Parallel Protocol Verification: The Two-Phase Algorithm and Complexity Analysis

Maria C. Yuang and A. Kershenbaum
Bell Communications Research, Inc., Piscataway, N. J.
Department of EECS, Polytechnic University, Brooklyn, N. Y.

Abstract

Protocol verification detects the existence of logic errors in protocol design specifications. Various verification approaches have been proposed to deal with the state space explosion problem resulting from the reachability analysis. This paper proposes a parallel protocol verification algorithm, called the Two-Phase algorithm, in an attempt to provide a maximum of verification with a minimum of state space. This algorithm allows verification for all FSMs to be executed in parallel through exploring fewer states. To quantify the reduction in state space, the paper provides the state space complexity comparison between the reachability analysis and the Two-Phase algorithm. The paper defines four protocol models giving the lower and upper bound state space complexity according to both state and channel synchronization characteristics of protocols. For each model, the state space complexity of these two verification algorithms are analyzed and compared. The Two-Phase algorithm is shown to require much smaller state space. To support the analytical result, this paper also gives experimental results on several protocols, including the Call Set-Up and Termination phases of the CCITT X.25 and X.75 protocols.

1. Introduction

Protocol verification [1,2,4,10,11] detects the existence of logic errors in protocol design specifications. Various verification approaches have been proposed to deal with the state space explosion problem resulting from the conventional reachability analysis [17]. Our previous survey [14] classified these verification approaches into five categories: (a) closed covers [5], (b) modified reachability analysis [13], (c) divide-and-conquer [7,12,16], (d) partial state exploration [8,9], and (e) localized approach [3,6]. Among them, the localized approach was deemed the most promising. Formerly, approaches (in (a)-(d)) considered an execution of all processes as a whole. The localized approach, however, considers N executions of N processes separately [3]. That is, instead of creating one large global reachability tree, a local expanded tree for each individual FSM is constructed and augmented with external information. Based on the local expanded trees, protocol verification is performed.

Based on the localized approach, this paper proposes a parallel protocol verification algorithm, called the Two-Phase algorithm. The Two-Phase algorithm constructs the expanded tree in the first phase and performs error detection in the second phase. By separating verification into two phases, this algorithm allows verification for all processes to be accomplished in parallel. Moreover, because the algorithm employs a simple method of the construction of the expanded tree and a new piece of external information, the algorithm requires smaller run-time and fewer explored states. To quantify the reduction in state space, the paper provides the state space complexity comparison between the reachability analysis and the Two-Phase algorithm. This, in turn, proves that our Two-Phase algorithm requires smaller state space.

In this paper, Section 2 overviews the Two-Phase algorithm. Section 3 gives definitions of terms used in this paper. Section 4 describes the first phase (the expanded tree construction phase), and Section 5 describes the second phase (the error detection phase) of the algorithm. Section 6 defines four protocol models and then analyzes the state space complexity for each protocol model. Finally, Section 7 provides experimental results on several realistic protocols.

2. Algorithm Overview

The Two-Phase algorithm constructs the expanded tree for each process in the first phase, and performs error detection in the second phase. By separating error detection from expanded tree construction, the algorithm allows verification for one process to be totally independent from verification for other processes. Verification of all processes can thus be accomplished in parallel. That is, to perform protocol verification for N processes, N copies of the Two-Phase algorithm can be run simultaneously in an N-processor parallel machine (see Figure 1).

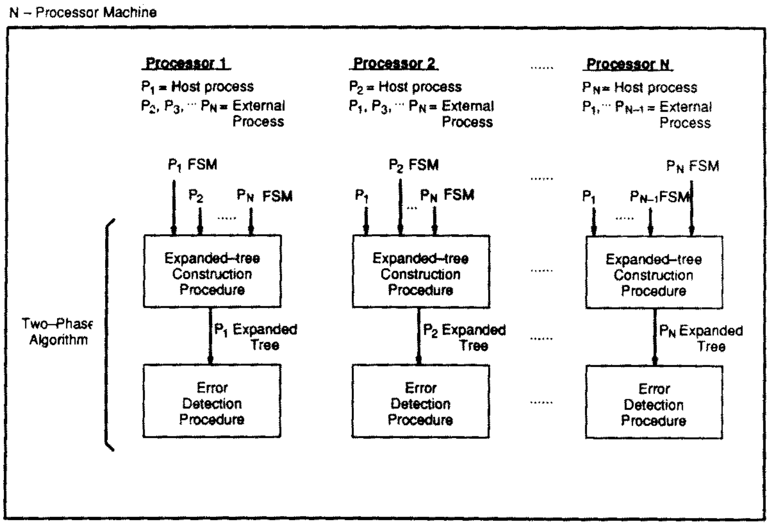


Figure 1. Parallel protocol verification for N processes: the Two-Phase algorithm.

The process being verified is called the *host process*, and all other processes are called *external processes*. The expanded tree consists of nodes and links. A node is composed of a state and the two-set external information - the least state pair and the synched set. Links are the same as transitions in FSMs.

The first phase of the algorithm constructs the expanded tree for the host process. Starting from the initial state of the host process, the tree is expanded, one state at a time, by adding all of the state's exit transitions onto the tree. For each transition, a new state is generated and the two-set external information is computed. This computation will be described in Section 4.1. The construction of the expanded tree terminates from a given state when either the further generation from this state only results in the repetition of a state already generated, or the further generation is not possible (e.g. errors). The termination rules are formally described in the T/E (Termination/Error) algorithm to be discussed in Section 4.2.

