h}{\underset{n}{\rightarrow}}{}^*(S',\sigma')$ means that starting from a state σ, the statement S can evolve into the statement S' in n computation steps, producing a communication history h and transforming σ into σ'. Let Λ denote the "empty" statement. Without proof, we claim that the following relation exists between our semantics and the one from [A83] ($\tau \neq \perp$):

For any statement (or program) S:

$$<\tau,h> \in M(S)(\sigma) \text{ iff } \exists n \quad <S,\sigma> \overset{h}{\underset{n}{\rightarrow}}{}^* (\Lambda,\tau)$$

```
construct proof outlines for $\{r_1\}S_1\{q_1\}$ and $\{r_2\}S_2\{q_2\}$
that cooperate w.r.t. GI ($r = r_1 \wedge r_2 \wedge GI$)
for p=1,2 do
        for any command $[\Box g_i \rightarrow L_i]$ in $S_p$
                with a pre-assertion t and at least one mixed
                alternative and one local alternative, where
                $\{g'_{j_1}, \ldots, g'_{j_l}\}$ is the set of boolean parts of
                the mixed guards and $\{g''_{i_1}, \ldots, g''_{i_m}\}$
                is the set of pure boolean guards
        do
                show $I \models \neg(q_{3-p} \wedge t \wedge GI \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{k=1}^{m} g''_{i_k})$
        od
od
---------------------------------------------
$GTCC([S_1 \| S_2], r)$
```

Figure 1. The rule GTCC for n=2

```
construct proof outlines for $\{r_1\}S_1\{q_1\}, \ldots, \{r_n\}S_n\{q_n\}$
that cooperate w.r.t. GI $(r = \bigwedge_{k=1}^{n} r_k \wedge GI)$
for $p=1..n$ do
        for any command $[\Box g_i \rightarrow L_i]$ in $S_p$
                with a pre-assertion $t$ and at least one mixed
                alternative and one local alternative, where
                $\{g'_{j_1}, \ldots, g'_{j_l}\}$ is the set of boolean parts of the
                mixed guards, referring to processes $S_{j_1}, \ldots, S_{j_l}$
                in their i/o parts, and $\{g''_{i_1}, \ldots, g''_{i_m}\}$ is the set
                of pure boolean guards
        do
                show $I \models \neg \bigvee_{k=1}^{l} (t \wedge q_{j_k} \wedge GI \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{k=1}^{m} g''_{i_k})$
        od
od
```

$GTCC([S_1 \| \cdots \| S_n], r)$

Figure 2. The rule GTCC

In other words, soundness and completeness of the CSP proof system still holds if this paper's semantics is used.

5.1. Soundness of the rule.

With that out of the way, we prove that the proposed *GTCC* rule is sound in the sense of the following theorem. Here, $I \vdash_{GTCC}$ denotes derivability within the [AFR80] proof system augmented with the GTCC rule of Figure 2, and $I \models GTCC(\mathbf{S}, r)$ is defined by $GTCC_I(\mathbf{S}, \{r \mid I, r \models r\})$. N denotes the standard interpretation of Peano arithmetic. I.e., N consists of the natural numbers together with the standard operations (+, *, Suc) on them and the constant 0.

Theorem 1:

For any layer $\{r\}\mathbf{S}\{q\}$,
if N $\vdash_{GTCC}$ $GTCC(\mathbf{S}, r)$ then N $\models$ $GTCC(\mathbf{S}, r)$

Proof:

Non closedness can only occur if a layer can terminate regardless of whether it performs a certain communication or not. Also, by assumption, layers terminate. Now observe that the only way in which a layer can be both non closed and terminating, is if at least one of its components contains an alternative statement of the special form as considered in the *GTCC* rule. In other words, the set of syntactic configurations as defined in the rule entails all potential sources of non closedness.

