RZ 1053 (#37725) 1/6/81
Communications

Research Report

ON COMMUNICATING FINITE-STATE MACHINES

Daniel Brand and Pitro Zafiropulo

IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

IBM Research Division
San Jose · Yorktown · Zurich

RZ 1053 (#37725) 1/6/81
Communications

ON COMMUNICATING FINITE-STATE MACHINES

Daniel Brand and Pitro Zafiropulo

IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

ABSTRACT. We investigate a model of communications protocols based on finite-state machines. The problem addressed is how to ensure certain generally desirable properties, which make protocols "well-formed", i.e., specifying a response to those and only those events that can actually occur. We determine to what extent the problem is solvable, and describe one approach to solving it.

First author's present address: IBM Thomas J. Watson Research Center,
P.O. Box 218, Yorktown Heights, N.Y. 10598, U.S.A.

October 13, 1980

1. Introduction

The trend towards distributed computing and computer networks is increasing the complexity of communication protocols. Formal methods of specification and analysis are being gradually introduced to handle the complexity. We will briefly review the most common protocol representations and relate our representation to them. There are several excellent surveys[10,23,18,6] on the subject of protocol specification and verification.

The most general model describes protocols as parallel programs[22,14,7]. In this framework one can specify all protocols and most of their properties. The cost of this generality is the undecidability of most properties. Therefore, existing methods of analysis use human assistance, or take advantage of the fact that many protocol features do not use all the generality available. The latter allows a protocol to be analyzed as if it were described in a less general formalism.

A Petri net[17,24,19,12] is a less general model. In this formalism, protocols are more easily analyzable, and some properties undecidable for programs are decidable for Petri nets (for example, boundedness[15]). The reduction in expressive power by comparison with programming languages is evident in some protocols that require a very large Petri-net description.

The least powerful model is that of a single finite-state machine describing the system including all the component processes and interconnecting channels[1,9,13,4]. In this model only certain protocols can be described. (For example, a protocol allowing an arbitrary number of messages in transit cannot be described.) But describable protocols are relatively easy to analyze in the sense that all properties are decidable by exhaustive analysis.

In order to improve the expressive power of finite-state machines and Petri nets, as well as to reduce the size of their descriptions, a number of extensions has been

proposed[17,5,21,27]. These extensions usually employ some programming language capability, which in turn makes analysis harder.

Our model[25,29] uses explicit finite-state machines to represent processes, and implicit queues to represent channels (details in Section 2). The processes communicate by sending messages to one another via the channels. The queues modeling the channels have unbounded capacity to represent protocols allowing an arbitrary number of messages in transit. In a physical implementation all channels must be bounded, but the bound may be too large to be of practical use. Moreover, since protocols are supposed to operate over different channels with different capacities, a channel of unbounded capacity is the proper abstraction.

This model is in a certain sense as powerful as programming languages, for the unbounded channels can in principle be used as memory devices (see Appendix A). However, this is not the intention; in modeling a given physical communication system, each process is supposed to be represented by only one finite-state machine, and the channels are to be used for communication only. With such a restriction, the model is less powerful in modeling processes than Petri nets (e.g., an unbounded counter cannot be represented). But it is more powerful in modeling channels (e.g., the FIFO property is built in).

The suitability of the model for representing and analyzing communication protocols has been demonstrated[26,20,21] on existing protocol designs. It is particularly appropriate in situations where the propagation delay is not negligible (so that several messages can be in transit at one time), and in situations where it is natural to describe the protocol parties and the communication medium as separate entities.

Our protocol model is described in Section 2. Section 3 defines when a protocol is well-formed, and Section 4 presents an approach to ensuring this property. Section 4 first characterizes all well-formed protocols if the processes are restricted to being trees, rather

than general transition graphs. This characterization is then used in a (partial) solution for general protocols.

The approach has been developed for a protocol synthesizer described previously[29,8] and is based on syntactical properties derived from notions of physical causality and completeness[28]. Because this paper is not directly concerned with the practical aspects of such a protocol synthesizer, we will describe only the essential properties of the model and the approach.

2. *Model of Protocols*

Our model of protocols employs a simple and commonly used representation of the communicating processes[1,4], namely, each process is a finite-state machine. Each pair of communicating processes is connected by a full-duplex, error-free, FIFO channel.

Our notation for expressing transmissions and receptions is illustrated in Figure 1. A "-" sign identifies the transmission of a message, a "+" sign its reception. When a process is in a state from which there is an arc labelled -m then it can traverse the arc and enter a new state. (For example, process USER when in state READY can traverse the arc labelled -REQ and enter state WAIT.) By traversing such an arc the message m is transmitted to the destination process via the connecting channel. (The destination process of a message will not be indicated explicitly by its transmission; this information will be either in an accompanying text or can be inferred by observing which process contains receptions of the message.)

There are no assumptions on the time that a process spends in a state before sending a message, and there are no assumptions on the time that a message spends in a channel before it is delivered to its destination. If and when a message m arrives at a process that is in a state s, then the process will enter a new state by traversing arc labelled +m. (If there is no arc +m attached to state s then the protocol is incorrect -- see Section 3.)

The example of Figure 1 is a simple protocol between two processes, USER and SERVER. Initially, both are in their initial states -- READY and IDLE. USER can send a request (message REQ) to SERVER. After receiving REQ, SERVER enters state SERVICE; when it is finished with the service it goes back to state IDLE while sending the message DONE to USER. Between sending REQ and receiving DONE, USER stays in state WAIT.

While idle, SERVER can detect a fault in itself. If so, it informs USER about it by sending the message ALARM. When USER receives an ALARM, it registers it and directs SERVER back to its IDLE state by the message ACK.

Now we define the model formally.

Definition 2.1. A *protocol* is a quadruple

$$<<S_i>_{i=1}^N , <o_i>_{i=1}^N , <M_{ij}>_{i,j=1}^N , succ>, \quad (1)$$

where

N is a positive integer [representing the number of processes],

$<S_i>_{i=1}^N$ are N disjoint finite sets [S_i represents the set of states of process i],

each o_i is an element of S_i [representing the initial state of process i],

$<M_{ij}>_{i,j=1}^N$ are N^2 disjoint finite sets with M_{ii} empty for all i [M_{ij} represents the messages that can be sent from process i to process j],

succ is a partial function mapping for each i and j

$S_i \times M_{ij} \rightarrow S_i$ and $S_i \times M_{ji} \rightarrow S_i$

[succ(s, x) is the state entered after a process transmits or receives message x in state s. It is a transmission if x is from M_{ij} , it is a reception if x is from M_{ji}].

For example, in Figure 1,

$N = 2$

S_1 = {READY, WAIT, REGISTER},

S_2 = {IDLE, FAULT, SERVICE},

o_1 = READY,

o_2 = IDLE,

M_{12} = {REQ, ACK},

M_{21} = {ALARM, DONE},

succ(READY, REQ) = WAIT,

succ(WAIT, DONE) = READY,

and so on according to Figure 1. The function succ is defined only for eight argument pairs, for there are eight arcs in Figure 1. An example of an undefined succ value is succ(READY, DONE) or succ(SERVICE, ALARM).

The requirement that all the sets M_{ij} be disjoint is merely a notational convenience and does not represent a restriction in practice. If two processes can send the same message then one of the messages can be renamed to achieve distinctness.

There are extensions[25,21,29,8] of the model, which are made for the handling of practical protocols. For example, a state can be marked as transient[25] so that it cannot receive messages, or the channels may not be FIFO and allow taking over of messages. Such extensions have little impact on the results of this paper and will not be discussed.

Notation.

(i) We will use subscripts to identify processes. Thus, s_i will always refer to a state in process i, that is, to a member of S_i. Similarly, x_{ij} will refer to a member of M_{ij} , that is, a message that can be sent from process i to process j. If it is not necessary to indicate where a state s or a message x belongs, then we will not use indices. Unsubscripted capital letters Q,

R, S, T will be used for sets of states, each from a different process. X, Y, Z will be used for sequences of messages.

(ii) An N-tuple S will always consist of N states $<s_1, ..., s_N>$, one from each process. If a function f is defined for states then we will extend it to N-tuples component-wise :

$f(<s_1, ..., s_N>) = <f(s_1), ..., f(s_N)>$.

If a relation ρ is defined between states then we will extend it component-wise to N-tuples:

$S \rho S'$ iff $s_i \rho s_i'$ for all i.

(iii) We will use juxtaposition to express concatenation of message sequences. For example, if x and y are messages and X and Y are message sequences, then x, xy, xY, XY are examples of message sequences. The first one, x, is of length one, the second xy is of length two. We will use the symbol ε to denote the empty sequence. If a relation ρ is defined between messages then we extend it component-wise to sequences:

$\varepsilon \rho \varepsilon$, and $xY \rho x'Y'$ iff $x \rho x'$ and $Y \rho Y'$.

(iv) We will extend the function succ to a sequence X of messages in the natural way:

$succ(s, \varepsilon) = s$, and $succ(s, xY) = succ(succ(s,x), Y)$.

(v) For readability purposes we use "+" sign to identify a reception and a "-" sign to identify a transmission. For example, we may write $succ(s_i, +x_{ji})$ or $succ(s_i, -y_{ij})$; here the "+" and "-" signs provide no new information because the indices indicate what is a reception and what is a transmission. If we write succ(s, +x) then the "+" sign does provide some information; namely, that there exist i and j so that s is in S_i and x is in M_{ji} (rather than M_{ij}). For a message sequence X if we write $succ(s, +X) = s'$, then X contains receptions only.

Properties of a protocol will be defined in terms of its execution. An execution is a sequence of global states[25], where each global state gives the state of all the processes and contents of the channels. An execution must start from the initial global state where each

process i is in its initial state o_i and all the channels are empty. The execution can then proceed; at each step either

(i) a message is sent (process i sends a message x_{ik} , i.e., it is appended to the right of the channel c_{ik}), or

(ii) a message is received (process k receives a message x_{ik} , i.e., it is removed from the left of the channel c_{ik}).

Formally we define:

Definition 2.2. A *global state* [for a protocol of the form (1)] is a pair $<S,C>$ where S is an N-tuple of states s_1 , ..., s_N [s_i represents the current state of process i], and C is an N^2-tuple $<c_{11}$, ..., c_{1N} , c_{21} , ..., $c_{NN}>$ where each c_{ij} is a sequence of messages from M_{ij}. [The message sequence c_{ij} represents the contents of the channel from process i to j. Note that every c_{ii} is empty, for M_{ii} is empty.]

Definition 2.3. We define a binary relation $|\!-$ on global states [meaning that one global state can produce the other in one step of execution]:
$<S,C> \; |\!- \; <S',C'>$ iff there exist i, k, and x_{ik} satisfying one of the two following conditions.

(i) All the elements of $<S,C>$ and $<S',C'>$ are equal except
$s_i' = succ(s_i , -x_{ik})$ and $c_{ik}' = c_{ik}x_{ik}$.

(ii) All the elements of $<S,C>$ and $<S',C'>$ are equal except
$s_k' = succ(s_k , +x_{ik})$ and $c_{ik} = x_{ik}c_{ik}'$.

Definition 2.4. Let $|\!-\!^*$ be the reflexive and transitive closure of $|\!-$. Then we say that a global state $<S,C>$ is *reachable* iff $<S^\circ,C^\circ> \; |\!-\!^* \; <S,C>$, where $<S^\circ,C^\circ> = <<o_i>_{i=1}^N , <\varepsilon>_{i,j=1}^N>$. [$<S^\circ,C^\circ>$ is the initial global state with all channels empty and each process i in its initial state o_i.]

3. The Problem

We do not try to prove that a protocol performs its intended function. Instead, we consider certain properties of interest to all protocols independent of their intended functions. As discussed in [20,25,29] missing or unexecutable receptions usually indicate an error of serious consequence in the protocol. Avoiding errors of this kind is the main subject of this paper. Additionally, some interesting properties (e.g., deadlock) can also be discovered from so called stable N-tuples -- reachable global states with all channels empty .

A protocol has a reception missing if a message x can arrive at a process when it is in a state s, but the protocol does not specify what state should be entered upon receiving x in state s. For example, in Figure 1 consider the execution where USER sends REQ and SERVER sends ALARM before either message arrives at its destination. Message ALARM arrives when USER is in state WAIT, and the protocol does not specify what should happen in that situation. Similarly, there is a missing reception of message REQ in state FAULT.

Now consider Figure 2 which is obtained from Figure 1 by adding the two receptions discussed above. It has all executable receptions specified. (An example of an unexecutable reception would be obtained, for example, by adding a reception +DONE to state READY.) We say that a protocol with all executable receptions specified and only executable receptions specified is well-formed.

While the protocol of Figure 2 is well-formed, it contains the possibility of a deadlock. In the execution considered above, the two processes first send the messages REQ and ALARM. They are both received, and then the two processes remain in states WAIT and FAULT for ever because no other message can arrive. We say that a protocol can be deadlocked if it is possible to reach a global state where all channels are empty and there is no transmission from any state. It has been shown[3,25,21] that N-tuples of states reachable

with all channels empty (in this paper called stable N-tuples) are also useful for detecting losses of synchronization. Therefore, it is important to identify such stable N-tuples.

Definition 3.1. In a protocol of the form (1):

(i) A *reception* is a pair <s, x> for a state s and a message x. A reception <s, x> is *specified* iff succ(s, +x) is defined.

(ii) A reception <s, x> is *executable* iff there exists a reachable global state <S,C> where for some i and k, $s = s_k$ and c_{ik} is of the form xY for some sequence Y.

(iii) A protocol is *well-formed* provided each reception is specified iff it is executable.

(iv) We say that an N-tuple S of states is *stable* iff $<S, <\varepsilon>_{i,j=1}^{N}>$ is reachable.

Experience[26,20,21] shows that it is important for a protocol to be well-formed independently of its function, and also that the knowledge of stable N-tuples allows the detection of significant errors. There have been two strategies for assisting a protocol designer in these aspects. Protocol validation[25,28] assumes a given protocol and informs the designer of any missing or unexecutable receptions, deadlocks, etc. Protocol synthesis[29,8] assumes input given interactively by the designer and constructs mechanically a protocol guaranteed to be well-formed. From a theoretical point of view, both approaches have to solve the same problem:

Given a protocol, identify all executable receptions and all stable N-tuples. (2)

In the case of validation, the given protocol is the one constructed by a human designer; in the case of synthesis, it is the partially constructed protocol at a certain stage during the protocol's construction.

We cannot expect a general solution for (2). Appendix A contains a proof that it is undecidable whether a given reception in a given protocol is executable. This implies

undecidability of other questions of interest -- Is a given protocol well-formed? Is a given N-tuple stable? Is deadlock possible? Therefore, we can expect a solution only for some classes of protocols; we can get an idea what classes might or might not be solvable by considering the proof of Appendix A. It reduces the halting problem to problem (2) using the channels to represent the tape of a Turing machine. This can be done because in our model the channels have infinite capacity. Therefore, we can expect that solvability will depend on restricting what messages can be in transit at any one time.

Definition 3.2. We say that the channel from process i to j is *bounded* by a constant h iff for every reachable global state $<S,C>$, c_{ij} is a sequence of length at most h. If no such constant h exists then we say that the channel is *unbounded.*

All the channels in the protocols of figs. 1 and 2 are bounded by one. A simple example of an unbounded channel is given by any protocol where succ(s, -x) = s (i.e., a state s has a loop transmitting x). Such a protocol can transmit x any number of times before the first x is received by the other party.

Boundedness of channels is again, in general, undecidable, but many practical protocols do have all their channels bounded. Protocols with unbounded channels usually use them in a simple manner, which makes them worth considering. Appendix C gives a sufficient condition for bounded channels.

The problem (2) is solvable for the class of protocols with all channels bounded. As we will show below, the problem is also solvable for the class of protocols with N = 2 processes and only one channel unbounded. In contrast, the problem is unsolvable for N > 2 when any one channel is unbounded. More generally, the problem is unsolvable for the class of protocols with an unbounded channel, say from process i to j, if there exists another way of passing information from i to j. For example, if we have three processes then we can use the third process just as a relaying station between processes i and j bypassing the unbounded

channel. The same situation would exist for N = 2 if we extended our model by allowing more than one channel from process i to j.

For those protocols where a complete solution to problem (2) is not available, one can provide approximate solutions. For example, [25] uses a channel bound h as a parameter of validation. A validated protocol is guaranteed to have all those receptions specified that are executable using global states with channels shorter that h.

4. A Solution

Most existing approaches[23,25,14] to problem (2) are based on a search of reachable global states. We present an alternative approach; instead of considering an execution of all the processes as a whole, we consider N executions of the N processes separately. This has a potential for reducing the combinatorial explosion because in a very crude analysis, the number of ways to reach a global state grows faster than the number of ways to reach a state of one process. However, this is a consideration about worst-case behavior, which is not necessarily a relevant measure here; because in both approaches a protocol can be analyzed successfully only if its behavior is far from the worst case, as is true for protocols designed in practice. Therefore, some experimentation will be necessary before practical effectiveness of our approach can be determined.

There are reasons to expect the existence of a solution that does not search all reachable global states. Figure 3 contains a portion of a protocol where we wish to decide whether state s can receive message x. The reader can see that the answer is in the affirmative, provided state t can receive x; because an execution, which delivers x before transmitting y, implies another possible execution that differs only in that it is slower in delivering x than in transmitting y. Note that the answer to this question was independent of the rest of the protocol; we only needed the information that s = succ(t, -y) and that succ(t, +x) is

defined. Using this motivation, we aim for rules characterizing all the executable receptions with minimum information about the rest of the protocol.

The method proposed here builds N trees, one for each process, representing all the possible executions of each process. The N trees constitute a protocol according to Definition 2.1, but because of its restricted form we will call it a tree protocol. Figure 4 shows a tree protocol corresponding to the general protocol of Figure 2. Traversing a tree is like traversing the corresponding graph, but with two differences. First, one state s of a process in Figure 2 is represented in Figure 4 by several separate states s.0, s.1, s.2, ...; each corresponds to a different way of reaching state s from the initial state. (For example, in Figure 4 states READY.0, READY.1, READY.2 all represent the same state READY of Figure 2.) Secondly, messages are renamed so that no message is transmitted from two different states in the tree. (For example, -REQ.0, -REQ.1, and -REQ.2 represent three different transmissions of the same message REQ.) It should be stressed that the messages REQ.0 and REQ.1 (as an example) are completely different from the point of view of the tree protocol. They merely share their reasons for being in the tree protocol.

It should be clear that given a general protocol as in Figure 2 we can construct tree protocols (like Figure 4) by tracing all possible executions. In general, there is an infinite number of such tree protocols depending on how long we trace the executions. Conversely, given a tree protocol we can construct the corresponding general protocol by merging equivalent states and messages. This is the basis of our approach; we grow the N trees and interpret any findings about the trees (executable receptions, stable N-tuples) in terms of the general protocol.