The second phase performs error detection upon the completion of the first phase. Protocol errors, such as deadcode, deadlocks, and channel overflows, can be easily detected by the simple "look-through" of the expanded tree. Unspecified receptions are detected by a two-rule procedure to be discussed in Section 5.

3. Definitions

Communication protocols can be modeled as communicating FSMs consisting of processes and channels between the processes. This model assumes FIFO and error-free channels.

[Definition 1]

A protocol system is defined as N FSMs $\{m_1, m_2, ..., m_N\}$. Each FSM m_i is defined by a quadruple (S_i, o_i, M_i, $succ$), where

1. S_i represents a finite set of states of process i.
2. $o_i \in S_i$ represents the initial state of process i.
3. M_i represents the messages from and to process i, i.e. $M_i = M_{ij} \cup M_{ji}$, where M_{ij} represents the messages sent from process i to process j, and M_{ji} represents the messages sent from process j to process i.
4. $succ$ is a partial function mapping, where $succ : S_i \times M_{ij} \rightarrow S_i$ and $S_i \times M_{ji} \rightarrow S_i$, from the states and messages of m_i to the states of m_i. In other words, $succ(s,x)$ specifies the transitional state from state s after transmitting or receiving message x. If $succ(s,x)=t$, s is called the from_state of transition x, denoted as from_state(x)=s, and t is called the to_state of transition x, denoted as to_state(x)=t. If $succ(s,x)=t$ and $succ(t,y)=r$, transition x is called an entry transition of state t, and y an exit transition of the state.

[End of Definition 1]

In this FSM model, each process or FSM is represented by a directed labeled graph with nodes and edges representing states and transitions, respectively. Figure 2 shows a protocol example with two processes P_1 and P_2. In the figure, circles denote states, arrows denote transitions. A minus sign "-" shows a transmission transition while a plus sign "+" shows a reception transition. As shown in P_1 of Figure 2, since $succ(1,+4)=3$, states 1 and 3 are the from_state and to_state of transition +4, respectively. Besides, since $succ(3,+3)=0$, transition +4 is an entry transition of state 3, and transition +3 is the exit transition of state 3.

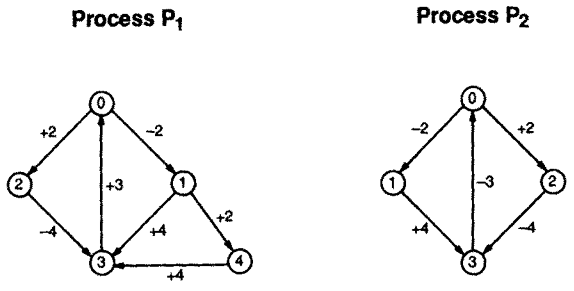


Figure 2. Protocol example 1.

Next, we define the expanded tree for protocols with only two processes.

[Definition 2]

The expanded tree of the host process P_x is a directed labeled acyclic tree composed of an ordered pair (V,E) where $V=\{n_i \mid i=1,...,M\}$ represents M *nodes*, and E represents $M-1$ *links* specifying the function $succ$ defined before. Each *node* i (i = the node number) is defined as a tuple $(s.m, L_i, Y_i)$, where

1. $s.m$ denotes the state, where s represents the state visited in the traversal of the FSM, and m is an ordinal number that identifies a particular visit to state s during the expanded tree construction. State s in the FSM is called the *corresponding state* of state $s.m$.
2. $L_i = (l_i, C_{Oi})$ is the *least state pair* of node i, where
   i. l_i denotes the least state in the external process for state $s.m$. The *least state* in external

process P_y for state s in host process P_x is defined as the last state in external process P_y that must have been reached before state s in host process P_x is reached.

ii. C_{Oi} denotes the output channel for node i when the host process is at state $s.m$ and the external process is at state l_i.

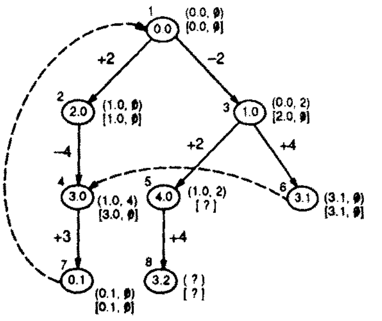
3. $Y_i = [\, y_{ik}\, , C_{Iik}\,]_{k=1,W}$ (W = a finite number) is the *synched set* of node i, where

i. y_{ik} denotes the k^{th} synched state in the external process for state $s.m$. A *synched state* in external process P_y for state s in host process P_x is defined as the first state in a possible path of the external process P_y's FSM that will eventually be reached and *synchronized* with state s. A state t in P_2 is said to be *synchronized* with state s in P_1 iff (1) all messages transmitted by P_1 before state s is reached have been received by P_2 before state t is reached, and (2) all messages received by P_1 before state s is reached have been sent by P_2 before state t is reached.

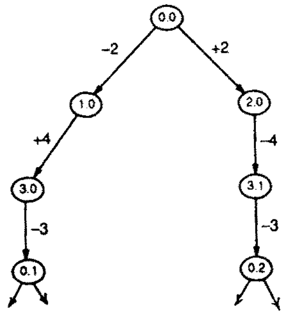
ii. C_{Iik} denotes the input channel for node i when the host process is at state $s.m$ and the external process is at state y_{ik}.