Next assume that $\mathbf{S} = [S_1 \| \cdots \| S_n]$ is not *GTCC*. First note, that the *CSP* proof system used for the partial corectness proof of the layer **S** is complete with respect to 'expressive' interpretations such as N ([A83]). To derive the contradiction, let q_i, t and GI be the assertions from the presumed proof outlines showing *GTCC*, and q'_i, t' and GI', be the corresponding assertions used in the *CSP* proof system completeness proof [A83]. By definition of the above assertions: $N \models q'_i \wedge t' \wedge GI' \rightarrow q_i \wedge t \wedge GI$.

Choose any state τ. By assumption $N,\tau \models \neg (q_i \wedge t \wedge GI \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{k=1}^{m} g''_{i_k})$, and hence the following holds: $N,\tau \models \neg (q'_i \wedge t' \wedge GI' \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{i=1}^{m} g''_{i_k})$. By definition of the completeness assertions, $N,\tau \models q'_i \wedge t' \wedge GI' \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{i=1}^{m} g''_{i_k})$ claims the existence of a transition $(\mathbf{S},\sigma) \overset{h}{\underset{n}{\rightarrow}}{}^{*} (\mathbf{S}',\tau')$ for some communication history h such that $N,\sigma \models r$, in $\mathbf{S}'$ the i-th component has terminated and the p-th component is at a guarded statement of the required form, and τ' coincides with τ on the variables of the i-th and p-th component. Now, since **S** is terminating, w.l.o.g. we may assume that the above transition is such that only the p-th component in $\mathbf{S}'$ has not terminated yet. By the correspondence with our semantics this means that $<\tau',h> \in M(\mathbf{S})(\sigma)$ and that $< \cdots ,h \mid_i \hat{}\alpha) \in M(\mathbf{S}_i)(\sigma)$ for any α corresponding to an i/o command addressing the i-th process in one of the open mixed guards. Hence, **S** is not *GTTC* with respect to r. The contradiction follows from contraposition.

5.2. Completeness of the rule.

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

Theorem 4:

For any layer **S**, if $N \models GTCC(\mathbf{S},r)$ then $N \vdash_{GTCC} GTCC(\mathbf{S},r)$

Proof:

By contraposition. Since the post-assertion of a layer can be choosen freely in a GTCC proof, cooperating proof outlines can always be constructed. So, $N \not\vdash_{GTCC} GTCC(\mathbf{S},r)$ can only mean that the second premis of the GTCC-rule cannot be made to hold. Assume that the completeness proof outlines and GI are used. By assumption, there is at least some p and some t such that $N \models \neg \bigvee_{k=1}^{l} (t \wedge q_{j_k} \wedge GI \wedge \bigvee_{k=1}^{l} g'_{j_k} \wedge \bigvee_{k=1}^{m} g''_{i_k})$ (notation as in the GTCC rule). hence, there is a τ such that $N,\tau \models \bigvee_{k=1}^{l} (\cdots)$. The rest of the argument is as in the proof of theorem 3.

Example 3:

We demonstrate the usefulness of the *GTCC* rule by using it to complete the formal reasoning in the transformational system of [M85]. In [M85] A .Moitra proposes a novel method for the automatic construction of *CSP* programs out of sequential nondeterministic programs w.r.t. a given data distribution. The method suggests translating each statement of the sequential program into an equivalent *CSP* fragment and then composing sequentialy the resulting fragments to form a complete *CSP* program. The ability to transform each statement separately is an important feature of the transformation based on the so called preservation of sequencing, which is expressed as follows: Let T denote the transformation and S_1, S_2 denote statements in the sequential program. *T preserves sequencing* if

$$T[S_1;S_2]=T[S_1];T[S_2] \qquad (*)$$

In [M85] it is claimed without proof that the suggested transformation preserves sequencing. By using the *GTCC* rule, we are able to prove that for any sequential statement S, the *CSP* fragment $T[S]$, generated by the [M85] transformation, is a *GTCC* layer and thus that (*) holds for T. The author in [M85] remarks that the transformation generates a *CC* layer for each statement. Actually, in order to be able to view translation of each statement in isolation, the notion of *CC* is not sufficient as it is defined relatively to a given context of the statement. The *GTCC* notion captures precisely the required condition.