Thus, we reduce the general problem (2) into two subproblems. First, how to solve problem (2) for tree protocols. And secondly, when to terminate growing of the trees without missing any receptions or stable N-tuples in the corresponding general protocol. The first

problem is solved in Section 4.1. The second problem is, in general, unsolvable and a limited solution is presented in Section 4.2. But first we will define a tree protocol.

Definition 4.1. We will call a protocol of the form (1) a *tree protocol* iff the following two conditions are satisfied:

(i) For every state s_i there is a unique sequence X of messages so that $s_i = succ(o_i , X)$.

(ii) For every message x there is a unique state t so that succ(t, -x) is defined.

Condition (i) is the usual definition of a tree. The unique sequence X will be called the *branch* leading from the *root* o_i to state s_i. Condition (ii) is independent of any tree-like qualities; we introduce it because it will be convenient to have a unique state from which a message can be transmitted. If succ(t, -x) is defined then we will call t the *departure* state of the transmission of x and will call succ(t, -x) the *entry* state of the transmission.

Condition (ii) does not constitute a restriction on the protocols that the approach can handle. It merely implies that in constructing a tree protocol from a general protocol, every new transmission must be renamed.

Notation. In a tree protocol:

(i) $s \leq s'$ iff there exists a (possibly empty) sequence X so that succ(s, X) = s'. We call X the *path* from s to s'. If X is not empty then $s < s'$.

For example, in Figure 4 READY.0 $\leq$ WAIT.1, but neither REGISTER.0 $\leq$ WAIT.1 nor WAIT.1 $\leq$ REGISTER.0. In particular, note that $s \leq s'$ only if s and s' are in the same process.

(ii) $s \lesseqgtr s'$ iff $s \leq s'$ or $s' \leq s$. [That is, s and s' are on the same branch.]

(iii) $\max(s, s') = s'$ if $s \leq s'$,

$\max(s, s') = s$ if $s' \leq s$, and

$\max(s, s')$ is undefined if it is not the case that $s \lesseqgtr s'$.

4.1 ARC SPECIFICATION FOR TREE PROTOCOLS. This section shows how to construct well-formed tree protocols, independent of any general protocol that might correspond to the tree protocol. It will give necessary and sufficient conditions for a reception to be executable. These conditions will be expressed in terms of three functions: From, To, and L. All three take two arguments -- a process identifier i and a state s.

$\text{From}_i(s)$ is the last message received from process i on the branch to state s. If no such message exists then $\text{From}_i(s) = *$, where * is a special symbol used for this purpose. For example, in Figure 4 $\text{From}_2(\text{REGISTER.2}) = \text{ALARM.1}$, $\text{From}_2(\text{WAIT.0}) = *$. In general, $\text{From}_i(s_i) = *$ for any s_i.

$\text{To}_i(s)$ is the last message transmitted to process i by the branch to state s. For example, $\text{To}_2(\text{READY.1}) = \text{REQ.0}$, $\text{To}_1(\text{SERVICE.0}) = *$. In general, $\text{To}_i(s_i) = *$ for any s_i.

$L_i(s)$ is the last state in process i that *must* have been reached before state s can be reached. (L is precisely defined in Definition 4.3.) For illustration consider the protocol of Figure 5. Process 2 sends message 1 to process 1 followed by message 2 to process 3. Upon receiving message 2 process 3 sends message 3 to process 1. Process 1 follows two different paths, depending on whether message 1 or 3 arrives first. In determining the value of $L_2(s_1)$ we take the position that we can observe the state of process 1 only, and try to say as much as possible about the state of process 2. Since +1 has been received, process 2 cannot be in state o_2; it must be in state t_2 or further. But we know even more. Since message 3 cannot be sent without message 2 being sent first, we know that process 2 must have entered state s_2. In fact, in this example $L_2(s_1) = s_2$ and $L_3(s_1) = s_3$.

Using the same reasoning $L_2(t_1) = s_2$. Thus, when process 1 reaches state t_1 it knows that process 2 must have already sent message 1. This ability of a process to find out that a message has been sent to it before receiving the message is not present for protocols with $N = 2$ processes only. Therefore, for two processes the solution[29] is different from the general solution presented here.

When a process i reaches a state s_i then, of course, it knows that process i is in state s_i; therefore always $L_i(s_i) = s_i$. Thus, the function L assigns an N-tuple to every state in the protocol: $L(s) = <L_1(s), ..., L_N(s)>$.

Using the function L we express three conditions, which we will later prove to be necessary and sufficient in order that a state r_k can receive a message x_{ik} (see Figure 6):

(i) Process k can receive x_{ik} only after it has received all messages previously sent to it by process i. This first condition is expressed by

$$From_i(r_k) = To_k(t_i). \tag{3}$$

(ii) If r_k is to receive x_{ik} then the knowledge possessed by processes i and k about any third process j must be consistent in the following sense: When process k is in state r_k then process j must have reached state $L_j(r_k)$. After process i transmits x_{ik} entering state t_i' then process j must have reached state $L_j(t_i')$. These two states of process j must lie on the same branch, or using the notation introduced before Section 4.1,

$$L_j(t_i') \lesseqgtr L_j(r_k) \text{ for all } j \neq k. \tag{4}$$

For $j = i$, (4) simplifies to $t_i' \lesseqgtr L_i(r_k)$.

(iii) In order that r_k can receive x_{ik} , condition (4) must also be satisfied for $j = k$, in which case it can be simplified to $L_k(t_i) \lesseqgtr r_k$. But this is not sufficient because it allows $r_k < L_k(t_i)$. If that were the case then process k must have passed state r_k by the time x_{ik} has been transmitted, and therefore r_k could not receive x_{ik}. Therefore, we require

$$L_k(t_i) \leq r_k. \tag{5}$$

Using these three conditions we now define a constructible reception and a constructible protocol. Later we will prove that "constructible" is equivalent to "executable". The function L is defined in Definition 4.3 together with a constructible tree protocol because they are interdependent -- constructible protocols are characterized in terms of the function L, and the function L exists only for constructible tree protocols.

Definition 4.2. Assume a tree protocol of the form (1) in Definition 2.1 and a function $L: S_1 \cup \ldots \cup S_N \rightarrow S_1 \times \ldots \times S_N$.

A reception $<r_k , x_{ik}>$ is *constructible* iff conditions (3), (4), and (5) are satisfied, where t_i and t_i' are the departure and entry states of $-x_{ik}$, respectively.

The next definition states that a tree protocol is constructible iff it is obtainable in any number of steps:

(i) starting from the roots,

(ii) by adding a new transmission -x to any state t_i , or

(iii) by adding any constructible reception.

Definition 4.3. A tree protocol P of the form (1) is *constructible,* and

$L: S_1 \cup \ldots \cup S_N \rightarrow S_1 \times \ldots \times S_N$

is a *function associated* with P iff P and L are obtainable by the following inductive definition.

(i) P is constructible if $S_i = \{o_i\}$ for all i and M_{ij} is empty for all i, j.

The following function L is associated with P: $L(o_i) = \langle o_1, \ldots, o_N \rangle$ for all i.

(ii) Let P be a constructible tree protocol of the form (1), let $i \neq k$ be two fixed process indices, let t_i be a state in S_i, and let x and s be new symbols. Let P' be the tree protocol obtained from P by replacing S_i with $S_i \cup \{s\}$, M_{ik} with $M_{ik} \cup \{x\}$, and by extending succ to $succ(t_i, -x) = s$. Then P' is a constructible tree protocol.

If L is associated with P then the following extension L' is associated with P':

$L'_i(s) = s, \quad L'_j(s) = L_j(t_i)$ for all $j \neq i$.

(iii) Let P be a constructible tree protocol of the form (1), let L be a function associated with P, let $\langle r_k, x_{ik} \rangle$ be an unspecified constructible reception (with respect to L), and let s be a new symbol. Let P' be the tree protocol obtained from P by replacing S_k with $S_k \cup \{s\}$, and by extending succ to $succ(r_k, +x_{ik}) = s$. Then P' is a constructible tree protocol.

If L is associated with P then the following extension L' is associated with P':

$L'_k(s) = s, \quad L'_j(s) = \max(L_j(r_k), L_j(t'_i))$ for all $j \neq k$,

where t'_i is the entry state of $-x_{ik}$. [Note that $L'_j(s)$ is well defined because (4) is satisfied.]

Condition (i) of Definition 4.3 is the basis of the inductive definition. It defines the initial protocol, where each process has only one state (the root o_i) and there are no messages. The associated function L has the same value for all the roots o_i, namely, the N-tuple of roots.

Condition (ii) says that to any state t_i of a constructible tree protocol P we can append the transmission -x of a new message x and again obtain a constructible tree protocol. In order that the result of the addition is a tree again, we have to create a new state s for the entry state of -x. The function L' will have the same value on s as on t_i because by sending a message x we have learned nothing new about the other processes. For process i itself we know that after sending x it is in state s. Therefore, $L_i(s) = s$.

Condition (iii) describes the expansion of a constructible tree protocol by the addition of a reception $<r_k , +x_{ik}>$. As in (ii) we have to create a new state s as the entry state of $+x_{ik}$. In order that the new reception is executable, $+x_{ik}$ can be attached only where the conditions (3), (4), and (5) are satisfied. The extension of L to s expresses that after a reception new information about the other processes can be gained. When process k receives x_{ik} then the transmitting process i must have reached state t_i'. And for that to happen any process j must have reached $L_j(t_i')$. However, it is possible that r_k has more information about a process j than t_i' has; therefore we take the maximum of the two.

Definition 4.3 allows many ways of obtaining the same constructible tree protocol. It is conceivable that different ways might lead to different associated functions. The next Lemma 4.1 states that this is not so, and therefore from now on L will always be *the* function associated with a constructible tree protocol. Proofs for the following lemma and theorem are in Appendix B.

LEMMA 4.1. *If L and L' are two functions associated with a constructible tree protocol then $L = L'$.*

THEOREM 4.1. *A tree protocol is constructible iff each of its specified receptions is executable.*

COROLLARY 4.1. *A tree protocol is well-formed iff it is constructible and all constructible receptions are specified.*

Theorem 4.1 says that (3), (4), and (5) are necessary and sufficient conditions for receptions to be executable. Corollary 4.1 implies that well-formed tree protocols can be constructed by applying Definition 4.3, where step (iii) is used to specify all the receptions allowed by conditions (3), (4), and (5).

Such a construction can be made more efficiently than suggested by Definition 4.3. In the rest of this section we will outline an algorithm for finding all the receptions that must be specified as a result of a transmission. This is followed by an algorithm to find all the stable N-tuples. Appendix B establishes the correctness of the algorithms.

Assume a well-formed tree protocol that has been expanded by adding the transmission $succ(t_1 , -x) = t_1'$ (see Figure 7).
We will identify all the new receptions that must be specified in order that the resulting tree protocol is again well-formed. Part of the construction is also the calculation of the function L for newly introduced states. The algorithm is described for the case of the new message x being sent from process 1 to process 2; the general case is obtained by renumbering the processes.

The first step of the algorithm, called propagation along negative arcs, copies the positive subtree of t_1 below t_1'. That is, for every $s_1 = succ(t_1 , +Y)$ for some sequence Y, a new state $s_1' = succ(t_1' , +Y)$ is generated.
$L_1(s_1') = s_1'$ and $L_j(s_1') = L_j(s_1)$ for all $j \neq 1$.

The second step finds all the executable receptions of x in process 2 by testing conditions (3), (4), and (5). (This can be done relatively efficiently by taking advantage of the tree structure.)

The last step is called propagation along positive arcs. It is done for every state of process 2 bottom up; that is, it is applied to a state r_2 only after applying it to the whole subtree of r_2. If there are z, q, q′ so that $succ(r_2 , +z) = q$, $succ(r_2 , +x) = q'$, and

succ(q, +x) = u then we create a new state $u' = succ(q', +z)$ and copy the subtree of u below u'. (Note that the subtree of u contains only receptions, for the subtree was created in previous iterations of this step, which generates only receptions.) The copying is done as in the propagation along negative arcs: A new state $s_2' = succ(u', +W)$ is created for every $s_2 = succ(u, +W)$.

$L_2(s_2') = s_2'$ and $L_j(s_2') = L_j(s_2)$ for all $j \neq 2$.

At the same time as all the new receptions are being added we can also identify all the new stable N-tuples. Assume a list of all the stable N-tuples in the original protocol before the addition of -x. We will expand the list by all the new stable N-tuples in the new protocol.

During the second step (finding all the receptions of x) identify all the stable N-tuples $<s_1, q, s_3, ..., s_N>$, where

$s_1 = succ(t_1, +Y)$ for some sequence Y and $succ(q, +x) = u$.

Add $<s_1', u, s_3, ..., s_N>$ to the list of stable N-tuples, where $s_1' = succ(t_1', +Y)$.

During the third step (propagation along positive arcs) identify all the stable N-tuples $<s_1', s_2, s_3, ..., s_N>$, where

$s_1' = succ(t_1', +Y)$ for some sequence Y and

$s_2 = succ(u, +W)$ for some sequence W.

Add $<s_1', s_2', s_3, ..., s_N>$ to the list of stable N-tuples, where $s_2' = succ(u', +W)$.

4.2 TERMINATION. In this section, we will assume a solution to problem (2) for tree protocols and therefore will be dealing with well-formed tree protocols only. In contrast to the previous section, we will assume the existence of a general protocol, for which problem (2) is supposed to be solved. As explained at the beginning of Section 4, the solution is obtained by expanding a tree protocol looking for missing receptions and stable N-tuples. The purpose of this section is to determine when to stop this expansion. Appendix D contains proofs of validity, namely, that stopping the expansion of the trees will not cause any receptions or

stable N-tuples in the general protocol to be missed. It also contains proofs of termination, namely, for what classes of general protocols the method guarantees that the tree expansion be stopped [and hence solves problem (2)].

More specifically, this section assumes a given well-formed tree protocol and a given equivalence relation $\sim$ on states and messages of that tree protocol. This equivalence relation represents all the information we need about the general protocol: Two states of the tree protocol are equivalent if they represent the same state of the general protocol, and similarly for equivalent messages. Therefore, we will assume that $\sim$ relates states from the same process S_i and messages from the same set M_{ij}. Moreover, $\sim$ is a congruence relation with respect to succ:

If $s \sim s'$ and $x \sim x'$ then $succ(s, x) \sim succ(s', x')$ whenever both sides are defined.

In Figure 4, naming is used to indicate the equivalence; two states or messages are equivalent if they have the same name except for the number following the period. For example, READY.0, READY.1, READY.2 are equivalent (and no other states belong to this equivalence class).

This section gives a partial solution to the problem of deciding whether further expansion of the tree protocol can uncover new properties of the general graph -- a new reception or stable N-tuple. It operates by marking states; we will call such a marked state, as well as all states in its subtree, dead. The criteria for marking a state guarantee that the subtree of a dead state cannot contribute any new information to the given general graph. Thus, the expansion of the tree protocol terminates when new transmissions can be appended below dead states only.

For example, suppose that in Figure 4 our criteria allow states READY.1, READY.2, IDLE.1, and IDLE.2 to be marked dead (as is actually the case). Then we do not need to generate the transmissions REQ.1, REQ.2, ALARM.1, and ALARM.2. Thus, the tree

construction can stop after building three levels only, and state WAIT.2 as well as all the other states on the fourth level are not needed.

There are two criteria under which a state can be marked dead, yielding two types of dead states -- types 0 and 1. The criterion for type 0 is analogous to the termination mechanism of the perturbation method[25]. That method generates reachable global states; a newly generated global state can be ignored if it is equal to a previously generated global state. Our criterion for marking a state s dead of type 0 considers only that global state containing s which is reached in the least number of steps. These steps do not have to be carried out because this minimal global state is given by <L(s), C>, where C = Channels(L(s)). (See Definition 4.4.) The state s can be marked dead if this minimal global state is equivalent to the minimal global state for some previous state s'.

Definition 4.4. For an N-tuple S in a tree protocol we define *Channels*(S), provided there exists a reachable global state <S,C>. In that case Channels(S) = C.

Note that it is a proper definition independent of the particular global state <S,C>; if s_i and s_j are in S then the channel from i to j consists of all the messages on the branch to s_i that have been sent to process j, but not received on the branch to s_j. Thus, calculating Channels(S) is a purely syntactic process, which does not involve searching for a reachable global state <S,C>. The condition under which Channels(S) are defined can also be stated without reference to execution (Lemma B.7 in Appendix B). For the method, it is necessary to determine Channels only for N-tuples of the form L(s), and Lemma B.7 guarantees that Channels(L(s)) are always defined.

Definition 4.5. We will say that a state s' is a *precursor* of a state s, provided

(i) $s' < s$,

(ii) $L(s') \sim L(s)$, and

(iii) $Channels(L(s')) \sim Channels(L(s))$.

As an example we show why in Figure 4 READY.0 is a precursor of READY.1.

L(READY.0) = <READY.0, IDLE.0>,

Channels(L(READY.0)) = <ε, ε, ε, ε>,

L(READY.1) = <READY.1, IDLE.1>,

Channels(L(READY.1)) = <ε, ε, ε, ε>.

Hence all three conditions for being a precursor are satisfied.

Having a precursor is the condition for a state s to be marked dead of type 0. The criterion for a state s to be marked dead of type 1 is motivated by the following natural expectation: If all receptions of a message can be ignored (because they are below dead states) then the transmission can be ignored, too. This condition is not valid for protocols with more than two processes, but a stronger condition expressed in Definition 4.6 is valid.

According to this stronger condition the subtree of a state t_i can be ignored if t_i "can send" to dead states only. That is, if any message sent from t_i can be received at dead states only. The relation "can send" is obtained from Definition 4.2 of a constructible reception by substituting the condition (6) for (4). Condition (6) is implied by (4) and differs from (4) only for $j = i$, [(6) refers to t_j rather than t_i']. It does not require that the knowledge of state r_k be consistent with the transmission of any specific message. Thus, it allows the possibility that r_k will be consistent with any message sent from t_i. This could be a message only later introduced into the protocol.

Definition 4.6. A state t_i *can send* to a state r_k provided the three conditions (3), (5), and

$$L_j(t_i) \lesseqgtr L_j(r_k) \text{ for all } j \neq k. \tag{6}$$

Definition 4.7. A tree protocol is *well-marked* iff it is a well-formed tree protocol with two distinguished subsets of all the states -- states marked dead of type 0 and states marked dead of type 1 -- satisfying the three conditions below:

(i) If s and t are two marked states then it is not the case that $s \lesssim t$. [So every dead state s has a type; it is the type of the unique marked state on the branch to s.]