[End of Definition 2]

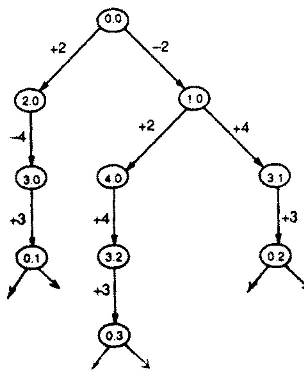
In Figure 3(d), for example, state 1.0 and external information $L_2 = (0.0\, , 2\,)$ and $Y_2 = [\, 2.0, \varnothing] + [\, 4.0\, , 2]$ compose node 2. In other words, when P_2 is at state 1.0, P_1 must have reached state 0.0 and P_2's output channel contains message 2. This is because message 2 sent by P_2 may or may not be received by P_1. Besides, this node has two synched states, 2.0 and 4.0, in the synched set. The P_2's input channel for synched state 2.0 is empty, and the input channel for synched state 4.0 contains message 2.



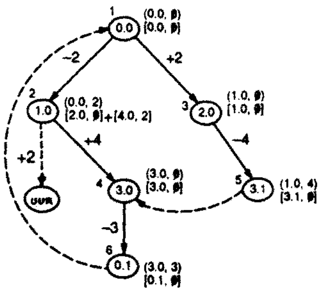
(a) P_1's expanded tree.



(b) P_2's acyclic FSM.



(c) P_1's acyclic FSM.



(d) P_2's expanded tree.

Figure 3. Protocol verification for protocol example 1 in Figure 2.

[Definition 3]

1. States (least states or synched states) $s.m$ and $t.n$ are *equivalent*, denoted as $s.m \sim t.n$, iff $s = t$.
2. Two input channels C_{Iij} and C_{Ikl} (or output channels C_{Om} and C_{On}) are said to be *equivalent*, denoted as $C_{Iij} = C_{Ikl}$ (or $C_{Om} = C_{On}$), if the sequences of messages within these two channels are the same.
3. Assuming that $L_i = (l_i, C_{Oi})$ and $L_j = (l_j, C_{Oj})$, then L_i and L_j are said to be *equivalent*, denoted as $L_i \equiv L_j$, iff $l_i \sim l_j$ and $C_{Oi} = C_{Oj}$.
4. Assuming that $Y_p = [y_{p\,i}, C_{Ipi}]_{i=1,N}$ and $Y_q = [y_{qj}, C_{Iqj}]_{j=1,M}$, then Y_p is said to be an *improper subset* of Y_q, denoted as $Y_p \subseteq Y_q$, iff (i) $N \le M$, (ii) for every $i, 1 \le i \le N$, there exists a j, $1 \le j \le M$, such that $y_{p\,i} \sim y_{qj}$ and $C_{Ipi} = C_{Iqj}$.
5. Assuming that $Y_p = [y_{p\,i}, C_{Ipi}]_{i=1,N}$ and $Y_q = [y_{qj}, C_{Iqj}]_{j=1,M}$, then Y_p and Y_q are said to be *equivalent*, denoted as $Y_p \equiv Y_q$, iff $Y_p \subseteq Y_q$ and $Y_q \subseteq Y_p$.

[End of Definition 3]

For example, in Figure 3(a), $3.1 \sim 3.0$, $L_7 \equiv L_1$, and $Y_6 \equiv Y_4$.

4. The First Phase - Expanded Tree Construction

For the sake of argument, two-process protocols are assumed throughout the rest of the paper. In addition, the external process' FSM (see Figure 3(b) or 3(c)) is shown in acyclic form. This is only for the ease of illustration and is not required in the real implementation.

4.1 External Information

When the host process progresses from an old state to a new state due to the "transmission" of message x, the least state of the new state remains the same. Message x is added into the output channel for the new state. The synched states are those states to which the external process progresses after message x has been received. All messages transmitted by the external process before receiving message x are added into the input channel for the new state.

For example, in Figure 3(d), P_2 moves to state 1.0 from state 0.0 due to the transmission of message 2. Since message 2 may or may not received by P_1, the least state of state 1.0 remains 0.0, and the output channel contains message 2. In addition, since there are two paths in P_1's acyclic FSM (Figure 3(c)), $X_1 = 0.0 \xrightarrow{+2} 2.0$ and $X_2 = 0.0 \xrightarrow{-2} 1.0 \xrightarrow{+2} 4.0$, P_1 will move to either state 2.0 or state 4.0 after it has received message 2. Thus, states 2.0 and 4.0 become the synched states. Since there is no transmission message within path X_1, the input channel for synched state 2.0 is empty. Since message 2 is the only transmission message within path X_2, the input channel for synched state 4.0 contains message 2.

On the other hand, when the host process progresses from an old state to a new state due to the "reception" of message x, the least state of the new state is the state to which the external process progresses after message x has been transmitted. All messages received before message x has been transmitted are deleted from the output channel for the new state. If the computed output channel is empty, the synched set is the same as the least state pair for the new state. Otherwise, the computation of the synched set is the same as that for the transmission case described above.

In Figure 3(d), for example, due to the reception of message 4, P_2 moves from state 1.0 to state 3.0. The least state of state 3.0 is state 3.0. This is because P_1 will progress to state 3.0 after message 4 has been transmitted. The output channel becomes empty since message 2 has been received by P_1 before transmitting message 4. For the synched set of state 3.0, since $C_{O4} = \varnothing, Y_4 = L_4 = [3.0, \varnothing]$.

