Verification of Protocols Using Presburger Array Theory

Masaaki Mori

Abstract

One of the verification methods of programs is to show certain predicates about program variables to be invariant. In general, it is undecidable whether or not a given predicate is invariant. In this paper, a class of concurrent program systems which include array variables is introduced, and a class of predicates about program variables is defined. The invariance problem for a predicate in the class is still undecidable. But it is shown that it is decidable whether or not a given predicate in the class satisfies a certain condition which is a sufficient one for the predicate to be invariant. An abstract protocol which is an example of the concurrent program systems is described, and it is formally shown that some predicates which assert properties of the abstract protocol are invariant.

I. Introduction

In this paper, a verification problem of concurrent programs is considered. One of verification methods of concurrent programs is to show certain predicates about program variables to be invariant. However, in general, it is undecidable whether or not a given predicate is invariant. Therefore, it is interesting to find conditions on predicates and concurrent program systems such that (i) if the conditions hold, then the predicates are invariant, and (ii) it is

Keywords: Communication protocols, verification, Presburger array theory, inductive, invariant.
decidable whether or not the conditions hold.

We formalize a class CS of concurrent program systems which use array variables and a class PS of predicates which are represented by the Presburger array theory [10], and it is shown that it is decidable whether or not a given predicate in PS is “strongly inductive” in a concurrent program system of CS. Being strongly inductive is a sufficient condition for the predicate to be inductive [5], and being inductive is a sufficient condition for the predicate to be invariant.

These results can be applied to verify a communication protocol. That is, an abstract protocol [9] is cited as an example of concurrent program systems, and it is shown that the predicates stated in [9] are strongly inductive, therefore, they are invariant.

II. Definitions and problems

Definition 1. Unquantified formulas of Presburger array theory (for short, P-array formulas) are defined as follows [10]:

1. constants and variables of sort integer are terms of sort integer,
2. if $i_1$ and $i_2$ are terms of sort integer, so are $i_1 + i_2$ and $i_1 - i_2$,
3. variables of sort array are terms of sort array,
4. if $M$ is a term of sort array and $i$ is a term of sort integer, then $M(i)$ is a term of sort integer representing the value of the $i$-th element of array $M$,
5. if $M$ is a term of sort array, and $i_1$ and $i_2$ are terms of sort integer, then $\text{Assign}(M, i_1, i_2)$ is a term of sort array representing an array obtained by assigning the value represented by $i_2$ to the $i_1$-th element of array $M$,
6. if $i_1 = i_2$ and $i_1 > i_2$ are P-array formulas, so are $\neg(B_1), (B_1) \lor (B_2)$ and $(B_1) \land (B_2)$. 

The above (6) and (7) are all the P-array formulas.

Let \( \mathbf{V} \) be a finite sequence \((v_1, v_2, \ldots, v_{|\mathbf{V}|})\) of program variables of sort integer and \( \mathbf{A} \) be a finite sequence \((A_1, A_2, \ldots, A_{|\mathbf{A}|})\) of program variables of sort array of integer. For convenience, we occasionally regard a sequence (such as \( \mathbf{V} \)) as a set, and write as \( v \in \mathbf{V} \) if \( v \) is a member of \( \mathbf{V} \). Let \( \mathbf{N} \) denote the set of nonnegative integers and \( \mathcal{K} \) denote the set of all infinite sequences of nonnegative integers. The values of program variables in \( \mathbf{V} \) will be denoted by a sequence \( \mathbf{X} = (x_1, x_2, \ldots, x_{|\mathbf{V}|}) \) where \( x_i (\in \mathbf{N}) \) is the value of \( v_i (1 \leq i \leq |\mathbf{V}|) \). The values of program variables in \( \mathbf{A} \) will be denoted by an infinite sequence \( \mathbf{Y} = (y_{11}^{(1)}, y_{12}^{(1)}, \ldots, y_{1|\mathbf{A}|}^{(1)}; y_{21}^{(2)}, y_{22}^{(2)}, \ldots, y_{2|\mathbf{A}|}^{(2)}; \ldots) \) where \( y_{kj}^{(i)} (\in \mathbf{N}) \) is the value of the \( k \)-th element of \( A_j \in \mathbf{A} \). A state of a "concurrent program system" using variables in \( \mathbf{V} \) and \( \mathbf{A} \) is defined by a pair of sequences \( (\mathbf{X}, \mathbf{Y}) \) in \( \mathcal{K} \). We will use state variables such as \( s, s' \), to stand for states of a concurrent program system. A concurrent program system executes actions. Let \( \mathbf{E} \) be a set of actions. In some cases, actions may take parameters in addition to program variables as their arguments. For such an action \( a \), let \( D_a \) denote a finite sequence of parameters \((p_1, p_2, \ldots, p_{|\mathbf{D}_a|})\) of sort integer such that \( D_a \cap \mathbf{V} = \phi \). Let \( a(\overline{d}) \) denote an action \( a \) with actual parameters \( \overline{d} \) in \( \mathbf{N}^{|\mathbf{D}_a|} \). An execution of \( a(\overline{d}) \) will cause a modification of values of program variables, that is, a state transition, say, from \( s \) to \( s' \).