(ii) If s is marked dead of type 0 then s has a precursor.

(iii) If t is marked dead of type 1 and t can send to r then r must be dead of type 0.

For example, in the protocol in Figure 4 we can mark states READY.1 and IDLE.2 as dead of type 0 because they have precursors READY.0 and IDLE.0, respectively. This also makes states WAIT.2, REGISTER.1, SERVICE.2, and FAULT.2 dead of type 0. State IDLE.1 can send to state READY.1 and WAIT.2 only, both of which are dead of type 0, so IDLE.1 can be marked dead of type 1. Similarly, READY.2 can be marked dead of type 1.

Appendix D proves that this approach guarantees tree-growth termination for protocols with all channels bounded, even if we restrict ourselves to type-0 dead states. If we allow dead states of type 1 then we can terminate the growth of some protocols with unbounded channels; in particular, we prove that termination is guaranteed for protocols with only two processes and one channel unbounded.

For an implementation the following consideration must be taken into account. A state dead of type 0 can remain dead even when new transmissions are introduced into the protocol. This is not necessarily the case for a state t dead of type 1. A new transmission may cause new states to be created to which t can send. If one of such new states is not dead of type 0 then t must be "revived". Therefore, states dead of type 1 should be used only in a final check on a completed protocol.

5. Conclusions

The purpose of this paper was to investigate a model of communications protocols from the point of view of certain properties (executable receptions and stable N-tuples). As we documented in the Introduction, this model and the properties considered provide useful information for protocol designers. Namely, first, the class of errors prevented often have a crippling effect in an implementation, and second, experience has shown that designers are prone to making errors of this class.

After showing that the properties of interest are undecidable in general, we concentrated on one approach to establishing those properties for some classes of protocols. This approach has its goal in considering the execution of each process separately, as compared with considering executions of the global system of all the processes.

The whole problem was divided into two subproblems -- first, growing of trees in the search for all executable receptions and stable N-tuples, and second, determining when the tree growth can be stopped. We feel that the first problem was solved successfully. We found necessary and sufficient conditions characterizing all executable receptions. In contrast, terminating the tree growth is much more difficult. First of all, it is undecidable when we can stop growing the trees. Therefore, we have to expect only partial solutions. We described a procedure that is guaranteed to terminate the tree growth if all channel capacities are finite (without any restriction on the number of processes). Termination of tree growth is also guaranteed if there are only two processes and one channel is unbounded. This is as far as we can relax the need for bounding the channels and keep the problem solvable. Because of the important role played by channel capacity, Appendix C describes how to estimate it.

The limitations of the approach lie in the termination. Since a complete solution cannot be expected, the user must be able to specify a parameter limiting the search for a solution. Such a limiting parameter should be natural to the user, should be easy to guess for common

protocols, and should be easy to incorporate into a termination procedure. This area requires more research.

ACKNOWLEDGEMENTS. We are grateful to D.D. Cowan, R. Hauser, H. Rudin, M. Sherman and C.H. West for valuable discussions and careful reading of the manuscript.

REFERENCES

1. K. A. Bartlett, R. A. Scantelbury, and P. T. Wilkinson, A Note on Reliable Full-Duplex Transmission over Half-Duplex Links, *Communications of the ACM, Vol. 12,* No. 5, May 1969, pp. 260 - 261.

2. W.W. Bledsoe, P. Bruell and R.E. Shostak, A Prover for General Inequalities, Mathematics Department Memo ATP-40A, University of Texas at Austin, Feb. 1979.

3. G.V. Bochmann, Communication Protocols and Error Recovery Procedures, *ACM Operating Systems Review, Vol. 9,* No. 3, July 1975.

4. G. V. Bochmann, Finite State Description of Communications Protocols, Proceedings of the Computer Network Protocols Symposium, Liege, Belgium, Feb. 1978, pp. F3-1 - F3-11.

5. G. V. Bochmann and J. Gescei, A Unified Method for Specification and Verification of Protocols, Proc. IFIP 77, Toronto, Canada, Aug. 1977, pp. 229-234.

6. G. V. Bochmann and C. Sunshine, Use of Formal Methods in Communication Protocol Design, *IEEE Transactions on Communications, Vol. COM-28,* No. 4, April 1980, pp. 624-631.

7. D. Brand and W. H. Joyner, Jr., Verification of Protocols Using Symbolic Execution, *Computer Networks 2,* No. 4/5, Sept./Oct. 1978, pp. 351-360.

8. D. Brand and P. Zafiropulo, Synthesis of Protocols for an Unlimited Number of Processes, Proc. of Trends and Applications 1980: Computer Network Protocols, National Bureau of Standards, Gaithersburg, Maryland, USA, May 1980.

9. CCITT, Recommendation X21 (Revised), AP VI, No. 55-E, Geneve, Switzerland, 1976.

10. A. Danthine, editor, Proceedings of the Computer Network Protocols Symposium, Liege, Belgium, 1978. See also special issue on Computer Network Protocols, *Computer Networks,* Vol. 2, No. 4/5, Sept./Oct. 1978.

11. M. Davis, *Computability and Unsolvability,* McGraw-Hill 1958.

12. M. Devy and M. Diaz, Multilevel Specification and Validation of the Control in Communication Systems, *Distributed Computing Systems,* Oct. 1979, pp. 43-50.

13. M. G. Gouda and E. G. Manning, On the Modelling, Analysis and Design of Protocols -- A Special Class of Software Structures, Proc. Second Int. Conf. on Software Engineering, San Francisco, U.S.A., Oct. 1976, pp. 256-262.

14. J. Hajek, Automatically Verified Data Transfer Protocols, Proc. International Conf. on Computer Communications, Kyoto, Japan, Sept. 1978, pp. 749-756.

15. R.M. Karp and R.E. Miller, Parallel Program Schemata: A Mathematical Model for Parallel Computation, Proc. 8th Switching and Automata Theory Symposium, Oct. 1967, pp.55-61.

16. J.C. King, A Program verifier, Ph. D. thesis, Carnegie Mellon University, 1969.

17. P. M. Merlin, A Methodology for the Design and Implementation of Communication Protocols, *IEEE Trans. on Comm., Vol. COM-24,* No. 6, June 1976, pp. 614-621.

18. P. M. Merlin, Specification and Validation of Protocols, *IEEE Trans. Comm., Vol. COM-27,* No. 11, Nov. 1979, pp. 1671-1680.

19. J.L. Peterson, Petri Nets, *Computing Surveys, Vol. 9,* No. 3, Sept. 1977, pp. 223-251.

20. H. Rudin, C. H. West, and P. Zafiropulo, Automated Protocol Validation: One Chain of Development, Proceedings of the Computer Network Protocols Conference, Liege, Belgium, Feb. 1978, pp. F4-1 - F4-6.

21. G.D. Schultz, D.B. Rose, C.H. West, J.P. Gray, Executable Description and Validation of SNA, *IEEE Transactions on Communications, Vol. COM-28,* No. 4, April 1980, pp. 661-667.

22. N. V. Stenning, A Data Transfer Protocol, *Computer Networks 1,* No. 2, Sept. 1976, pp. 99-110.

23. C. A. Sunshine, Survey of Protocol Definition and Verification Techniques, Proceedings of the Computer Network Protocols Symposium, Liege, Belgium, Feb. 1978, pp. F1-1 - F1-4.

24. F. J. W. Symmons, Modelling and Analysis of Communication Protocols Using Petri Nets, Report No. 140, Telecomm. Systems Group, Dept. of EE, University of Essex, England, Sept. 1976.

25. C. H. West, General Technique for Communications Protocol Validation, *IBM Journal of Research and Development, Vol. 22,* No. 4, July 1978, pp. 393-404.

26) C.H. West and P. Zafiropulo, Automated Validation of a Communications Protocol: the CCITT X.21 Recommendation, *IBM Journal of Research and Development, Vol 22,* No.1, Jan. 1978, pp. 60-71.

27. M. Yoeli and Z. Barzilai, Behavioral Description of Communication Switching Systems Using Extended Petri Nets, *Digital Processes 3,* 1977, pp. 307-320.

28. P. Zafiropulo, Protocol Validation by Duologue-Matrix Analysis, *IEEE Transactions on Communications, Vol. COM-26,* No. 8, Aug. 1978, pp. 1187-1194.

29. P. Zafiropulo, C.H. West, H. Rudin, D.D. Cowan and D. Brand, Towards Analyzing and Synthesizing Protocols, *IEEE Transactions on Communications, Vol. COM-28,* No. 4, April 1980, pp. 651-661.

Appendix A -- Proof of unsolvability

The proof is based on showing how to simulate a Turing machine by a protocol. For that we will first state our conventions for Turing machines following [11].

A Turing machine has:

(i) A finite set Q of states with a distinguished initial state I, (we will refer to these states as Q-states to avoid any confusion with states in a protocol);

(ii) A finite set T of tape symbols, with one distinguished symbol #, called "blank";

(iii) A finite set of instructions of three possible forms: $qtt'q'$, $qtRq'$, $qtLq'$, where, q and q' are Q-states, t and t' are tape symbols. No two instructions are allowed to start with the same pair qt.

A configuration of a Turing machine is a finite string of the form:

$$t_{-n} \ldots t_{-1}\ q\ t_0\ t_1 \ldots t_m, \tag{A1}$$

where $n \geq 0$ and $m \geq 0$.

It means that the whole doubly infinite tape is blank (i.e. filled with the tape symbol #), except for the sequence of tape symbols $t_{-n} \ldots t_{-1}\ t_0\ t_1 \ldots t_m$. The head is scanning the symbol t_0 and the Turing machine is in Q-state q. Note that an arbitrary number of blanks (#) on either side of the representation (A1) does not change its meaning.

The interpretation of the instructions is the following. From any initial configuration the Turing machine determines a computation, a finite or infinite sequence of configurations. If (A1) is a configuration in an execution then the next configuration is:

$t_{-n} \ldots t_{-1}\ q'\ t'\ t_1 \ldots t_m$ if there is an instruction $qt_0t'q'$,

$t_{-n} \ldots t_{-1}\ t_0\ q'\ t_1 \ldots t_m$ if there is an instruction qt_0Rq',

$t_{-n} \ldots q'\ t_{-1}\ t_0\ t_1 \ldots t_m$ if there is an instruction qt_0Lq'.

If there is no instruction qt... then there is no next configuration, i.e., the Turing machine halts.

It is proved in [11] that there is a Turing machine Z° for which the "halting problem" is unsolvable. This means that there is no program that takes as input a sequence X of tape symbols and always returns the correct answer to the question whether Z° halts after the initial configuration IX. This is the configuration when Z° is in the initial state I, scanning the first symbol of X (if any), and the rest of the tape is blank.

In this Turing machine Z° there is at least one pair qt so that no instruction starts with qt because otherwise Z° would never terminate. We will construct a new Turing machine Z from Z°: We add a new Q-state H to Z° and add instructions of the form qtRH so as to ensure that H# is the only pair not starting any instruction. It should be clear that Z terminates on an initial configuration iff the original Z° does. (If Z° terminates then Z terminates after positioning its head to a blank.) Thus, the halting problem is also undecidable for Z.

Based on this Turing machine Z we will construct a protocol PZ that in a certain sense simulates Z. The protocol PZ has two processes, which use the channels between them to represent a Turing-machine configuration (see Figure A.1). The configuration is bent into a ring with a special message "|" separating the two ends. Besides using the channel, the processes also remember some symbols in their finite memory. Thus, for example, the global state of Figure A.1 represents the Turing-machine configuration

a b b d q e # d d e b #.

The ring constantly rotates in clockwise direction. Process 2 keeps the ring moving by sending back everything it receives. Process 1, apart from keeping the ring moving, also simulates the Turing machine. Each time a Q-state q is inside this process, one step of the Turing machine Z is simulated.

However, a message may arrive (and must be received) before the work associated with the previous message is done. If this happens, then the process will enter a special "sink" state, where it remains without transmitting anything. Consequently, not every execution of PZ simulates Z. (It would not be necessary to introduce the sink state if the model allowed transient states[25], internal states that cannot receive any messages. With such a facility we could write a protocol whose every execution simulates Z, which would make the proof more straightforward.)

Now we define PZ precisely. Assume that Q and T are disjoint and that neither contains the symbol "|". Let $M = Q \cup T \cup \{|\}$ [M is the set of messages]. Let $\underline{M}$ be a set with the same number of elements as M, but disjoint from M. We consider $\underline{M}$ as consisting of the same elements as M but renamed. This renaming is used only for the following definition of PZ because according to Definition 2.1 the two processes must send distinct messages. But later we will ignore this renaming, for no confusion can arise.

The protocol PZ is of the form (1), where

$N = 2$,

$S_1 = M \times M \cup M \times M \times M \cup \{sink\}$,

$S_2 = \underline{M} \cup \underline{M} \times \underline{M} \cup \{|\#\} \times \underline{M} \cup \{\underline{sink}\}$,

$o_1 = \#I$,

$o_2 = \underline{|}$,

$M_{12} = M$,

$M_{21} = \underline{M}$.

Before defining succ we explain the representation of states. In addition to state sink, process 1 has states of the form ab or of the form abc, where each a, b, and c is a Q-state, or a tape symbol, or the symbol "|". A state of the form ab represents that process 1 remembers the symbol a followed by b from a configuration (A1). In addition to state $\underline{sink}$, process 2 has states of the form $\underline{a}$ or $\underline{ab}$ or $\underline{|\#c}$, where each $\underline{a}$, $\underline{b}$, $\underline{c}$ is from Q, T, or $\{|\}$

renamed by the underlining. Similarly, the messages sent by process 2 are the same as those sent by process 1, except for the renaming. In the sequel we will ignore this renaming and will always state which is the process under discussion.

Definition of succ for process 2 (see Figure A.2):

For any a, b from M,

succ(a, +b) = ab.

For any a from Q ∪ T and any b from M,

succ(ab, -a) = b,

succ(|b, -#) = |#b, succ(|#b, -|) = #b.

For any s from, M×M ∪ {|#}×M ∪ {sink} and any c from M

succ(s, +c) = sink.

Process 2 just sends back whatever it receives from process 1. It also pads "|" with a blank on each side. Namely, when process 2 is in state | receiving any b, then it will first transmit the messages #, |, and # before entering state b.

All the states of process 2 of the form ab or |#b are supposed to be temporary before a transmission is made. Therefore, if a message c arrives in one of these temporary states, then the process enters the sink state, where it does not transmit anything, thus abandoning the simulation. Note that all the receptions for process 2 are specified (whether or not they are executable).

Definition of succ for process 1. (Figure A.3 shows a portion of process 1 resulting from a Turing machine with the instructions $q_1tt'q_1'$, q_2tRq_2', and q_3tLq_3'.)

For any a from M, any q from Q, and any t from T,

succ(aq, +t) = aq′t′ if Z has qtt′q′, (A2)

succ(aq, +t) = atq′ if Z has qtRq′, (A3)

succ(aq, +t) = q′at if Z has qtLq′, (A4)

succ(aH, +#) is undefined. (A5)

For any a, b, c from M such that b is not in Q or c is not in T (i.e., the above definition does not apply)

succ(ab, +c) = abc. (A6)

For any a, b, c from M

succ(abc, -a) = bc. (A7)

For s from M×M×M ∪ {sink} and any b from M

succ(s, +b) = sink. (A8)

Equations (A2), (A3), (A4) simulate a step of the Turing machine Z. Equations (A6) and (A7) keep the ring (representing a configuration of Z) moving. As in the definition for process 2, when a message is received in one of the temporary states of the form abc then the simulation ends in the sink state (A8). Note that in process 1 all receptions are specified, except for the receptions of the form <aH, #>.

Definition A.1. A global state of PZ is *normal* if it does not contain either of the two sink states.

Definition A.2. The *contour* of a normal global state $<<s_1 , s_2>, <c_{11} , c_{12} , c_{21} , c_{22}>>$ is defined as follows. (Informally, we take the ring of messages in the channels, including the messages remembered by the processes, break it at the message |, and straighten it.) If the concatenation $s_1c_{21}s_2c_{12}$ can be written as A|B in a unique way (i.e., it contains exactly one occurrence of "|") then the contour is the sequence BA. Otherwise the contour is the empty sequence.

Notation: In the sequel $|\!\!-_P$ indicates that $|\!\!-$ relates to a protocol P rather than to some other protocol.

LEMMA A.1. *Let X be a sequence of tape symbols and let* $G_X = <<\#I, |>, <\epsilon, X>>$. *Assume that* $G_X \ |\!\!-^*_{PZ} \ G$, *for some normal global state G. Then the contour of G is a configuration of the Turing machine Z during a computation started from IX.*

PROOF. (by induction on the number of steps to reach G).

The lemma is true if $G = G_X$, when the contour is #IX.

Assume the lemma for G and let $G \ |\!\!-_{PZ} \ G'$. If G′ is obtained by (A2), (A3), or (A4) then the contour of G′ can be obtained from the contour of G by the corresponding step of Z. If G′ is obtained by (A6) or (A7) then the contour of G′ is the same as the contour of G. If G′ is obtained by equation (A8) then G′ is not normal.

PROPOSITION A.1. *It is undecidable whether a given reception in a given protocol is executable.*

PROOF. Suppose the contrary and let EXECUTABLE(P,R) be a program that returns "yes" if the reception R is executable in the protocol P, and returns "no" otherwise. We will construct a program HALTS(X) that returns "yes" if Z halts when started from the configuration IX, and returns "no" otherwise.

HALTS(X) extends the protocol PZ into a protocol P by initializing the channel from process 2 to process 1 to contain X (see Figure A.4). Let $X = x_1 \ldots x_k$ $(k \geq 0)$.

$P = \langle\langle S_1 , S_2' \rangle, \langle o_1 , o_2' \rangle, \langle \phi, M_{12} , M_{21} , \phi \rangle, succ' \rangle$, where

$S_2' = S_2 \cup \{v_1 , \ldots, v_k\}$, where v_i are some new symbols,

$o_2' = v_1$ if $k > 0$, $o_2' = o_2 = |$ if $k = 0$,

$succ'(v_i , -x_i) = v_{i+1}$ for $1 \leq i < k$,

$succ'(v_k , -x_k) = |$, and $succ'$ is an extension of succ.

Then HALTS(X) calls EXECUTABLE(P, <aH, #>) for each a from M. If each of these calls returns "no" then HALTS(X) returns "no", otherwise it returns "yes".

I. Assume that EXECUTABLE(P, <aH, #>) returns "yes". We prove that if Z is started from the configuration IX then it will reach a configuration ...aH#..., i.e., it will halt. By Definition 3.1, P has at least one reachable global state of the form

$$\langle\langle aH, s_2 \rangle, \langle c_{11} , c_{12} , \#Y, c_{22} \rangle\rangle. \tag{A9}$$

Let G be a global state of the form (A9) that is reachable in the minimal number of steps of Definition 2.3.