When the computation of the least state pair produces multiple least state pairs, the new node is split to multiple nodes and each of them owns one least state pair. The computation of the synched set for each new nodes can be similarly computed. When the computation of the least state pair fails, a flag associated with each node i, called Term_flag$_i$, is set to be "DL (Deadlock) + DC (Dead Code)" [15].

4.2 Tree Termination - T/E Algorithm

[T/E Algorithm]

1. **Termination Rule**
   A node $(s.n, L_i, Y_i)$ is called a *precursor of type* $t1$ of another node $(t.m, L_j, Y_j)$, and the second node is said to be *marked* $t1$, if $t.m \sim s.n$ and $L_j \equiv L_i$. When a node is marked $t1$, the construction from this node terminates.
2. **Merge Rule**
   A node $(s.n, L_i, Y_i)$ is called a *precursor of type* $m1$ of another node $(t.m, L_j, Y_j)$, and the second node is said to be *marked* $m1$, if $t.m \sim s.n$ and $Y_j \subseteq Y_i$. When a node is marked $m1$, the construction from the node is said to be merged to the construction from its precursor of type $m1$.
3. **Error Rule**
   A node $(s.n, L_i, Y_i)$ is marked $e1$ if the least state cannot be computed and Term_flag_i = "DL+DC". When a node is marked $e1$, the construction from this node terminates.

[End of Algorithm]

In Figures 3(a) and 3(d), a curved dashed arrow links a node to its precursor of type $t1$ or $m1$. For example, in Figure 3(a), according to the Termination Rule, node 1 (state 0.0) is a precursor of type $t1$ of node 7 (state 0.1). According to the Merge Rule, node 4 (state 3.0) is a precursor of type $m1$ of node 6 (state 3.1). Therefore, the construction of the expanded tree from nodes 7 and 6 terminate. Finally, according to the Error Rule, node 8 (state 3.2) is marked $e1$ because Term_flag_8="DL" (Deadlock) (explained in Section 4.4).

4.3 Tree-Construction Procedure

Variables	Definitions	Domain of values
Term_flag_i	Termination flag for node i	= NULL/DC/DL
Node_queue	The FIFO queue of expandable nodes	= EMPTY/NOT_EMPTY
Etr_num(s)	The number of exit transitions of state s	> 0
Node_num	The next available node number	> 0

Table 1. The variable table for Tree-Construction Procedure.

[Tree-Construction Procedure]

```
Initialize Node_num=1;
Initialize Node_queue to have initial node (0.0, (0.0,∅), [0.0,∅]);
Initialize the expanded tree to have root node (0.0, (0.0,∅), [0.0,∅]);
while (Node_queue="NOT_EMPTY") do
{ Dequeue the first node $i$ $(s.n, L_i, Y_i)$ from Node_queue;
  for Each exit transition $x_k$, $k = 1,..,Etr\_num(s)$, of the corresponding state $s$
   do {Create a node with state $t.m$, if $succ(s, x_k) = t$ and $m$ is the next occurrence of state $t$;
       Add the node onto the expanded tree as a child node of node $i$;
       Assign Node_num to this child node, say $j$;
       Node_num ← Node_num+1;
       Compute the least state pair and the synched set for node $j$;
       if Any of the rules in the T/E algorithm can be applied to node $j$
           then Mark node $j$;
           else Add node $j$ into Node_queue; };
  if There exists at least one child node $c$ of node $i$ such that $\text{Term\_flag}_c$="NULL"
      then for Those child nodes with Term_flag = "DL+DC" do Term_flag = "DC";
      else for Those child nodes with Term_flag = "DL+DC" do Term_flag = "DL";
};
[END].
```

4.4 Example

The protocol example shown in Figure 2 has logic errors. The expanded trees for processes P_1 and P_2 are shown in Figures 3(a) and 3(d), respectively. In the following, we give a step-by-step illustration of the construction of the P_1's expanded tree (Figure 3(a)). In this example, P_1 is the host process and P_2 is the external process. The construction starts from the initial node 0.0.

