数理解析研究所講究録
第655巻 1988年 84-103

An Algebraic Method For Verifying Progress Property of Communication Protocols

東野 輝夫　谷口 健一　嵩 忠雄
Teruo HIGASHINO[+], Kenichi TANIGUCHI[+], Tadao KASAMI[+],
藤井 護　森 将豪
Mamoru FUJII[+] and Masaaki MORI[++]

+ : Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University, Toyonaka, Osaka 560, JAPAN
++ : Department of Management Science,
Faculty of Economics, Shiga University, Hikone, Shiga 522, JAPAN

ABSTRACT In this paper, a method is presented for verifying the progress property of a communication protocol whose specification is described algebraically in a style of an "abstract sequential machine", and Stenning's data transfer protocol is used as a running example. At first, the set of sequences of states is introduced, and several theorems (invariants) on the specification are verified by algebraic methods, and then, they are transformed into the theorems for the sequences of states. Next, such a hypothesis as "if the packets with the same contents have been retransmitted repeatedly, at least one of them will be eventually received by the receiver" is also formulated as theorems for the sequences of states. Finally, by using those theorems and several axioms which denote inference rules of Temporal logic, the formula (term) which expresses the progress property is proved to be true by using our interactive verification support system for algebraic specifications.

1. INTRODUCTION

In order to specify a protocol formally and to verify the correctness of the protocol, the protocol must be exactly specified. There have been some approaches to specifying protocols such as state machine approach, the high-level programming language approach and algebraic approach [1,2,3,4,5,6]. We have algebraically specified protocols such as the HDLC procedures and Stenning's Data Transfer Protocol [3] (for short, SDT protocol) in a style of an "abstract sequential machine" [6,7], and verified some theorems (invariants) on the specifications, for example, a theorem on SAFENESS property of the protocols, etc. [6,8].

To specify a protocol in a style of an abstract sequential machine, we introduce a set of abstract states of the whole communication system as a sort, state transition functions corresponding to the basic operations of the protocol such as transmission/reception of the stations, and output functions which extract necessary information from an abstract state. Each abstract state is represented as the sequence of state transition functions. The semantics of output functions is given by axioms which describe relations to be held between the values of output functions at the state just before the execution of a state transition function and those at the state just after its execution.

Algebraically specifying a protocol in a style of an abstract sequential machine has the following advantage : The meaning of a description is defined formally and concisely because of using the congruence relation [13] only, and therefore its verification is carried out formally

in the same framework as the algebraic specification. And, by using output functions, it overcomes the problem of state explosion where the state machine approach would require a large number of states. In general, the high-level programming language (an ordinary procedural language) approach has such a drawback that protocol procedures are specified uniquely and therefore the non-deterministic behaviors can't be described naturally. But, in our model, basic operations such as transmission/reception behaviors of each station are specified as operations independent of the other behaviors in a quite natural way.

There have been some studies [11,12,14,16] on the DEADLOCK-free property and PROGRESS property of protocols that the packets sent out from the sender will be eventually received by the receiver. In this paper, we present a method to formally carry out verification of PROGRESS property of a protocol that is algebraically specified in a style of an abstract sequential machine, and apply the method to SDT protocol.

As for verification of PROGRESS property, we describe the notion of Temporal logic algebraically. First we consider a sequence of states which are reachable from the present state by executing allowable behaviors successively, and second we verify some invariants which hold on any state in a sequence of states and those which hold between the state just before the execution of a state transition function and the state just after its execution. These invariants are represented as theorems [10] on the algebraic specification. And then, we transform these theorems into the the theorems for the sequences of states by using some transformation rules which will be introduced in Section 3.2.3. Furthermore, such hypotheses on stations and communication lines that the sender will eventually send a packet if the transmission of the packet is allowed repeatedly and that communication lines are not in cut-off state are formulated as the properties for the sequences of states. Then, we try to prove a predicate that asserts PROGRESS property of SDT protocol to be true by using the the transformed theorems, hypotheses on stations and communication lines, and the axioms which denote inference rules on operators "□"(always), "◇"(eventually) and "○"(next) in Temporal logic.

In our formalism, we can treat the verification of PROGRESS property in the same framework as the algebraic specification of SDT protocol. Since we use our interactive verification support system for verification of theorems on a given algebraic specification [10] which provides such facilities as term reduction (by axioms which are regarded as rewrite rules) and a decision procedure for Presburger arithmetic, the whole verification works are carried out formally. Here, SDT protocol is used as a verification example, because SDT protocol is considerably complicated one where the occurrence of loss, corruption, duplication and/or re-ordering of packets in transit are taken into account, and is suitable in scale for examining our verification method, and moreover we have already proved some theorems on the algebraic specification of SDT protocol necessary for verifying PROGRESS property [8].

2. AN ALGEBRAIC SPECIFICATION OF THE WHOLE COMMUNICATION SYSTEM

2.1 AN ABSTRACT SEQUENTIAL MACHINE MODEL

For specifying the whole communication system algebraically, we introduce a sort "state" which denotes the set of states of the whole communication system, and represent basic operations such as transmission or reception in a station as "state transition functions" of the whole communication system. There is a constant "INITIAL" in sort "state" which corresponds to the initial state of the whole communication system. For a state transition function g, $g(s, i_1, \ldots, i_n)$ denotes the resulting state when state transition function g with parameter

$i_1,\ldots,i_n$ is executed at state s. Here, a state is considered as a sequence of basic operations (state transition functions) applied on INITIAL. That is, a state is represented as a term consisting of state transition functions and INITIAL. We also introduce "output functions" which extract some information such as the value of a register or the data to be sent next in each station from a state. The semantics of those functions is given by axioms. Axioms describe relations between the values of output functions at the state just before the execution of a state transition function and those at the state just after its execution.

For example, an output function, say, "f" may have the axioms of the following form :

(1) $f(INITIAL) == 0$

(2) $f(g(s,i_1,\ldots,i_n)) == IF\ f_1(s) \geqq f_2(s)+1\ THEN\ f(s)+1\ ELSE\ f(s)$

Here, "g" denotes a state transition function, "s" a variable of sort state, and "i_1",...,"i_n" variables of sorts other than sort state, respectively. The axiom (1) defines the initial value of output function f, and the axiom (2) defines the value of output function f at state $g(s,i_1,\ldots,i_n)$ by using the values of output functions at state s ($f(s)$, $f_1(s)$ and $f_2(s)$, etc.) and some operators and/or values (IF-function, $\geqq$, + and 1, etc.) of Base algebra [6,7]. For every pair of output function f and state transition function g, axioms in the form of (2) are described.

In the set of states (terms of sort state) of the abstract sequential machine stated above, there may exist meaningless states, for example, which are reachable from the initial state by executing only receiving behaviors. To distinguish valid states (meaningful states) from meaningless states, we introduce a special output function "Valid" which indicates whether or not the present state is reachable from the initial state by executing only allowable actions successively, and regard states such that the value of function Valid is TRUE (FALSE) as valid states (meaningless states) of the protocol. We describe the validity of the states by the following form of axioms :

(1') $Valid(INITIAL) == TRUE$

(2') $Valid(g(s,i_1,\ldots,i_n)) == IF\ Valid(s)\ THEN\ P(s,i_1,\ldots,i_n)\ ELSE\ FALSE$

Here, $P(s,i_1,\ldots,i_n)$ is a term of sort bool which consists of output function symbols, variables $s,i_1,\ldots,i_n$ and some operator symbols and/or values (IF-function, $\geqq$, +, TRUE, FALSE, 0, 1, etc.) $i_1,\ldots,i_n$ is allowable at a state s if and only if the state s is a valid state and predicate $P(s,i_1,\ldots,i_n)$ is TRUE at state s. For example, if "g" is a state transition function which represents a receiving action and $P(s,i_1,\ldots,i_n)$ is "f(s)>0", then the axiom shows that receiving action g is executable when the value of output function f is greater than 0.

2.2 AN ALGEBRAIC SPECIFICATION OF SDT PROTOCOL

Stenning's Data Transfer Protocol [3] (for short SDT protocol) defines actions of two processes, a sender and a receiver, by means of Pascal-like programming language, and the term "packet" is used to denote a unit sent by the sender or the receiver. A packet sent by the sender consists of a message and a sequence number attached to the message, and a packet sent by the receiver, called an acknowledgement packet, consists of only a sequence number, which means that all messages with sequence numbers up to this number have been received and released to a sink of the receiver correctly. It is assumed that sequence numbers may increase infinitely. The sender gets a message from a source and sends a packet to the receiver through a communication line. The receiver in turn accepts the packet from the line, releases the

received message to a sink, and acknowledges its correct reception by sending an acknowledgement packet to the sender via another communication line. Since the communication lines are unreliable, it is assumed that packets traveling in either direction may be completely lost, corrupted, duplicated and/or reordered. It is also assumed that there are no undetected transmission errors (i.e., that corruptions can be detected by a checksum test on a received packet). Any received packet which fails the checksum test is immediately discarded and is therefore effectively "lost in transit". In order to ensure correct delivery of a packet under such environment, SDT protocol uses a conventional positive acknowledgement/retransmission on timeout technique.