First, we show that G is normal. Assume otherwise and consider a minimal derivation of G. Let G_i be the last normal global state in this derivation. Thus,

$G_0 = \langle\langle o_1 , o_2' \rangle, \langle \varepsilon, \varepsilon, \varepsilon, \varepsilon \rangle\rangle \mid\!\!-_P^* G_i \mid\!\!-_P G_{i+1} \mid\!\!-_P^* G_n = G.$

If G is not normal then necessarily s_2 = junk, which must also be in G_{i+1}. Therefore, by deleting all those steps of the derivation $G_i \mid\!\!-_P^* G$ that advance process 2, we get in fewer steps $G_i \mid\!\!-_P G'$. The global state G' is of the form (A9), thus contradicting the minimality of G. Therefore, G must be normal.

Case I.1: $s_2 = v_i$ for some i.

Then in the derivation $G_0 \mid\!\!-_P^* G$ process 2 has sent only $x_1 , \ldots, x_{i-1}$, which do not include any Q-state. Therefore, process 1 could have made steps according to equations (A3) and

(A7) only. (If any other equation were used then in G process 1 could not be in state aH.) Equation (A3) can be simulated by Z with a move to the right and equation (A7) can be simulated by Z with no action. Therefore, when started in IX, Z will reach a configuration of the form ...aH#x_i...x_k#.

Case I.2: $s_2 \neq v_i$ for any i.

We will show that for G_X = <<#I, |>, <ε, ε, X, ε>>

$$G_0 \vdash_P^* G_X \vdash_P^* G. \tag{A10}$$

so that we will be able to use Lemma A.1. Out of all the derivations $G_0 \vdash_P^* G$ consider one with the maximal number of steps of process 2 before the first step of process 1. In this derivation, process 1 must stay in state #I at least until process 2 reaches state |; because any sequence of steps performed by process 1 can be performed after process 2 reaches | (which must happen before G is derived). This establishes (A10).

By the definition of P, $G_X \vdash_P^* G$ implies $G_X \vdash_{PZ}^* G$. By Lemma A.1 the contour of G, which is of the form ...H#..., will be reached by Z when started from IX. So Z will halt.

II. Now assume that the Turing machine Z halts when started from IX and we will show that P has a reachable global state of the form (A9); hence, EXECUTABLE(P, <aH, #>) returns "yes". A global state (A9) can be reached in two stages. First, process 2 transmits the sequence X; this results in a global state of the form G_X in Lemma A.1. Then PZ (and hence P) can simulate the steps of Z, resulting in a global state of the form (A9).

Discussion

The same argument can be used to show the unsolvability of other problems in other settings. We will discuss some of them.

It was essential for the proof of Proposition A.1 for both channels to be unbounded. It is impossible to simulate a Turing machine with a protocol that has only two processes and keeps

one channel bounded; the main body of the paper states and Appendix D proves that problem (2) is solvable for protocols with two processes and one channel bounded.

In contrast, we can construct a protocol simulating a Turing machine with only one channel unbounded, provided we can use more than two processes. Processes 1 and 2 can perform the same function as in PZ, but they will make sure that the channel from process 2 to process 1 contains at most one message at any one time. Process 2 will not send a message until process 1 acknowledges the previous message; the acknowledgement is sent via process 3. The same reasoning shows unsolvability for two processes and only one channel unbounded, provided we allow two parallel channels from process 1 to process 2. One would be used to keep the tape of the Turing machine and the other would allow process 1 to acknowledge each message from process 2. As above, a sink state would be necessary because not every computation of the protocol simulates a computation of the Turing machine.

The other part of problem (2), namely, identifying all the stable N-tuples, is also unsolvable. To see this, extend PZ by defining succ(aH, +#) = u for all a and some new symbol u. When process 1 reaches u let it send a special message to process 2, that makes process 2 enter a state w. Both processes stay in states u and w without sending anything. Hence <u, w> is a stable pair, provided it is reachable. But it is reachable only if Z halts.

Also the problem of whether a protocol has bounded channels is unsolvable. If it were solvable then we could determine whether Z halts when started from IX as follows. Ask whether the protocol P constructed by HALTS(X) has all channels bounded. If not, then Z cannot halt. If yes, than we can analyze P (for example using the method described in this paper) to determine whether any reception <aH, #> is executable. And this gives the answer to whether Z halts.

Appendix B -- Proofs for Section 4.1

LEMMA B.1. *In a tree protocol if* $s \leq r$ *and* $s' \leq r$ *then* $s \lesseqgtr s'$.

PROOF. Let X and X′ be such that r = succ(s, X) and r = succ(s′, X′). All three states s, s′, and r must be in the same process; let o be the initial state of that process.

Let s = succ(o, Y) and s′ = succ(o, Y′).

Then r = succ(o,YX) and r = succ(o, Y′X′).

By Definition 4.1 YX = Y′X′. Depending on whether Y or Y′ is longer $s' \leq s$ or $s \leq s'$.

Therefore, $s \lesseqgtr s'$.

In the next Definition B.1 we define an ordering <<. It will have the property that if t << r then state t must be reached before an execution can reach state r. Therefore, we will be able to base induction on this ordering.

Definition B.1. For two states t and r in a tree protocol, t << r iff either t < r or there exist states t°, r°, and a message x so that r = succ(r°, +x) and t = succ(t°, -x). The relation $<<^+$ is the transitive closure of <<.

LEMMA B.2. *In a constructible tree protocol with an associated function L,*

$t \leq r$ *implies* $L(t) \leq L(r)$.

PROOF. If $t \leq r$ then r = succ(t, X) for some message sequence X. It is enough to prove the lemma for X of length zero and one. For longer X the lemma follows by induction and by transitivity of $\leq$.

If X is of length zero then t = r and L(t) = L(r). If X is of length one (i.e., X = x) then we have two cases depending on whether <t, x> is a transmission or a reception.

If r = succ(t, -x) then by case (ii) of Definition 4.3 (where i is the index of the process containing t and r),

$L_i(r) = r > t = L_i(t)$

$L_j(r) = L_j(t)$ for $j \neq i$.

Hence $L(t) \leq L(r)$.

If $r = succ(t, +x)$ then by case (iii) of Definition 4.3 (where k is the index of the process containing t and r, and t_i' is the entry state of -x),

$L_k(r) = r > t = L_k(t)$,

$L_j(r) = \max(L_j(t), L_j(t_i')) \geq L_j(t)$ for $j \neq k$.

Hence $L(t) \leq L(r)$.

LEMMA B.3. *In a constructible protocol with an associated function L*

$t_i <<^+ r_k$ *implies* $L(t_i) \leq L(r_k)$. *Moreover,* $L_k(t_i) < r_k$.

PROOF. We will prove the lemma by induction on the length of the chain $t_i << \ldots << r_k$.

Basis: $t_i << r_k$.

If $t_i < r_k$ then the claim follows by Lemma B.2. Therefore, we can assume the other case of Definition B.1, namely, $r_k = succ(r_k^\circ, +x)$ and $t_i = succ(t_i^\circ, -x)$.

The result follows from Definition 4.3:

$L_k(r_k) = r_k > r_k^\circ \geq L_k(t_i)$ -- the last inequality by (5).

$L_j(r_k) = \max(L_j(r_k^\circ), L_j(t_i)) \geq L_j(t_i)$ for $j \neq k$.

Induction step: $t_i <<^+ s_j << r_k$, assume $L(t_i) \leq L(s_j)$.

The argument in the basis step shows $L(s_j) \leq L(r_k)$. Therefore, by transitivity of $\leq$, $L(t_i) \leq L(r_k)$. The argument in the basis step also shows $L_k(s_j) < r_k$. Thus,

$L_k(t_i) \leq L_k(s_j) < r_k$.

LEMMA B.4. *In a constructible tree protocol* $<<^+$ *is a well founded ordering.*

PROOF. If there existed an infinite descending chain then some tree k would have to contain an infinite number of members from the chain. This is impossible, for by Lemma B.3, if $t_k <<^+ r_k$ then $t_k = L_k(t_k) < r_k$,
where L is any function associated with the protocol.

PROOF of Lemma 4.1. The proof is by induction on the well founded ordering $<<^+$. Namely, we will show that $L'(r_k) = L(r_k)$ follows from assuming $L'(t) = L(t)$ for all $t <<^+ r_k$. The state r_k could be either a root, or the entry state of a transmission or the entry state of a reception, which gives us three cases corresponding to Definition 4.3:

(i) If r_k is a root then both $L'(r_k)$ and $L(r_k)$ are the N-tuple of roots.

(ii) If $r_k = succ(r_k^\circ , -x)$ then we can apply induction hypothesis to r_k° and get for all $j \neq k$

$L'_j(r_k) = L'_j(r_k^\circ) = L_j(r_k^\circ) = L_j(r_k)$.

(iii) If $r_k = succ(r_k^\circ , +x)$ let t' be the entry state of the transmission -x. We can apply induction hypothesis to both r_k° and t':

for all $j \neq k$

$$\begin{aligned} L'_j(r_k) &= \max(L'_j(r_k^\circ), L'_j(t')) \\ &= \max(L_j(r_k^\circ), L_j(t')) \\ &= L_j(r_k). \end{aligned}$$

In cases (ii) and (iii) for $j = k$, $L'_k(r_k) = r_k = L_k(r_k)$.

LEMMA B.5. *In a constructible tree protocol $L_i(L_j(s)) \leq L_i(s)$ for all s, i, j.*

PROOF. If s is in process j or if $i = j$ then the result is immediate. So assume otherwise and the proof will be by induction on the ordering $<<^+$. Assume the lemma true for all states $<<^+$ s.

Case 1: s is a root. Then both sides of the inequality evaluate to the root o_i.

Case 2: $s = succ(s^\circ, -x)$ for some s° and x.

$L_i(L_j(s)) = L_i(L_j(s^\circ)) \leq L_i(s^\circ) \leq L_i(s)$,

where the first inequality follows by induction hypothesis and the second by Lemma B.2.

Case 3: $s = succ(s^\circ, +x)$ for some s° and x.

Let t and t' be the departure and entry states of -x. Then

$$L_j(s) = \max(L_j(s^\circ), L_j(t')). \tag{B1}$$

Case 3.1: $L_j(s^\circ) \geq L_j(t')$.

Then $L_i(L_j(s)) = L_i(L_j(s^\circ)) \leq L_i(s^\circ) \leq L_i(s)$.

Case 3.2: $L_j(s^\circ) < L_j(t')$. (B2)

Case 3.2.1: s is in S_i.

$L_i(L_j(s)) = L_i(L_j(t'))$	by (B1) and (B2)
$\leq L_i(t')$	by induction hypothesis
$= L_i(t)$	by Definition 4.3
$\leq s^\circ$	by (5)
$< s = L_i(s)$.	

Case 3.2.2: s is not in S_i.

$L_i(L_j(s)) = L_i(L_j(t'))$	by (B1) and (B2)
$\leq L_i(t')$	by induction hypothesis
$\leq \max(L_i(s^\circ), L_i(t'))$	max exists by (4)
$= L_i(s)$	by Definition 4.3.

LEMMA B.6. *Let $<S,C>$ be a reachable global state in a tree protocol (1). Let $i \neq k$ and let X be the sequence consisting of the symbol * followed by all the messages from M_{ik} appearing on the branch to s_i. Let $From_i(s_k) = w$. Then for some Y, $X = Ywc_{ik}$.*

PROOF. The proof is by induction on the number of steps to reach <S,C>. If this number is zero then the lemma is satisfied with Y empty.

Assume the lemma for some <S,C> and let <S,C> |— <S′,C′>. Let X′ consist of * followed by all the messages from M_{ik} appearing on the branch to s_i'. <S′,C′> satisfies the lemma if $X' = X$, $c_{ik}' = c_{ik}$, and $From_i(s_k') = From_i(s_k)$.

So we will check only those two ways of obtaining <S′,C′> from <S,C> that do not preserve one of the three equalities. And for both we have to show the existence of Y′ to satisfy the lemma for <S′,C′>.

Case 1: $s_i' = succ(s_i , -z_{ik})$.

Then $X' = Xz_{ik}$, $c_{ik}' = c_{ik}z_{ik}$, and $From_i(s_k') = From_i(s_k)$. The lemma is satisfied with $Y' = Y$.

Case 2: $s_k' = succ(s_k , +z_{ik})$.

Then $X' = X$, $c_{ik} = z_{ik}c_{ik}'$, and $From_i(s_k') = z_{ik}$.

$X' = X = Ywc_{ik} = Ywz_{ik}c_{ik}'$.

Thus, the lemma is satisfied with $Y' = Yw$.

Definition B.2. In a tree protocol we say that an N-tuple S is *reachable* iff there exists C so that the global state <S,C> is reachable. Note that if C exists then it is unique and C = Channels(S) (see Definition 4.4). For two reachable N-tuples S and S′,

S |— S′ iff <S,Channels(S)> |— <S′, Channels(S′)>.

Definition B.3: For two states s_i and s_k in a tree protocol *Channel*(s_i , s_k) is defined iff there exists a reachable global state <S,C> containing both s_i and s_k. In that case $Channel(s_i , s_k) = c_{ik}$.

Note that $Channel(s_i , s_k)$ is well defined because, as for Definition 4.4, $Channel(s_i , s_k)$ is determined by the branches to s_i and s_k.

LEMMA B.7. *An N-tuple S in a constructible tree protocol is reachable iff*

$L_i(s_j) \leq s_i$ *for all i, j.* *(B3)*

PROOF.

I. Assume that S is reachable. We prove (B3) by induction on the number steps to reach S.

I.1. If S consists of roots only then (B3) is satisfied with equalities.

I.2. Assume that S satisfies (B3) and that $S \mid\!\!- S'$ using case (i) of Definition 2.3 (transmission). Without loss of generality we can assume that $s'_1 = \text{succ}(s_1 , -x_{12})$.
It is enough to check (B3) for $i = 1$ and for $j = 1$, in both cases $i \neq j$:

$L_1(s'_j) = L_1(s_j) \leq s_1 < s'_1$.

$L_i(s'_1) = L_i(s_1) \leq s_i = s'_i$.

I.3. Assume that S satisfies (B3) and that $S \mid\!\!- S'$ using case (ii) of Definition 2.3 (reception). Again without loss of generality we can assume $s'_2 = \text{succ}(s_2 , +x_{12})$ and check (B3) for $i = 2$ and for $j = 2$, in both cases $i \neq j$. Let t_1 and t'_1 be the departure and entry states of $-x_{12}$, respectively.

$L_2(s'_j) = L_2(s_j) \leq s_2 < s'_2$.

$$\begin{aligned} L_i(s'_2) &= \max(L_i(s_2), L_i(t'_1)) \\ &\leq \max(L_i(s_2), L_i(s_1)) \\ &\leq \max(s_i , s_i) = s_i = s'_i , \end{aligned}$$

where the first inequality follows from Lemma B.2 using $t'_1 \leq s_1$ (x_{12} must be transmitted before it can be received), and the second inequality follows by induction hypothesis.

II. Assume that S satisfies (B3). We will show that S is reachable by induction on the total distance of S from the roots. Assume that whenever an N-tuple S° satisfies (B3) and the

sum of the lengths of the branches for all the states in S° is smaller than for S, then S° is reachable.

If S consists of all the roots then we are done, so assume otherwise. Consider any fixed sequence of the steps in Definition 4.3 used to construct the protocol. At each step exactly one state is added to the protocol; without loss of generality assume that s_1 was added later than any other state in S. Therefore, for any $i \neq 1$, $L_1(s_i) \neq s_1$, and in combination with (B3):

$$L_1(s_i) < s_1 \text{ for all } i \neq 1. \tag{B4}$$

Let $s_1 = \text{succ}(s_1^\circ , x)$. By Lemma B.2

$$L_i(s_1^\circ) \leq L_i(s_1) \text{ for all i.} \tag{B5}$$

The N-tuple $S^\circ = \langle s_1^\circ , s_2 , \ldots, s_N \rangle$ satisfies (B3) by (B4) and (B5). By induction hypothesis S° is reachable. If $s_1 = \text{succ}(s_1^\circ , -x)$ then $S^\circ \vdash\!\!- S$ by part (i) of Definition 2.3 and S is also reachable. So consider the other case -- that s_1 is the entry state of a reception, and without loss of generality we can assume that $s_1 = \text{succ}(s_1^\circ , +x_{21})$. Thus, to show that S is reachable it is sufficient to show that $\text{Channel}(s_2, s_1^\circ)$ is of the form $x_{21}Z$ (i.e., x_{21} is ready to be received).

Let t_2 and t_2' be the departure and entry states of the transmission $-x_{21}$, respectively. We must have $t_2' \leq s_2$ because

$t_2' = L_2(t_2') \leq \max(L_2(s_1^\circ), L_2(t_2')) = L_2(s_1) \leq s_2$.

(That means that x_{21} has been transmitted.)

We apply Lemma B.6 to the N-tuple S° with $i = 2$ and $k = 1$. Since $\text{succ}(s_1^\circ , +x_{21})$ is defined in a constructible tree protocol, $\text{To}_1(t_2) = \text{From}_2(s_1^\circ) = w$ (for some w that is a message or *). Let X be defined as in Lemma B.6; then in X, x_{21} follows w, i.e., $X = Ywx_{21}Z$ for some Y and Z. By Lemma B.6 $\text{Channel}(s_2 , s_1^\circ) = x_{21}Z$. Therefore, S is reachable.

LEMMA B.8. *Let $<R,C>$ be a reachable global state in a constructible tree protocol. If c_{ik} is of the form $x_{ik}Z$ for a message x_{ik} and a sequence Z then the reception $<r_k , x_{ik}>$ is constructible.*

PROOF. Let t_i and t_i' be the departure and entry states of $-x_{ik}$, respectively. We prove (3), (4), (5).

PROOF OF (3). Let X consist of * followed by all the messages from M_{ik} appearing on the branch to r_i. By Lemma B.6 there exists Y so that

$X = Ywc_{ik} = Ywx_{ik}Z$, where $w = From_i(r_k)$. By definition of the function To,

$To_k(t_i) = w = From_i(r_k)$.

PROOF OF (4). We show that for all j, $L_j(t_i') \leq r_j$ and $L_j(r_k) \leq r_j$. By Lemma B.1 this implies that $L_j(t_i') \lesseqgtr L_j(r_k)$.

$L_j(t_i') \leq L_j(r_i) \leq r_j$,

where the first inequality is by Lemma B.2 and the second by Lemma B.7.

$L_j(r_k) \leq r_j$ by Lemma B.7.

PROOF OF (5). $L_k(t_i) \leq L_k(r_i) \leq r_k$,

where the first inequality follows from Lemma B.2 ($t_i \leq r_i$ because x_{ik} had to be transmitted before $<R,C>$ was reached) and the second inequality follows from Lemma B.7.