1. Initialize Node_num=1, Node_queue to have initial node (0.0, (0.0,∅), [0.0,∅]), and the expanded tree to have root node (0.0, (0.0,∅), [0.0,∅]).
2. Dequeue node 1 from Node_queue.
3. Since node 1's corresponding state 0 in P_1's FSM has two exit transitions, *succ*$(0,+2)=2$ and *succ*$(0,-2)=1$, two links and two child nodes are added onto the expanded tree ($0.0 \xrightarrow{+2} 2.0$ and $0.0 \xrightarrow{-2} 1.0$). These two nodes are assigned as node 2 and node 3, respectively. Increment Node_num (=4).
4. Compute the external information, and we get $L_2=(1.0,\varnothing)$, $Y_2=[1.0,\varnothing]$, $L_3=(0.0,2)$, and $Y_3=[2.0,\varnothing]$.
5. Since $L_1 \not\equiv L_2 \not\equiv L_3$ and $Y_1 \not\equiv Y_2 \not\equiv Y_3$, nodes 2 and 3 are not marked. Add nodes 2 and 3 into Node_queue.
6. Dequeue node 2 from Node_queue.
7. Since node 2's corresponding state 2 has one exit transition, *succ*$(2,-4)=3$, one link and one child node are added onto Expanded_tree ($2.0 \xrightarrow{-4} 3.0$). This node is assigned as node 4. Increment Node_num (=5).
8. Compute the external information, we get $L_4=(1.0,4)$ and $Y_4=[3.0,\varnothing]$.
9. Since $L_4 \not\equiv L_1 \not\equiv L_2 \not\equiv L_3$ and $Y_4 \not\equiv Y_1 \not\equiv Y_2 \not\equiv Y_3$, node 4 is not marked. Add node 4 into Node_queue.
10. Dequeue node 3 from Node_queue.
11. Since node 3's corresponding state 1 has two exit transitions, *succ*$(1,+2)=4$ and *succ*$(1,+4)=3$, two links and two child nodes are added onto the expanded tree ($1.0 \xrightarrow{+2} 4.0$ and $1.0 \xrightarrow{+4} 3.1$). These two nodes are assigned as node 5 and node 6, respectively. Increment Node_num (=7).
12. Compute the external information, and we get $L_5=(1.0,2)$, $Y_5=$"UNKNOWN" (cannot be computed), $L_6=(3.1,\varnothing)$, and $Y_6=[3.1,\varnothing]$.
13. Since $L_5 \not\equiv L_1 \not\equiv L_2 \not\equiv L_3 \not\equiv L_4$ and $Y_5 \not\equiv Y_1 \not\equiv Y_2 \not\equiv Y_3 \not\equiv Y_4$, node 5 is not marked. Add node 5 into Node_queue. However, since $Y_6 \equiv Y_4$, node 6 is marked *m* 1 and the construction from node 6 is merged to the construction from node 4.
14. Dequeue node 4 from Node_queue.
15. Since node 4's corresponding state 3 has one exit transition, *succ*$(3,+3)=0$, one link and one child node are added onto the expanded tree ($3.0 \xrightarrow{+3} 0.1$). This node is assigned as node 7. Increment Node_num (=8).
16. Compute the external information, and we get $L_7=(0.1,\varnothing)$ and $Y_7=[0.1,\varnothing]$.
17. Since $L_7 \equiv L_1$, node 7 is marked *t* 1. The construction from node 7 terminates.
18. Dequeue node 5 from Node_queue.
19. Since node 5's corresponding state 4 has only one exit transition, *succ*$(4,+4)=3$, one link and one child node are added onto the expanded tree ($4.0 \xrightarrow{+4} 3.2$). This node is assigned as node 8. Increment Node_num (=9).

20. Compute the external information, and the computation fails. Node 8 is thus marked $e1$. Since node 8 is the only child node of node 5, Term_flag$_8$="DL".
21. Since Node_queue is empty, the construction terminates.

5. The Second Phase - Error Detection

Deadcode are detected by examining the expanded tree generated in the first phase. Deadlocks are detected by the presence of nodes flagged "DL". For example, in Figure 3(a), a deadlock is detected at node 8. Channel overflows are detected by the presence of nodes in which the outbound channel within the least state pair has the number of messages exceeding a predefined number (buffer size).

An unspecified reception is a reception that is executable but not specified in the FSM. It may occur in two different situations: Unconditional Unspecified Reception (UUR), and Conditional Unspecified Reception (CUR). A UUR represents a situation in which the reception of the first message in the channel is not specified at a state and no further transmission from this state can be made. A CUR represents another situation in which the reception is not specified at a state, but from this state and through transmission transitions, the process can move to another state at which the reception of the message is specified. UUR and CUR are detected by two rules: the Induction Rule, and the Propagation Rule.

[Induction Rule]
Search through the expanded tree for a reception message $+n$ such that $succ(s.x,+n)=d.y$ and the output channel of node $d.y$ is $m_1+m_2+\cdots+m_p$ where $p\geq 1$, then $succ$ (from_state($-m_i$),$+n$)="CUR", for $i=1,..,p$, are created and detected.

[End of Induction Rule]

[Propagation Rule]
Search through the expanded tree for a reception message $+n$ such that $succ(s.x,+n)=d.y$. If there exists a path X after state $s.x$ and before state $t.r$, such that $succ(s.x,-m_1)=t_1$, $succ(t_1,-m_2)=t_2$, ..., $succ(t_{q-2},-m_{q-1})=t_{q-1}$, and $succ(t_{q-1},-m_q)=t.r$, where $q\geq 1$, and $t.r$ has only reception exit transitions, then (1) if $succ(t_i,+n)$ does not exist, create $succ(t_i,+n)=$"CUR", for $i=1,..,q-1$ and $q\geq 2$, and (2) if $succ(t.r,+n)$ does not exist, create $succ(t.r,+n)=$"UUR".

[End of Propagation Rule]

In Figure 3(d), according to the Propagation Rule, the reception of message +2 should also be specified at state 1.0, thus $succ(1.0,+2)=$ "UUR" is created and detected.

6. State Space Complexity Analysis

The state-space complexity is determined by two factors: state synchronization and channel synchronization of protocols. State synchronization defines how states in one process are synchronized to states in another process. Based on this characteristic, protocols can be classified as either tightly-synchronous-oriented or loosely-synchronous-oriented. Protocols which allow a state in one process to be coexisted with a small number of states in another process are categorized as tightly-synchronous-oriented. On the other hand, protocols which allow a state in one process to be coexisted with a large number of states in another process are categorized as loosely-synchronous-oriented. The more loosely-synchronous the protocol is, the larger state-space complexity the protocol requires, and vice versa.