An algebraic specification of SDT protocol is shown in Table 1. In Table 1, state transition functions "TRANSM", "RETRANSM", "RECA", and "FINDTO" are introduced corresponding to basic actions "transmission", "retransmission", "reception" and "timeout" of the sender, respectively. As for the receiver, a state transition function "RECM&TRANSA" which corresponds to basic action "reception of a message and transmission of an acknowledgement" is introduced.

As output functions, we introduce four types of output functions described below :

[1] The output functions corresponding to the functions (or predicates) which are introduced in [3] to define the behavior of the stations.

(1-1) The output functions of the sender

- highest_sent(s) : It denotes the sequence number of the highest numbered packet sent so far at state s.
- lowest_unacked(s) : It denotes the sequence number of the lowest numbered packet sent by the sender and still requiring acknowledgement from the receiver at state s.
- timer_running(s, i): It indicates whether or not the timer for the packet with sequence number i is running at state s.
- retrans_mode(s) : It indicates whether or not state s is in retransmission mode due to the occurrence of timeout.
- retrans_position(s) : It denotes the sequence number of a packet to be retransmitted subsequently at state s.
- TWS : It is an abbreviation of "Transmitter Window Size" and it is the maximum number of

(1-2) The output functions of the receiver

- next_required(s) : It denotes the sequence number of a packet to be received by the receiver such that all packets numbered up to the sequence number minus one have been correctly received by the receiver at state s.
- already_received(s, j): It indicates whether or not the packet with sequence number i has been correctly received by the receiver at state s.

[2] The output functions corresponding to the functions (or predicates) which are introduced in [3] to verify some assertions on the behavior of SDT protocol.

(2-1) The output functions of the sender and the receiver

- SMB(i) : SMB is an abbreviation of "Source Message Buffer" and it denotes the messages held in a source of the sender. Message SMB(i) is sent from the sender as the packet with sequence number i.
- rmb(s, j) : It denotes the message which was received as the packet with sequence number j and released to a sink of the receiver.

(2-2) The output functions of the communication lines

Table 1 : An algebraic specification of Stenning's data transfer protocol (PART I)

```
text  STENNING ;
 project  primitives ;{the declaration to use the operators and/or values of Base algebra}
 define   seq_number := nn_integer ;

  label constructors :
    state -> INITIAL ;
          -> TRANSM( state ) ;
          -> RETRANSM( state , seq_number ) ;
          -> RECA( state , seq_number ) ;
          -> FINDTO( state , seq_number ) ;
          -> RECM&TRANSA( state , seq_number , seq_number );
  endlabel constructors ;
  label output_functions :
    seq_number -> highest_sent( state ) ;
               -> lowest_unacked( state ) ;
    bool       -> timer_running( state , seq_number ) ;
               -> retrans_mode( state ) ;
    seq_number -> retrans_position( state ) ;
    nn_integer -> k( state ) ;
    seq_number -> S( state , nn_integer ) ;
    data       -> datapart( state , nn_integer ) ;
    seq_number -> next_required( state ) ;
    bool       -> already_received( state , seq_number ) ;
    data       -> rmb( state , seq_number ) ;
    nn_integer -> p( state ) ;
    seq_number -> A( state , nn_integer ) ;
  endlabel output_functions ;
    bool       -> exist_S( state , seq_number ) ;
               -> exist_A( state , seq_number ) ;
    data       -> data_S( state , seq_number ) ;
    nn_integer -> pos_S( state , nn_integer , seq_number ) ;
               -> pos_A( state , nn_integer , seq_number ) ;
    seq_number -> pos_r( state , seq_number ) ;
    bool       -> Valid( state ) ;
    data       -> SMB( seq_number ) ;
    data       -> NULL ;
    seq_number -> TWS ;

    state       s ;
    seq_number  n , l , h , i , j , a ;
    nn_integer  m ;

AX1   : highest_sent(INITIAL) == 0 ;
AX2   : lowest_unacked(INITIAL) == 1 ;
AX3   : timer_running(INITIAL,n) == FALSE ;
AX4   : retrans_mode(INITIAL) == FALSE ;
AX5   : retrans_position(INITIAL) == 0 ;
AX6   : k(INITIAL) == 0 ;
AX7   : S(INITIAL,m) == 0 ;
AX8   : datapart(INITIAL,m) == NULL ;
AX9   : next_required(INITIAL) == 1 ;
AX10  : already_received(INITIAL,n) == FALSE ;
AX11  : p(INITIAL) == 0 ;
AX12  : A(INITIAL,m) == 0 ;
AX13  : rmb(INITIAL,n) == NULL ;
AX14  : Valid(INITIAL) == TRUE ;

AX15  : highest_sent(TRANSM(s)) == highest_sent(s)+1 ;
AX16  : for each 'Q' in ( 'RETRANSM(s,h)' , 'RECA(s,j)' , 'FINDTO(s,n)' ) ,
          highest_sent(Q) == highest_sent(s) ;
AX17  : lowest_unacked(RECA(s,j)) == if j=lowest_unacked(s) then j+1 else lowest_unacked(s) ;
AX18  : for each 'Q' in ( 'TRANSM(s)' ,'RETRANSM(s,h)' , 'FINDTO(s,n)' ) ,
            lowest_unacked(Q) == lowest_unacked(s) ;
AX19  : timer_running(TRANSM(s),n) == n=highest_sent(s)+1 OR timer_running(s,n) ;
AX20  : timer_running(RETRANSM(s,h),n) == n=h OR timer_running(s,n) ;
```

Table 1 : An algebraic specification of Stenning's data transfer protocol (PART II)

```
  AX21  : timer_running(FINDTO(s,1),n) == if highest_sent(s)=n AND n>=1
                                          then FALSE else timer_running(s,n) ;
  AX22  : timer_running(RECA(s,j),n) == if j>n AND n>=lowest_unacked(s)
                                        then FALSE else timer_running(s,n) ;
  AX23  : retrans_mode(RETRANSM(s,h)) == if highest_sent(s)=h then FALSE else retrans_mode(s) ;
  AX24  : retrans_mode(RECA(s,j)) == if j>highest_sent(s) then FALSE else retrans_mode(s) ;
  AX25  : retrans_mode(FINDTO(s,n)) == TRUE ;
  AX26  : retrans_mode(TRANSM(s)) == retrans_mode(s) ;
  AX27  : retrans_position(RETRANSM(s,h)) == retrans_position(s)+1 ;
  AX28  : retrans_position(RECA(s,j)) == if j>retrans_position(s)
                                         then j else retrans_position(s) ;
  AX29  : retrans_position(FINDTO(s,n)) == n ;
  AX30  : retrans_position(TRANSM(s)) == retrans_position(s) ;
  AX31  : for each 'Q' in ( 'TRANSM(s)' , 'RETRANSM(s,h)' ), k(Q) == k(s)+1 ;
  AX32  : for each 'Q' in ( 'RECA(s,j)' , 'FINDTO(s,n)' ) ,  k(Q) == k(s) ;
  AX33  : S(TRANSM(s),m) == if m=k(s)+1 then highest_sent(s)+1 else S(s,m) ;
  AX34  : S(RETRANSM(s,h),m) == if m=k(s)+1 then h else S(s,m) ;
  AX35  : datapart(TRANSM(s),m) == if m=k(s)+1 then SMB(highest_sent(s)+1) else datapart(s,m) ;
  AX36  : datapart(RETRANSM(s,h),m) == if m=k(s)+1 then  SMB(h) else datapart(s,m);
  AX37  : for each 'Q' in ( 'RECA(s,j)' , 'FINDTO(s,n)' ) ,
            for each f in ( S , datapart ) , f(Q,m) == f(s,m) ;

  AX38  : next_required(RECM&TRANSA(s,i,j)) == if i=next_required(s)
                                               then pos_r(s,i+1) else next_required(s) ;
  AX39  : already_received(RECM&TRANSA(s,i,j),n) == if i=n then TRUE else already_received(s,n);
  AX40  : rmb(RECM&TRANSA(s,i,j),n) == if i=n then data_S(s,i) else rmb(s,n) ;
  AX41  : p(RECM&TRANSA(s,i,j)) == p(s)+1 ;
  AX42  : A(RECM&TRANSA(s,i,j),m) == if m=p(s)+1 then j else A(s,m) ;

  AX43  : for each f in ( highest_sent , lowest_unacked , retrans_mode , retrans_position , k ),
               f(RECM&TRANSA(s,i,j)) == f(s) ;
          for each g in ( timer_running , S , datapart ) ,
              g(RECM&TRANSA(s,i,j),a) == g(s,a) ;
  AX44  : for each 'Q' in ( 'TRANSM(s)' , 'RETRANSM(s,h)' , 'RECA(s,j)' , 'FINDTO(s,n)' ) ,
          begin
             for each f in ( next_required , p ) , f(Q) == f(s) ;
             for each g in ( already_received , A , rmb ) , g(Q,a) == g(s,a) ;
          end;

  AX45  : exist_S(s,i) == NOT(pos_S(s,k(s),i)=0) ;
  AX46  : exist_A(s,j) == NOT(pos_A(s,p(s),j)=0) ;
  AX47  : data_S(s,i)  == datapart(s,pos_S(s,k(s),i)) ;
  AX48  : pos_S(s,m,i) == if m=0 then 0 else if S(s,m)=i then m else pos_S(s,m-1,i) ;
  AX49  : pos_A(s,m,i) == if m=0 then 0 else if A(s,m)=i then m else pos_A(s,m-1,i) ;
  AX50  : pos_r(s,n)   == if already_received(s,n) then pos_r(s,n+1) else n ;

  AX51  : Valid(TRANSM(s)) == if Valid(s) then NOT(retrans_mode(s)) AND
                                               TWS-1>highest_sent(s)-lowest_unacked(s)
                                          else FALSE ;
  AX52  : Valid(RETRANSM(s,h)) == if Valid(s) then retrans_mode(s) AND h=retrans_position(s)
                                              else FALSE ;
  AX53  : Valid(RECM&TRANSA(s,i,j)) == if Valid(s)
                                       then exist_S(s,i) AND
                                            {(i=next_required(s)) AND (j=pos_r(s,i+1)-1) OR
                                             NOT(i=next_required(s)) AND (j=next_required(s)-1)}
                                       else FALSE ;
  AX54  : Valid(FINDTO(s,n)) == if Valid(s) then timer_running(s,n) else FALSE ;
  AX55  : Valid(RECA(s,j)) == if Valid(s) then exist_A(s,j) else FALSE ;
end STENNING ;
```