**Definition 2.** A concurrent program system \( S = (\mathbf{V}, \mathbf{E}, \mathbf{R}, \mathbf{Q}, I) \) consists of the followings:

1. \( \mathbf{V} \) is a pair of sequences \( \mathbf{V} \) and \( \mathbf{A} \).
2. \( \mathbf{E} \) is a set of actions in \( S \). Predicates \( R_a \) and \( Q_a \) are defined for each action \( a (\in \mathbf{E}) \).
(3) \( R \) is a set of predicates: \( \{R_a(V, V', D_a)|a \in E\} \). \( R_a(V, V', D_a) \) is a predicate corresponding to the action \( a \) and represents the relation on the values of variables of \( V \) at the time before and after executing the action \( a \), where \( V' = \{x'|x \in V\} \). The unprimed variables and the primed variables represent the values of program variables before and after executing the action \( a \), respectively. We assume that the predicates \( R_a \)'s take the form of P-array formulas.

(4) \( Q \) is a set of predicates: \( \{Q_a(V, D_a)|a \in E\} \). An action \( a(\tilde{d}) \) is executable at state \( s \) if and only if \( Q_a(s, \tilde{d}) \) is true. For \( s \) and \( s' \) in \( K \), we write \( s \rightarrow s' \) and call \( s' \) a next state of \( s \) if and only if there exist an action \( a \) and actual parameters \( \tilde{d} \) such that \( Q_a(s, \tilde{d}) \land R_a(s, s', \tilde{d}) \) is true. In general, for \( s \) and \( \tilde{d} \), there exist one or more actions \( a(\tilde{d}) \)'s such that \( Q_a(s, \tilde{d}) \) is true, and there exist one or more next states of \( s \) after executing action \( a(\tilde{d}) \).

(5) \( I \) is a predicate of the form \( I(V) \) which designates the initial states of \( S \). A state \( s \) satisfying \( I(s) = \text{true} \) is called an initial state of \( S \).

**Definition 3.** Let "\( \longrightarrow^* \)" denote the reflexive, transitive closure of the binary relation "\( \longrightarrow \)" on \( K \). \( s' \) is said to be "reachable" from \( s \) if and only if \( s \rightarrow^* s' \).

Let \( P(V) \) be a predicate on program variables in \( S \) and \( s_0 \in K \) be an initial state.

**Definition 4.** \( P(V) \) is said to be inductive in \( S \) if

1. \( P(s_0) \) is true, and
2. \( (\forall s, s' \in K)([(P(s) = \text{true}) \land (s \rightarrow s')] \rightarrow P(s')) \) is true.

**Definition 5.** \( P(V) \) is said to be invariant in \( S \) if
is true. That is, the predicate $P(V)$ is said to invariant if there
exists an initial state $s_0$ such that $P(V)$ is true at $s_0$ and at any
state $s$ which is reachable from $s_0$.