Channel synchronization defines how the channel in one process is synchronized to the channel in another process. For example, in Figure 7, when P_1 is at state s_r and P_2 at state t_r, P_1's channel may contain different numbers of messages and so may P_2's channel. This occurs when contention is detected by both process, each process will progress to its recovery state from which messages in the channel are cleared and the system is then recovered. On the other hand, for the protocol shown in Figure 6, when P_1 is at state s_r and P_2 at state t_r, P_1 may contain different numbers of messages in its channel; whereas P_2 can only have an empty channel. This occurs when errors are detected by one process, this process then progresses to a recovery state from which messages in the channel are cleared. The more numbers of processes which allow multiple messages in their channels a protocol has, the larger state-space complexity the protocol requires, and vice versa.

Accordingly, we define four protocol models:

(a) Model 1: tightly-synchronous state synchronization model;
(b) Model 2: loosely-synchronous state synchronization model;
(c) Model 3: single channel synchronization model; and
(d) Model 4: complete channel synchronization model.

Based on the state synchronization characteristic of protocols, Model 1 gives the lower bound state-space complexity and Model 2 gives the upper bound state-space complexity. Based on the channel synchronization characteristic of protocols, Model 3 gives the lower bound state-space complexity and Model 4 gives the upper bound state-space complexity. Note that, for the sake of argument, protocols are shown with only two processes.

The state-space complexity is represented by the following sums:

$S(R)$ = The total number of global states generated using the reachability analysis;
$S(T)$ = The total number of local states generated using the Two-Phase algorithm.

6.1 Model 1

Model 1 (Figure 4) defines a protocol in which process P_x is tightly synchronous with process P_y among k sets of coupled transitions (transmissions and corresponding receptions), where

m_i = the number of coupled transitions in the i_{th} set, and $m_i \geq 1, i = 1$ to k.

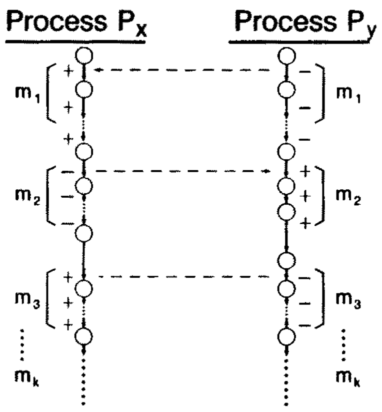


Figure 4. Model 1.

We compute and get:

$$S(R) = \sum_{i=1}^{k} \frac{(m_i+2)(m_i+1)}{2} - (k-1) = \frac{\sum_{i=1}^{k} (m_i^2+3m_i)}{2} + 1 \tag{1}$$

$$S(T) = 2[(m_1+1)+(m_2+1)+ \cdots +(m_k+1)-(k-1)] = 2(\sum_{i=1}^{k} m_i + 1). \tag{2}$$

In comparison with the reachability analysis, by subtracting (1)-(2), we get

$$S(R)-S(T) = \frac{\sum_{i=1}^{k} (m_i^2-m_i)}{2} - 1 \tag{3}$$

In (3), the state space of the two T-trees is shown to be not always smaller than that of the reachability tree. This indicates that the Two-Phase algorithm is not favored all the time for tightly-synchronous-oriented protocols.

6.2 Model 2

Model 2 (Figure 5) defines a protocol in which process P_x and P_y are allowed to simultaneously transmit n and m messages, respectively, without acknowledgements, where

n = the maximum number of consecutive unacknowledged transmissions allowed for P_x;
m = the maximum number of consecutive unacknowledged transmissions allowed for P_y;
and $n, m \geq 1$.

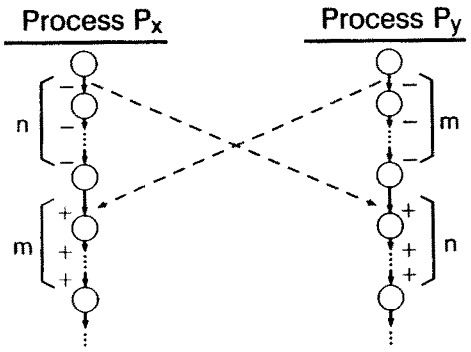


Figure 5. Model 2.

We compute and get:

$$S(R) = \frac{n^2+3n}{2} + \frac{m^2+3m}{2} + 2mn + 1 \tag{4}$$

$$S(T) = 2(n+m+1)\ . \tag{5}$$

In comparison with the reachability analysis, by subtracting (4)-(5), we get

$$S(R) - S(T) = \frac{n^2-n}{2} + \frac{m^2-m}{2} + 2mn - 1 > 0\ . \tag{6}$$

Therefore, for Model 2, the state space of two T-trees is shown to be always smaller than that of the reachability tree. This indicates that the Two-Phase algorithm is always favored for loosely-synchronous protocols.

6.3 Model 3

Model 3 (Figure 6) defines a protocol in which process P_x initiates an error recovery procedure from any of the states s_0, s_1, ..., and s_{m+n}, by sending message a to P_y, and moves to the synchronized state s_r; and process P_y executes the error recovery procedure upon receiving message a, and moves to the synchronized state t_r. Consequently, when P_x is at state s_r, multiple messages are allowed in its channel; whereas when P_y is at state t_r, it can only have an empty channel. In this model,

m = the number of coupled transitions in the first set, before an error recovery takes place;
n = the number of coupled transitions in the second set, before an error recovery takes place; $m, n \geq 1$.

