Theorems on Computations of Distributed Systems

K. Mani Chandy

Computer Science Department
California Institute of Technology

Caltech-CS-TR-88-6

Theorems on Computations of Distributed Systems

K. Mani Chandy[1]
California Institute of Technology
24 April 1988

Caltech-CS-TR-88-6

Abstract

This paper presents theorems that are helpful in developing algorithms for the detection of stable properties, recording global snapshots and tracing the execution of distributed systems. The theorems are based on a property of channels.

1 Introduction

A distributed system consists of processes and channels. A process can record its computation, i.e., the sequences of actions it performs. A channel is an interface between two processes. Thus, the actions of a channel are actions of the two processes that use the channel. A channel is passive, unlike a process, and it is not possible for a channel to record the sequence of actions performed on it. A system computation, a historical record of what happened in the system, is a sequence of actions of component processes. This paper presents theorems that are helpful in designing algorithms for constructing a system-wide record — the system computation — from local records – the process computations. Such algorithms are useful in detecting stable properties [2] and in debugging distributed systems.

The organization of this paper is as follows. The remainder of this section has two parts: Section 1.1 consists of definitions and Section 1.2 has examples that illustrate the definitions. Section 2 contains an overview of the results and outlines of the proof. Detailed proofs are found in the Appendix. Applications of the theorems are proposed in Section 3. Section 4 is the conclusion.

[1]On leave of absence from the University of Texas at Austin.

1.1 Definitions

1.1.1 Standard Concepts

In this section we present a few well-known concepts such as 'process', 'channel', 'projection', and 'interleaving' from the literature of concurrent systems, see [7] for instance.

A *prefix* of a sequence y is an initial subsequence of y. A sequence is a prefix of itself. We use the notation, $x \preceq y$, to denote that x is a prefix of y. A prefix-closed set of sequences is one in which all prefixes of sequences in the set are also in the set [7].

A *distributed system* is a set of *processes* and a set of *channels*. A *process* is a set of actions (called its alphabet) and a prefix-closed set of sequences of its actions (called its set of computations). Distinct processes have disjoint alphabets. A *channel* is defined in the same way as a process: it is a set of actions and a prefix-closed set of sequences of its actions. A channel is *used by* precisely two processes. The actions of a channel are drawn from the actions of the processes that use the channel, i.e., the alphabet of a channel is a subset of the union of the alphabets of the processes that use the channel.

Letter p represents processes; c represents channels; x, y, z represent sequences. In the remainder of this paper we restrict attention to one distributed system. Universal quantification over all processes of the system is to be assumed where p appears, and universal quantification over all channels of the system is to be assumed where c appears (unless otherwise indicated). Hereafter, the term *sequence* represents a sequence of actions from the (union of the) alphabets of the processes of the system. The symbol *null* represents the empty sequence. Define a binary relation *uses* between a process and a channel as follows: p *uses* c holds if and only if p is one of the two processes that uses channel c.

Associated with each process p is a boolean function F_p on sequences where $F_p(x)$ holds if and only if x is a computation of p. Associated with each channel c is a function $F_c(x)$ with the analogous meaning. Since the sets of process and channel computations are prefix closed:

$$F_p(y) \wedge (x \preceq y) \Rightarrow F_p(x) \tag{1}$$

$$F_c(y) \wedge (x \preceq y) \Rightarrow F_c(x) \tag{2}$$

Let D be a set of processes. We define the *projection* of a sequence on a set of processes as follows: The projection of x on D is the sequence obtained from x by deleting all actions in x that are not actions of processes in D. We use the notation $G_D(x)$ to denote the projection of x on D. We shall use a short-form for the special case where D is a singleton set: we write G_p instead of $G_{\{p\}}$. The notation is extended to channels in the obvious way: $G_c(x)$ is the sequence obtained from x by deleting all actions of x that are not actions of c.

A *system computation* is defined as a sequence x such that the projection of x on p is a computation of p, all p, and the projection of x on c is a computation of c, all c. Define a boolean function H on sequences where $H(x)$ holds if and only if x is a system computation.

Therefore:

$$H(x) = [\forall p : F_p(G_p(x))] \wedge [\forall c : F_c(G_c(x))] \tag{3}$$

We shall use letters U, V, W for system computations. Let x_p be a sequence of actions of p. Define a binary relation $\otimes$ (which is read 'interleaves') between a sequence y and a set of sequences, $\{x_p | p \in D\}$, as follows:

$$y \otimes \{x_p | p \in D\} = [\forall p : (p \in D \Rightarrow G_p(y) = x_p) \wedge (p \notin D \Rightarrow G_p(y) = null)] \tag{4}$$

The elements of y (where $y \otimes \{x_p | p \in D\}$) are the elements of the x_p, all p, and the ordering among elements of each of the x_p is preserved in y.

The notation $x; y$ represents the sequence obtained by catenating x and y. The concepts discussed so far are from the literature of concurrent systems. Next, we introduce four concepts that are helpful in defining our problem and solution.

1.1.2 New Concepts

Let q and r be the processes that use a channel c, and let x_q and x_r be sequences of actions from q and r respectively. Sequences x_q and x_r are said *to match with respect to* c if and only if there exists an interleaving of $\{x_q, x_r\}$ which is a computation of c. Define binary relation $\overset{c}{\approx}$ (which is read ' c-matches') between x_q and x_r as: $x_q \overset{c}{\approx} x_r$ if and only if x_q, x_r match with respect to c, i.e.,

$$(x_q \overset{c}{\approx} x_r) \;=\; [\exists z : (z \otimes \{x_q, x_r\}) \wedge F_c(G_c(z))] \tag{5}$$

The notation $x; y$ represents the sequence obtained by catenating x and y. Define a partial ordering, $\sqsubseteq$ (which is read as 'slice'), between sequences as:

$$(x \sqsubseteq y) \;=\; [\forall p : G_p(x) \preceq G_p(y)] \tag{6}$$

Define an equivalence relation, $\sim$ (which is read as 'like'), between sequences as:

$$(x \sim y) \;=\; [\forall p : G_p(x) = G_p(y)] \tag{7}$$

Consider x, y such that $x \sqsubseteq y$. We define two operators $-$ and $\uparrow$ (read as 'minus' and 'takes' respectively) between sequences x and y as follows. For each p, let n_p and m_p be the lengths of $G_p(x)$ and $G_p(y)$ respectively. Because $x \sqsubseteq y$, see (6), $G_p(x) \preceq G_p(y)$; hence $n_p \leq m_p$ and $G_p(x)$ is the sequence consisting of the first n_p actions of p in y. Define $y - x$ as the sequence obtained from y by deleting the *first* n_p actions of p from y, all p. Define $x \uparrow y$ as the sequence obtained from y by deleting the *last* $(m_p - n_p)$ actions of p from y, all p.

Note: We adopt the convention that minus and takes have precedence over catenation, and therefore $x; z - y = x; (z - y)$ and $x; z \uparrow y = x; (z \uparrow y)$.