- k(s) : It denotes the number of packets sent by the sender. Let $p_1, p_2, \ldots, p_{k(s)}$ be the sequence of packets sent by the sender (p_1 is the first packet and $p_{k(s)}$ is the last packet sent by the sender). We call packet p_i "the i-th packet" sent by the sender.
- S(s, n) : It denotes the sequence number of n-th packet sent by the sender.
- datapart(s, n) : It denotes the message of n-th packet sent by the sender.
- p(s) : It denotes the number of packets sent by the receiver.
- A(s, m) : It denotes the sequence number of m-th acknowledgement packet sent by the receiver.

[3] Auxiliary functions

In [3], a few assertions concerning the receiving procedures of messages or acknowledgements are expressed by using existential quantifiers. However, existential quantifiers cannot be used directly in our axiom. Here, we newly introduce auxiliary functions "exist_S", "exist_A" and "data_S" which correspond to the Skolem functions, in order to delete existential quantifiers and variables bounded by them appeared in the assertions.

"Pos_S", "pos_A" and "pos_r" are auxiliary functions which are introduced to recursively define the functions "exist_S", "exist_A" and "next_required", respectively.

[4]A special function "Valid"

As stated in Section 2.1, the value of function "Valid" is TRUE for a state which is reachable from the initial state by executing allowable operations successively, and FALSE when an unallowable operation has been executed at least once.

2.3 VERIFICATION OF STATIC PROPERTIES

For a term P of sort bool, let $\{x_1, \ldots, x_n\}$ be the set of distinct variables occurring in P. For simplicity, we write $P(x_1, \ldots, x_n)$. Let $P(d_1, \ldots, d_n)$ denote the term obtained from $P(x_1, \ldots, x_n)$ by substituting values $d_1, \ldots, d_n$ for variables $x_1, \ldots, x_n$, respectively (d_i must be a value of the sort of x_i). Here, a value of a sort is represented by a term (constructor term) consisting only of special function symbols (constructors) of the sort. For example, in Table 1, TRUE, FALSE and 0, 1, 2, ... are constructor terms of sort bool and nn_integer, respectively. All state transition functions described in Section 2.2 are constructors of sort state. Therefore, constructor terms of sort state are INITIAL, TRANSMES(INITIAL), ...

For terms t and t' without variables and an axiom system E, we write $t \overset{E}{\equiv} t'$ if and only if t has the congruence relation [13] with t' under E.

We write $P(x_1, \ldots, x_n) \overset{E}{\equiv} \text{TRUE}$ and call $P(x_1, \ldots, x_n) \overset{E}{\equiv} \text{TRUE}$ "a theorem" of the axiom system E, if and only if it holds that $P(d_1, \ldots, d_n) \overset{E}{\equiv} \text{TRUE}$ for any value d_i of the sort of x_i ($1 \leqq i \leqq n$).

In our abstract sequential machine model, properties which we want to verify are described as the following form.

$[\text{Valid}(s) \supset Q(s, x_1, \ldots, x_n) \overset{E}{\equiv} \text{TRUE}]$

(here, s is a variable of sort state and x_i ($1 \leqq i \leqq n$) is a variable of a sort other than state.)

The verification is usually carried out by using structural induction on the depth of nesting of terms of sort state [5, 6, 10].

In Table 2, we list some theorems which we have already proved on the algebraic specification of SDT protocol in Table 1 [8]. For example, the following property (Theorem 11)

Valid(s) AND i(next_required(s) $\supset$ equal(rmb(s, i), SMB(i)) $\overset{E}{\equiv}$ TRUE

asserts that the contents of data received at the receiver equal to those sent from the sender for all data with sequence numbers less than the value of function next_required.

Table2 Theorems which we have proved on the algebraic specification of SDT protocol in Table 1

Theorem1 : Valid(s) AND timer_running(s, i)
⊃ highest_sent(s)≧i ≡ TRUE

Theorem2 : Valid(s) AND k(s)≧n AND n≧1
⊃ highest_sent(s)≧S(s, n) ≡ TRUE

Theorem3 : Valid(s) ⊃ lowest_unacked(s)+TWS>highest_sent(s)
≡ TRUE

Theorem4 : Valid(s) AND p(s)≧n AND n≧1
⊃ next_required(s)≧A(s, n) ≡ TRUE

Theorem5 : Valid(s) AND already_received(s, i)
⊃ highest_sent(s)≧i ≡ TRUE

Theorem6 : Valid(s) ⊃ highest_sent(s)+1≧next_required(s)
≡ TRUE

Theorem7 : Valid(s) ⊃ next_required(s)≧lowest_unacked(s)
≡ TRUE

Theorem8 : Valid(s) AND timer_running(s, i)
⊃ i≧lowest_unacked(s) ≡ TRUE

Theorem9 : Valid(s) AND k(s)≧n AND n≧1
⊃ equal(datapart(s, n), SMB(S(s, n)))
≡ TRUE

Theorem10: Valid(s) AND already_received(s, i)
⊃ equal(rmb(s, i), SMB(i)) ≡ TRUE

Theorem11: Valid(s) AND i(next_required(s)
⊃ equal(rmb(s, i), SMB(i)) ≡ TRUE

However, Theorem 11 doesn't assert that packets will be received by the receiver. In Section 3, we discuss PROGRESS property of SDT protocol.

3. VERIFICATION OF PROGRESS PROPERTY

In this section, we present a verification of "PROGRESS property" of SDT protocol that the packets sent out from the sender will be eventually received by the receiver.

In Section 3.2, we formulate the above PROGRESS property in terms of sequences of states in the abstract sequential machine. To verify a property of sequences of states, it is convenient to use the notations of Temporal logic [11] which provides operators for representing the present and future states. We introduce the concept of Temporal logic into our formalism.

3.1 TEMPORAL LOGIC

In Table 3, we describe the notion of Temporal logic algebraically. First, a new sort "formula" which denotes the set of formulas of Temporal logic is introduced into the algebraic specification of SDT protocol in Table 1. Next, three modal operators ("□"(always), "◇"(eventually), "○"(next)), logical symbols (TRUE, FALSE, AND, OR, NOT, ⊃) and some atomic predicate