We compute and get:

$$S(R) = \frac{3m^2+n^2+7m+5n+6}{2} \tag{7}$$

$$S(T) = 2[(m+n+1)+1]\ . \tag{8}$$

Compared to the reachability analysis, by subtracting (7)-(8), we get

$$S(R) > S(T)\ . \tag{9}$$

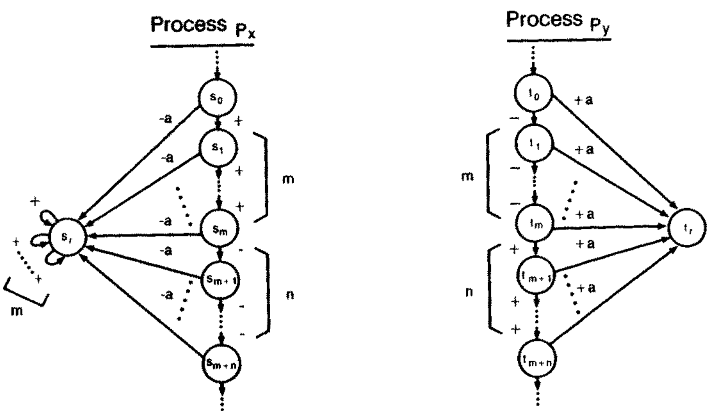


Figure 6. Model 3.

We thus prove that, for Model 3, the state space using the Two-Phase algorithm is always smaller than that using the reachability analysis. This indicates that the Two-Phase algorithm is favored over the reachability analysis for protocols which allow multiple messages in only one channel; or which provide the general-error recovery capability, in practice.

6.4 Model 4

Model 4 (Figure 7) defines a protocol in which process P_x starts an error recovery procedure from any of the states s_1, s_2, ..., and s_n, when the process receives an unexpected message b and interprets it as a contention, and then moves to the synchronized state s_r; and similarly for P_y. Consequently, when P_x is at state s_r and P_y at state t_r, multiple messages are allowed in both processes' channel. In this model,

m = the maximum number of consecutive unacknowledged transmissions allowed for P_y;
n = the maximum number of consecutive unacknowledged transmissions allowed for P_x;
and m, $n \geq 1$.

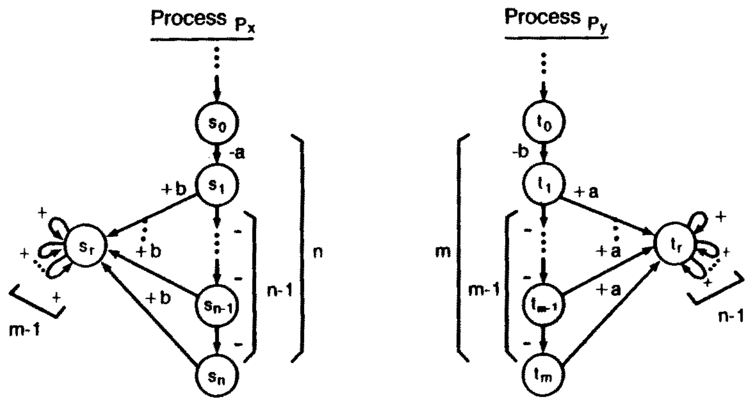


Figure 7. Model 4.

We compute and get:

$$S(R)=\frac{m^2n^2+m^2n+mn^2+9mn}{4}+\frac{m^2+n^2+m+n}{2}+2 \tag{10}$$

$$S(T)=[m+(n+1)]+[n+(m+1)]=2(m+n+1)\ . \tag{11}$$

Compared to the reachability analysis, by subtracting (10)-(11), we get

$$S(R)\gg S(T)\ . \tag{12}$$

We thus prove that, for Model 4, the state space using the Two-Phase algorithm is much smaller tha that using the reachability analysis. This indicates that the Two-Phase algorithm is favored over th reachability analysis for protocols which allow multiple messages in both channels; or which provide contention-error recovery capability, in practice.

6.5 State Space Complexity Summary

Given N the number of consecutive unacknowledged transmissions, the complexity analysis yields th following results (Table 2):

State-Space Complexity	Reachability Analysis	Two-Phase Algorithm
Model 1	N^2	N
Model 2	N^2	N
Model 3	N^2	N
Model 4	N^4	N

Table 2. The state space complexity for two-process protocols.

In summary, the state space using the reachability analysis always grows non-linearly with the numbers of states and messages in the channel. By contrast, for the Two-Phase algorithm, the state space only grows linearly with the numbers of states and messages. Moreover, the increase of the state space will be magnified by higher numbers of communicating processes in the protocol.

7. Experimental Results

To quantify verification state space required by the reachability analysis and the Two-Phase algorithm, we implemented these two algorithms and conducted experiments on several realistic protocols. The numbers of states and transitions of each protocol are listed in Table 3. The numbers of explored states using the reachability analysis and the Two-Phase algorithm are shown in Table 4. These experimental results show that the Two-Phase algorithm requires much fewer explored states than the reachability analysis does.

	P_x states	P_x transitions	P_y states	P_y transitions
Simple Data Transfer	8	12	8	12
X.25 Call Set-Up/Clearing	7	22	7	21
X.75 Call Set-Up/Clearing	12	69	12	69
Modified BISYNC	68	174	68	174

Table 3. The states and transitions of experimented protocols.