For each process q we define a function, cut_q on sequences as follows: $cut_q(x)$ is the smallest prefix of x containing all the actions from q in x i.e

$$[\forall y : (y \preceq x) \wedge (G_q(y) = G_q(x)) \;\Rightarrow\; cut_q(x) \preceq y] \wedge$$
$$G_q(cut_q(x)) = G_q(x) \;\wedge\; (cut_q(x) \preceq x) \tag{8}$$

(Note: The last conjunct in (8), $(cut_q(x) \preceq x)$, follows from the other two conjuncts in (8); we retain it for convenience.)

The Channel Property We shall restrict attention to channels that satisfy the following property: Let x, y be sequences of actions of a channel c, and let $x \sqsubseteq y$. If x and y are computations of c then so is $x; y - x$, i.e.,

$$(x \sqsubseteq y) \wedge F_c(x) \wedge F_c(y) \Rightarrow F_c(x; y - x) \tag{9}$$

The results of this paper are based on the channel property.

1.2 Examples

The examples in this section are informal; we use terms such as 'message', 'send', 'receive', 'buffer', 'read' and 'write' without defining them.

Note: A sequence is enclosed by '$\ll$' and '$\gg$'.

1.2.1 Example 1

This example illustrates many of the functions and operators introduced in Section 1.1. Let $a0, a1, a2$ be actions of a process r; let $b0, b1$ be actions of a process q. Let x, y be the following sequences:

$$x = \ll a0, a1, b0, a0 \gg$$

$$y = \ll a0, b0, a1, b0, a0, b1, a2 \gg$$

Then:

$$G_r(x) = \ll a0, a1, a0 \gg$$
$$G_r(y) = \ll a0, a1, a0, a2 \gg$$
$$G_q(x) = \ll b0 \gg$$
$$G_q(y) = \ll b0, b0, b1 \gg$$
$$G_q(x) \preceq G_q(y)$$
$$G_r(x) \preceq G_r(y)$$

$$x \sqsubseteq y$$

$$x \not\sim y$$

$$x \otimes \{\ll a0, a1, a0 \gg, \ll b0 \gg\}$$

$$x \uparrow y = \ll a0, b0, a1, a0 \gg$$

$$y - x = \ll b0, b1, a2 \gg$$

$$x; y - x = \ll a0, a1, b0, a0, b0, b1, a2 \gg$$

$$(x \uparrow y); (y - x) = \ll a0, b0, a1, a0, b0, b1, a2 \gg$$

$$y \neq x; y - x$$

$$y \sim x; y - x$$

$$y \neq (x \uparrow y); (y - x)$$

$$y \sim (x \uparrow y); (y - x)$$

$$cut_q(x) = \ll a0, a1, b0 \gg$$

$$cut_q(y) = \ll a0, b0, a1, b0, a0, b1 \gg$$

$$cut_r(x) = x$$

1.2.2 Example 2

Let c be a first-come-first-served bounded buffer with a buffer capacity of 2. The buffer is empty initially. Let $send(i)$, $receive(i)$ be the actions that send value i and receive value i (respectively) on the channel. Let q and r be the processes that send and receive (respectively) on channel c. Let x and y be computations of c where:

$$x = \ll send(0), receive(0) \gg$$

$$y = \ll send(0), send(1), receive(0) \gg$$

Then:

$$G_q(x) = \ll send(0) \gg \wedge G_q(y) = \ll send(0), send(1) \gg$$

$$G_r(x) = \ll receive(0) \gg \wedge G_r(y) = \ll receive(0) \gg$$

$$x \sqsubseteq y$$

$$y - x = \ll send(1) \gg$$

$$x; y - x = \ll send(0), receive(0), send(1) \gg$$

The channel property requires that $x; y - x$ be a computation of c (and indeed it is).

1.2.3 Example 3

Let c be a variable shared by processes p and q, where p and q can read and write c in any order. We show that c does not satisfy the channel property. Let:

$$x = \ll p \textit{ writes } 0,\ q \textit{ writes } 1 \gg$$

$$y = \ll q \textit{ writes } 1,\ p \textit{ writes } 0,\ q \textit{ reads } 0 \gg$$

Then:

$$x \sqsubseteq y$$

$$y - x = \ll q \textit{ reads } 0 \gg$$

$$x; y - x = \ll p \textit{ writes } 0,\ q \textit{ writes } 1,\ q \textit{ reads } 0 \gg$$

Though x and y are computations of c, the sequence $x; y-x$ is not a computation of c because q cannot read 0 from shared variable c immediately after q writes 1 into it. Therefore a shared variable that can be read and written in any order, does not satisfy the channel property.

1.2.4 Example 4

Here we give several instances of channels that satisfy the channel property. Assume that the channels are empty initially (though the arguments apply for the general case as well). We shall describe channels in terms of properties of computations of the channels (in addition to the requirement that the set of channel computations be prefix-closed). The descriptions are informal and we leave it to the reader to formalize these descriptions of channels, and to show that they satisfy the channel property.

First-In-First-Out Unbounded Channels A sequence of actions of the channel is a computation of the channel if and only if the sequence of messages received along the channel is a prefix of the sequence of messages sent along the channel.

First-In-First-Out Bounded Channels A sequence of actions of the channel is a computation of the channel if and only if the sequence of messages received along the channel is a prefix of the sequence of messages sent along the channel, and the number of messages sent along the channel exceeds the number of messages received along the channel by at most the buffer size.

Lossy Channels in which Message-Order can be Permuted A sequence of actions of the channel is a computation of the channel if and only if the bag of messages received along the channel is included in the bag of messages sent along the channel.

Shared Variables in which Processes Read and Write Alternately Let q and r be the processes that use channel c. The computation is a repetition of the following cycle: q writes a value, say m, into variable c, then r reads c and gets value m, then r writes a value, say n, into c and then q reads n from c.

2 Theory

This section gives the theorems with outlines of proofs. Detailed proofs are found in the Appendix. Most of the steps in the proofs given in the Appendix consist of the substitution of one term in an expression by another equal term; therefore these proofs are easily verifiable.

2.1 Inheritance of The Channel Property

Theorem 1 *If U and V are system computations and $U \sqsubseteq V$ then $U;V-U$ is also a system computation.*

$$H(U) \wedge H(V) \wedge (U \sqsubseteq V) \;\Rightarrow\; H(U;V-U)$$

Proof: The projection of V on p is a computation of p because V is a system computation. The projection of $U;V-U$ on p is a computation of p because it is the same as the projection of V on p.

The projections of U and V on c are computations of c because U and V are system computations. Hence, the projection of $U;V-U$ on c is a computation of c from the channel property.