6. Automatic closedness enforcement, revisited

To facilitate distributed program synthesis with communication closed layers, [EF82] and [SF83] advocate the use of an automatic layer communication closedness enforcement facility. We discuss here the feasability of such a facility in view of our better understanding of the mechanism of communication closedness. The proposed closedness enforcement facility should be used as follows: the programmer suplies code for component layers and specifies their required composition using a special syntactic construct. The compiler generates code to be inserted between the suplied component layers and composes the resulting program from the suplied 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 suplied layer and commands in the synchronyzing code. Let us consider a specific non *GTCC* layer L and some synchronyzing code Z^2 as in example 4 below.

Example 4:

$$Z^2=[Z_1 \| Z_2]$$
$$L = [P_1::[P_2?x \rightarrow skip \,\Box P_2!a \rightarrow skip \,\Box true \rightarrow skip\,] \,\|$$
$$P_2::[P_1?x \rightarrow skip \,\Box P_2!a \rightarrow skip \,\Box true \rightarrow skip\,]].$$

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 synchronyzing layer is impossible, unless an extra level of synchronization is introduced. The *tag* 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 closedness violation.

7. Conclusions and future work

We showed that closedness of a layer can be formally proven in terms of the layer itself. For this, we strengthened the original notion of closedness. The basic strategy in devising the proof rule was the same as for formulating the deadlock freedom test in [AFR80]: Find a *syntactically* determined set of program configurations that include all potential closedness violations, and show that none of these configurations is semantically reachable. We did this for the language CSP. In the full paper we indicate how to do this for Ada, based on [GR84] and for monitor based languages, based on [GR86].

The proof rule is meant as an extension of the [AFR80] proof system. A notable drawback is the non syntax directedness and non compositionality of that proof system. It would be worthwhile to see whether closedness can be shown in a compositional way that is closer to [ZRB85], possibly along the lines of [GR86].

Finally, *GTCC* is a sufficient but by no means necessary condition for the safe composition of two layers. Necessary is only that possible closedness violations of a layer do not find a matching communication in an appended layer. We have formulated a generalization of closedness which allows such conditions to be stated but is nevertheless independent from a layer's environment. We are currently constructing proof rules for verifying such more general properties.

Acknowledgements

We would like to thank Nissim Francez, Orna Grumberg and Gadi Taubenfeld for helpful discussions and Luc Bouge for pointing out a mistake in definition 4. The first author thanks the Technion for its hospitality.

References

[A83] APT K.R. (1983) , Formal justification of a proof system for Communicating Sequential Processes, *J. Assoc. Comput. Mach.* **30**, pp. 197-216.

[A85] APT K.R. (1984), Proving correctness of CSP programs - a tutorial, *in* "Control Flow and Data Flow: Concepts of Distributed Programming (M. Broy, Ed.)", pp. 441-474, NATO ASI Series, Vol. F14, Springer-Verlag, New York.

[AFR80] APT, K.R., FRANCEZ, N., DE ROEVER, W.P. (1980), A proof system for communicating sequential processes, *ACM Trans. Programm. Lang. Systems* **2-3**, pp. 359-380.

[deB80] DE BAKKER, J.W. (1980), **Mathematical Theory of Program Correctness**, Prentice Hall.

[EF82] ELRAD, T., FRANCEZ, N. (1982), Decomposition of distributed programs into communication-closed layers, *Science of Computer Programming* **2**, pp. 155-173.

[FLP80] FRANCEZ, N., LEHMANN, D., PNUELI, A. (1980), A linear-history semantics for CSP, *in* "Proceedings 21st IEEE Confer. Foundat. of Comput. Science".

[FLP84] FRANCEZ, N., LEHMANN, D., PNUELI, A. (1984), A linear-history semantics for languages for distributed programming, *Theoretical Computer Science* **32**, pp. 25-46.

[GR84] GERTH, R., DE ROEVER, W.P. (1984), A proof system for concurrent Ada programs, *Science of Computer Programming*, **4-2**, pp. 159-204.