Table 3 : An algebraic specification to verify PROGRESS property of SDT protocol (PART I)

```
text PROGRESS_PROPERTY ;
 include STENNING ;{the declaration to use the subtext STENNING}
     formula  -) formula  ≡ formula  | formula1 ;
     formula1 -) formula1 ⊃ formula1 | formula2 ;
     formula2 -) formula2 OR formula2 | formula3 ;
     formula3 -) atomic | TRUE | FALSE ;
              -) formula3 AND formula3 | NOT( formula ) ;
              -) □( formula ) | ◇( formula ) | ○( formula ) ;
              -) ( formula ) ;
     bool     -) Val( formula , SEQ ) ;
     state    -) car( SEQ ) ;
     SEQ      -) cdr( SEQ ) ;

     SEQ         w ;
     nn_integer  m ;
     state       s ;
     formula     x , y , z ;
     seq_number  i , j , n , h ;

IR1  :  □(x AND y) ≡ (□(x) AND □(y)) == TRUE ;
IR2  :  x ⊃ ◇(x) == TRUE ;
IR3  :  □(x ⊃ ○(x) OR ○(y)) ⊃ □(x ⊃ □(x) OR ◇(y)) == TRUE ;
IR4  :  □(x ⊃ y) ⊃ □(◇(x) ⊃ ◇(y)) == TRUE ;
IR5  :  ○(x) OR ○(y) ≡ ○(x OR y) == TRUE ;
IR6  :  ◇(x) ⊃ ◇(x OR y) == TRUE ;
IR7  :  ◇(□(x)) OR □(x) ⊃ ◇(□(x)) == TRUE ;
IR8  :  □(x ⊃ y) ⊃ (□(x) ⊃ □(y)) == TRUE ;
IR9  :  □(x AND ○(x) ⊃ y) ⊃  (□(x) ⊃ □(y)) == TRUE ;
IR10 :  □(x) ⊃ □(○(x)) == TRUE ;
IR11 :  NOT(○(x)) ⊃ ○(NOT(x)) == TRUE ;
IR12 :  □(○(x)) ⊃ ○(□(x)) == TRUE ;
IR13 :  □(x) ⊃ □(◇(x)) == TRUE ;
IR14 :  NOT(◇(x) AND □(NOT(x))) == TRUE ;
IR15 :  □(x OR y) ⊃ (◇(□(x)) OR ◇(□(y)) OR (□(◇(x)) AND □(◇(y)))) == TRUE ;
IR16 :  ◇(□(◇(x))) ≡ □(◇(x)) == TRUE ;
IR17 :  □(◇(○(x))) ≡ □(◇(x)) == TRUE ;
IR18 :  ◇(◇(x)) ≡ ◇(x) == TRUE ;
IR19 :  ◇(□(x AND y)) ≡ (◇(□(x)) AND ◇(□(y))) == TRUE ;
IR20 :  □(□(x) ⊃ ◇(y)) ≡ □(□(x) ⊃ □(◇(y))) == TRUE ;
IR21 :  □(□(◇(x)) ⊃ ◇(y)) ≡ □(□(◇(x)) ⊃ □(◇(y))) == TRUE ;
IR22 :  □(◇(x AND y)) ⊃ □(◇(x)) AND □(◇(y)) == TRUE ;

LR1  :  x ⊃ (x OR y) == TRUE ;
LR2  :  TRUE AND x == x ;
LR3  :  FALSE OR x == x ;
LR4  :  x AND y ⊃ x == TRUE ;
LR5  :  TRUE ⊃ x == TRUE ;
LR6  :  NOT(NOT(x)) == x ;
LR7  :  TRUE ≡ x == x ;
LR8  :  (x ⊃ (y ⊃ z)) ≡ ((x AND y) ⊃ z) == TRUE ;
LR9  :  ((x ⊃ z) AND (y ⊃ z)) ≡ ((x OR y) ⊃ z) == TRUE ;
LR10 :  ((x ⊃ y) AND (y ⊃ z)) ⊃ (x ⊃ z) == TRUE ;
LR11 :  NOT(x OR y)  ≡ (NOT(x) AND NOT(y)) == TRUE ;
LR12 :  NOT(x AND y) ≡ (NOT(x) OR  NOT(y)) == TRUE ;
LR13 :  x AND y == y AND x ;
LR14 :  x OR  y == y OR  x ;
LR15 :  x ≡  y == y ≡  x ;
LR16 :  □(TRUE) == TRUE ;

VL1  :  Val(TRUE, w)      == TRUE ;
VL2  :  Val(FALSE, w)     == FALSE ;
VL3  :  Val(x AND y, w) == Val(x, w) AND Val(y, w) ;
VL4  :  Val(x OR  y, w) == Val(x, w) OR  Val(y, w) ;
VL5  :  Val(NOT(x) , w) == NOT(Val(x, w)) ;
VL6  :  Val(x ⊃  y, w) == Val(x, w) ⊃  Val(y, w) ;
```

Table 3 : An algebraic specification to verify PROGRESS property of SDT protocol (PART II)

```
VL7  :  Val(□(x) , w)  == Val(x,w) AND Val(□(x),cdr(w)) ;
VL8  :  Val(◇(x) , w)  == Val(x,w) OR  Val(◇(x),cdr(w)) ;
VL9  :  Val(○(x) , w)  == Val(x,cdr(w)) ;
VL10 :  Val((x) , w)   == Val(x,w) ;
VL11 :  Val(x ≡ y, w)  == (Val(x,w) ⊃ Val(y,w)) AND (Val(y,w) ⊃ Val(x,w)) ;

   for each 'Q' in ( 'TRANSM' , 'RETRANSM[ seq_number ]' , 'RECA[ seq_number ]' ,
                     'FINDTO[ seq_number ]' , 'RECM&TRANSA[ seq_number , seq_number ]' ) ,
   begin
     atomic -) last_Q ;
            -) ok_Q ;
     bool   -) last_Q( state ) ;
            -) ok_Q( state ) ;
   end;
   for each 'Q' in ( 'INITIAL' , 'RETRANSM(s,i)' , 'RECA(s,i)' , 'FINDTO(s,i)' ,
                     'RECM&TRANSA(s,i,j)' ) ,
     LA1  :  last_TRANSM(Q) == FALSE ;
   for each 'Q' in ( 'INITIAL' , 'TRANSM(s)' , 'RECA(s,i)' , 'FINDTO(s,i)' ,
                     'RECM&TRANSA(s,i,j)' ) ,
     LA2  :  last_RETRANSM[n](Q) == FALSE ;
   for each 'Q' in ( 'INITIAL' , 'TRANSM(s)' , 'RETRANSM(s,i)' , 'FINDTO(s,i)' ,
                     'RECM&TRANSA(s,i,j)' ) ,
     LA3  :  last_RECA[n](Q) == FALSE ;
   for each 'Q' in ( 'INITIAL' , 'TRANSM(s)' , 'RETRANSM(s,i)' , 'RECA(s,i)' ,
                     'RECM&TRANSA(s,i,j)' ) ,
     LA4  :  last_FINDTO[n](Q) == FALSE ;
   for each 'Q' in ( 'INITIAL' , 'TRANSM(s)' , 'RETRANSM(s,i)' , 'RECA(s,i)' ,
                     'FINDTO(s,i)' ) ,
     LA5  :  last_RECM&TRANSA[n,h](Q) == FALSE ;
LA6  :  last_TRANSM(TRANSM(s)) == TRUE ;
LA7  :  last_RETRANSM[n](RETRANSM(s,i)) == n=i ;
LA8  :  last_RECA[n](RECA(s,i)) == n=i ;
LA9  :  last_FINDTO[n](FINDTO(s,i)) == n=i ;
LA10 :  last_RECM&TRANSA[n,h](RECM&TRANSA(s,i,j)) == (n=i) AND (h=j);
OK1  :  ok_TRANSM(s) == Valid(TRANSM(s)) ;
OK2  :  ok_RETRANSM[n](s) == Valid(RETRANSM(s,n)) ;
OK3  :  ok_RECA[n](s) == Valid(RECA(s,n)) ;
OK4  :  ok_FINDTO[n](s) == Valid(FINDTO(s,n)) ;
OK5  :  ok_RECM&TRANSA[n,h](s) == Valid(RECM&TRANSA(s,n,h)) ;
   for each 'Q' in ( 'TRANSM' , 'RETRANSM[i]' , 'RECA[i]' , 'FINDTO[i]' , 'RECM&TRANSA[i,j]' ) ,
   begin
     AT1  :  Val(last_Q,w) == last_Q(car(w)) ;
     AT2  :  Val(ok_Q,w)   == ok_Q(car(w)) ;
   end;
   for each "R" in ( "Q" , "Q'" , "Q1" , "Q2" , "Q3" , "Q4" , "Q5" ) ,
   begin
     atomic -) R[ seq_number ] ;
     bool   -) R( state , seq_number ) ;
     AT3  :  Val(R[i],w) == R(car(w),i) ;
   end;
AT4  :  Q(s,i)  == Valid(s) AND lowest_unacked(s)=i ;
AT5  :  Q'(s,i) == Valid(s) AND lowest_unacked(s)>i ;
AT6  :  Q1(s,i) == Valid(s) AND lowest_unacked(s)=i AND highest_sent(s)+1<=i ;
AT7  :  Q2(s,i) == Valid(s) AND lowest_unacked(s)=i AND highest_sent(s)+1>i
                            AND timer_running(s,i)  AND NOT(already_received(s,i)) ;
AT8  :  Q3(s,i) == Valid(s) AND lowest_unacked(s)=i AND highest_sent(s)+1>i
                            AND NOT(timer_running(s,i)) AND NOT(already_received(s,i)) ;
AT9  :  Q4(s,i) == Valid(s) AND lowest_unacked(s)=i AND highest_sent(s)+1>i
                            AND timer_running(s,i)  AND already_received(s,i) ;
AT10 :  Q5(s,i) == Valid(s) AND lowest_unacked(s)=i AND highest_sent(s)+1>i
                            AND NOT(timer_running(s,i)) AND already_received(s,i) ;
end PROGRESS_PROPERTY ;
```

symbols (details are described in Section 3.2.2 and 3.2.3) are introduced as functions whose range are sort formula. A formula of Temporal logic is represented by a term consisting of only above modal operators, logical symbols and atomic predicate symbols. And several inference rules of Temporal logic are introduced as axioms (the axioms IR1,...,IR22 and LR1,...,LR16).