Definition B.4: For two protocols P and P′, $P \subseteq P'$ iff

$P = <<S_i>_{i=1}^N, <o_i>_{i=1}^N , <M_{ij}>_{i,j=1}^N , succ>$,

$P' = <<S_i'>_{i=1}^N , <o_i>_{i=1}^N , <M_{ij}'>_{i,j=1}^N , succ'>$,

with $S_i \subseteq S_i'$ for all i, $M_{ij} \subseteq M_{ij}'$ for all i, j, and $succ \subseteq succ'$.

PROOF OF THEOREM 4.1.

I. Assume a tree protocol P with all specified receptions executable. We will show that P is constructible.

Let G_0 , ..., G_n be all the reachable global states of P ordered in such a way that whenever $0 < m \leq n$ then there is $i < m$ so that $G_i \mid\!\!- G_m$.
Such an ordering can be accomplished by generating all the reachable global states starting from $G_0 = \langle\langle o_i\rangle_{i=1}^N , \langle\varepsilon\rangle_{i,j=1}^N\rangle$.

We will construct a sequence $P_0 \subseteq \ldots \subseteq P_n$ of constructible tree protocols such that every $P_i \subseteq P$, and whenever $0 \leq i \leq m \leq n$ then G_i is a reachable global state of P_m. The direction (I) of the theorem will be established by showing $P_n = P$.

P_0 is the protocol with no messages and only roots as states. Clearly G_0 is reachable in P_0. Assume that we have constructed a sequence of constructible tree protocols $P_0 \subseteq \ldots \subseteq P_m \subseteq P$ $(0 \leq m \leq n)$ so that G_i is reachable in P_m for all $i \leq m$.

Case 1: $m < n$.

Let i be such that $G_i \mid\!\!- G_{m+1}$. We will write $G_i = \langle S,C\rangle$, $G_{m+1} = \langle S',C'\rangle$.

Case 1.1: $\langle S,C\rangle \mid\!\!- \langle S',C'\rangle$ according to part (i) of Definition 2.3 (transmission). Without loss of generality we can assume $s_1' = \mathrm{succ}(s_1 , -x)$. If state s_1' exists in P_m then G_{m+1} is also a reachable global state of P_m and we can set $P_{m+1} = P_m$. Otherwise we obtain P_{m+1} by adding the transmission of x to P_m according to step (ii) of Definition 4.3.

Case 1.2: $\langle S,C\rangle \mid\!\!- \langle S',C'\rangle$ according to part (ii) of Definition 2.3 (reception). Without loss of generality we can assume $s_2' = \mathrm{succ}(s_2 , +x_{12})$. If state s_2' exists in P_m then G_{m+1} is also a reachable global state of P_m and we can set $P_{m+1} = P_m$. Otherwise we obtain P_{m+1} by specifying the reception $\langle s_2 , +x_{12}\rangle$ according to step (iii) of Definition 4.3. We can do that because the reception is constructible by Lemma B.8.

Case 2: $m = n$.

We show that $P_n = P$. For that it is enough to show $P \subseteq P_n$. And for that it is enough to show that all the branches of P exist in P_n. We will prove this by induction on the length of a branch. It is based on the fact that all global states reachable in P are also reachable in P_n.

All the branches of length 0 are in P_n because they are in $P_0 \subseteq P_n$. Assume that the branch to a state s of P is also in P_n and we will show that $s' = succ(s,x)$ is also in P_n. By construction of P_n state s must be in a reachable global state G.

If $s' = succ(s, -x)$ then the global state G' obtained from G by transmitting -x is reachable in P and hence in P_n. Therefore, s' is also in P_n.

If $s' = succ(s,+x)$ then s' must be in a global state G' reachable in P (because P has all specified receptions executable). Hence G' is also reachable in P_n and s' is in P_n.

II. Assume a constructible tree protocol. We will show that it has all specified receptions executable by showing that for every state s there exists a reachable global state containing s. This global state is (L(s), Channels(L(s)). It is reachable by Lemma B.7; to apply it we have to show (B3): $L_i(L_j(s)) \leq L_i(s)$, which is true by Lemma B.5. This proves the theorem.

The next Propositions B.1 and B.2 are justifications for the algorithm constructing well-formed tree protocols as described at the end of Section 4.1. Proposition B.3 is a justification for the algorithm detecting stable N-tuples. All three propositions follow Figure 7.

PROPOSITION B.1. *Let* $P \subseteq P'$, *where*

$P = \langle\langle S_i\rangle_{i=1}^{N}, \langle o_i\rangle_{i=1}^{N}, \langle M_{ij}\rangle_{i,j=1}^{N}, succ\rangle$ *and*

$P' = \langle\langle S'_i\rangle_{i=1}^{N}, \langle o_i\rangle_{i=1}^{N}, \langle M'_{ij}\rangle_{i,j=1}^{N}, succ'\rangle$

are well-formed tree protocols. Assume that $M'_{ij} = M_{ij}$ *for all i,j, except* $M'_{12} = M_{12} \cup \{x\}$. *Let* $t'_1 = succ'(t_1, -x)$, *where* t_1 *is in* S_1. *We make three claims:*

(i) If P contains $s_1 = succ(t_1, +Y)$ *for some sequence Y then P' contains* $s'_1 = succ'(t'_1, +Y)$ *and* $L_j(s'_1) = L_j(s_1)$ *for all* $j \neq 1$.

(ii) If $\langle r_2, +x\rangle$ *is a constructible reception in P' then P' contains* $s'_2 = succ'(r_2, +x)$.

(iii) If P' contains $s_2 = succ'(r_2, +z\ +x\ +W)$ *for some* r_2 *in P, some message z and some sequence W and if* $succ'(r_2, +x)$ *is defined, then P' also contains* $s'_2 = succ'(r_2, +x\ +z\ +W)$. *Moreover,* $L_j(s'_2) = L_j(s_2)$ *for all* $j \neq 2$.

PROOF.

(i) Since P is well-formed, there exists a reachable global state G containing s_1. Let G° be the global state obtained by the same execution steps as G, except process 1 stops when it reaches t_1. All the other steps can be performed as for G; hence the channels in G° contain the messages Y ready to be received. Starting from G° we can first transmit -x followed by receiving Y. Since P' is well-formed it must contain the resulting state s'_1. That $L_j(s'_1) = L_j(s_1)$ for all $j \neq 1$ follows by induction on the length of Y.

(ii) Follows directly from Corollary 4.1.

(iii) As in (i) we consider a global state G containing s_2, and obtain G° differing from G in that G° contains r_2 rather than s_2. All the messages z, x, W are in the channels of G° ready to be received. We can then receive them so as to arrive at s'_2, provided that we can

receive +x before +z. And this is possible because z is not from M_{12}. [For by (3) $From_1(r_2)$ and $From_1(q)$ are both equal to $To_2(t_1)$.]

We will show that for $j \neq 2$, $L_j(u') = L_j(u)$. The same claim for any s_2' and s_2 follows by induction on the length of W. Let t_i' be the entry state of -z.

$$\begin{aligned} L_j(u') &= \max(L_j(q'), L_j(t_i')) \\ &= \max(\max(L_j(r_2), L_j(t_1')), L_j(t_i')) \\ &= \max(L_j(t_1'), \max(L_j(r_2), L_j(t_i'))) \\ &= \max(L_j(t_1'), L_j(q)) \\ &= L_j(u). \end{aligned}$$

PROPOSITION B.2. *Let* $P \subseteq P'$, *where*

$P = <<S_i>_{i=1}^N, <o_i>_{i=1}^N, <M_{ij}>_{i,j=1}^N, succ>$ *and*

$P' = <<S_i'>_{i=1}^N, <o_i>_{i=1}^N, <M_{ij}'>_{i,j=1}^N, succ'>$

are well-formed tree protocols. Assume that $M_{ij}' = M_{ij}$ *for all i,j, except* $M_{12}' = M_{12} \cup \{x\}$.

Let $t_1' = succ'(t_1, -x)$, *where* t_1 *is in* S_1.

Let s' *be a state in* P', *but not in P. Then one of the following three cases is true.*

(i) $s' = succ'(t_1', +Y)$ *for some sequence Y and* $s_1 = succ(t_1, +Y)$ *is defined.*

(ii) $s' = succ'(r_2, +x)$ *for some* r_2 *in* S_2.

(iii) $s' = succ'(r_2, +x\ +z\ +W)$ *for some message z and sequence W, and* $s_2 = succ'(r_2, +z\ +x\ +W)$ *is defined.*

PROOF. Since s' is in the well-formed protocol P' there is a reachable (in P') global state G' containing s'. Let

$$G_0 \mid\!\!-\!\!* G_m \mid\!\!- G_{m+1} \mid\!\!-\!\!* G_n = G' \qquad \text{(B6)}$$

be a sequence of global states leading to G' according to Definition 2.3, where G_{m+1} is the first global state that is reachable in P' but not in P. There are many such sequences of the

form (B6); we assume that (B6) contains the minimal number of global states that are reachable in P' but not in P.

First note that G_{m+1} must be obtained from G_m by transmitting -x. Because with any other step G_{m+1} would be reachable in P. Secondly, for $i > m+1$, G_i is obtained from G_{m+1} by receiving messages in process 1 or 2 only. Because any other step can be performed in P contradicting the minimality of (B6).

Therefore, s' must be in S'_1 or S'_2. First suppose that s' is in S'_1 and we will prove claim (i). Since G_{m+1} is obtained by transmitting -x, $s' \geq t'_1$. Moreover, there can be no transmissions on the path from t'_1 to s' because x is the only message in P' that is not in P. Thus, there is a message sequence Y so that $s' = succ'(t'_1 , +Y)$. To show the existence of $s_1 = succ(t_1 , +Y)$ note that no transmission is used in obtaining G_{m+2} , ..., G_n. Therefore, all the messages Y must have been in the channels in G_m. Thus, by receiving them, rather than transmitting x, $G_m \mid\!\!-\!\!* G$, where G contains the sought state s_1.

Now suppose that s' is in S'_2. Let r_2 be that state of S_2 contained by G_m. The first step after G_m involving process 2 must be a reception of x. (Because any other step can be performed in P.) Therefore, $s' \geq succ'(r_2 , +x)$. And as in the proof of (i) there must be a message sequence Z so that $s' = succ'(r_2 , +x +Z)$. This sequence Z cannot contain any messages from process 1, for there are no transmissions after -x.

If Z is empty then claim (ii) is satisfied. If Z is not empty, i.e., $Z = zW$ for some message z and sequence W, then we will prove claim (iii). By the same argument as in the proof of (i), the channels of G_m contain all the messages in zW. Therefore, G_m can first receive +z, then transmit -x, then receive +x, and lastly receive +W. This gives us the existence of $s_2 = succ'(r_2 , +z +x +W)$.

PROPOSITION B.3. *Let* $P \subseteq P'$, *where*

$P = \langle\langle S_i\rangle_{i=1}^{N}, \langle o_i\rangle_{i=1}^{N}, \langle M_{ij}\rangle_{i,j=1}^{N}, succ\rangle$ *and*

$P' = \langle\langle S'_i\rangle_{i=1}^{N}, \langle o_i\rangle_{i=1}^{N}, \langle M'_{ij}\rangle_{i,j=1}^{N}, succ'\rangle$

are well-formed tree protocols. Assume that $M'_{ij} = M_{ij}$ *for all i,j, except* $M'_{12} = M_{12} \cup \{x\}$.

Let $t'_1 = succ'(t_1, -x)$, *where* t_1 *is in* S_1.

An N-tuple $\langle s'_1, s'_2, s_3, \ldots, s_N\rangle$ *is stable in* P' *but not in P iff*

$s'_1 = succ'(t'_1, +Y)$ *for some sequence Y, and one of the two following conditions is satisfied:*

(a) $s'_2 = succ'(q, +x)$ *for some q in* S_2, *and*

$\langle s_1, q, s_3, \ldots, s_N\rangle$ *is a stable N-tuple of P, where* $s_1 = succ(t_1, +Y)$.

(b) $s'_2 = succ'(r_2, +x\ +z\ +W)$ *for some* r_2 *in* S_2, *a message z and a sequence W, and* $\langle s'_1, s_2, s_3, \ldots, s_N\rangle$ *is also a stable N-tuple of* P', *where* $s_2 = succ'(r_2, +z\ +x\ +W)$.

PROOF.

I. If $s'_1 = succ'(t'_1, +Y)$ and one of the conditions (a), (b) is satisfied then $\langle s'_1, s'_2, s_3, \ldots, s_N\rangle$ is reachable by the same argument as in the proof of Proposition B.1. That the channels are empty follows by considering the relevant branches. Therefore, the N-tuple is stable in P'; it is not stable in P because s'_1 and s'_2 are not in P.

II. Let $\langle s'_1, s'_2, s_3, \ldots, s_N\rangle$ be a stable N-tuple of P' but not in P. By Proposition B.2, s'_1 must be of the form given by case (i) of that proposition, or s'_2 must be of the form (ii) or (iii). In case (i), -x has been transmitted and the requirement that all channels be empty implies that s'_2 must have +x on its branch. Conversely, if s'_2 has +x on its branch then s'_1 must have -x on its branch in order that the N-tuple be reachable. Therefore, in both cases $s'_1 = succ'(t'_1, +Y)$ for some sequence Y.

By the above argument, s'_2 is of the form (ii) or (iii) given in Proposition B.2. These cases correspond to cases (a) and (b) of this proposition, respectively. The reachability of the

N-tuples claimed in (a) and (b) follows by the same argument as in Proposition B.2; the fact that their channels are empty can be seen by considering the relevant branches.

Appendix C -- Test for bounded channels

Since the number of messages that can be in transit impacts the feasibility of solving problem (2), it is important to provide some estimate of this number. We cannot expect a necessary and sufficient condition, for boundedness is undecidable; we will therefore only give a sufficient condition. This condition will be expressed in the form of a test, which takes as input a protocol containing messages x, y, ... and outputs for each message an upper bound on its occurrences in a channel. For example, the test could result in an upper bound of 3 for x, but no finite upper bound for y. That means that during execution there will never be more than three copies of message x in transit, whereas there might be any number of copies of y in transit. If for each message the test provides a finite upper bound then all the channels are bounded.

The first step of the test is to identify all the loops in the protocol.

A loop is a pair <s, X> where s is a state and X is a non-empty sequence of messages with two properties:

(i) succ(s, X) = s and

(ii) if X' is obtained from X by removing some messages and still succ(s, X') = s, then X' must be empty (i.e., loops do not contain other loops).

Two loops <s, X> and <s', X'> are equivalent if for some Y and Z

$X = YZ$, $X' = ZY$, and $s' = succ(s,Y)$.

For the test we need to identify only one representative of each equivalence class (i.e., only one starting point of a loop).

For example, in the protocol of Figure C.1 there are seven loops:

1.) <a, xy>,

2.) <a, xz>,

3.) <a, w>,

4.) <c, xy>,

5.) <c, xz>,

6.) <c, w>, and

7.) <d, w>.

The second step of the test is to form a number of inequalities, which will be solved in the third step.

The inequalities involve the following variables -- for each loop i, a variable n_i and for each message x, a variable m_x. The inequalities will reflect conditions that must be satisfied at any point of an execution. In those conditions n_i represents the number of times the loop i has been traversed, and m_x represents the number of copies of message x in transit at that point of execution. Our aim will be to show that the inequalities imply $m_x \leq h$ for some constant h.

If the protocol has L loops and M messages then there will be L+3M inequalities. For each loop i there is an inequality

$$n_i \geq 0.$$

For each message x there are three inequalities:

$$m_x \geq 0,$$

$$\sum_{i=1}^{L} t_{xi} n_i - \sum_{i=1}^{L} r_{xi} n_i - m_x \leq r_x \qquad \text{(C1)}$$

$$\sum_{i=1}^{L} t_{xi} n_i - \sum_{i=1}^{L} r_{xi} n_i - m_x \geq -t_x, \qquad \text{(C2)}$$

where

t_{xi} is the number of transmissions -x in loop i,

r_{xi} is the number of receptions +x in loop i,

t_x is the total number of transmissions -x appearing in the protocol, and

r_x is the total number of receptions +x appearing in the protocol.

Note that in $\sum_{i=1}^{L} t_{xi}n_i$ all the loops i without -x will have $t_{xi} = 0$ and thus will not contribute to the sum. (Similarly for the loops without +x.)

Before justifying (C1) and (C2) we list the eight inequalities that they produce for the protocol of Figure C.1. These are in addition to the inequalities asserting that all the variables are non-negative.

x1) $n_1 + n_2 - n_4 - n_5 - m_x \leq 1$

x2) $n_1 + n_2 - n_4 - n_5 - m_x \geq -1$

y1) $n_4 - n_1 - m_y \leq 1$

y2) $n_4 - n_1 - m_y \geq -1$

z1) $n_5 - n_2 - m_z \leq 1$

z2) $n_5 - n_2 - m_z \geq -1$

w1) $n_3 - n_6 - n_7 - m_w \leq 2$

w2) $n_3 - n_6 - n_7 - m_w \geq -1$.

The reasoning behind inequalities (C1), (C2) is the fact that at any point during execution the number of messages x sent (denoted out_x) equals the number received (in_x) plus the number in transit (m_x). That is,

$$0 = out_x - in_x - m_x. \tag{C3}$$

Each time a loop i is executed t_{xi} copies of x are sent. Thus,

$$out_x \geq \sum_{i=1}^{L} t_{xi}n_i. \tag{C4}$$

However, it is possible to transmit x without going around a whole loop. Any one of the t_x transmissions of x can be executed (at most) once without executing any loop containing the transmission. Thus,

$$out_x \leq t_x + \sum_{i=1}^{L} t_{xi}n_i. \tag{C5}$$

Using the same reasoning for receptions we get

$$in_x \geq \sum_{i=1}^{L} r_{xi} n_i \ , \tag{C6}$$

$$in_x \leq r_x + \sum_{i=1}^{L} r_{xi} n_i . \tag{C7}$$

By substituting (C4) and (C7) into (C3) we get (C1), and by substituting (C5) and (C6) into (C3) we get (C2).

The third step of the test is to solve the inequalities for the variables m_x. Since the inequalities are linear, relatively efficient inequality solvers can be used[2,16]. In our example by adding inequalities x1, y1, z1 we get

$$m_x + m_y + m_z \leq 3.$$

Notice that the actual upper bound is 0, 1, or 2 depending on where the initial states are. In contrast, the inequalities do not imply any upper bound for the message w.

The discrepancy between the computed bound of three and the actual bounds illustrates that the test represents only a sufficient condition. In general, the computed bounds may not be achievable because the test does not take into account what global states are reachable. Especially, it does not take into account what are the initial states.