2.2 Condition for $w \uparrow V$ to be a System Computation

Theorem 2 *Let V be a system computation, and let w be a sequence such that $w \sqsubseteq V$. Sequence $w \uparrow V$ is a system computation if and only if, for all channels c, the projections of w on the the processes that use c match with respect to c, i.e., for all w, V, such that $H(V)$ and $w \sqsubseteq V$:*

$$[\forall c, q, r : (q \textit{ uses } c) \wedge (r \textit{ uses } c) \Rightarrow G_q(w) \overset{c}{\approx} G_r(w)] = H(w \uparrow V)$$

Proof: We shall prove that if for all channels c, the projections of w on the processes that use c match with respect to c, then $(w \uparrow V)$ is a system computation. We leave to the reader the (trivial) proof that if $(w \uparrow V)$ is a system computation then the projections of w on the processes that use c match with respect to c.

Assume that for all channels c, the projections of w on the processes that use c match with respect to c. Our proof obligation is to show that the projection of $w \uparrow V$ on p and c are computations of p and c respectively.

The projection of $w \uparrow V$ on p is the same as the projection of w on p, which is a prefix of the projection of V on p. Since V is a system computation, the projection of V on p is a computation of p, and by prefix-closure the projection of $w \uparrow V$ on p is also a computation of p.

Let z be the projection of $w \uparrow V$ on c. Our remaining proof obligation is to show that z is a computation of c. Let q and r be the processes that use c. Assume without loss of generality that $cut_q(z) \preceq cut_r(z)$, i.e., there are no actions of q in z, or the last action from q appears earlier in z than the last action from r. Let $x = cut_q(z)$. There is an interleaving t of $G_q(w)$ and $G_r(w)$ such that the projection of t on c is a computation of c, because $G_q(w)$ and $G_r(w)$ match with respect to c. Let y be the projection of t on c. The proof that z is a computation of c has three parts:

1. Prove $y \sim z$
2. Prove $z = x; y - x$
3. Prove $F_c(x; y - x)$, i.e., prove that $x; y - x$ is a computation of c.

Part 1 is used in proving Part 2. The result follows from Parts 2 and 3.

Part 1: The projections of t and w on q (and on r) are identical because t is an interleaving of the projections of w on q and r. The projections of $w \uparrow V$ on any process p is the same as the projection of w on p. Hence the projections of t and $w \uparrow V$ on q (and on r) are identical. Since y and z are projections of t and $w \uparrow V$ on c, the projections of y and z on q (and on r) are identical, and also y and z consist of actions of only q and r. Hence $y \sim z$.

Part 2: Since $x = cut_q(z)$, x is a prefix of z. Hence $z = x; z - x$. Next we shall show that $z - x = y - x$ and hence $x; z - x = x; y - x$, and this completes the proof of Part 2. Sequences x, y and z are sequences of actions of channel c because y and z are projections on c, and x is a prefix of z; hence they contain actions of only q and r. Because $x = cut_q(z)$, the projections of x and z on q are the same. From Part 1, the projections of y and z on any q are the same. Therefore, x, y and z have identical projections on q. Hence $z - x$ and $y - x$ contain actions only from r. From Part 1, the projections of x and y on r are identical, and hence $z - x = y - x$.

Part 3: We shall first show that x is a computation of c, by showing that x is a prefix of the projection of system computation V on c, and then employing prefix-closure. From the defintions of projection and $\uparrow$, $z = G_c(w) \uparrow G_c(V)$. Sequence z is obtained from $G_c(V)$ by deleting elements where: If an action of a process p is deleted from $G_c(V)$, then all later actions of p in $G_c(V)$ are also deleted from $G_c(V)$. From our assumption, the last action of q appears earlier than the last action of r in z. Therefore, no action of r that appears earlier than

the last action of q in z is deleted to get z from $G_c(V)$. Therefore all elements of $G_c(V)$ that appear before the last action of q in z are retained in z. Hence $cut_q(z)$ is a prefix of $G_c(V)$, and since $x = cut_q(z)$, x is a prefix of $G_c(V)$.

Since x is a prefix of z: $x \sqsubseteq z$. Since $y \sim z$ (from Part 1), $x \sqsubseteq y$. From the definition of y, it is a computation of c. From the Channel Property, $x; y - x$ is a computation of c.

3 Applications of the Theorems

3.1 States

A state of a system is its memory; system states satisfy the following property: the set of future behaviors of a system given its current state is independent of its past. In this section we shall study states of distributed systems in which all channels satisfy the channel property.

3.1.1 A Graph Representation of Sequences

We define a labeled directed graph STD (for State-Transition-Diagram) as follows. Each vertex of the graph is a set of process computations, one computation from each component process; thus a vertex is:

$$\{x_p | p \textit{ is a process} \wedge F_p(x_p)\}$$

Define the initial vertex as follows. The initial vertex is:

$$\{x_p | p \textit{ is a process} \wedge (x_p = null)\}$$

For brevity we shall denote $\{x_p | p \textit{ is a process}\}$ by $\{x_p\}$. An edge is labeled with an action. There is an edge labeled e from $\{x_p\}$ to $\{y_p\}$ if and only if there exists a q such that:

$$(y_q = x_q; e) \wedge [\forall p : (p \neq q) \Rightarrow (y_p = x_p)]$$

A path is uniquely represented by the vertex from which the path originates and the sequence of edge-labels along the path. Therefore, we use the same notation for paths as for sequences of actions. For example, a path $\ll e[1], e[2], .. \gg$ from the initial vertex is the path from the initial vertex to vertex $\{G_p(\ll e[1] \gg)\}$ to vertex $\{G_p(\ll e[1], e[2] \gg)\}$, .., and so on.

Hereafter, restrict attention to sequences that are interleavings of process computations. Thus, for every sequence x we assume that $F_p(G_p(x))$ holds, for all p. It is not necessary, however, that $F_c(G_c(x))$ holds, and therefore x need not be a system computation.

Observation 1 *A sequence x is a path from the initial vertex to vertex $\{G_p(x)\}$.*

Observation 2 *For all sequences x, y: $x \sim y$ if and only if x, y are paths from the intial vertex to the same vertex.*

Observation 3 *For all sequences x, y: $x \sqsubseteq y$ if and only if there is a path from $\{G_p(x)\}$ to $\{G_p(y)\}$.*

We extend the definition of the partial ordering $\sqsubseteq$ to vertices in the obvious way:

$$\{x_p\} \sqsubseteq \{y_p\} = [\forall p : x_p \preceq y_p]$$

Therefore, $\{x_p\} \sqsubseteq \{y_p\}$ if and only if there is a path from vertex $\{x_p\}$ to vertex $\{y_p\}$.

3.1.2 States and System Computations

We define a *feasible vertex* and a *feasible path* as follows: A vertex $\{x_p\}$ is feasible if and only if there exists a system computation which is an interleaving of $\{x_p\}$. A path is feasible if and only if all vertices on the path are feasible.

Theorem 3 *A path from the initial vertex is a system computation if and only if the path is feasible.*

Proof Let y be a feasible path from the initial vertex with k edges, $k >= 0$. We shall show by induction on k that $H(y)$ holds.

Base Case, k=0: For $k = 0$, $y = null$, and $H(null)$ holds.