In our formalism, if P≒TRUE holds for a term P of sort formula, then the formula P is shown to be TRUE by using typical inference rules of Temporal logic (for example, those in [11]).

Next, we define sequences of states and the interpretation of atomic predicate symbols (the values of atomic predicate symbols for each sequence of states). For convenience sake, we use the following notations:

For a state transition function g (g≠INITIAL) with k parameters whose sorts are $s_1,\ldots,s_k$, a value (term) t of sort state (for short, state t) and values $a_1,\ldots,a_k$ of sorts $s_1,\ldots,s_k$, respectively, let $g[a_1,\ldots,a_k](t)$ denote $g(t,a_1,\ldots,a_k)$. We call each $g[a_1,\ldots,a_k]$ "γ-function". Let Γ denote the set consisting of all γ-functions (in general, Γ may be infinite).

We call state t satisfying Valid(t)≒TRUE "a valid state". If t and $\alpha(t)$ ($\alpha\in\Gamma$) are both valid states, then we denote this by $t \Rightarrow \alpha(t)$ and call $\alpha(t)$ a next state of state t.

In general, there doesn't necessarily exist at least one next state for any valid state t. But, in case of SDT protocol, there exists at least one next state for any valid state t. That is, the following property holds :

$$\exists \alpha\in\Gamma \ \{\mathrm{Valid}(t) \supset \mathrm{Valid}(\alpha(t)) \not\equiv \mathrm{TRUE}\} \qquad \text{---(3-1)}$$

We have verified this property by using structural induction on the depth of nesting of terms of sort state [9].

[The sequences of states]

If $\forall i(i\geqq 0)\ s_i \Rightarrow s_{i+1}$ holds for a infinite sequence $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$ of states, then we call w "a valid sequence". In Table 3, a new sort "SEQ" which denotes the set of valid sequences are introduced. We assume that all states in $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$ are valid states, and that $\forall i(i\geqq 0)\ s_i \Rightarrow s_{i+1}$ holds for the given infinite sequence $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$ of states. We call the first state s_0 in a valid sequence $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$ "a present state" and denote it by car(w).

SEQ isn't empty by property (3-1).

[Atomic predicate symbols and their values]

Suppose that a term $P(s,a_1,a_2,\ldots,a_k)$ of sort bool satisfies the following (1) and (2).

(1) s is a variable of sort state.

(2) $a_1,a_2,\ldots,a_k$ are values whose sorts are other than sort state.

We introduce $P[a_1,\ldots,a_k]$ as an atomic predicate symbol if the value of the term $P(t,a_1,\ldots,a_k)$ is defined as TRUE or FALSE for any valid state t. The value of an atomic predicate symbol $P[a_1,\ldots,a_k]$ is defined as follows:

For a infinite list $w(\triangleq[s_0 s_1 s_2 \cdots s_n \cdots])$ of states, let Val(P,w) denote the value of the formula P for the infinite list w of states, and cdr(w) denote the infinite list $[s_1 s_2 \cdots s_n \cdots]$ of states which is removed the present state s_0 from the infinite list $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$ of states. In this paper, for any atomic predicate symbol $P[a_1,\ldots,a_k]$ and valid sequence $w(\triangleq[s_0 s_1 \cdots s_n \cdots])$, $\mathrm{Val}(P[a_1,\ldots,a_k],w)$ is defined as the value of $P(s_0,a_1,\ldots,a_k)$.

We also define the value of a formula which contains modal operators and/or logical symbols as follows:

We define that Val(○(P),w) is equivalent to Val(P,cdr(w)) for any formula P. And, we also

define that Val(□(P),w) is equivalent to Val(P,w) AND Val(□(P),cdr(w)), and that Val(◇(P),w) is equivalent to Val(P,w) OR Val(◇(P),cdr(w)). These definitions of the values of formulas are described as the axioms VL1,VL2,...,VL11 in Table 3. If the value of a formula (term of sort formula) Q for any valid sequence (term of sort SEQ) is TRUE, then we denote it by "Val(Q,w)≧TRUE" (w is a variable of sort SEQ).

In Table 3, seven kinds of atomic predicate symbols (Q[i], Q'[i],Q1[i],Q2[i],Q3[i],Q4[i],Q5[i]) are introduced. The meanings of these atomic predicate symbols are described in Section 3.2.2 and 3.2.3.

For example, since we define Q(s,i) as follows,

Q(s,i) ≜ Valid(s) and lowest_unacked(s)=i

then Q[1], Q[2],... are atomic predicate symbols. Q[i] is an atomic predicate symbol which indicate whether the value of lowest_unacked at present state is "i" or not.

We also introduce two kinds of special atomic predicate symbols "last_α" and "ok_α". The meanings of last_α and ok_α are as follows :

(A) last_α : The predicate symbol which indicates whether the last action at the present state is α or not. The value of last_α is defined as the axioms LA1,LA2,...,LA10 in Table 3.

(B) ok_α : The predicate symbol which indicates whether the execution of action α is allowed at the present state or not. We define the value of ok_α as the axioms OK1,OK2,...,OK5 in Table 3.

3.2 THE PROCESS OF THE VERIFICATION OF PROGRESS PROPERTY

In this section, we explain the process of verifying PROGRESS property of SDT protocol.

3.2.1 PROGRESS PROPERTY OF SDT PROTOCOL

The axioms of the algebraic specification (Table 1) of SDT protocol and its theorems (Table 2) state that every packet with a sequence number less than the value of "next_required" has been received correctly by the receiver but the packet with the sequence number equal to the value of "next_required" has not yet received. Hence, we can express the property that "SDT protocol will progress" as follows :

"The value of function "next_required" grows unboundedly."

Theorem 7 in Table 2 states that the value of function next_required is always greater than or equal to that of function lowest_unacked which denotes the lowest sequence number that the sender has sent the packet of the sequence number but not yet received the acknowledgement of reception of the packet from the receiver. Therefore, if the value of function lowest_unacked grows unboundedly, the value of function next_required does, too. In this paper, we show the value of function lowest_unacked grows unboundedly.

But, in general, a specification of a protocol doesn't contain the following hypotheses (A1) and (A2) for the stations and the communication lines, respectively :

(A1)If the transmission/retransmission of a packet with the same sequence number (whose sequence number is identical) is allowed repeatedly, then the action transmission/retransmission will be eventually done.

(A2)If packets with the same sequence number have been retransmitted repeatedly, at least one of them will be eventually received by the receiver.

In order to verify PROGRESS property, such hypotheses as (A1) and (A2) are necessary. In Section 3.2.2, these hypotheses are formulated algebraically. In Sections 3.2.3 and 3.2.4, we

show the following two properties :

(1) Consistency of the hypotheses that there exists at least a valid sequence satisfying the given hypotheses.

(2) PROGRESS property that the value of function lowest_unacked grows unboundedly for any valid sequence satisfying the given hypotheses.

3.2.2 THE HYPOTHESES FOR THE STATIONS AND COMMUNICATION LINES

In this section, we describe how to express the hypotheses for the stations and communication lines algebraically.

There are two types of actions at each station in SDT protocol. One is an active action such as "transmission" or "retransmission", and the other is a passive action such as "reception".

In general, for the execution of a passive action (such as "reception" in a station), it is necessary to have executed some active action (such as transmission in the other station) in advance. We call such an active action a "Λ-action" for the passive one. TRANSM and RETRANSM[i] are Λ-actions for RECM&TRANSA[i,j] and RECM&TRANSA[i,j] is an Λ-action for RECA[j].

In this paper, we give the following two hypotheses.

(B1)If the execution of an active action such as transmission is allowed repeatedly, then the action will be done eventually.

(B2)If the Λ-action for a passive action has been done repeatedly and the execution of the passive action is allowed repeatedly, then the passive action will be done eventually. []

Let I denote the set of sequence numbers. By using predicate symbols last_α and ok_α introduced in Section 3.1, we express the above hypotheses as follows:

[The formal definition of the hypothesis (B1)]

If a given valid sequence w satisfies the hypothesis (B1), then

$$Val(\Box(\Box(\Diamond(ok\_\alpha)) \supset \Diamond(last\_\alpha)), w) \Eqv TRUE$$

holds for any active action $\alpha \in \{TRANSM, FINDTO[i], RETRANSM[i] \mid i \in I\}$. That is, the following (B-1), (B-2) and (B-3) hold for any valid sequence w satisfying hypothesis (B1):

(B-1) $Val(\Box(\Box(\Diamond(ok\_TRANSM)) \supset \Diamond(last\_TRANSM)), w) \Eqv TRUE$

(B-2) $\forall i \in I\{ Val(\Box(\Box(\Diamond(ok\_RETRANSM[i])) \supset \Diamond(last\_RETRANSM[i])), w) \Eqv TRUE \}$

(B-3) $\forall i \in I\{ Val(\Box(\Box(\Diamond(ok\_FINDTO[i])) \supset \Diamond(last\_FINDTO[i])), w) \Eqv TRUE \}$

From now, the above (B-1), (B-2) and (B-3) are abbreviated as $Val(B_1, w) \Eqv TRUE$, $\forall i \in I\ \{Val(B_2[i], w) \Eqv TRUE\}$, $\forall i \in I\{Val(B_3[i], w) \Eqv TRUE\}$, respectively.