Appendix D -- Proofs for Section 4.2

We will need to distinguish clearly between a state marked dead and a state dead just by virtue of being in the subtree of a marked state. Therefore, we will refer to the former as killed. Thus, there are three kinds of states in a well-marked protocol:

(i) killed states, states that are explicitly marked as dead subject to the conditions of Definition 4.7;

(ii) states that are dead, but not killed, i.e., merely in the subtree of a killed state, and

(iii) live states -- all the states that are not dead.

In this Appendix we shall extend the definition of a precursor. It is necessary because even though a state killed of type 0 has a precursor, a state s in its subtree does not necessarily have a precursor $< s$. But there is always a state s' satisfying conditions (ii) and (iii) of Definition 4.5. These are the essential conditions and they are preserved in the new Definition D.3. Definition D.3 could have been given instead of Definition 4.5, but then the main body of the paper would have to be complicated by the ordering "before" (see Definition D.1).

Below we will show that the approach is valid (i.e., dead states can be ignored) even with the more liberal Definition D.3. This will imply that the approach is valid also with the more restrictive Definition 4.5. In Propositions D.2 and D.3 we prove under what conditions the approach guarantees termination of the tree growth; and for that we will use the more restrictive Definition 4.5.

As stated at the beginning of Section 4.2, we assume every protocol to be a well-formed tree protocol with an equivalence relation $\sim$. In general, we will assume that whenever $P'' \subseteq P$ then the equivalence relation on P is an extension of the equivalence relation on P''.

Moreover, (for the next Definition D.1) we assume that every protocol has all messages linearly ordered in an arbitrary but fixed manner. We will call this ordering "generated

before". This is extended in the usual way to a lexical ordering of message sequences. Then we say that s' is before s if the branch to s' is lexically before the branch to s.

Definition D.1. For two states s, s' in the same process we say that s' is *before* s iff either

(i) the branch to s' is shorter than the branch to s, or

(ii) the branches are of the same length,

the branch to s is of the form XyZ, and

the branch to s' is of the form $Xy'Z'$, where y' is a message generated before y and X, Z, Z' are (possibly empty) message sequences.

Note that the ordering before is well founded.

Definition D.2: For two states s, s'

$s' <\sim s$ iff $s' \sim s$ and s' is before s,

$s' \leq\sim s$ iff $s' <\sim s$ or $s' = s$.

For two N-tuples S and S'

$S' \leq\sim S$ iff $s'_i \leq\sim s_i$ for all i,

$S' <\sim S$ iff $S' \leq\sim S$ and for some i $s'_i \neq s_i$.

Note that $<\sim$ is a well founded ordering on states as well as N-tuples, and that it has the following property: Whenever $s' <\sim s$ then $succ(s', X) <\sim succ(s, X)$, for any message sequence X for which both sides of the inequality are defined. Similarly, whenever $s' \leq\sim s$ then $succ(s', X) \leq\sim succ(s, X)$.

We use the symbol $<$ because the ordering before is a generalization of the ordering $<$. (If $s' < s$ then s' is before s.) In this light, the following definition should be seen. It extends Definition 4.5 in that instead of requiring $s' < s$ it is sufficient when s' is before s. [If s' is a precursor of s according to Definition 4.5 then it is also a precursor by Definition D.3 -- condition (ii) below follows by Lemma B.2.]

Definition D.3 A state s' is a *precursor* of a state s iff the following three conditions are satisfied:

(i) $s' \neq s$,

(ii) $L(s') \leq\sim L(s)$,

(iii) $\text{Channels}(L(s')) \sim \text{Channels}(L(s))$.

Notation. For an N-tuple S, $S(i : s')$ denotes an N-tuple identical to S in all components, except for the i-th component, which is s'.

LEMMA D.1: *Let Q be a reachable N-tuple and assume that* $Q \geq L(s)$ *for some state s, but it is not the case that* $Q = L(s)$. *Then there exists a reachable N-tuple* $R \geq L(s)$ *so that* $R \mid\!\!- Q$.

PROOF. Consider any sequence of steps of Definition 4.3 forming the protocol. Out of all the states

$$q_i > L_i(s) \tag{D1}$$

select that q_i that was introduced last into the protocol. Let $q_i = \text{succ}(r_i , x)$ and let $R = Q(i : r_i)$. By (D1) $R \geq L(s)$.

To show that R is reachable we apply Lemma B.7. We have to check that for all $j \neq i$, (i) $L_j(r_i) \leq q_j$ and (ii) $L_i(q_j) \leq r_i$.

(i) $L_j(r_i) \leq L_j(q_i) \leq q_j$,

where the first inequality follows by Lemma B.2 and the second by Lemma B.7 applied to Q.

(ii) If $q_j = L_j(s)$ then $L_i(q_j) = L_i(L_j(s)) \leq L_i(s) \leq r_i$,

where the first inequality follows by Lemma B.5 and the second by (D1).

If $q_j > L_j(s)$ then q_j was introduced into the protocol earlier than q_i , and therefore $L_i(q_j) < q_i$, whence $L_i(q_j) \leq r_i$.

This proves that R is reachable. To show that $R \mid\!\!- Q$ we have two cases.

If $q_i = succ(r_i , -x_{ik})$ then R |— Q by step (i) of Definition 2.3.

If $q_i = succ(r_i , +x_{ki})$ then R |— Q by step (ii) of Definition 2.3, provided that $Channel(q_k , r_i)$ is of the form $x_{ki}Y$. Since Q is reachable, $-x_{ki}$ appears on the branch to q_k. By considering the branches to q_k , q_i , and r_i , $Channel(q_k , r_i)$ does contain x_{ki} , but $Channel(q_k , q_i)$ does not, and therefore x_{ki} must be at the front of $Channel(q_k , r_i)$.

LEMMA D.2. *Assume a protocol with the following property.*

whenever succ(t, -x) is defined and $t' <\sim t$ *then* (D2)

succ(t', -x') is defined for some $x' \sim x$.

Let s' be a precursor of s and let Q be a reachable N-tuple with

$Q \geq L(s)$. (D3)

Then there exists a reachable N-tuple $Q' \geq L(s')$ *such that* $Q' \leq\sim Q$ *and* $Channels(Q') \sim Channels(Q)$.

Moreover, whenever $q_i = succ(L_i(s), X)$ *then* $q_i' = succ(L_i(s'), X')$ *for some message sequence* $X' \sim X$.

PROOF. We will prove the lemma by induction on the total distance of Q from L(s). Let for each i, X_i be the path from $L_i(s)$ to q_i , i.e., $q_i = succ(L_i(s), X_i)$. The induction is on the value $|X_1| + ... + |X_N|$ (where $|X_i|$ is the length of X_i). The basis -- when Q = L(s) -- follows from definition of precursor by setting $Q' = L(s')$.

For the induction step by Lemma D.1 there exists R |— Q so that $R \geq L(s)$. Therefore, by induction hypothesis there exists $R' \geq L(s')$ such that $R' \leq\sim R$ and $Channels(R') \sim Channels(R)$.

We will show that R' |— Q', where Q' has the sought properties. Since R |— Q, there is i so that $Q = R(i : q_i)$.

Case 1: $q_i = succ(r_i , -x)$.

Since the protocol satisfies (D2), there exists $q_i' = succ(r_i', -x')$ for some $x' \sim x$. Let $Q' = R'(i : q_i')$; then $R' \mid\!\!- Q'$. By comparing the N-tuples Q, R, R', and Q' we see that $Q' \leq\sim Q$, Channels(Q') ~ Channels(Q), and the paths from L(s') to Q' are equivalent to the paths from L(s) to Q.

Case 2: $q_i = succ(r_i, +x)$

Since Channels(R') ~ Channels(R) and the protocol is well-formed, there exists $q_i' = succ(r_i', +x)$. By the same reasoning as in Case 1, $Q' = R'(i : q_i')$ satisfies the lemma.

LEMMA D.3. *Assume a protocol and quantities s, s', Q, Q' as given in Lemma D.2. Further assume that for some k*

$$Q = L(q_k) \text{ and} \tag{D4}$$

$$L(q_k') \geq L(s'). \tag{D5}$$

Then $Q' = L(q_k')$.

PROOF. Suppose the contrary and we will derive a contradiction. Thus, there is at least one $q_i' \neq L_i(q_k')$. Out of all such q_i' select the one that was introduced last into the protocol according to some fixed but arbitrary sequence of steps of Definition 4.3. Q' is reachable by Lemma D.2 and by Lemma B.7 $L_i(q_k') \leq q_i'$. Therefore,

$$L_i(q_k') < q\, L_i(q_k') < q_i'. \tag{D6}$$

Consider the paths X and X' to q_i and q_i', i.e.,

$q_i = succ(L_i(s), X)$, $q_i' = succ(L_i(s'), X')$.

From (D5) and (D6) X' is not empty, thus there is a sequence Y' and message z' so that X' = Y'z'. By Lemma D.2, X ~ X'; therefore there is Y ~ Y' and z ~ z' so that X = Yz. Because of (D6) $i \neq k$ and because of (D4) q_i is the entry state of a transmission; thus the occurrences of z and z' in X and X' must be transmissions.

Let the message z be directed to process j. Then +z must appear on the path from $L_j(s)$ to q_j, i.e., for some V and W, $q_j = succ(L_j(s), V +z W)$. The reasoning is the following:

The branch to q_j must contain $+z$ because otherwise $Q(i : succ(L_i(s), Y))$ would be reachable, which is not possible by Lemma B.7. On the other hand the branch to $L_j(s)$ cannot contain $+z$ because otherwise $q_i \leq L_i(L_j(s)) \leq L_i(s)$,

where the first inequality follows from Definition 4.3 (after receiving $+z$ process j knows that process i is in state q_i or further) and the last inequality follows by Lemma B.5. In combination with (D3) $q_i = L_i(s)$, hence by Lemma D.2 $q_i' = L_i(s')$, contradicting (D5) and (D6). Therefore, $+z$ must appear on the path from $L_j(s)$ to q_j.

By Lemma D.2 there exist $V'' \sim V$, $z'' \sim z$, and $W'' \sim W$ so that $q_j' = succ(L_j(s'), V'' +z'' W'')$. We shall show that $z'' = z'$. Let $Channel(L_i(s), L_j(s))$ contain c messages and let Y contain d transmissions directed to process j. Then V contains c+d receptions from process i (because of FIFO channels). By Lemma D.2, $Channel(L_i(s'), L_j(s'))$ also contains c messages and Y' contains d transmissions to process j. Thus, $+z'$ will be received as the $c+d+1^{st}$ message from process i (counting from $L_j(s')$), and that message is z''. Hence $z'' = z'$.

Since the branch to q_j' contains $+z'$, q_j' must have been introduced into the protocol after q_i' , and therefore $q_j' = L_j(q_k')$. Thus,

$$q_i' \leq L_i(q\ q_i' \leq L_i(q_j') = L_i(L_j(q_k')) \leq L_i(q_k'),$$

where the first inequality follows from the fact that q_i' is the entry state of $-z'$ and the branch to q_j' contains $+z'$, the second inequality follows from Lemma B.5, and constitutes a contradiction to (D6).

LEMMA D.4. *Assume a protocol satisfying (D2). Let s' be a precursor of s and let $r = succ(s, x)$. Then r has a precursor $r' = succ(s', x')$ for some $x' \sim x$.*

PROOF. We use Lemma D.2 with $Q = L(r)$, which is $\geq L(s)$ by Lemma B.2. There is Q' with the properties given in the conclusions of Lemma D.2. Let k be the index of the process having states s, s', r. We let $r' = q'_k$. We will be done if we show that $Q' = L(r')$. This follows from Lemma D.3, which we can apply because condition (D5) follows from Lemma B.2.

COROLLARY D.1. *In a protocol satisfying (D2) if s has a precursor and $r \geq s$ the r also has a precursor.*

LEMMA D.5. *Assume a protocol satisfying (D2). Suppose that $s_i = succ(t_i , -x_{ik})$, $q_k = succ(r_k , +x_{ik})$, and that t_i has a precursor t'_i. Then q_k also has a precursor.*

PROOF. Let t'_i be a precursor of t_i and let

$$Q = L(q_k). \tag{D7}$$

In order to apply Lemma D.2 we prove that $Q \geq L(t_i)$:

$q_j = L_j(q_k)$ by (D7)

$\geq L_j(L_i(q_k))$ by Lemma B.5

$\geq L_j(s_i)$ by definition of $L_i(q_k)$

$\geq L_j(t_i)$ by Lemma B.2.

Therefore, there is Q' with the properties given by Lemma D.2. We will show that q'_k is a precursor of q_k.

We apply Lemma D.3 to prove that $Q' = L(q'_k)$. For that we have to show that $L(q'_k) \geq L(t'_i)$. By Lemma D.2 [namely, that the paths from $L(t'_i)$ to Q' are equivalent to the paths from $L(t_i)$ to Q] there are

$x'_{ik} \sim x_{ik}$, $s'_i \sim s_i$, $x''_{ik} \sim x_{ik}$, and $r'_k \sim r_k$, so that

$s'_i = succ(t\ s'_i = succ(t'_i , -x'_{ik})$ and $q'_k = succ(r'_k , +x''_{ik})$.

By the same argument as in the proof of Lemma D.3, $x''_{ik} = x'_{ik}$. Thus, for $j \neq k$,

$$L_j(q'_k) = \max(L_j(r'_k), L_j(s'_i)) \geq L_j(s'_i) \geq L_j(t'_i).$$

For $j = k$, $L_k(q'_k) \geq L_k(t'_i)$ because Lemma D.2 provides $Q' \geq L(t'_i)$.

Thus, by Lemma D.3 $Q' = L(q'_k)$. This establishes conditions (ii) and (iii) of Definition D.3.

To show condition (i) of Definition D.3 assume the contrary, i.e., $q'_k = q_k$. Then $q'_i = L_i(q'_k) = L_i(q_k) = q_i$.

By Lemma D.2 the path from t'_i to q'_i is of the same length as the path from t_i to q_i, and therefore $t'_i = t_i$, contradicting the fact that t'_i is a precursor of t_i. Therefore, $q'_k \neq q_k$, and q'_k is a precursor of q_k.

LEMMA D.6. *If t_i can send to r_k and $t'_i \leq t_i$ then t'_i can send to some $s_k \leq r_k$.*

PROOF. It is sufficient to prove the lemma for the case $t_i = \text{succ}(t'_i, x)$. Because for $t'_i = t_i$ the lemma is satisfied with $s_k = r_k$, and if the path from t'_i to t_i is longer than one then the lemma follows by induction.

Case 1: x is from M_{ik}.

$\text{From}_i(r_k) = \text{To}_k(t_i) = x$, therefore there exists s_k so that $r_k = \text{succ}(s_k, +x\ Y)$ for some (possibly empty) sequence Y. t'_i can send to r_k because conditions (3), (4), (5) imply (3), (5), (6).

Case 2: x is not from M_{ik}.

We prove that t'_i can send to r_k.

(3): $\text{From}_i(r_k)$	$= \text{To}_k(t_i)$	by (3) for t_i can send to r_k
	$= \text{To}_k(t'_i)$	by Definition of To.
(5): $L_k(t'_i)$	$\leq L_k(t_i)$	by Lemma B.2
	$\leq r_k$	by (5) for t_i can send to r_k.

(6): $L_j(t_i') \leq L_j(t_i)$ by Lemma B.2

$\leq \max(L_j(t_i), L_j(r_k))$ by (6) for t_i can send to r_k

$L_j(r_k) \leq \max(L_j(t_i), L_j(r_k))$

$L_j(t_i') \lesseqgtr L_j(r_k)$ by Lemma B.1

LEMMA D.7. *Let S be a reachable N-tuple with Channel(s_i , s_k) empty. Then s_i can send to s_k.*

PROOF.

(3) follows from Channel(s_i , s_k) being empty because that implies that the branches to s_i and s_k contain the same messages from M_{ik}.

(5): $L_k(s_i) \leq s_k$ by Lemma B.7

(6): $L_j(s_i) \lesseqgtr L_j(s_k)$ will follow from Lemma B.1 if we show that both sides are $\leq s_j$. And that follows from Lemma B.7.

LEMMA D.8. *If s is a state in a well-marked tree protocol satisfying (D2) then there is a live state $s'' <\sim s$.*

PROOF by induction on the ordering $<\sim$. Assume the lemma for every state $s' <\sim s$. If s is live then $s'' = s$ satisfies the lemma. If s is dead of type 0 then by Corollary D.1, s has a precursor $s' <\sim s$. By induction hypothesis there exists live $s'' <\sim s' <\sim s$.

So for the rest of the proof assume that s is dead of type 1, and let s° be the state on the branch to s that is killed of type 1. Let i be the process containing s and let k be any other process.

Consider all the reachable N-tuples S where

$s_i = s$ and Channel(s_i , s_k) is empty. (D8)

There is certainly at least one such N-tuple. To see that, pick any reachable N-tuple containing s; it exists because the protocol is well-formed. Then receive all the messages in the channel from process i to k. As a result, process k reaches a state s_k with Channel(s_i , s_k) empty. The resulting N-tuple is reachable and satisfies (D8).

From all the reachable N-tuples satisfying (D8) select that S where s_k is minimal in the ordering $<\sim$. By Lemma D.7, s_i can send to s_k. By Lemma D.6, s° can send to some $s_k^\circ \leq s_k$. By condition (iii) of Definition 4.7, s_k° is dead of type 0. By Corollary D.1, s_k has a precursor $s_k' <\sim s_k$. By Lemma B.7, $S \geq L(s_k)$. By Lemma D.2, there exist S' containing s_k' with $S' \leq\sim S$ and Channels(S') $\sim$ Channels(S).

The N-tuple S' cannot contain s because then S' would satisfy (D8) contradicting the minimality of S. (S' contains $s_k' <\sim s_k$.) Therefore, $s_i' <\sim s_i$ and we can apply induction hypothesis to s_i' obtaining $s'' <\sim s_i' <\sim s$. This proves the lemma.

The next proposition considers the growing of tree protocols in an attempt to discover new information about a general protocol, as described at the beginning of Section 4. Assume that we have constructed a well-marked tree protocol P'', which need not be expanded further according to Section 4.2. That is, live states have all possible transmissions attached. It is sufficient to express this by (D9) below, for Lemma D.8 implies that there will always be a live state to which to attach an executable transmission appearing in the general protocol.