The following lemma is due to N. Suzuki and D. Jefferson [10].

**Lemma 1.** It is decidable whether P-array formulas are true.

Now, we introduce a new type of variables other than program
variables and parameters. They are called index variables of sort
integer and appear as indexes of arrays in predicates. Let $\bar{q}=(q_1, q_2, 
\ldots, q_n)$ be a finite sequence of index variables. Then, we use
the notation $\forall \bar{q}$ and $\exists \bar{q}$ to represent $\forall q_1\forall q_2\ldots\forall q_n$ and $\exists q_1\exists q_2\ldots\exists q_n$, respectively.

**Proposition 1.** For a concurrent program system $S$, assume that the
following conditions (C1) and (C2) are satisfied:

(C1) For each action $a$, the predicate $Q_a(V, D_a)\land R_a(V, V', D_a)$ is
of the form

$$\forall j \exists \bar{w} T_a(V, V', j, \bar{w}, D_a) \quad (1)$$

where $j$ and $\bar{w}$ are disjoint sets of index variables and $T_a(V,
V', j, \bar{w}, D_a)$ is a P-array formula.

(C2) The predicate $I(V)$ is of the form

$$\forall h I_0(V, h) \quad (2)$$

where $h$ is a set of index variables and $I_0(V, h)$ is a P-array
formula.

Furthermore, assume that for any predicate given to verify, the
following condition (P1) is satisfied:

(P1) The predicate $P(V)$ is of the form

$$\forall \bar{i} P_0(V, \bar{i}) \quad (3)$$

where $\bar{i}$ is a set of index variables and $P_0(V, \bar{i})$ is a P-array
formulas.

Then the followings are hold.

[1] For a given predicate $P(V)$ and each action $a$, if there exist mappings $\sigma_1 : \tilde{j} \rightarrow \tilde{i}$ and $\sigma_2 : \tilde{h} \rightarrow \tilde{i}$ such that the following formulas (4) and (5) are true respectively, then the predicate $P(V)$ is invariant.

\[
\begin{align*}
(P_0(V, \tilde{i}) \land T_a(V, V', \sigma_1(j, \tilde{i}), \tilde{w}, D_a) \supset P_0(V', \tilde{i})) & \quad (4) \\
I_0(V, \sigma_2(h, \tilde{i})) \supset P_0(V', \tilde{i}) & \quad (5)
\end{align*}
\]

where $\sigma_1(j, \tilde{i})$ and $\sigma_2(h, \tilde{i})$ are the abbreviation for $(\sigma_1(j_1), \sigma_1(j_2), \ldots, \sigma_1(j_{\bar{m}}))$ and $(\sigma_2(h_1), \sigma_2(h_2), \ldots, \sigma_2(h_{\bar{m}}))$, respectively.

[2] It is decidable whether each of formulas (4) and (5) is true.

\[\square\]

**Proof.** [1] Assume that the formula (4) is true. For notational simplicity, write (4) as follows:

\[
P_0(i) \land T_a(\sigma_1(j, \tilde{i}), \tilde{w}) \supset P_0(i') (6)
\]

Since the formula (6) is equivalent to $\forall \tilde{w}\{P_0(i') \land T_a(\sigma_1(j, \tilde{i}), \tilde{w}) \supset P_0(i')\}$, it holds that

\[
P_0(i') \land \exists \tilde{w} T_a(\sigma_1(j, \tilde{i}), \tilde{w}) \supset P_0(i') (7)
\]

Note that

\[
\{\forall i' \forall j(P_0(i') \land \exists \tilde{w} T_a(j, \tilde{w})) \supset (P_0(i') \land \exists \tilde{w} T_a(\sigma_1(j, \tilde{i}), \tilde{w}))\} (8)
\]

is true. Then, it holds from formulas (7) and (8) that

\[
\{\forall i' \forall j(P_0(i') \land \exists \tilde{w} T_a(j, \tilde{i})) \supset P_0(i')\} (9)
\]