State Space	Reachability Analysis	Two-Phase Algorithm
Simple Data Transfer	42	18
X.25 Call Set-Up/Clearing	111	17
Modified BISYNC	292	172
X.75 Call Set-Up/Clearing	1534	46

Table 4. The numbers of explored states for experimented protocols in Table 3.

8. Conclusions

This paper proposed a parallel protocol verification algorithm, called the Two-Phase algorithm, in an attempt to provide a maximum of verification with a minimum of state space. The algorithm constructs the expanded trees in the first phase and performs error detection in the second phase. By separating expanded tree construction from error detection, the algorithm allows verification of all processes to be accomplished in parallel. This parallelism tremendously reduces the run time for verifying protocols with higher numbers of processes. In addition, the algorithm employs a simple tree construction procedure and a new piece of external information, "synched states", the algorithm thus requires fewer explored states.

This paper then provided state space complexity comparison between the reachability analysis and our algorithm. To analyze the complexity, we defined four protocol models, giving the lower and upper bound state space complexity according to both state and channel synchronization characteristics of the protocols. For each model, the state space complexity of each algorithm is computed. In practice, Model 1 applies to protocols with a high degree of interaction. Model 2 applies to protocols which allow processes to simultaneously transmit unacknowledged messages. Model 3 applies to protocols with the error recovery capability for handling general errors. Finally, Model 4 applies to protocols with the error recovery capability for handling the simultaneous transmissions of messages or contention.

In summary, the state space using the reachability analysis grows polynomially when the numbers of states and messages in the channel increase. By contrast, for the Two-Phase algorithm, the state space only grows linearly with the numbers of states and messages. The increase of the state space will be magnified by higher numbers of communicating processes in the protocol. To support the analytical result shown above, we also experimented on several protocols. Our experimental results showed that, for the X.75 protocol, the total numbers of explored states using the reachability analysis and the Two-Phase algorithm are 1534 and 46, respectively. Results proved the superiority of Two-Phase algorithm over the reachability analysis.

9. Acknowledgments

The authors would like to especially thank Dr. A. L. P. Chen, Dr. C. H. Chow, Dr. I. Frisch, and Dr. M. Koblentz for their giving valuable comments and suggestions.

References

[1] G. V. Bochmann, "Finite State Description of Communication Protocols," Computer Networks, Vol. 2, Oct. 1978, pp. 361-372.

[2] G. V. Bochmann and C. A. Sunshine, "Formal Methods in Communication Protocol Design," IEEE Trans. on Communications, Vol. COM-28, No. 4, April 1980, pp. 624-631.

[3] D. Brand and P. Zafiropulo, "On Communicating Finite-State Machines," Journal of the ACM, Vol 30, No. 2, April 1983, pp. 323-342.

[4] A. Danthine, "Protocol Representation with Finite State Models," IEEE Trans. on Communications, Vol. COM-28, No. 4, April 1980, pp. 632-643.

[5] M. G. Gouda, "Closed Covers: To Verify Progress of Communicating Finite State Machines," IEEE Trans. on Software Engineering, Nov. 1984, Vol. SE-10, No. 6, pp. 846-855.

[6] Y. Kakuda, Y. Wakahara, and M. Norigoe, "A New Algorithm For Fast Protocol Validation," Proc. IEEE COMPSAC, 1986, pp. 228-236.

[7] S. Lam and A. Shankar, "Protocol Verification via Projections," IEEE Trans. on Software Engineering, Vol. SE-10, No. 4, July 1984, pp. 325-342.

[8] F. Lin, P. Chu, and M. Liu, "Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies," Proc. ACM SIGCOMM, Aug. 1987.

[9] N. F. Maxemchuk and K. Sabnani, "Probabilistic Verification of Communication Protocols," Protocol Specification, Testing, and Verification, VII, pp. 307-320, North-Holland, 1987.

[10] P. M. Merlin, "Specification and Validation of Protocols," IEEE Trans. on Communications, Vol. COM-27, No. 11, Nov. 1979, pp. 1671-1680.

[11] "Communication Protocol Modeling," Edited by Carl A. Sunshine, Artech House.

[12] S. T. Vuong and D. D. Cowan, "A Decomposition Method for the Validation of Structured Protocols," Proc. IEEE INFOCOM, April 1982.

[13] S. T. Vuong, D. D. Hui, and D. D. Cowan, "VALIRA - A Tool for Protocol Validation Via Reachability Analysis," Protocol Specification, Testing, and Verification, VI. North-Holland, 1986.

[14] M. C. Yuang, "Survey of Protocol Verification Techniques Based on Finite State Machine Models," Proc. NBS Computer Networking Symposium, 1988, pp. 164-172.

[15] M. C. Yuang and A. Kershenbaum, "Parallel Protocol Verification Using the Localized Approach: A Two-Phase Algorithm," Proc. Ninth International Symposium on Protocol Specification, Testing, and Verification, 1989.

[16] P. Zafiropulo, "Protocol Validation by Duologue-Matrix Analysis," IEEE Trans. Commun. COM-26, 8(August, 1978), pp. 1187-1194.

[17] P. Zafiropulo, et al., "Towards Analyzing and Synthesizing Protocols," IEEE Trans. on Communications, COM-28, 4(April, 1980), pp. 651-661.