Induction Step: We show that if all feasible paths, from the initial vertex, with at most k edges are system computations then all feasible paths from the intital vertex with at most $k + 1$ edges are system computations. Let y be a feasible path from the initial vertex with at most $k + 1$ edges, where $k \geq 0$. Let e be the label of the last edge traversed in y, and let $y = x; e$. Then x is a feasible path from the initial vertex with at most k edges. By the induction assumption $H(x)$ holds. Path y goes to vertex $\{G_p(y)\}$. Since y is feasible, $\{G_p(y)\}$ is a feasible vertex. Hence there exists a system computation V which is an interleaving of $\{G_p(y)\}$, i.e., $H(V)$ holds and $G_p(V) = G_p(y)$. Since there is a path from $\{G_p(x)\}$ to $\{G_p(y)\}$, there is a path from $\{G_p(x)\}$ to $\{G_p(V)\}$, and from Observation 3, $x \sqsubseteq V$. Therefore, from Theorem 1, $H(x; V - x)$ holds. But $V - x = e$. Hence $H(y)$ holds.

Our remaining proof obligation is to show that every system computation is a feasible path from the initial vertex. This proof is a straightforward application of the prefix-closed property and is left to the reader.

A corollary of the theorem is: the feasible vertices of graph STD are states of the system. The definition of state is as follows: The future behaviors of a

system given its current state are independent of the past. A future behavior of a distributed system that has executed a system computation V, is a sequence x such that $V; x$ is a system computation. It is possible to determine whether $V; x$ is a system computation given only the vertex $\{G_p(V)\}$ *without being given* V because $V; x$ is a system computation if and only if x is a feasible path from vertex $\{G_p(V)\}$.

The state space, i.e., the graph STD, has interesting properties that can be exploited in developing algorithms. We explore a few of the properties next.

Observation 4 *Let v_0, v_1, v_2, be feasible vertices where $v_1 \sqsubseteq v_2$. Let x be a feasible path from v_0 to v_1, and let y be a feasible path from v_0 to v_2. Then, $y - x$ is a feasible path from v_1 to v_2. Also, $x \uparrow y$ is a feasible path from v_0 to v_1. Therefore $(x \uparrow y); (y - x)$ and $x; (y - x)$ are feasible paths from v_0 to v_2 via v_1.*

The proof follows directly from Theorems 1 and 2.

Observation 5 *For all feasible vertices v_1, v_2: $v_1 \sqsubseteq v_2$ if and only if there is a feasible path from v_1 to v_2.*

The proof follows directly from the previous observation; there are system computations U_1, and U_2 to vertices v_1 and v_2 respectively, and hence $U_2 - U_1$ is a feasible path from v_1 to v_2.

3.2 Determining Whether a Vertex is Feasible

From the definition, a vertex $\{x_p\}$ is feasible if and only if there exists a system computation which is an interleaving of $\{x_p\}$. Therefore one way of determining whether $\{x_p\}$ is feasible is to inspect all interleavings of $\{x_p\}$ to determine if any of them is a system computation. Theorem 2 gives us a more efficient scheme to determine whether $\{x_p\}$ is feasible for the case where $\{x_p\} \sqsubseteq \{y_p\}$ and where $\{y_p\}$ is feasible. All we need to do is to determine, for each channel c, whether the computations of the processes that use c match with respect to c. It is more efficient to determine if *two* process computations match than to generate all interleavings of a large number of process computations.

For the remainder of this section we restrict attention to vertices $\{x_p\}$ where there is a feasible vertex $\{y_p\}$ such that $\{x_p\} \sqsubseteq \{y_p\}$; therefore Theorem 2 is applicable. There are many types of channels for which it is not necessary to generate all interleavings of process computations x_q and x_r to determine if they match with respect to c. Consider the channels in Example 4 from Section 1.

3.2.1 Unbounded FIFO Channels

Let q be the process that sends messages on channel c and let r be the process that receives messages on c. Let n_s and n_r be the number of messages sent and received (respectively) along channel c in x_q and x_r (respectively).

Lemma 1 *Process computations x_q and x_r match with respect to channel c if and only if the number of sends along c in x_q is at least the number of receives along c in x_r:*

$$(x_q \overset{c}{\approx} x_r) = (n_s \geq n_r)$$

Proof Let sy and ry be the sequences of messages sent and received along c in y_q and y_r (respectively). Let sx and rx be the sequences of messages sent and received along c in x_q and x_r (respectively). Since $\{y_p\}$ is feasible $ry \preceq sy$. Since $\{x_p\} \sqsubseteq \{y_p\}$ it follows that $rx \preceq ry$, and $sx \preceq sy$. Therefore both rx and sx are prefixes of sy. Hence rx is a prefix of sx if and only if the length of rx is at most the length of sx.

The lemma tells us $\{x_p\}$ is feasible if and only if the number of messages sent on each channel in $\{x_p\}$ is at least the number of messages received on that channel in $\{x_p\}$.

3.2.2 Bounded First-In-First-Out Channels

For a channel c, let $n_s(c)$ and $n_r(c)$ be the number of messages sent and received (respectively) along c in $\{x_p\}$. Then, $\{x_p\}$ is feasible if and only if:

$$[\forall c : b(c) \geq (n_s(c) - n_r(c)) \geq 0]$$

where $b(c)$ is the buffer size of channel c. The proof for this observation is the same as for the previous case.

3.2.3 Lossy Channels

From the definition of lossy channels, process computations x_q and x_r match with respect to channel c if and only if the bag of messages received along c is included in the bag of messages sent along c. In the case of error-free first-in-first-out channels, we could employ the facts that $\{y_p\}$ is feasible and $\{x_p\} \sqsubseteq \{y_p\}$ to reduce the determination of whether x_q and x_r match with respect to c to a comparision of the numbers of messages sent and received along c. For lossy channels, no such simplification is readily apparent. Therefore, we employ the obvious test: $\{x_p\}$ is feasible if and only if for each channel c, the bag of messages received along c is included in the bag of messages sent along c.

3.2.4 Shared Variables into which Processes Read and Write Alternately

Let q be the process that carries out the first action on the shared variable. Let n_q and n_r be the number of actions of processes q and r (respectively) on the shared variable in x_q and x_r (respectively) . We leave it to the reader to show that $\{x_p\}$ is feasible if and only if:

$$1 \geq (n_q - n_r) \geq 0$$

3.3 Recording Global States

We are given a distributed system in which channels are unbounded and first-in-first-out (and error-free). Next we derive the algorithm for global state recording given in [2]. Solutions to this problem are presented in [3,5,6]. We restrict attention to the recording of one global state. As execution proceeds, each process p records the computation it has carried out so far in a local variable, say x_p. A process p records its computation in x_p precisely once.

3.3.1 Problem Specification

Let U be the system computation at the point at which global state recording is initiated, and let V be the system computation at the point at which it terminates. We require that there exist a system computation that takes the system to state $\{G_p(U)\}$ and later to state $\{x_p\}$, and later to state $\{G_p(V)\}$; in other words, we require that there exists a system computation V' with prefixes U' and X' such that:

$$(V' \sim V) \wedge (U' \sim U) \wedge (\{G_p(X')\} = \{x_p\})$$

3.3.2 Algorithm

The algorithm consists of the following three rules:

1. When a process records its computation it sends a special message called a *marker* along each of its outgoing channels.