[GR86] GERTH, R., DE ROEVER, W.P. (1986), Proving monitors revisited: a first step towards verifying object oriented systems, *Fundamenta Informaticae* **IX-4**, North-Holland, to appear.

[H78] HOARE, C.A.R. (1987), Communicating Sequential Processes, *Communications ACM* **21-8**, pp. 666-677.

[M85] MOITRA, A. (1985), Automatic construction of CSP programs from sequential non-deterministic programs, *Science of Computer Programming* **5**, pp. 277-307.

[SF85] SHRIRA, L., FRANCEZ, N. (1985), "A program transformation regarded as a proof transformation", Report TR-371, Department of Computer Science, Technion, Haifa 32000, Israel.

[SFR83] SHRIRA, L., FRANCEZ, N., RODEH, M. (1983), Distributed k-selection: from a sequential to a distributed algorithm, *in* "Proceedings 2nd ACM Confer. Principles of Distributed Computing".

[ZRB85] ZWIERS, J., DE ROEVER, W.P., VAN EMDE BOAS, P. (1985), Compositionality and Concurrent Networks: Soundness and Completeness of a Proofsystem, *in* "Proceedings 12th ICALP", LNCS **194**, pp. 509-520, Springer-Verlag, New York.

86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failure semantics for communicating processes
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview

Available Reports from the Theoretical Computing Science Group

	Author(s)	Title	Classification EUT	Classification DESCARTES
TIR83.1	R. Koymans, J. Vytopil, W.P. de Roever	Real-Time Programming and Synchronous Message passing (2nd ACM PODC)		
TIR84.1	R. Gerth, W.P. de Roever	A Proof System for Concurrent Ada Programs (SCP4)		
TIR84.2	R. Gerth	Transition Logic - how to reason about temporal properties in a compositional way (16th ACM FOCS)		
TIR85.1	W.P. de Roever	The Quest for Compositionality - a survey of assertion-based proof systems for concurrent progams, Part I: Concurrency based on shared variables (IFIP85)		
TIR85.2	O. Grünberg, N. Francez, J. Makowsky, W.P. de Roever	A proof-rule for fair termination of guarded commands (Inf.& Control 1986)		
TIR85.3	F.A. Stomp, W.P. de Roever, R. Gerth	The μ-calculus as an assertion language for fairness arguments (Inf.& Control 1987)		
TIR85.4	R. Koymans, W.P. de Roever	Examples of a Real-Time Temporal Logic Specification (LNCS207)		
TIR86.1	R. Koymans	Specifying Message Passing and Real-Time Systems (extended abstract)	CSN86/01	
TIR86.2	J. Hooman, W.P. de Roever	The Quest goes on: A Survey of Proof Systems for Partial Correctness of CSP (LNCS227)	EUT-Report 86-WSK-01	

TIR86.3	R. Gerth, L. Shira	On Proving Communication Closedness of Distributed Layers (LNCS236)	CSN86/07	
TIR86.4	R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, S. Arun Kumar	Compositional Semantics for Real-Time Distributed Computing (Inf.&Control 1987)	CSN86/08	
TIR86.5	C. Huizing, R. Gerth, W.P. de Roever	Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like Language	CSN86/09	PE.01
TIR86.6	J. Hooman	A Compositional Proof Theory for Real-Time Distributed Message Passing	CSN86/10	TR.4-1-1(1)
TIR86.7	W.P. de Roever	Questions to Robin Milner - A Responder's Commentary (IFIP86)	CSN86/11	
TIR86.8	A. Boucher, R. Gerth	A Timed Failure Semantics for Communicating Processes	CSN86/12	TR.4-4(1)
TIR86.9	R. Gerth, W.P. de Roever	Proving Monitors Revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)	CSN86/13	
TIR86.10	R.Koymans	Specifying Message Passing Systems Requires Extending Temporal Logic	CSN86/14	PE.02
TIR87.1	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept	CSN87/01	