Suppose that we do not stop the growth of the trees, and append some transmissions even to dead states. This is followed by specifying all the necessary receptions in order to obtain a well-formed protocol P. All the transmissions appended to dead states are equivalent to some already existing transmissions in P'' -- (D10) below. The proposition states that P cannot provide any more information about the general protocol than the original P''. More exactly, by the extension to P we will not discover in the general protocol any

I. new executable receptions, or

II. new stable N-tuples.

PROPOSITION D.1. *Let* $P'' \subseteq P$, *where*

$P'' = <<S_i''>_{i=1}^N , <o_i>_{i=1}^N , <M_{ij}''>_{i,j=1}^N , succ''>$ *and*

$P = <<S_i>_{i=1}^N , <o_i>_{i=1}^N , <M_{ij}>_{i,j=1}^N , succ>$.

Assume that killed states are only in P'' *and* P'' *is well-marked. (P is not necessarily well-marked, but we will call a state of P dead if it is* $\geq$ *a killed state of* P''.*) Assume that* P'' *has the following property:*

whenever $succ''(t, -x)$ *is defined,* $t' \sim t$ *and* t' *is live then* (D9)

there exists $x' \sim x$ *so that* $succ''(t', -x')$ *is defined.*

Further assume that

whenever $succ(t, -x)$ *is defined then* $succ''(t'', -x'')$ *is defined* (D10)

for some $t'' \sim t$ *and* $x'' \sim x$.

Then the following two claims are true:

I. If $succ(r_k , +y)$ *is defined for some* r_k *and y in P then* $succ''(r_k'' , +y'')$ *is defined for some* r_k'' , y'' *in* P'' *such that* $r_k'' \sim r_k$ *and* $y'' \sim y$.

II. If S is a stable N-tuple in P then P'' *has a stable N-tuple* $S'' \sim S$.

PROOF. Without loss of generality we can assume that P has property (D2). Because otherwise we could add to P all the messages needed to satisfy (D2). The number of such added messages is finite because a state can be the departure state of only a finite number of transmissions, and the branch to such a departure state cannot be longer than the longest branch in P. (This is independent of where the added messages would be in the ordering "generated before".) Note that even after adding any such messages to satisfy (D2) both (D9) and (D10) remain true.

We can also assume that no two equivalent messages x and x' of P share the same departure state. Consider such a situation and let x be generated before x'. Then for any reachable global state of P there is an equivalent global state reachable without using the

message x'. (For x and x' bring the processes into equivalent states.) Therefore, it is enough to prove the proposition with the assumption that no two such messages x and x' exist. Then (D9) and (D10) imply that

if succ(t'', -x) is defined for some t'' from P'' and some x not in P'' then t'' is dead. (D11)

First we prove three lemmas about P and P''.

LEMMA D.9. *A state s is in P'' iff the branch to s'' consists of messages from P'' only.*

PROOF. This follows from Definition 4.3: In constructing P from P'' all new states introduced have on their branches either -x or +x for some x not in P''.

LEMMA D.10. *If s = succ(s'', x) for some s'' in P'', but s is not in P'' then s either has a precursor or is dead of type 1.*

PROOF. The proof is by induction on the well founded ordering <<+; namely, we assume that the lemma is true for any r = succ(r'', y) if r <<+ s, where r'' is in P'' and r is not.

If s = succ(s'', -x) then by (D11) s'' (and hence s) is dead. Therefore, the lemma is satisfied.

Assume s = succ(s'', +x). Let t' = succ(t, -x). Since x is not in P'', neither is t'; let y be the first message not in P'' on the branch to t'. Thus, there exists r'' in P'' and r not in P'' so that r = succ(r'', y) $\leq$ t'.

Case 1: r = succ(r'', -y).

By (D11) r'' is dead and so is t. If t is dead of type 0 then by Corollary D.1, t has a precursor, and by Lemma D.5, s also has a precursor. If t is dead of type 1 then by Lemma D.6 and Corollary D.1 (using the fact that t can send to s''), s has a precursor.

Case 2: $r = succ(r'', +y)$

Then $y \neq x$ and $r \leq t$. Therefore, $r <<+ s$ and we can apply induction hypothesis to r. If r has a precursor then so does t and by Lemma D.5, s has a precursor. If r is dead of type 1 then as in Case 1, s has a precursor.

COROLLARY D.2. *If s is not in P'' then s either has a precursor or is dead of type 1.*

LEMMA D.11. *Suppose that a state t_i dead of type 1 can send to r_k. Then r_k has a precursor.*

PROOF. Let $t_i \geq t_i''$, which is killed of type 1. By Lemma D.6 there exists r_k' to which t_i'' can send. If r_k' is in P'' then it must be dead of type 0, and by Corollary D.1, r_k has a precursor. So assume that r_k' is not in P''.

By Lemma D.9 the branch to r_k' contains some messages not in P''. Let x be the first such message on the branch to r_k'. Then there are q_k and q_k' so that

$$q_k' = succ(q_k , x) \leq r_k'.$$

By Lemma D.10,

q_k' either has a precursor or is dead of type 1. (D12)

We will show that q_k' cannot be dead of type 1 and therefore it must have a precursor.

We will use the fact that t_i'' can send to r_k' to show that the N-tuple $S = \max(L(t_i''), L(q_k))$ is reachable. First we have to show that S is well defined:

For $j \neq k$,

$L_j(q_k) \leq L_j(r_k')$ by Lemma B.2

$\leq \max(L_j(r_k'), L_j(t_i''))$ max exists by (6)

$L_j(t_i'') \leq \max(L_j(r_k'), L_j(t_i''))$ max exists by (6)

$L_j(t_i'') \lesssim L_j(q_k)$ by Lemma B.1.

For $j = k$,

$$L_k(t_i'') \leq q_k \qquad \text{(D13)}$$

because by (5) $L_k(t_i'') \leq r_k'$, $L_k(t_i'')$ must be in P'' as t_i'' is, and q_k is the last state in P'' that is $\leq r_k'$.

Therefore, S is well defined. To show that S is reachable we apply Lemma B.7. We have to prove that for all m and j, $L_m(s_j) \leq s_m$.

$$\begin{aligned} L_m(s_j) &= L_m(\max(L_j(t_i''), L_j(q_k))) && \text{by Definition of S} \\ &= \max(L_m(L_j(t_i'')), L_m(L_j(q_k))) && \text{by Lemma B.2} \\ &\leq \max(L_m(t_i''), L_m(q_k)) && \text{by Lemma B.5} \\ &= s_m && \text{by Definition of S} \end{aligned}$$

From S an execution can proceed by receiving all the messages present in the channels from process i to k, arriving at a state q_k''. This state is in P'', for S is and P'' is well-formed. By Lemma D.7, s_i can send to q_k''. By the definition of S, $s_i \geq t_i''$. By Lemma D.6, t_i'' can send to some $r_k'' \leq q_k''$. Therefore, r_k'' (and hence q_k'') is dead of type 0. By (D13), $s_k = q_k$ and therefore $q_k \leq q_k''$. By condition (i) of Definition 4.7, q_k cannot be dead of type 1. Since q_k' is not in P'' it cannot be killed and therefore cannot be dead of type 1 either. By (D12), q_k' has a precursor and by Corollary D.1, r_k also has a precursor. This proves the lemma.

PROOF OF CLAIM I. The proof is by induction on the well founded ordering before. Let t_i be the departure state of -y. As the induction hypothesis we assume that the claim (I) is true for any r_k' and y', with -y' having departure state t_i' , provided $<r_k' , t_i'>$ is lexically before $<r_k , t_i>$; that is, r_k' is before r_k , or $r_k' = r_k$ and t_i' is before t_i. We will prove (I) for r_k and y.

By Theorem 4.1 and Definition 3.1 there is a reachable N-tuple R containing r_k with Channel(r_i , r_k) = yW for some sequence W. If r_k has a precursor r_k' then we are done. Because then by Lemma D.2 there exists R' containing r_k' so that

Channels(R′) ~ Channels(R). Therefore, succ(r'_k , $+y'$) is defined for some $y' \sim y$. By induction hypothesis there exist r''_k and y'' satisfying claim (I).

We are also done if t_i has a precursor t'_i for the following reason. $R \geq L(t_i)$ using Lemmas B.2, B.7 and the fact that Channels(R) contain y. By Lemma D.2 there exists $R' <\sim R$ with Channels(R′) ~ Channels(R). Thus, succ(r'_k , $+y'$) is defined for some $y' \sim y$. Moreover, since the paths from $L(t'_i)$ to R′ are equivalent to the paths from $L(t_i)$ to R, t'_i is the departure state of $-y'$. Therefore, we can apply induction hypothesis to r'_k , y' and t'_i to obtain the sought r''_k and y''. Therefore, the claim will be established as soon as we prove that r_k or t_i has a precursor.

Case 1: r_k is in P″.

Case 1.1: y is in P″. Then $r''_k = r_k$ and $y'' = y$ satisfy (I).

Case 1.2: y is not in P″.

If t_i is in P″ then by (D11) t_i is dead. If it is dead of type 0 then by Corollary D.1, t_i has a precursor. If t_i is dead of type 1 then by Lemma D.11, r_k has a precursor. Therefore, claim (I) is true in both cases.

If t_i is not in P″ then by Corollary D.2, t_i either has a precursor or is dead of type 1. Therefore, the same argument as in the above paragraph applies.

Case 2: r_k is not in P″.

Case 2.1: The branch to r_k contains $+x$ for some x not in P″.
Let s be the departure state of $-x$. Depending on whether or not s is in P″ we apply (D11) or Corollary D.2. In both cases, s either has a precursor or is dead of type 1. If it has a precursor then by Lemma D.5 and Corollary D.1, r_k also has a precursor. If s is dead of type 1 then using Lemma D.11 and Corollary D.1, r_k has a precursor.

Case 2.2: All the messages received on the branch to r_k are from P''.

Since r_k is not in P'', by Lemma D.9 the branch to r_k contains some messages not in P''. Let x be the first message on the branch to r_k that is not in P''. Then it must be a transmission -x and its departure state s_k is in P''. By (D11), s_k is dead. If s_k is dead of type 0 then by Corollary D.1, r_k has a precursor, and we are done. So assume that s_k is dead of type 1.

Consider all the reachable N-tuples

R containing r_k where Channel(r_k , r_i) is empty and (D14)

Channel(r_i , r_k) ~ yW for some sequence W.

At least one such N-tuple exists for the following reason. By Theorem 4.1 and Definition 3.1 there is an N-tuple R° containing r_k where Channel(r_i° , r_k) is of the form yW. Starting from R° process i can receive all the messages that are in the channel from process k, reaching an N-tuple R satisfying (D14).

Out of all the N-tuples satisfying (D14) select one where r_i is minimal in the ordering before. By Lemma D.7, r_k can send to r_i and by Lemma D.11, r_i has a precursor r_i'. By Lemma B.7, $R \geq L(r_i)$; applying Lemma D.2 we obtain a reachable N-tuple $R' <\sim R$ containing r_i' with Channels(R') ~ Channels(R). If R' contained r_k then it would be of the form (D14) contradicting the minimality of R. Therefore, $r_k' <\sim r_k$ and we can apply induction hypothesis to r_k' obtaining the sought r_k'' and y''.

PROOF OF CLAIM II.

The proof is by induction on the well founded ordering $<\sim$. Assume that the claim is true for any stable N-tuple $S' <\sim S$, and we will prove it for S.

If S lies in P'' then $S'' = S$ satisfies the claim, so assume that some s_i does not lie in P''. By Corollary D.2, s_i either has a precursor or is dead of type 1. If s_i is dead of type 1 then all other states of S have a precursor (using Lemmas D.7 and D.11).

Therefore, some state of S has a precursor. By Lemma B.7 we can apply Lemma D.2 yielding $S' <\sim S$ with Channels(S') empty. Therefore, we can apply induction hypothesis to S' to obtain a stable N-tuple $S'' <\sim S' <\sim S$ with S'' in P''. This proves the proposition.

The next Proposition D.2 states that the tree growth is guaranteed to terminate for a general protocol, provided its channels are bounded by a constant h, each process has a finite number n of states, and each process can receive a finite number m of messages. This is true even if we use the more restrictive definition of a precursor -- Definition 4.5 rather than D.3, and even if we do not kill any states of type 1. However, it is necessary that no state with a precursor be left live. The bound b is very rough and is used to prove termination only, not any complexity results.

PROPOSITION D.2. *Assume a well-marked tree protocol with all channels bounded by a constant h. Let the relation $\sim$ divide all the states of each process into at most n equivalence classes and let $\sim$ divide all the messages receivable by any one process into at most m equivalence classes. Assume that no live state has a precursor according to Definition 4.5. Then no live state has a branch longer than* $b = n^N(m+1)^{N(N-1)h}$.

PROOF. In the protocol there can be at most n^N non-equivalent N-tuples. In counting the number of non-equivalent Channels(L(s)), there are N(N-1) non-empty channels each with at most h positions. Thus, there is a total of N(N-1)h positions to be filled with (m+1) items (m messages and a blank). Therefore, on a branch longer than b there must be s and s' with $L(s') \sim L(s)$ and $\text{Channels}(L(s')) \sim \text{Channels}(L(s))$.

LEMMA D.12. *Assume a well-formed tree protocol with $N = 2$ processes and let $i \neq k$.*

If $\text{From}_i(s_k) = *$ then $L_i(s_k) = o_i$.

If $\text{From}_i(s_k) = x$ then $L_i(s_k)$ is the entry state of $-x$.

PROOF. The proof is by induction on the well founded ordering $<$. That is, assume that the lemma is true for any $r_k < s_k$.

Case 1: $s_k = o_k$

Then $From_i(s_k) = *$ and $L_i(s_k) = o_i$.

Case 2: $s_k = succ(r_k , -y)$ for some r_k and y.

$From_i(s_k) = From_i(r_k)$ and $L_i(s_k) = L_i(r_k)$. Therefore, by induction hypothesis the lemma is satisfied.

Case 3: $s_k = succ(r_k , +y)$ for some r_k and y.

Since N = 2, y must be sent from process i, so $From_i(s_k) = y$. Let t_i' be the entry state of -y. Using the induction hypothesis, $L_i(r_k) < t_i'$ independently whether $From_i(r_k)$ is a message or *. Therefore,

$L_i(s_k) = \max(L_i(r_k), t_i') = t_i'$.

The next Proposition D.3 states that the tree growth is guaranteed to terminate for a general protocol, provided it has only two processes and one channel is bounded by a constant h. The other channel may be unbounded. The proposition is stated for the case when the channel from process 2 to process 1 is bounded; by renumbering the processes we obtain the result for the other possibility. The quantities n and m have the same meaning as in Proposition D.2. Also as in Proposition D.2 we have to assume that no state is kept live if it could be killed.

PROPOSITION D.3. *Assume a protocol with N = 2 processes and the channel from process 2 to process 1 bounded by h. Let the relation ~ divide the set of states of each process into at most n equivalence classes, and let ~ divide all the messages receivable by any one process into at most m equivalence classes. Assume that every state of process 2 with a precursor is dead of type 0, and that no live state of process 1 can send to type 0 dead states only. Then no live state has a branch longer than* $b = n^2(m+1)^h$.

PROOF. The same argument as in Proposition D.2 shows that any state s_2 with branch longer than b must have a precursor. The only difference is that Channel($L_1(s_2)$, s_2) is always empty, which follows from Lemma D.12.

Now consider a state s_1 whose branch is longer than b. We will show that s_1 can send only to states with precursors and therefore s_1 cannot be live. If s_1 can send to s_2 then by (5) and Lemma D.12 the branch to s_2 must contain transmissions of all the messages received on the branch to s_1. By (3) it must also contain receptions of all the messages transmitted on the branch to s_1. Therefore, the branch to s_2 is longer than b and hence s_2 has a precursor.

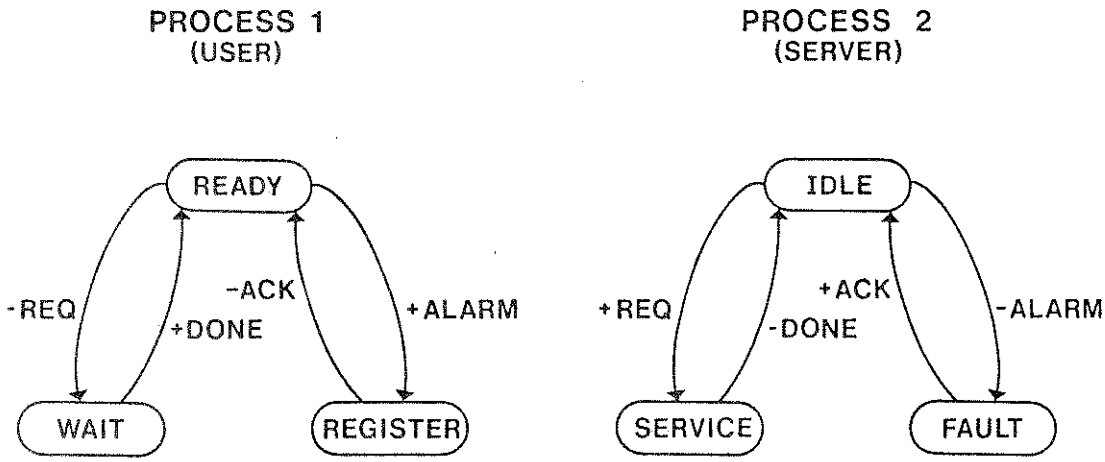

PROCESS 1
(USER)
READY
-REQ
-ACK
+DONE
+ALARM
WAIT
REGISTER
PROCESS 2
(SERVER)
IDLE
+REQ
+ACK
-DONE
-ALARM
SERVICE
FAULT

fig. 1

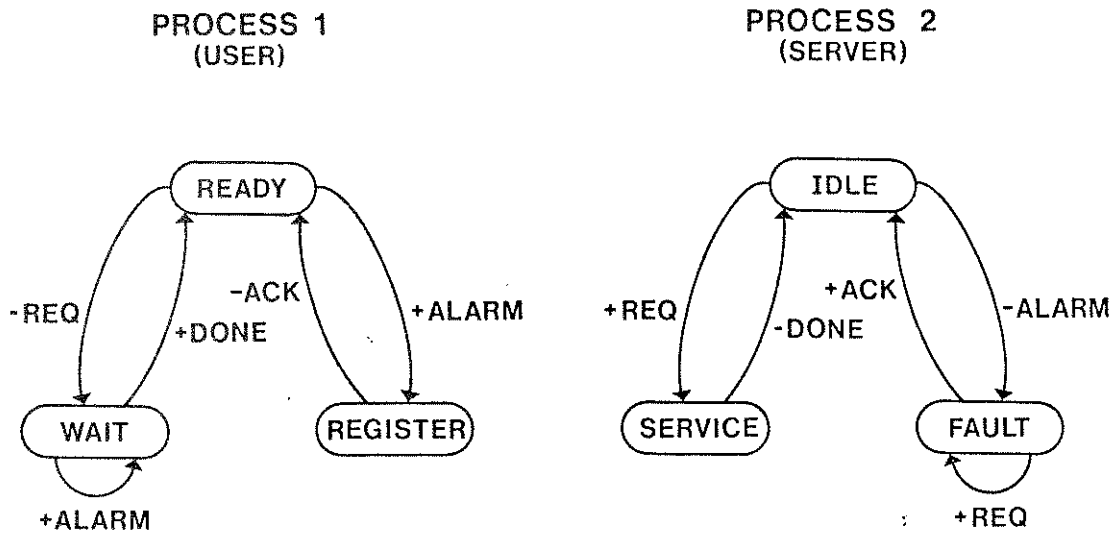

PROCESS 1
(USER)
READY
-REQ
-ACK
+DONE
+ALARM
WAIT
REGISTER
+ALARM
PROCESS 2
(SERVER)
IDLE
+REQ
+ACK
-DONE
-ALARM
SERVICE
FAULT
+REQ