2. When a process receives a marker it records its computation if it has not done so.

3. Eventually, every process records its computation.

3.3.3 Proof of Correctness

Consider a channel c and let n_s and n_r be the number of messages sent and received (respectively) along c in x_p. To prove that $\{x_p\}$ is feasible, our only obligation is to show $n_s \geq n_r$. From Rule 1, n_s is the number of messages sent along c before the marker along c. From Rule 2, n_r, the number of messages received along c when the receiver records its state, does not exceed the number of messages sent along c before the marker along c. Therefore $n_s \geq n_r$. Hence $\{x_p\}$ is feasible.

Since (from the definitions of U and V) p records x_p no earlier than U and no later than V:

$$(G_p(U) \preceq x_p) \wedge (x_p \preceq G_p(V))$$

Hence:

$$\{G_p(U)\} \sqsubseteq \{x_p\} \ \wedge \ \{x_p\} \sqsubseteq \{G_p(V)\}$$

From Observation 5, there is a feasible path from the initial vertex to $\{G_p(U)\}$ to $\{x_p\}$ to $\{G_p(V)\}$. The result follows from Theorem 3.

3.4 Detecting Stable Properties

Let S be a predicate on states of the system. Predicate S is defined to be a stable property of the system if and only if, for all states v_1, v_2 where there is an execution that takes the system from v_1 to v_2: if S holds for v_1 then S holds for v_2. Thus once S holds, it continues to hold forever thereafter. In terms of graph STD, if S holds for a feasible vertex v_1, and there is a feasible path from v_1 to v_2, then S holds for v_2.

Observation 6 *For all feasible vertices v_1, v_2 where $v_1 \sqsubseteq v_2$: if S holds for v_1 then S holds for v_2.*

This observation follows directly from the definition of stability and Observation 5.

An important problem in distributed systems is to detect whether a given stable property holds. Algorithms to detect a stable property S have the following specification.

3.4.1 Problem Specification

The algorithm has a boolean variable cl (which stands for *claim*) where cl must not hold before S holds, and cl must hold some time after S holds. Thus, the algorithm is required to have the invariant $cl \Rightarrow S$, and the progress property that if S holds then eventually cl holds as well.

3.4.2 Algorithm

An algorithm for detecting a stable property S is as follows. Initially cl is $false$. While cl remains $false$ each process p records its process computation into a local variable, say x_p, in such a way that $\{x_p\}$ is a feasible vertex (as shown in Section 3 — for instance), and cl is set to *true* if S holds at $\{x_p\}$.

3.4.3 Proof of Correctness

The algorithm follows directly from Observation 6. First consider the proof of the invariant. Assume that cl is *true* when the system computation is V. Let y_p be the value of x_p when cl is set to *true*; therefore S holds for $\{y_p\}$ and $y_p \preceq G_p(V)$. Hence $\{y_p\} \sqsubseteq \{G_p(V)\}$. Therefore, from Observation 6, S holds for $\{G_p(V)\}$.

Next consider the proof of progress. Assume that S holds after a computation V, i.e., S holds at $\{G_p(V)\}$. Since each process p repeatedly records its computation while cl remains $false$, either cl holds at V or each p records its

computation at or after V. Therefore, either cl holds at V or a feasible vertex $\{x_p\}$ is recorded where for all p : $G_p(V) \preceq x_p$. Hence $\{G_p(V)\} \sqsubseteq \{x_p\}$. From Observation 6, since S holds at $\{G_p(V)\}$, it follows that S holds at $\{x_p\}$ as well. Hence cl becomes *true*.

3.5 Passive Termination Detection

3.5.1 Problem Specification

We are given a distributed system in which channels are first- in-first-out bounded buffers. Consider the following problem: A process is either *idle* or *active*. An idle process remains idle at least until it receives a message. An idle process sends no messages. The problem is to detect the condition: All processes are idle and all channels are empty. This condition is stable. Next we derive the solution to this problem given in [1,4,8].

Assume that initially all channels are empty and all processes are active. (The extension of the algorithm to the general case is left to the reader.) Also assume that every process uses at least one channel.

3.5.2 Algorithm

One of the processes (or possibly an external process) is designated the *observer*. The algorithm consists of the following two rules.

> *Action taken by each process:* When an active process becomes idle it sends a special message called a status *message* to the observer. A status message sent by a process p contains the following information: For each outgoing channel c of p , the number $n_s(c)$ of messages sent by p along c, and for each incoming channel c of p, the number $n_r(c)$ of messages received by p along c. (Status messages are excluded from these counts.)
>
> *Action taken by the observer:* The observer has local integer variables $m_s(c)$ and $m_r(c)$ for each channel c in the system. When the observer receives a status message from a process p, the observer sets $m_s(c) := n_s(c)$ for each outgoing channel c of p, and sets $m_r(c) := n_r(c)$ for each incoming channel c of p. All channels are empty and all processes are idle if for all channels c: $m_s(c) = m_r(c)$.

To guarantee that $m_s(c) \neq m_r(c)$ unless status messages are sent by both the sender and receiver along c, we choose initial values of $m_s(c)$ and $m_r(c)$ such that $m_s(c) \neq m_r(c)$ and $m_s(c) < 0$, and $m_r(c) < 0$. For example, initially, for all c:

$$m_r(c) = -2, m_s(c) = -1$$

3.5.3 Proof of Correctness

Let us first prove the invariant, if $[\forall c : m_s(c) = m_r(c)]$ then all channels are empty and all processes are idle. If the observer has not received any messages from a process p, then since there is at least on channel incident on p, there is a channel c such that $m_s(c) \neq m_r(c)$ (from the initial conditions). Therefore assume that the observer has received at least one message from each process. Let L_p be the last message received by the observer from p. Let x_p be the computation of p at the point at which L_p is sent. Then after x_p is executed p is idle, and the number of messages p has sent along each of its outgoing chanels c is $m_s(c)$, and the number of messages p has received along each of its incoming channels c is $m_r(c)$. From Section 3.2, vertex $\{x_p\}$ is feasible if and only if:

$$[\forall c : b(c) \geq (m_s(c) - m_r(c)) \geq 0]$$

where $b(c)$ is the buffer size of channel c. Therefore, $\{x_p\}$ is feasible if $m_s(c) = m_r(c)$ for all channels c; furthermore, all channels are empty and all processes are idle in state $\{x_p\}$. Let $\{y_p\}$ be the current state. Since $\{x_p\} \sqsubseteq \{y_p\}$, from Observation 6, stable properties that hold in $\{x_p\}$ also hold in the current state $\{y_p\}$. Therefore all processes are idle and all channels are empty in the current state.

The proof of progress (i.e., the proof that if all processes are idle and all channels are empty then eventually $m_s(c) = m_r(c)$ for all channels c) is left to the reader.