[The formal definition of the hypothesis (B2)]

If a given valid sequence w satisfies the hypothesis (B2), then

$$Val(\Box(\Box(\Diamond(last\_\alpha)) \text{ AND } \Box(\Diamond(ok\_\beta)) \supset \Diamond(last\_\beta)), w) \Eqv TRUE$$

holds for any passive action $\beta \in \{RECM\&TRANSA[i,j], RECA[j] \mid i,j \in I\}$ and each Λ-action α (executable repeatedly) for β. That is, if hypothesis (B2) holds, then the following (B-4) and (B-5) hold for any valid sequence w satisfying hypothesis (B2):

(B-4) $\forall i,j \in I$ {

$$Val(\Box(\Box(\Diamond(last\_RETRANSM[i])) \text{ AND } \Box(\Diamond(ok\_RECM\&TRANSA[i,j])) \supset \Diamond(last\_RECM\&TRANSA[i,j])), w) \Eqv TRUE \}$$

(B-5) $\forall i, j \in I$ {

Val(□(□(◇(last_RECM&TRANSA[i, j])) AND

□(◇(ok_RECA[j])) ⊃ ◇(last_RECA[j])), w)≧TRUE }

Hereafter, the above (B-4) and (B-5) are abbreviated as $\forall i, j \in I$ {Val(B_4[i, j], w)≧TRUE}, $\forall i, j \in I$ {Val(B_5[i, j], w)≧TRUE}, respectively.

In case of SDT protocol, the receiver sends the acknowledgement whose value is between i and i+TWS (TWS means the transmitter window size (see Section 2.2)) when the receiver has received the packet with a sequence number "i". Let Val(B[i], w) denote the logical product of Val(B_1, w), Val(B_2[i], w), Val(B_3[i], w), Val(B_4[i, i], w), Val(B_4[i, i+1], w), ..., Val(B_4[i, i+TWS], w), Val(B_5[i, i], w), Val(B_5[i, i+1], w), ..., Val(B_5[i, i+TWS], w). If the hypotheses (B-1) to (B-5) hold, then $\forall k \in I$ {Val(B[k], w)≧TRUE} holds. We define that if Val(B[i], w)≧TRUE holds for a valid sequence w, then the valid sequence w satisfies the hypotheses necessary for the transmission/reception of the packet with a sequence number "i".

In this paper, we prove PROGRESS property of SDT protocol by showing that the following condition [PR] holds.

[PR] For any valid sequence w (≜[$s_0 s_1 \cdots s_m \cdots s_n \cdots$]) such that Val(B[j], w)≧TRUE holds for any sequence number j in I, the following (A) and (B) hold.

(A) There exists a sequence number i such that lowest_unacked(s_0)=i≧TRUE.

(B) For any sequence number i and valid state s_m in w such that lowest_unacked(s_m)=i≧TRUE holds, there exists a valid state s_n in w such that m(n and lowest_unacked(s_n))i≧TRUE.

From the axioms AX2, AX17, AX18 and AX43 in Table 1, the value of lowest_unacked(s) is defined for any valid state s. Then, for any valid state s_0, there exists a sequence number "i" such that lowest_unacked(s_0)=i≧TRUE holds. That is, the condition (A) holds.

Let Q[i] and Q'[i] denote the atomic predicate symbols such that

Val(Q[i], w) ≧Q(car(w), i)

Q(t, i) ≜ Valid(t) AND lowest_unacked(t)=i ---(3-2)

Val(Q'[i], w)≧Q'(car(w), i)

Q'(t, i) ≜ Valid(t) AND lowest_unacked(t)>i ---(3-3)

and let C[i] denote the formula such that

C[i] ≜ □(Q[i] ⊃ ◇(Q'[i])) ---(3-4)

We formulate the condition (B) as follows :

For any valid sequence w such that $\forall j \in I$ {Val(B[j], w)≧TRUE} holds,

Val(C[i], w)≧TRUE holds. ---(3-5)

In Sections 3.2.3 and 3.2.4, we prove (3-6) as the sufficient condition for (3-5).

Val(B[i], w) ⊃ Val(C[i], w)≧TRUE ---(3-6)

3.2.3 LEMMAS NECESSARY FOR THE VERIFICATION OF PROGRESS PROPERTY OF SDT PROTOCOL

In this section, we describe how to find out the lemmas necessary for the verification of PROGRESS property of SDT protocol and present them in the form of "Val(A_h[i], w)≧TRUE".

First, we classify valid states where the values of function "lowest_unacked" are equal to some sequence number (say, "i"). That is, we split the set {t|Q(t, i)≧TRUE} as follows :

(A) The subset where the packet with sequence number "i" hasn't yet been transmitted from the sender.

(B) The subset where the packet with sequence number "i" has been already transmitted but not

yet been received by the receiver.

(C) The subset where the packet with sequence number "i" has been already received by the receiver and the acknowledgement has already been sent by the receiver, but hasn't yet been received by the sender.

Furthermore, we split (B) (or (C)) into the following (B1) and (B2) (or (C1) and (C2)) depending on whether timeout has occurred or not.

(B1) The subset of (B) where timeout hasn't occurred.

(B2) The subset of (B) where timeout has occurred. The sender is preparing retransmission of the packet with a sequence number "i".

(C1) The subset of (C) where timeout hasn't occurred.

(C2) The subset of (C) where timeout has occurred.

We represent the these five subsets (A),(B1),(B2),(C1),(C2) as Situation(1),...,Situation(5), respectively.

We formulate the these five subsets using the following five predicates Q1(t,i),..., Q5(t,i) such that for each k ($1 \leqq k \leqq 5$) and state t, Qk(t,i)$\equiv$TRUE holds if and only if t is in Situation(k).

(1)Q1(t,i) $\triangleq$ Valid(t) AND lowest_unacked(t)=i
AND highest_sent(t)+1$\leqq$i

(2)Q2(t,i) $\triangleq$ Valid(t) AND lowest_unacked(t)=i
AND highest_sent(t)+1>i AND timer_running(t,i)
AND NOT(already_received(t,i))

(3)Q3(t,i) $\triangleq$ Valid(t) AND lowest_unacked(t)=i
AND highest_sent(t)+1>i AND NOT(timer_running(t,i))
AND NOT(already_received(t,i))

(4)Q4(t,i) $\triangleq$ Valid(t) AND lowest_unacked(t)=i
AND highest_sent(t)+1>i AND timer_running(t,i)
AND already_received(t,i)

(5)Q5(t,i) $\triangleq$ Valid(t) AND lowest_unacked(t)=i
AND highest_sent(t)+1>i AND NOT(timer_running(t,i))
AND already_received(t,i) []

Since Q1(t,i) OR Q2(t,i) OR ... OR Q5(t,i) $\equiv$ Valid(t) AND lowest_unacked(t)=i, it follows from the definition of Q(t,i) that

Q(t,i) $\equiv$ Q1(t,i) OR Q2(t,i) OR ... OR Q5(t,i) ---(3-7)

We introduce new situation (named Situation(6)) where the value of lowest_unacked(t) is greater than "i" at any valid state t in Situation(6). From (3-3) in Section 3.2.2, Q'(t,i)$\equiv$TRUE holds for any state t in Situation(6).

Now, let C'[i] be defined by

C'[i] $\triangleq$ $\Box$(Q1[i] OR Q2[i] OR ... OR Q5[i] $\supset$ $\Diamond$Q'[i])

Val(Qk[i],w) $\triangleq$ Qk(car(w),i) ($1 \leqq k \leqq 5$)

From (3-4), (3-6) and (3-7), "Val(B[i],w)$\supset$Val(C[i],w)$\equiv$TRUE" (C[i]$\triangleq\Box$(Q[i]$\supset\Diamond$Q'[i])) holds if and only if "Val(B[i],w)$\supset$Val(C'[i],w)$\equiv$TRUE" holds.

For any two Situation(i) and Situation(j) such that $i \neq j$, we examine whether there exists a γ-function α such that for any state t in Situation(i), α(t) belongs to Situation(j). If there exists such a γ-function, then we say that Situation(j) is a successor of Situation(i).

The following 〈1〉 to 〈3〉 are the theorems in the algebraic specification of SDT protocol

which represent what Situations are successors of Situation(1). These theorems have been proved formally by using our interactive verification support system [10].

〈1〉 [Q1(t,i) ⊃ ok_TRANSM(t) ≡ TRUE]

〈2〉 [Q1(t,i) ⊃ Q2(TRANSM(t),i) ≡ TRUE]

〈3〉 For any γ-function α

[Q1(t,i) ⊃ {NOT(Valid(α(t))) OR

Q1(α(t),i) OR Q2(α(t),i)} ≡ TRUE] []

Theorems 〈1〉 and 〈2〉 represent that the action of transmission (TRANSM) is allowed for any state t in Situation(1), and that if TRANSM is done, then the state just after the execution of TRANSM belongs to Situation(2). And theorem 〈3〉 represents that if any valid action is done, then the state just after the execution of the action belongs to Situation(1) itself or its successor Situation(2).