Since formula (9) is equivalent to $\forall \tilde{w}(\forall i P_0(i) \land \forall j \exists \tilde{w} T_a(j, \tilde{i}) \land P_0(h))$, we have $\forall i P_0(i) \land \forall j \exists \tilde{w} T_a(j, \tilde{w}) \supset \forall i P_0(i')$, that is

\[
P(V) \land Q_a(V, D_a) \land R_a(V, V', D_a) \supset P(V') (10)
\]

Similarly, if we assume that formula (5) holds, then we can show that $\forall \tilde{w} I_0(V, \tilde{w}) \supset \forall i P_0(V, \tilde{i})$ holds, that is, $I(V) \supset P(V)$ holds. Hence, if $s_0$ is a state satisfying that $I(s_0)$ is true, then $P(s_0)$ is true.
Furthermore, if $P(s)$ is true and $s \rightarrow s'$, then $P(s')$ is true by formula (10). Consequently, if $s_0 \xrightarrow{*} s$, then $P(s)$ is true. Hence, $P(V)$ is invariant by Definition 5.

The truth of formulas (4) and (5) is decidable by Lemma 1.

Let $CS$ be a class of concurrent program systems which satisfy conditions (C1) and (C2) in Proposition 1, and let $PS$ be a class of predicates which satisfy condition (P1) in Proposition 1. A predicate $P(V)$ in $PS$ is said to be "strongly inductive" in $S(\in CS)$ if it satisfies formulas (4) and (5) in Proposition 1. By definitions, a predicate $P(V)$ is inductive if it is strongly inductive.

The truth of formulas (4) and (5) is a sufficient condition for a predicate $P(V)$ to be invariant in $S$, if we assume that a system $S$ is in $CS$ and a predicate $P(V)$ is in $PS$. Note that it is decidable whether or not the sufficient condition holds. However, the truth of formulas (4) and (5) is not a necessary condition for $P(V)$ to be invariant, because it is undecidable [1] in general whether or not $P(V)$ is invariant in $S$ even if $P(V)$ is in $PS$ and $S$ is in $CS$. Moreover, the truth of formula (4) is undecidable if a form bounded by a universal quantifier at the head of $T_e$ in the formula (4) is allowed, because the (negation of) Hilbert's tenth problem [3] can be expressed by such a form of P-array formulas [10].

It is known that decision procedure for P-array formulas can be carried out by a nondeterministic Turing machine in polynomial time [10].

**III. Verification of a concurrent program system**

In formal verification of a protocol, the following approach may be promising [2], [9]: first we specify an abstract protocol in which
the values of all variables are not bounded, secondly verify the correctness of the abstract protocol, and lastly convert it into a concrete protocol where the values of all variables are bounded. In the following, we show that the abstract protocol stated in [9] can be formulated as a concurrent program system for which the conditions (C1) and (C2) are satisfied, and that the predicates which are shown by [9] to be invariant by tracing the protocol directly are strongly inductive (in the framework described above), and hence, invariant.

III. 1 Stenning's abstract protocol

[Assumptions] We assume that a message in transit may be lost completely, be corrupted and/or duplicated. Corruptions can be detected by a checksum test on a received message. Messages and replies are not necessarily received in the transmitted order. For simplicity, a message is assumed to be represented as a nonnegative integer.

[Properties] (1) The protocol is designed to handle such functions as transmission and reception of packets based on the full-duplexed high level data link control. A message is sent with a sequence number. (2) Sequence numbers increase indefinitely in order. (3) There exist timers for each transmitted sequence number, and retransmission of a message is carried out by checking the corresponding timer.

[Variables and system constants] (Fig. 1) 

\( H \) : a state variable of sort integer which represents the sequence number of the highest numbered message sent so far.