3.6 Debugging

One way of debugging a distributed system is to record each process computation locally, and to execute the system until it terminates or an error is detected. If an error occurs, the programmer wants to know the recent history of the system computation – say the last N steps of the computation, for some N. If the recent history is insufficient for identifying the error, the programmer may want to have earlier history – say the previous N steps. The problem is to reconstruct the recent histories of possible system computations from the recorded process computations.

Let x_p be the process computation recorded by p. Vertex $\{x_p\}$ is feasible because it occurred during a system computation. We wish to determine all feasible vertices that can lead to this vertex in k transitions, for increasing values of k. The problem reduces to determining all feasible vertices that can lead to the given vertex in a single transition (and k iterations of this step gives all feasible vertices k transitions away). Let e_p be the last element of x_p. A vertex $\{y_p\}$ that leads to $\{x_p\}$ in a single transition differs from $\{x_p\}$ by a single element, i.e., there is precisely one process, say q, for which y_q differs from x_q, and $y_q; e_q = x_q$. For each such vertex $\{y_p\}$, we determine whether the vertex is feasible using the methods given earlier. For example, for a system in which

channels are first-in-first-out bounded buffers, we merely determine whether, for each channel c, the number of sends along c in $\{y_p\}$ exceeds the number of receives along c in $\{y_p\}$ by at most the buffer size, and by at least zero. This simple test leads to efficient algorithms for determining the feasible vertices that lead to a given vertex.

4 Conclusion

A system computation is an interleaving of process computations such that the projection of the interleaving on each channel is a computation of that channel. If all channels satisfy the channel property, an interleaving of process computations is a system computation if and only if for all channels, the process computations of the processes that use the channel, match with respect to the channel. Thus, the test to determine whether an interleaving of process computations is a channel computation reduces to a set of *pair-wise* tests of process computations.

The computations of a distributed system, in which all channels satisfy the channel property, have several interesting properties. The channel property leads to straightforward algorithms for solving detection problems.

5 Acknowledgement

The author is grateful to J. Misra for his suggestions and constructive criticism, and to Jan van de Snepscheut for his advice and careful study of the detailed proofs.

Bibliography

[1] Chandy, K. M.[1987] 'A Theorem on Termination of Distributed Systems', TR-87-09, March 1987, Dept. of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712-1188.

[2] Chandy, K. M., and L. Lamport [1985]. 'Distributed Snapshots: Determining Global States of Distributed Systems,' ACM TOCS, 3:1, February 1985, pp. 63-75.

[3] Chandy, K. M. and J. Misra [1988] *Parallel Program Design: A Foundation*, Addison-Wesley, Reading, Massachusetts, 1988.

[4] Chandy, K. M. and J. Misra [1988] 'On Proofs of Distributed Algorithms with Application to the Problem of Termination Detection', submitted to Distributed Computing [1987].

[5] Dijkstra, E. W. [1985]. 'The Distributed Snapshot of K. M. Chandy and L. Lamport,' in Control Flow and Data Flow, ed. M. Broy, Berlin: Springer-Verlag, 1985, pp. 513-517.

[6] Fischer, M. J., N. D. Griffeth, and N. A. Lynch [1982]. 'Global States of a Distributed System,' IEEE Transactions on Software Engineering, SE-8:3, M May 1982, pp. 198-202.

[7] Hoare, C. A. R. [1984]. *Communicating Sequential Processes*, London: Prentice-Hall International, 1984.

[8] Raynal, M., J.-M. Helary, C. Jard, and N. Plouzeau [1987]. 'Detection of Stable Properties in Distributed Applications,' in Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, 1987, pp. 125-136.

A APPENDIX: Detailed Proofs

It is convenient to separate the theory into two parts; the first part deals with sequences and properties of sequences, and the second part develops theorems about computations without employing the fact that computations are sequences. The goal in the second part is to present proofs in which the proof steps consist of substituting a term in an expression by an equal term. We employ a surprisingly small number of results in the predicate calculus (in addition to the substitution of a term by another equal one).

The properties we use about sequences and operations on them are simple. We present them without formal proof.

A.1 Part 1: Properties of Operations on Sequences

Relationship between Prefix, Catenation and null

$$x \preceq y = [\exists z : (x; z) = y] \tag{10}$$

$$(x; y = null) = ((x = null) \wedge (y = null)) \tag{11}$$

$$[(x; y = x) = (y = null)] \wedge [(x; y = y) = (x = null)] \tag{12}$$

$$(x \preceq z) \wedge (y \preceq z) \Rightarrow (x \preceq y) \vee (y \preceq x) \tag{13}$$

Distributive Properties of Projection

$$G_D(x; y) = G_D(x); G_D(y) \tag{14}$$

$$(x \sqsubseteq y) \Rightarrow (G_D(x) \sqsubseteq G_D(y)) \tag{15}$$

$$(x \sqsubseteq y) \Rightarrow (G_D(y - x) = G_D(y) - G_D(x)) \tag{16}$$

$$(x \sqsubseteq y) \Rightarrow (G_D(x \uparrow y) = G_D(x) \uparrow G_D(y)) \tag{17}$$

Note The distributive properties also hold with G_D replaced by G_c.

Other Properties of Projection

$$G_c(G_p(x)) = G_p(G_c(x)) \tag{18}$$

$$(q \textit{ uses } c) \wedge (r \textit{ uses } c) \wedge (q \neq r) \wedge (p \neq q) \wedge (p \neq r) \Rightarrow G_p(G_c(x)) = null \tag{19}$$

$$(x = G_r(x)) = [\forall p : (p \neq r) \Rightarrow (G_p(x) = null)] \tag{20}$$

Properties of Minus and Takes

$$(x; y) - x = y \tag{21}$$

$$(x \uparrow (x; y)) = x \tag{22}$$

$$(x \sim y) \wedge (x \sqsubseteq z) \wedge (y \sqsubseteq z) \Rightarrow (z - x = z - y) \wedge (x \uparrow z = y \uparrow z) \tag{23}$$

Property of Cut Let c be a channel and let q, r be processes such that q *uses* c, and r *uses* c, and $q \neq r$. Let x and y be sequences of actions of c such that $x \sqsubseteq y$, and let $z = (x \uparrow y)$.

$$cut_q(z) \preceq cut_r(z) \Rightarrow cut_q(z) \preceq y \tag{24}$$

Justification If $cut_q(z) = null$, the result follows trivially; hence restrict attention to the case where $cut_q(z) \neq null$. Label the n-th element in y with n, for $n > 0$; retain these labels in z, and let $L[i]$ be the label of the i-th element in z, for $i > 0$. Sequences x, y, z consist of actions of only q and r. Let the last action of p in z be the m_p-th element of z for $p = q, r$. We shall show that $cut_q(z)$ is the same as the sequence consisting of the first $L[m_q]$ elements of y.