In this paper, we transform the following three types of theorems 〈A〉, 〈B〉 and 〈C〉 into the following theorems 〈A'〉, 〈B'〉 and 〈C'〉, respectively.

〈A〉 [P(t,i) ≡ TRUE]

〈B〉 [P(t,i) ⊃ P'(α(t),i) ≡ TRUE]

〈C〉 For any γ-function α

[P(t,i) ⊃ P'(α(t),i) ≡ TRUE]

〈A'〉 [Val(□(P[i]),w)≡TRUE]

〈B'〉 [Val(□(P[i] AND ○(last_α) ⊃ ○(P'[i])),w)≡TRUE]

〈C'〉 [Val(□(P[i] ⊃ ○(P'[i])),w)≡TRUE] []

The soundness of these transformation rules follows from the axioms in Table 3.

For example, the above theorems 〈1〉 to 〈3〉 are transformed into the following theorems.

〈1'〉 [Val(□(Q1[i] ⊃ ok_TRANSM),w)≡TRUE]

〈2'〉 [Val(□(Q1[i] AND ○(last_TRANSM) ⊃ ○(Q2[i])),w)≡TRUE]

〈3'〉 [Val(□(Q1[i] ⊃ (○(Q1[i]) OR ○(Q2[i]))),w)≡TRUE]

We have also proved the theorems of types 〈A〉, 〈B〉 and 〈C〉 related to Situation(2) to Situation(5), and then transformed them into the theorems of types 〈A'〉, 〈B'〉 and 〈C'〉. In Table 4, we give the list of 30 theorems written in the form of "Val(A_h[i],w)≡TRUE"($1 \leqq h \leqq 30$).

3.2.4 THE PROCESS OF VERIFICATION

In this section, we describe how to prove the consistency of hypotheses and PROGRESS property described in (3-6) in Section 3.2.2.

For the consistency of the hypotheses, we give the valid sequence W which is described below.

Now, let f (≜$f_0 f_1 f_2 \cdots f_{n-1} f_n \cdots$) denote the sequence of γ-functions such that

f ≜ INITIAL

TRANSM RECM&TRANSA[1,1] RECA[1]

TRANSM RECM&TRANSA[2,2] RECA[2]

.....

TRANSM RECM&TRANSA[m,m] RECA[m]

..... []

and let W(≜$s_0 s_1 s_2 \cdots s_n \cdots$) denote the sequence of states such that

W ≜ f_0 $f_1(f_0)$ $f_2(f_1(f_0))$ ... $f_n(f_{n-1}(\cdots(f_1(f_0))\cdots))$ ...

Table 4: Lemmas used for verifying PROGRESS property of SDT protocol

Lemma 1: Val(□(Q1[i]⊃ok_TRANSM), w)≧TRUE

Lemma 2: Val(□(Q1[i] AND ○(last_TRANSM)⊃○(Q2[i])), w)≧TRUE

Lemma 3: Val(□(Q1[i]⊃(○(Q1[i]) OR ○(Q2[i]))), w)≧TRUE

Lemma 4: Val(□(Q2[i]⊃ok_FINDTO[i]), w)≧TRUE

Lemma 5: Val(□(Q2[i]⊃(ok_RECM&TRANSA[i, i] OR ok_RECM&TRANSA[i, i+1]
OR ····· OR ok_RECM&TRANSA[i, i+TWS])), w)≧TRUE

Lemma 6: Val(□(Q2[i] AND ○(last_FINDTO[i])⊃○(Q3[i])), w)≧TRUE

Lemma 7: Val(□(Q2[i] AND (last_RECM&TRANSA[i, i] OR last_RECM&TRANSA[i, i+1]
OR ····· OR last_RECM&TRANSA[i, i+TWS])⊃○(Q4[i])), w)≧TRUE

Lemma 8: Val(□(Q2[i]⊃(○(Q2[i]) OR ○(Q3[i]) OR ○(Q4[i]))), w)≧TRUE

Lemma 9: Val(□(Q3[i]⊃ok_RETRANSM[i]), w)≧TRUE

Lemma10: Val(□(Q3[i]⊃(ok_RECM&TRANSA[i, i] OR ok_RECM&TRANSA[i, i+1]
OR ····· OR ok_RECM&TRANSA[i, i+TWS])), w)≧TRUE

Lemma11: Val(□(Q3[i] AND ○(last_RETRANSM[i])⊃○(Q2[i])), w)≧TRUE

Lemma12: Val(□(Q3[i] AND (last_RECM&TRANSA[i, i] OR last_RECM&TRANSA[i, i+1]
OR ····· OR last_RECM&TRANSA[i, i+TWS])⊃○(Q5[i])), w)≧TRUE

Lemma13: Val(□(Q3[i]⊃(○(Q2[i]) OR ○(Q3[i]) OR ○(Q5[i]))), w)≧TRUE

Lemma14: Val(□(Q4[i]⊃ok_FINDTO[i]), w)≧TRUE

Lemma15: Val(□(Q4[i]⊃(ok_RECM&TRANSA[i, i] OR ok_RECM&TRANSA[i, i+1]
OR ····· OR ok_RECM&TRANSA[i, i+TWS])), w)≧TRUE

Lemma16: Val(□(Q4[i]⊃(ok_RECA[i] OR ok_RECA[i+1]
OR ····· OR ok_RECA[i+TWS])), w)≧TRUE

Lemma17: Val(□(Q4[i] AND ○(last_FINDTO[i]) ⊃ ○(Q5[i])), w)≧TRUE

Lemma18: Val(□(Q4[i] AND (○(last_RECA[i]) OR ○(last_RECA[i+1])
OR ····· OR ○(last_RECA[i+TWS]))⊃○(Q'[i])), w)≧TRUE

Lemma19: Val(□(Q4[i]⊃(○(Q4[i]) OR ○(Q5[i]) OR ○(Q'[i]))), w)≧TRUE

Lemma20: Val(□(Q5[i]⊃ok_RETRANSM[i]), w)≧TRUE

Lemma21: Val(□(Q5[i]⊃(ok_RECM&TRANSA[i, i] OR ok_RECM&TRANSA[i, i+1]
OR ····· OR ok_RECM&TRANSA[i, i+TWS])), w)≧TRUE

Lemma22: Val(□(Q5[i]⊃(ok_RECA[i] OR ok_RECA[i+1]
OR ····· OR ok_RECA[i+TWS])), w)≧TRUE

Lemma23: Val(□(Q5[i] AND ○(last_RETRANSM[i])⊃○(Q4[i])), w)≧TRUE

Lemma24: Val(□(Q5[i] AND (○(last_RECA[i]) OR ○(last_RECA[i+1])
OR ····· OR ○(last_RECA[i+TWS]))⊃○(Q'[i])), w)≧TRUE

Lemma25: Val(□(Q5[i]⊃(○(Q4[i]) OR ○(Q5[i]) OR ○(Q'[i]))), w)≧TRUE

Lemma26: Val(□(Q2[i]⊃NOT(Q1[i]) AND NOT(Q3[i])), w)≧TRUE

Lemma27: Val(□(Q3[i]⊃NOT(Q1[i]) AND NOT(Q2[i])), w)≧TRUE

Lemma28: Val(□(Q4[i]⊃NOT(Q2[i]) AND NOT(Q3[i]) AND NOT(Q5[i])), w)≧TRUE

Lemma29: Val(□(Q5[i]⊃NOT(Q2[i]) AND NOT(Q3[i]) AND NOT(Q4[i])), w)≧TRUE

Lemma30: Val(□(Q'[i]⊃NOT(Q4[i]) AND NOT(Q5[i])), w)≧TRUE

That is, $\mathbb{W}$ denotes the sequence of states which are reachable from the initial state INITIAL by executing the actions $f_1, f_2, \ldots,$ successively. We can prove that $\mathbb{W}$ is a valid sequence, and that $\mathbb{W}$ is satisfied all hypotheses on stations and communication lines[9].

For verifying PROGRESS property (Val(B[i],w) $\supset$ Val(C'[i],w)$\eqsim$TRUE), we introduce constant K which represents an arbitrary sequence number, and verify

Val(B[K],w) $\supset$ Val(C'[K],w) $\eqsim$ TRUE. ----(3-8)

Even if B[K] and C'[K] are replaced by, say, B[10] and C'[10], respectively, the process of the verification is the same. Thus, we conclude that Val(B[i],w) $\supset$ Val(C'[i],w)$\eqsim$TRUE holds.

To verify (3-8), we use 30 lemmas (Val(A_h[i],w)$\eqsim$TRUE ($1 \leqq h \leqq 30$)) described in Section 3.2.3. Let E' denote the axiom system such that

E' $\triangleq$ {axioms IR1,...IR22, LR1,...,LR16, VL1,...,VL11 of the axiom system in Table 3}
+ {Val(A_h[K],w)==TRUE | $1 \leqq h \leqq 30$} + {Val(B_j[K],w)==TRUE | $1 \leqq j \leqq 5$}.