\( L \) : a state variable of sort integer which represents the sequence number of the lowest numbered message sent still requiring acknowled
Verification of Protocols Using Presburger Array Theory

**Sender**

\[ m_{13} \rightarrow m_{12} \rightarrow m_{11} \rightarrow m_{10} \rightarrow m_9 \rightarrow m_8 \rightarrow m_7 \rightarrow m_6 \rightarrow m_5 \rightarrow m_4 \rightarrow m_3 \rightarrow m_2 \rightarrow m_1 \]

**Receiver**

\[ 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \]

Fig. 1 Typical situation of the program variables of a communication system.

**Smb(·)**: a state variable of sort array of integer which represents the message buffer for messages to be transmitted. A message is buffered in \( \text{Smb}(i) \) such that \( i \) is equal to the sequence number of the message.

\[ T(\cdot) : \text{a state variable of sort array of integer whose indexes represent the sequence numbers of messages. } T(i) \text{ is used as a timer}^* \]

---

* Timers are set to "1" when they are set up and their values increase one by one according to executions of the action "tick" described later on.
such that its value represents the time elapsed after the transmission of a message with sequence number \( i \). This variable is not introduced in [9] explicitly.

\( Tn \) : a state variable of sort integer whose value is equal to the sequence number \( i \) when a timeout occurs in \( T(i) \). \( Tn \) is also used as a variable which represents the sequence number of a message to be retransmitted subsequently.

\( Data(\cdot), S(\cdot), A(\cdot) \) : state variables of sort array of integer whose indexes represent the sequence numbers of messages. \( Data(i) \) and \( S(i) \) represent the \( i \)-th message transmitted by the sender and its sequence number, respectively. \( A(i) \) represents the \( i \)-th reply sent by the receiver, which means that all messages with sequence numbers less than or equal to \( A(i) \) have been accepted by the sink process in the receiver.

\( Ps, Pr \) : state variables of sort integer which correspond to program counters in the sender and in the receiver, respectively.

\( data \) : a parameter which represents the content of a received message.

\( ackno \) : a parameter to which a reply sent by the receiver is assigned.

\( messno \) : a parameter to which a sequence number transmitted from the sender is assigned.

\( acheck \) : a parameter to which "0" or "1" is assigned according to whether or not corruptions were detected by a checksum test on the content of a received reply.

\( mcheck \) : a parameter to which "0" or "1" is assigned according to whether or not corruptions were detected by a checksum test on the content of a received message.

\( Ra \) : a state variable of sort integer to which "0" or "1" is assigned

** Newly introduced state variables for verifying protocols.
according to whether or not a reply from the receiver has been received and the action “rec-ACK” stated below has not yet executed. 

\( Rm \) : a state variable of sort integer to which “0” or “1” is assigned according to whether or not a message from the sender has been received and the action “rec-MES” stated below has not yet executed.

Here, we assume that the values of the state variables \( Ra, Rm \) and all the parameters are assigned by a lower level communication mechanism than that of the protocol described in this paper.

\( TWS, RWS \) : system constants named “window size” which are used at the sender and the receiver, respectively.

\( N \) : a state variable of sort integer which represents the sequence number of the lowest numbered message not yet received without error, that is, all messages with sequence numbers less than \( N \) have been accepted by the sink process in the receiver.

\( Tmb(\cdot) \) : a state variable of sort array of integer. A message with sequence number \( i \) which is received without error is temporally stored in \( Tmb(\cdot) \) until it is transferred to the sink process in the receiver.

\( Ari(\cdot) \) : a state variable of sort array of integer whose indexes represent the sequence numbers of messages. \( Ari(j)=1 \) (or \( Ari(j)=0 \)) shows that a message with sequence number \( j \) has been stored in \( Tmb \) (or has not been stored in \( Tmb \)).