Because $cut_q(z) \preceq cut_r(z)$ it follows that $m_q \leq m_r$. Since z is obtained from y by deleting elements (without permuting the retained elements), the labels in z are in increasing order; hence $L[m_q] \leq L[m_r]$. All actions of p in y up to the $L[m_p]$-th element of y, are retained in z, for $p = q, r$. Hence all actions (of both q and r) in y up to the $L[m_q]$-th element of y are retained in z, and hence are also retained in $cut_q(z)$. The result follows since the order of elements in y is retained in z, and hence in $cut_q(z)$ as well.

A.2 Part 2 : Proofs Derived from Formualae of Part 1

Next, we develop theorems based only on the formulae given earlier. Most steps of the proof consist merely of replacing one term in an expression by an equal term. Verifying the proofs is largely an issue of pattern matching and string substitution. Reducing a proof to substitution of a term by an equal term requires that no steps be left out, and therefore these proofs appear longer than the informal proofs (in which several obvious steps are left out).

Notes About Proof Notation Let PR be a predicate. The predicate obtained by replacing all instances of x in PR by y is written: $PR|x := y$. In the proof, $(*)$ denotes the previous equation, and $(* - i)$ denotes the $(i+1)$ -th previous equation. For example, $(* - 1|x := y)$ denotes the last but one equation with all instances of x replaced by y. The antecedent of implication (n) is denoted by $ante(n)$, and its consequent is denoted by $cons(n)$.

The format of the proof is as follows: the left side of the page contains the steps of the proofs, and the right side has the explanation for each step.

The first lemma consists of several (perhaps intuitively obvious formulae) that we need to prove our theorems.

Lemma 2

$$(x \preceq y) \Rightarrow (G_D(x) \preceq G_D(y)) \wedge (G_c(x) \preceq G_c(y)) \tag{25}$$

$$(x \preceq y) \Rightarrow ((x; y - x) = y) \wedge (x \uparrow y = x) \tag{26}$$

$$(x \sqsubseteq y) \Rightarrow (G_c(x; y - x) = G_c(x); G_c(y) - G_c(x)) \tag{27}$$

$$(x \sqsubseteq y) \Rightarrow G_p(x; y - x) = G_p(y) \tag{28}$$

$$(x \sqsubseteq y) \Rightarrow (G_p(x \uparrow y) = G_p(x)) \tag{29}$$

$$x - x = null \tag{30}$$

$$(x \preceq null) = (x = null) \tag{31}$$

Let $x \sqsubseteq y$, let c be a channel and let q, r be processes such that q uses c, and r uses c, and $q \neq r$. Let $z = G_c(x \uparrow y)$.

$$cut_q(z) \preceq cut_r(z) \Rightarrow cut_q(z) \preceq G_c(y) \tag{32}$$

proof of (25)

$(x \preceq y)$	ante(25)
Consider a z such that:	
$x; z = y$	such a z exists because, from (*), $x \preceq y$, see (10)
$G_D(x; z) = G_D(x); G_D(z)$	$(14 \mid y := z)$
$G_D(x) \preceq G_D(x); G_D(z)$	$(10 \mid x := G_D(x) y := G_D(x); G_D(z),\ z := G_D(z))$
cons(25)	$(*)$,$(* - 1)$, $(* - 2)$

proof of (26)

$(x \preceq y)$	ante(26)
Consider a z such that:	
$x; z = y$	such a z exists because, from (*), $x \preceq y$, see (10)
$(x; z) - x = z$	$(21 \mid y := z)$
$x; ((x; z) - x) = x; z$	append x to the head of both sides of (*)
$x; y - x = y$	$x; z = y$ from (*-2), $(* \mid (x; z) := y)$
$x \uparrow y = x$	similar argument using (22)

proof of (27)

$(14 \mid D := c, y := y - x)$ and then $(16 \mid D := c)$

proof of (28)

$G_p(x) \preceq G_p(y)$	ante(28), (6)
$G_p(x); G_p(y) - G_p(x) = G_p(y)$	(*), (26 $\mid x := G_p(x), y := G_p(y)$)
$G_p(x; y - x) = G_p(x); G_p(y) - G_p(x)$	same proof as for (27)
$G_p(x; y - x) = G_p(y)$	(*),(*-1)

proof of (29)

$x \sqsubseteq y$	ante(29)
$G_p(x \uparrow y) = G_p(x) \uparrow G_p(y)$	(*), $(17 \mid D := p)$
$G_p(x) \preceq G_p(y)$	(*-1), (6)
$G_p(x) \uparrow G_p(y) = G_p(x)$	(*),$(26 \mid x := G_p(x), y := G_p(y))$
$G_p(x \uparrow y) = G_p(x)$	(*),(*-2)

proof of (30)

$x; null = x$	(12)
$(x; null) - x = null$	(21 $\mid y := null$)
$x - x = null$	(*),(*-1)

proof of (31)

($10 \mid y := null$), ($11 \mid y := z$)

proof of (32)

$G_c(x) \sqsubseteq G_c(y))$	given $x \sqsubseteq y$, (15)
$z = G_c(x \uparrow y)$	given
$z = G_c(x) \uparrow G_c(y)$	(*),(17|$G_D := G_c$)
(32)	(24|$x := G_c(x), y := G_c(y)$)

Theorem 1 *If U and V are system computations and $U \sqsubseteq V$ then $U;V-U$ is also a system computation.*

$$H(U) \wedge H(V) \wedge (U \sqsubseteq V) \Rightarrow H(U;V-U) \tag{33}$$

Proof

$G_p(U;V-U) = G_p(V)$	$U \sqsubseteq V$ from ante(33), (28|$x := U, y := V$)
$F_p(G_p(V))$	$H(V)$ from ante(33) ,(3|$x := V$)
$F_p(G_p(U;V-U))$	(*),(*-1)
$G_c(U) \sqsubseteq G_c(V)$	$U \sqsubseteq V$ from ante(33) , (15|$G_D := G_c, x := U, y := V$)
$F_c(G_c(U))$	$H(U)$ from ante(33) ,(3|$x := U$)
$F_c(G_c(V))$	$H(V)$ from ante(33) ,(3|$x := V$)
$F_c(G_c(U);G_c(V)-G_c(U))$	(*-2),(*-1),(*), (9|$x := G_c(U), y := G_c(V)$
$F_c(G_c(U;V-U))$	(*),(27|$x := U, y := V$)
$F_p(G_p(U;V-U))$	proved earlier
$H(U;V-U)$	(*−1),(*),(3|$x := U;V-U$)

Theorem 2 *Let V be a system computation, and let w be a sequence such that $w \sqsubseteq V$. If for all channels c, the projections of w on the the processes that use c match with respect to c, then $w \uparrow V$ is a system computation.*

For all w, V, such that $H(V)$ and $w \sqsubseteq V$:

$$[\forall c, q, r : (q \text{ uses } c) \wedge (r \text{ uses } c) \Rightarrow G_q(w) \overset{c}{\approx} G_r(w)] \Rightarrow H(w \uparrow V) \tag{34}$$

Proof

$w \sqsubseteq V$	given
$G_p(w \uparrow V) = G_p(w)$	(*), (29|$x := w, y := V$)
$G_p(w) \preceq G_p(V)$	(*-1), (6|$x := w, y := V$)
$G_p(w \uparrow V) \preceq G_p(V)$	(*),(*-1)
$H(V)$	given
$F_p(G_p(V))$	(*),(3|$x := V$)
$F_p(G_p(w \uparrow V))$	(*),(*-2), (1|$x := G_p(w \uparrow V), y := G_p(V)$)

Our remaining proof obligation is to show $F_c(G_c(w \uparrow V))$. In the remainder of the proof consider a specific channel c, and let q and r be the processes that use c, i.e., restrict attention to q, r, where q *uses* c and r *uses* c and $q \neq r$.