If we can show that

Val(C'[K],w)) $\eqsim$ TRUE ----(3-9)

holds by using axiom system E', then we can conclude that (3-8) holds.

The outline of the inference process to show that (3-9) holds is as follows :

[INFERENCE PROCESS]

By "Val(A_3[K],w)==TRUE" in E' and inference rule IR3 and VL6 in Table 3, it follows that

Val($\Box(Q_1[K] \supset \Box(Q_1[K])$ OR $\Diamond(Q_2[K]))$,w)$\eqsim$TRUE --- F_1

By "Val(A_1[K],w)==TRUE" and "Val(B_1,w)==TRUE" in E' and several inference rules in Table 3, it follows that

Val($\Box(\Box(Q_1[K]) \supset \Diamond$(last_TRANSM)),w)$\eqsim$TRUE --- F_2

By "Val(A_2[K],w)==TRUE" in E' and several inference rules, it follows that

Val($\Box(\Box(Q_1[K])$ AND $\Diamond$(last_TRANSM)$\supset \Diamond(Q_2[K]))$,w)$\eqsim$TRUE --- F_3

By F_1, F_2, F_3, it follows that

Val($\Box(Q_1[K] \supset \Diamond(Q_2[K]))$,w)$\eqsim$TRUE --- F_4

Similarly, followings hold:

By "Val(A_8[K],w)==TRUE" in E' and several inference rules, it follows that

Val($\Box(Q_2[K] \supset \Box(Q_2[K])$ OR $\Diamond(Q_3[K])$ OR $\Diamond(Q_4[K]))$,w)$\eqsim$TRUE --- F_5

By "Val(A_{13}[K],w)==TRUE" in E' and several inference rules, it follows that

Val($\Box(Q_3[K] \supset \Box(Q_3[K])$ OR $\Diamond(Q_2[K])$ OR $\Diamond(Q_5[K]))$,w)$\eqsim$TRUE --- F_6

By F_5 and F_6, it holds that

Val($\Box((Q_2[K]$ OR $Q_3[K]) \supset$
$\Diamond(\Box(Q_2[K]))$ OR $\Diamond(\Box(Q_3[K]))$
OR ($\Box(\Diamond(Q_2[K]))$ AND $\Box(\Diamond(Q_3[K]))$)
OR $\Diamond(Q_4[K])$ OR $\Diamond(Q_5[K]))$,w)$\eqsim$TRUE --- F_7

By applying "{Val(A_h[K],w)==TRUE | h=4, 6, 27}" and "Val(B_3,w)==TRUE" in E' to F_7, F_7 is simplified as follows:

Val($\Box((Q_2[K]$ OR $Q_3[K]) \supset$
$\Diamond(\Box(Q_3[K]))$ OR ($\Box(\Diamond(Q_2[K]))$ AND $\Box(\Diamond(Q_3[K]))$)
OR $\Diamond(Q_4[K])$ OR $\Diamond(Q_5[K]))$,w)$\eqsim$TRUE --- F_8

Furthermore, by applying "{Val(A_h[K],w)==TRUE | h=9, 11, 26}" and "Val(B_2,w)==TRUE" in E' to F_8, it reduces to

Val($\Box((Q_2[K]$ OR $Q_3[K]) \supset$
($\Box(\Diamond(Q_2[K]))$ AND $\Box(\Diamond(Q_3[K]))$)

$$\text{OR } \Diamond(Q_4[K]) \text{ OR } \Diamond(Q_5[K])), w) \not\equiv \text{TRUE} \quad \text{--- } F_9$$

By applying "{Val(A_h[K], w)==TRUE|h=5, 7, 10, 12, 28, 29}" and "Val(B_4, w)==TRUE" in E' to F_9, it reduces to

$$\text{Val}(\Box((Q_2[K] \text{ OR } Q_3[K]) \supset \text{OR } \Diamond(Q_4[K]) \text{ OR } \Diamond(Q_5[K])), w) \not\equiv \text{TRUE} \quad \text{--- } F_{10}$$

Similarly, by using "{Val(A_h[K], w)==TRUE|h=4, . . 30}" and "{Val(B_j, w)==TRUE| j=2, 3, 4, 5}" in E', we have

$$\text{Val}(\Box((Q_4[K] \text{ OR } Q_5[K]) \supset \Diamond(Q'[K])), w) \not\equiv \text{TRUE} \quad \text{--- } F_{11}$$

By F_4, F_{10} and F_{11}, it follows that

$$\text{Val}(\Box((Q_1[K] \text{ OR } Q_2[K] \text{ OR } Q_3[K] \text{ OR } Q_4[K] \text{ OR } Q_5[K]) \supset \Diamond(Q'[K])), w) \not\equiv \text{TRUE} \quad \text{--- } F_{12}$$

that is,

$$\text{Val}(C'[K], w)) \not\equiv \text{TRUE}$$

holds. The details are omitted. []

4. CONCLUSION

In this paper, we have presented a method for verifying PROGRESS property of a protocol whose specification is described algebraically in a style of an abstract sequential machine, and applied the method to SDT protocol. The good feature of our method is that whole verification works can be carried out formally and mechanically. Our method can be applied for some verification problems such that the properties to be proved are formulated in the form of "Val(P, w)$\not\equiv$TRUE".

We have proved the theorems (lemmas) necessary for the verification of PROGRESS property of SDT protocol by using our interactive verification support system. For decreasing the work load in carrying out the verification process by using inference rules of Temporal logic, we have a plan to build decision procedures for an subclass of Temporal logic in our verification support system.

REFERENCES

[1] G. V. Bochmann and C. A. Sunshine, "Formal methods in communication protocol design," IEEE Trans. commun., vol. COM-28, pp. 624-631, Apr. 1980.

[2] A. S. Danthine, "Protocol representation with finite-state models," IEEE Trans. Commun., vol. COM-28, pp. 632-643, Apr. 1980.

[3] N. V. Stenning, "A data transfer protocol," Comput. Networks, vol. 1, pp. 99-110, Sept. 1976.

[4] G. D. Schuitz, D. B. Rose, C. H. West and J. P. Gray, "Executable description and validation of SNA," IEEE Trans. Commun., vol. COM-28, pp. 632-643, Apr. 1980.

[5] C. A. Sunshine, D. H. Thompson, R. W. Erickson, S. L. Gerhart and D. Schwabe, "Specification and verification of communication protocols in AFFIRM using state transition models," IEEE Trans. Software Eng., vol. SE-8, pp. 460-489, Sept. 1982.

[6] T. Higashino, M. Mori, Y. Sugiyama, K. Taniguchi and T. Kasami, "An algebraic specification of HDLC procedures and its verification," IEEE Trans. Software Eng., vol. SE-10, pp. 825-837, Nov. 1984.

[7] Y. Sugiyama, K. Taniguchi and T. Kasami, "A specification defined as an extension of a base

algebra," Trans. IECE Japan, vol. 64-D, pp. 324-331, Apr. 1981 (in Japanese).

[8] T. Kudo, T. Higashino, M. Higuchi, K. Taniguchi, T. Kasami and M. Mori, "An algebraic specification of Stenning's data transfer protocol and its verification," in Proc. Tech. Group Automaton, Lang., IECE Japan, Rep. AL83-64, Jan. 1984, pp. 93-105 (in Japanese).

[9] T. Higashino, K. Taniguchi and T. Kasami, "Verification of progress properties of an algebraic specification of Stenning's data transfer protocol," in Proc. summer LA sympo. Theoretical Computer Science, pp. 102-111, July 1985 (in Japanese).

[10] T. Higashino, T. Kudo, S. Nawata, Y. Sugiyama and K. Taniguchi, "Verification support system for algebraic specifications," Trans. IECE Japan, vol. 67-D, pp. 472-479, Apr. 1984(in Japanese).

[11] B. T. Hailpern, "Verifying concurrent processes using temporal logic," Lecture Notes in Computer Science 129 : Springer Verlag, 1982.

[12] B. T. Hailpern and S. Owicki, "Verifying network protocols using temporal logic," in Proc. Trends and applications 1980, IEEE Comput. Society, pp. 18-28, 1980.

[13] J. A. Goguen, J. W. Thatcher and E. G. Wagner, "An initial algebra approach to the specification, correctness and implementation of abstract data types," in Current Trends in Programming methodology, vol. IV, R. T. Yeh, Ed. Engledwood Cliffs, NJ: Prentice-Hall, 1978.

[14] R. L. Schwartz and P. M. Melliar-Smith, "From state machine temporal logic : specification methods for protocol standards," IEEE Trans. commun., vol. COM-30, pp. 2486-2496, Dec. 1982.

[15] T. Higashino, K. Taniguchi, T. Kasami, M. Fujii and M. Mori, "Verification of progress property of algebraically specified communication protocols," Trans. IECE Japan, vol. 69-D, 1471-1480, Oct. 1986 (in Japanese).

[16] Y. T. Yu and M. G. Gouda, "Deadlock detection for class of communicating finite state machines," IEEE Trans. Commun., vol. COM-28, pp. 2514-2518, Dec. 1982.