[Predicates representing actions] Stenning’s abstract protocol is represented by \( R_a(V, V', D_a) \)'s and \( Q_a(V, D_a) \)'s in Table 1, where only \( D_a \) is written as arguments explicitly. In \( R_a(V, V', D_a) \), \( \forall i (M_1 (i)=M (i)) \) is abbreviated as \( M_1=M_2 \) where \( M_1 \) and \( M_2 \) are terms of sort array, and subformulas of the form \( x'=x \) and \( M'=M \) are omitted.
Table 1. Predicates representing Stenning's abstract protocol

\[ I: I_{\text{sender}} \land I_{\text{receiver}} \]
\[ I_{\text{sender}}: Ps = 1 \land H = 0 \land L = 1 \land Tn = 0 \land Ra = 0 \land k = 0 \land \forall i(T(i) = 0) \]
\[ Q_{\text{trans-MES}}: (H < L + TWS - 1 \land Ps = 1) \lor (H < L + TWS - 1 \land Tn = H \land Ps = 4) \lor (H < L + TWS - 1 \land Ra = 0 \land \forall i(T(i) > \tau) \land Ps = 2) \]
\[ R_{\text{trans-MES}}: H' = H + 1 \land k' = k + 1 \land S' = \text{Assign}(S, k', H') \land Data' = \text{Assign}(Data, k', \text{Smb}(H)) \land T' = \text{Assign}(T, H', 1) \land Ps' = 1 \]
\[ Q_{\text{rec-ACK}}(\text{ackcheck}, \text{ackno}) : (H = L + TWS - 1 \land Ps = 1 \land Ra = 1 \land \text{ackcheck} = 0) \lor (Ps = 2 \land Ra = 1 \land \text{ackcheck} = 0) \]
\[ R_{\text{rec-ACK}}(\text{ackcheck}, \text{ackno}) : \exists w(1 \leq w \leq l \land \text{ackno} = A(w)) \land L \leq \text{ackno} \lor (\forall j(L \leq j \leq \text{ackno} \land T'(j) = 0) \land (j < L \lor \text{ackno} < j \lor T'(j) = T(j))) \land L = \text{ackno} + 1) \land Ps = 2 \]
\[ Q_{\text{get timeout No.}} : (H = L + TWS - 1 \land Ps = 1 \land Ra = 0 \land \exists i(T(i) > \tau) \land Ps = 2 \land Ra = 0 \land \exists i(T(i) > \tau) \land Ps = 2) \]
\[ R_{\text{get timeout No.}} : \exists w(T(S(w)) \geq \tau \land Tn = S(w) - 1 \land Ps = 3) \]
\[ Q_{\text{retrans-MES}} : Ps = 3 \lor (Ps = 4 \land Tn = H) \]
\[ R_{\text{retrans-MES}} : Tn' = Tn + 1 \land k' = k + 1 \land S' = \text{Assign}(S, k', Tn') \land Data' = \text{Assign}(Data, k', \text{Smb}(Tn')) \land T' = \text{Assign}(T, Tn', 1) \land Ps' = 4 \]
\[ Q_{\text{tick}} : \text{"true"} \]
\[ R_{\text{tick}} : \forall i(T(i) > 0 \Rightarrow T' = \text{Assign}(T, i, T(i) + 1)) \]
\[ I_{\text{receiver}} : Pr = 1 \land N = 1 \land Ra = 0 \land l = 0 \land \forall i(Ari(i) = 0) \]
\[ Q_{\text{ren-MES}}(\text{mcheck}, \text{messno}, \text{data}) : Pr = 1 \land Ra = 1 \land mcheck = 0 \]
\[ R_{\text{ren-MES}}(\text{mcheck}, \text{messno}, \text{data}) : \exists w(1 \leq w \leq k \land \text{messno} = S(w) \land \text{data} = D(w)) \lor \forall (Ari(\text{messno}) = 0 \land \text{messno} < N + \text{RWS}) \lor (Tmb' = \text{Assign}(Tmb, \text{messno}, \text{data}) \land Ari' = \text{Assign}(Ari, \text{messno}, 1) \land Update_N(\text{messno}, Ari, N, N')) \land Pr' = 2 \]
\[ \text{where Update_N(\text{messno}, Ari, N, N') : } [N = \text{messno} \lor (N' > N \land \forall i(N < i < N' \Rightarrow (Ari(i) = 1 \lor Ari(N') = 0))] \]
\[ Q_{\text{trans-ACK}} : Pr = 2 \lor (Ra = 0 \land Pr = 1) \]
\[ R_{\text{trans-ACK}} : l' = l + 1 \land A' = \text{Assign}(A, l', N - 1) \land Pr' = 1 \]