$G_q(w) \overset{c}{\approx} G_r(w)$	ante(34)
Consider a sequence t such that:	
$(t \otimes \{G_q(w), G_r(w)\}) \wedge F_c(G_c(t))$	such a t exists from (*) and $(5\|z := t, x_q := G_q(w), x_r := G_r(w))$

Let $z = G_c(w \uparrow V)$, $y = G_c(t)$; hereafter, y and z are as defined in these equations, and we no longer quantify over them. The proof has three parts:

PART 1: proof of $y \sim z$

$G_q(t) = G_q(w)$	from the definition of t, see (*), and $(4\|y := t, D := \{q, r\}, x_p := G_p(w))$
$G_q(w \uparrow V) = G_q(w)$	given $w \sqsubseteq V$,$(29\|p := q, x := w, y := V)$
$G_q(t) = G_q(w \uparrow V)$	(*),(*-1)
$G_c(G_q(t)) = G_c(G_q(w \uparrow V))$	apply G_c to both sides of (*)
$G_q(G_c(t)) = G_q(G_c(w \uparrow V))$	(*),(18|$p := q$, and first $x := t$, and then $x := w \uparrow V$)
$G_q(y) = G_q(z)$	(*), definition of y, z
$G_r(y) = G_r(z)$	(*), by symmetry of q and r
$(p \neq q) \wedge (p \neq r) \Rightarrow (G_p(G_c(w \uparrow V)) = null)$	$(19\|x := w \uparrow V)$
$(p \neq q) \wedge (p \neq r) \Rightarrow (G_p(z) = null)$	(*), definition of z
$(p \neq q) \wedge (p \neq r) \Rightarrow (G_p(y) = null)$	repeat (*-1) with $w \uparrow V := t$, and use definition of y
$(p \neq q) \wedge (p \neq r) \Rightarrow (G_p(y) = G_p(z))$	(*), (*-1)
$[\forall p : G_p(y) = G_p(z)]$	(*), $G_r(y) = G_r(z)$ see (*-4), $G_q(y) = G_q(z)$ see (*-5)
$y \sim z$	(*),$(7\|x := y, y := z)$

This completes the proof of Part 1.

$(cut_q(z) \preceq z) \wedge (cut_r(z) \preceq z)$	$(8\|x := z$ and then $q := r$, $x := z)$
$(cut_q(z) \preceq cut_r(z)) \vee (cut_r(z) \preceq cut_q(z))$	(*),$(13\ \|x := cut_q(z), y := cut_r(z))$

Consider the case where $cut_q(z) \preceq cut_r(z)$; a similar argument holds for the case where $cut_r(z) \preceq cut_q(z)$. Let $x = cut_q(z)$. Hereafter x refers to the specific sequence $cut_q(z)$, and we do not quantify over x. In the second part of the proof we show that $z = x; y - x$, and in the last part we show that $x; y - x$ is a computation of c.

Part 2: proof of $z = x; y - x$

$x \preceq z$	$cut_q(z) \preceq z$,see (*-1), and given $x = cut_q(z)$
$G_p(x) \preceq G_p(z)$	(*),$(25\|y := z, D := p)$
$x \sqsubseteq z$	(*),$(6\|y := z)$
$(p \neq q) \wedge (p \neq r) \Rightarrow G_p(z) = null$	proved earlier, see middle of Part 1.
$(p \neq q) \wedge (p \neq r) \Rightarrow G_p(x) = null$	(*),(*-2), $(31\|x := G_p(x))$
$(p \neq q) \wedge (p \neq r) \Rightarrow G_p(x) = G_p(z)$	(*-1), (*)
$G_q(cut_q(z)) = G_q(z)$	second conjunct of $(8\|x := z)$

$G_q(x) = G_q(z)$	given $x = cut_q(z)$,(*)
$(p \neq r) \Rightarrow G_p(x) = G_p(z)$	(*-2),(*)
$(p \neq r) \Rightarrow G_p(z) - G_p(x) = null$	(*),$(30\|x := G_p(z))$
$(p \neq r) \Rightarrow G_p(z - x) = null$	(*), $x \sqsubseteq z$ proved earlier, $(16\|y := z, D := p)$
$z - x = G_r(z - x)$	(*),$(20\|x := z - x)$
$z - x = G_r(z) - G_r(x)$	(*),$(16\|y := z, D := r)$
$y - x = G_r(y) - G_r(x)$	repeat last four steps with $z := y$ and using $G_p(y) = G_p(z)$ - from Part 1
$z - x = y - x$	(*-1),(*),and $G_r(y) = G_r(z)$ - from Part 1
$x; z - x = z$	$x \preceq z$ proved earlier, $(26\|y := z)$
$x; y - x = z$	(*),(*-1)

This completes the proof of Part 2. Lastly we prove that $x; y-x$ is a computation of z. We first prove $F_c(x)$, then $x \sqsubseteq y$, then $F_c(y)$, and finally use the channel property.

Part 3: proof of $F_c(x; y - x)$

$cut_q(z) \preceq cut_r(z)$	given
$cut_q(z) \preceq G_c(V)$	(*),$(32\|x := w, y := V)$, and definition of z, i.e., $z = G_c(w \uparrow V)$
$x \preceq G_c(V)$	(*), definition of x, i.e., $x = cut_q(z)$
$H(V)$	given
$F_c(G_c(V))$	(*), $(3\|x := V)$
$F_c(x)$	(*),(*-2),$(2\|y := G_c(V))$
$G_p(x) \preceq G_p(z)$	proved earlier — see second step of Part 2
$G_p(x) \preceq G_p(y)$	(*),and $G_p(y) = G_p(z)$, from Part 1
$x \sqsubseteq y$	(*),(6)
$F_c(y)$	given $F_c(G_c(t))$, and $y = G_c(t)$, see definitions of y and t.
$F_c(x; y - x)$	(*),(*-1),$F_c(x)$- see (*-4),(9)

This completes the proof of Part 3. For completeness we put the parts together:

$F_c(z)$	from Part 2 and Part 3
$F_c(G_c(w \uparrow V))$	(*),see definiton of z
$F_p(G_p(w \uparrow V))$	proved earlier, see immediately before Part 1
$H(w \uparrow V)$	(*),(*-1),$(3\|x := w \uparrow V)$