fig. 2

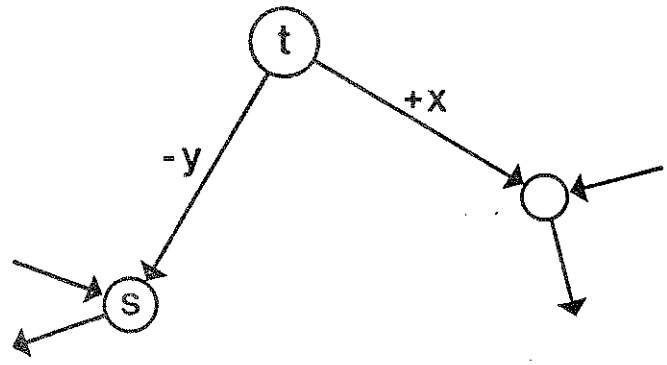

t
+x
-y
s

fig. 3

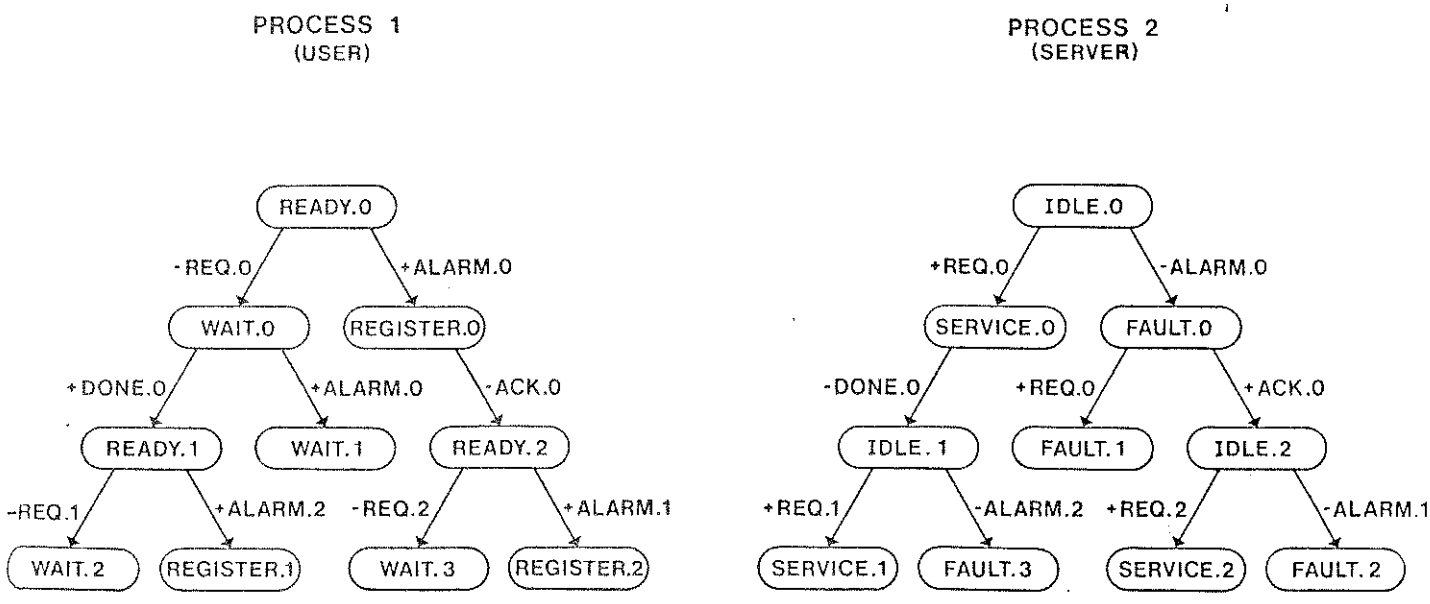
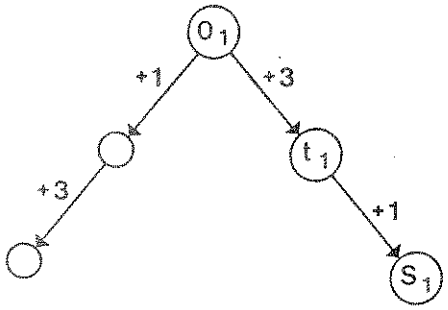

PROCESS 1
(USER)
READY.0
-REQ.0
+ALARM.0
WAIT.0
REGISTER.0
+DONE.0
+ALARM.0
-ACK.0
READY.1
WAIT.1
READY.2
-REQ.1
+ALARM.2
-REQ.2
+ALARM.1
WAIT.2
REGISTER.1
WAIT.3
REGISTER.2
PROCESS 2
(SERVER)
IDLE.0
+REQ.0
-ALARM.0
SERVICE.0
FAULT.0
-DONE.0
+REQ.0
+ACK.0
IDLE.1
FAULT.1
IDLE.2
+REQ.1
-ALARM.2
+REQ.2
-ALARM.1
SERVICE.1
FAULT.3
SERVICE.2
FAULT.2

fig. 4

PROCESS 1



PROCESS 2

PROCESS 3

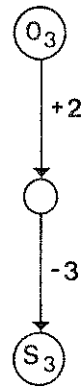
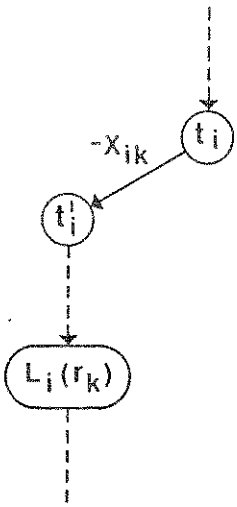
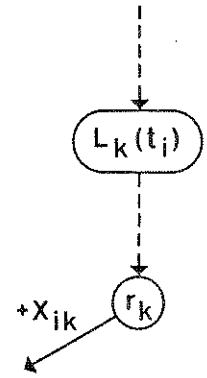


fig. 5

PROCESS i



PROCESS k



PROCESS j

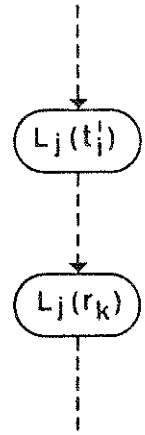


fig. 6

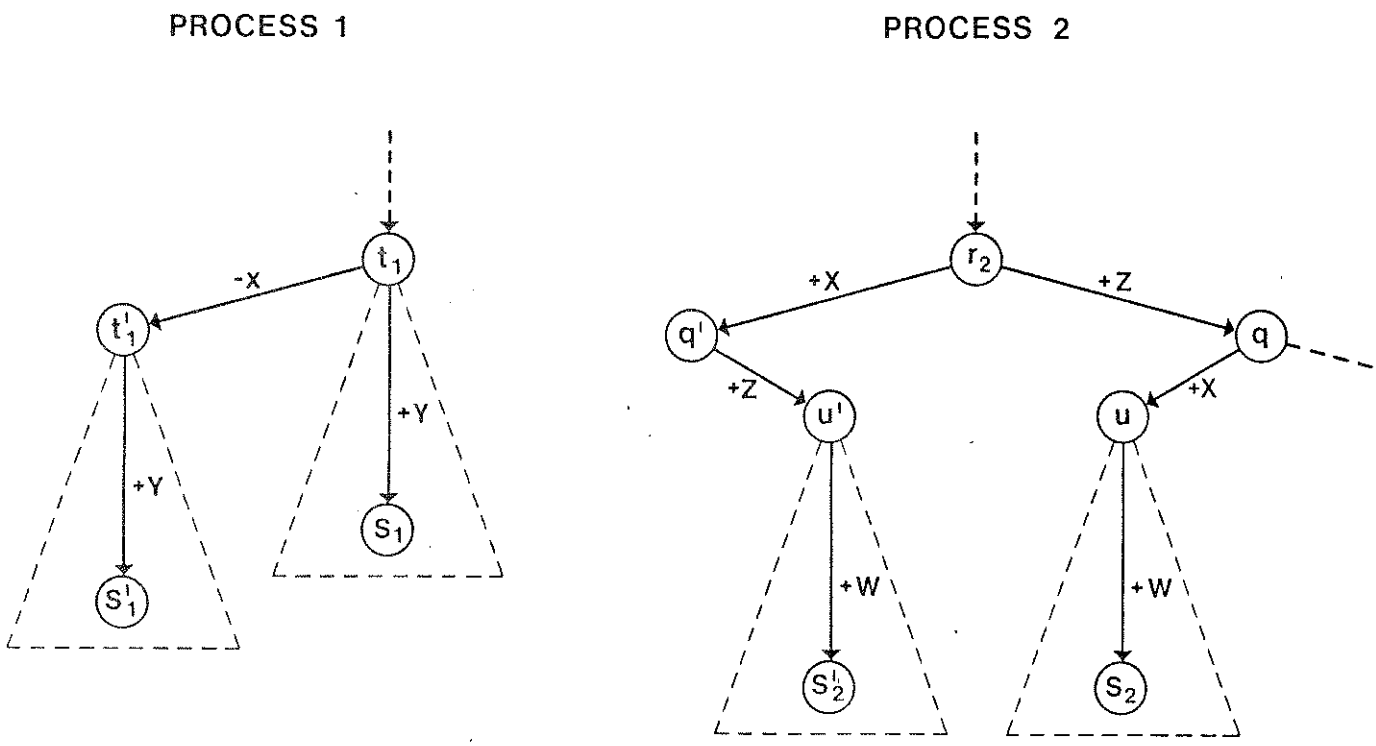

PROCESS 1
PROCESS 2
t1
-X
t'1
+Y
+Y
S1
S'1
r2
+X
+Z
q'
q
+Z
+X
u'
u
+W
+W
S'2
S2

fig. 7

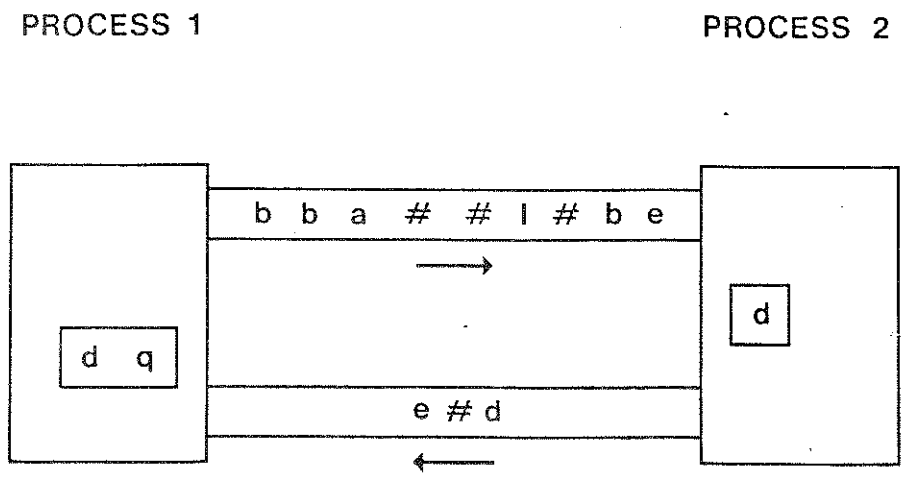

PROCESS 1
PROCESS 2
b b a # # l # b e
d q
d
e # d

fig. A1

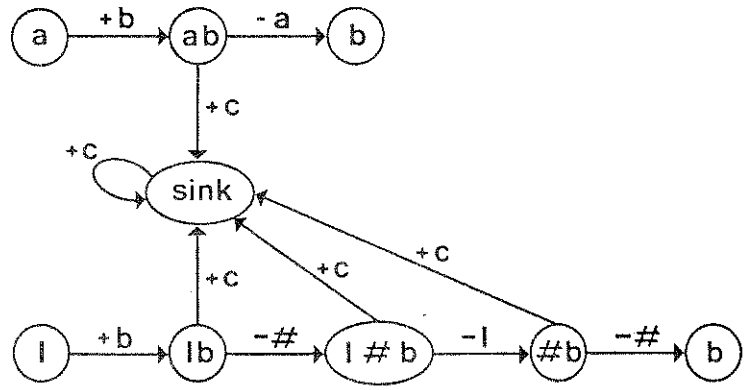

a
+b
ab
-a
b
+c
+c
sink
+c
+c
+c
l
+b
lb
-#
l # b
-l
#b
-#
b

fig. A2

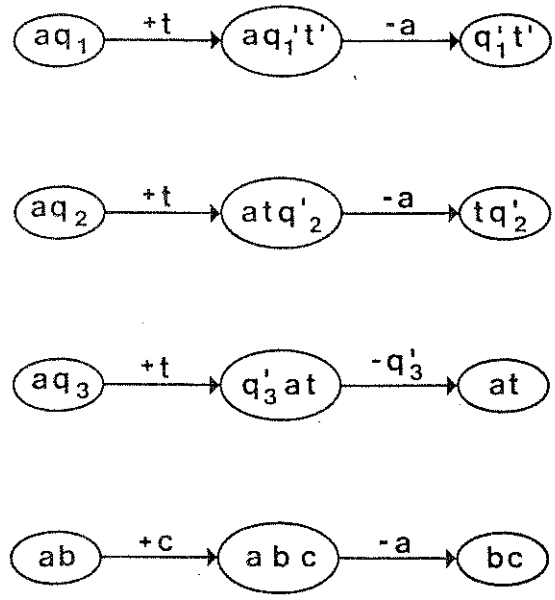

aq1
+t
aq1't'
-a
q1't'
aq2
+t
atq'2
-a
tq'2
aq3
+t
q'3at
-q'3
at
ab
+c
abc
-a
bc

fig. A3

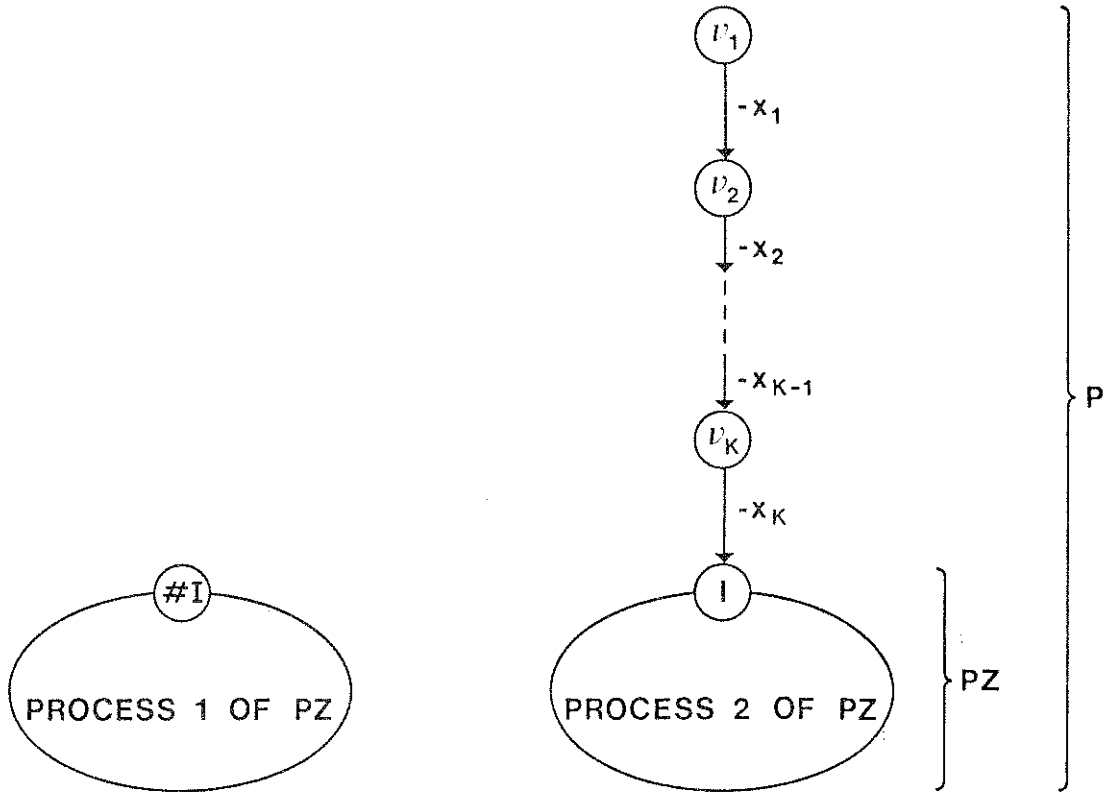

v_1
$-x_1$
v_2
$-x_2$
$-x_{K-1}$
v_K
$-x_K$
P
#I
PROCESS 1 OF PZ
I
PROCESS 2 OF PZ
PZ

fig. A4

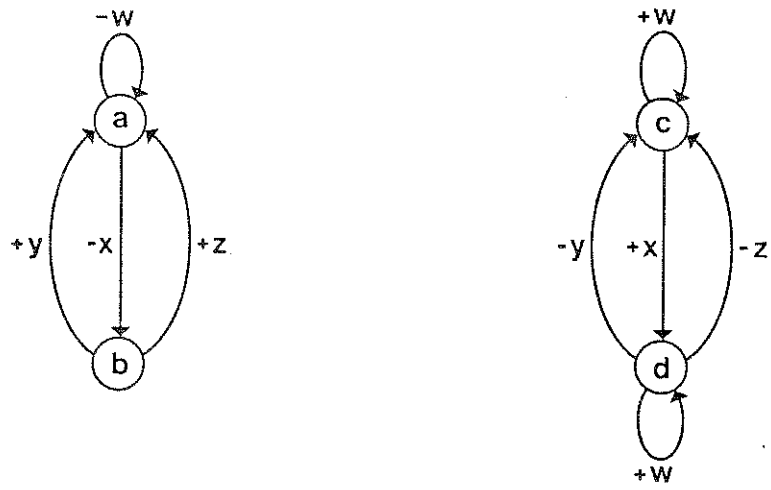

-w
a
+y
-x
+z
b
+w
c
-y
+x
-z
d
+w

fig. C1