\[ \star \quad k \text{ and } l \text{ are state variables of sort integer which represent the times of action "trans-MES" and "trans-ACK", respectively.} \]
\[ \star \star \quad \tau \text{ is a constant which represents the fixed time interval after which a timer expires.} \]
Each $Q_a(V, D_a) \land R_a(V, V', D_a)$ in Table 1 can be rewritten in the form of condition (C1) in Proposition 1 such that the abbreviated terms described above are expressly specified with universal quantifiers.

The formula (13) is a predicate which represents an action of the timeout and the formula (15) is one which represents an action such as to change the value of set up timers and is introduced to specify the mechanism of timeout for retransmission of messages. These are not described in the Stenning's abstract protocol explicitly. The action "tick" is not executed during executing another action since each action is assumed to be sequentially executed in order to satisfy the condition of mutual exclusion. The action "tick", however, can be executed many times between other actions without having an effect on the essential properties of Stenning's protocol. The meaning of the predicates for each action is explained in the following.

[Behaviour of protocol]

**Sender** Action 1 (transmission of messages): if $H < L + TWS - 1$, then $H + 1$ is assigned to $H'$ (that is, $H$ is increased by one), and the next message $Smb(H')$ is transmitted and the timer $T(H')$ is set simultaneously. This action, called "trans-MES", is repeatedly executed until $H = L + TWS - 1$ holds, and is followed by Action 2.

Action 2 (reception of replies from the receiver): if a reply has been received ($Ra = 1$) and it is correct ($acheck = 0$) (which shows that the reply has been assigned to parameter $ackno$) and $ackno \geq L$, then all timers whose indexes are in the range from $L$ to $ackno$ are reset (and stopped) and the value $ackno + 1$ is assigned to $L'$. This action, called "rec-ACK", is repeatedly executed until "0" is set to $Ra$, and is followed by Action 3.
Action 3 (detection of timeout sequence numbers and retransmission of messages) : if any timers have been expired, then one of the index values of expired timers is chosen by subaction “get timeout No.”, and the value decreased by one is assigned to $T_n$. After that, if $T_n < H$, then $T_n + 1$ is assigned to $T_n'$ and message $Smb(T_n')$ is retransmitted and simultaneously resets the timer with index value $T_n'$. This action, called “retran-MES”, is repeatedly executed until $T_n = H$ holds, and after completion of Action 3, Action 1 is executed again.

Receiver Action 4 (reception of messages and transmission of replies) : if a message has been correctly received ($Rm = 1$ and $mcheck = 0$, that is, the message and its sequence number have been assigned to parameters $data$ and $messno$ respectively) and $Ari(messno) = 0$ and $messno < N + RWS$, then the content of $data$ is assigned to $Tmb(messno)$ and “1” is assigned to $Ari(messno)$, and the following operations are executed subsequently ; while $Ari(N) = 1$ it is repeated to transfer the content of $Tmb(N)$ to sink process* and to increase $N$ by one. When this action called “rec-MES” has been completed, the value $Rm$ is equal to zero then subaction called “trans-ACK” by which the value $N-1$ at the time is transmitted to the sender as reply is executed, and Action 4 is executed again.

III. 2 Examples of invariants

Predicates which Stenning regards as the object to be verified are classified into two types : predicates with either variables of the sender or the receiver, and those with both variables of the sender.

* We assume that the operation is also executed by a lower level communication mechanism than that of the protocol described in this paper. Therefore, this action is not described in the formula (16) explicitly.
and the receiver. In many cases, for the former type of predicates, strong inductivity can be easily proved by tracing the one-sided behaviour. An example of such predicates is as follows:

$$\forall i (1 \leq i \leq l \rightarrow A(i) \leq N-1)$$  \hspace{1cm} (18)

An example of the latter type of predicates is as follows:

$$L \leq N$$  \hspace{1cm} (19)

The formula (4) to which the predicate (19) is substituted is not true for subaction "rec-ACK", so we consider the following predicate (20) which is a conjunct predicate of (18) and (19).

$$(L \leq N) \land \forall i (1 \leq i \leq l \rightarrow A(i) \leq N-1)$$  \hspace{1cm} (20)

There exist three kinds of subactions "rec-ACK", "rec-MES" and "trans-ACK" for which $Q_a(V, D_a)$ is true and by which some values of variables in the predicate (20) are changed. In the case of predicate $Q_{rec-ACK}$, the formula (4) is represented by a P-array formula described below, where the part of $Q_a(V, D_a)$ is omitted. Here, variables whose values do not change are represented by variables without prime.

$$[L \leq N \land (1 \leq i \leq l \rightarrow A(i) \leq N-1) \land (1 \leq w \leq l \rightarrow \text{ackno} = A(w))$$
$$\land (L \leq \text{ackno} \lor ((L \leq j \leq \text{ackno} \lor T'(j) = 0) \land (j < L \land \text{ackno} < j)$$
$$\lor T'(j) = T(j) \lor L' = \text{ackno} + 1)) \land Ps' = 2$$
$$\lor \{L' \leq N \land (1 \leq i \leq l \rightarrow A(i) \leq N-1)\}$$  \hspace{1cm} (21)

We introduce new variables $b_0$, $b_1$, $b_2$ and $b_3$ of sort integer such that $A(i) = b_0$, $A(w) = b_1$, $T'(j) = b_2$ and $T(j) = b_3$, and replace the formula (21) by the Presburger formula (22) [4] which does not contain any variables of sort array.

$$(i = w \lor b_0 = b_1) \lor [L \leq N \land (1 \leq i \leq l \rightarrow b_0 \leq N-1)$$
$$\land (1 \leq w \leq l \land \text{ackno} = b_1) \land \langle L \leq \text{ackno} \rangle [(L \leq j \leq \text{ackno} \rightarrow b_2 = 0)$$
$$\land (j < L \land \text{ackno} < j \rightarrow b_2 = b_3) \land L' = \text{ackno} + 1]]$$
\( \land P_s' = 2 \Rightarrow \{ L' \leq N \land (1 \leq i \leq l \Rightarrow b_a \leq N - 1) \} \) \hspace{1cm} (22)

Since (22) is an unquantified Presburger formula, the formula (22) is equivalent to a Presburger sentence where all variables are bounded by universal quantifiers at the front of the formula (22). Hence, we can effectively decide the truth of (22) by the decision procedure of Presburger sentences [8]. We can show that the predicate (20) is strongly inductive, that is, invariant in this way, hence the predicate (19) is invariant.

We can show other predicates stated by [9] to be invariant similarly.

Furthermore, if we show that the sequence numbers and acknowledge numbers can be represented by modulo some system constant, Stenning's protocol could be transformed into a concrete protocol easily. In order to realize a concrete protocol, it suffices that \( A(i) - L \) and \( S(i) - N \) are bounded by some system constants. These verification problems can be dealt with similarly.

### IV. Dynamic properties of concurrent program systems

Another problem on concurrent program systems is to verify a dynamic property whether a protocol progresses correctly if there occur no transmission errors for a certain time interval. On such a problem, we must consider not only the truth of predicates on the states before and after executing an action, but also the truth of predicates which represent the relation held before and after executing an arbitrary sequence of actions. In general, the truth of such predicates is undecidable even for a seemingly simple case [7]. This fact can be easily proved as follows: in unbounded state variables of an abstract protocol, if there exist variables of sort
integer in both the sender and the receiver, and we can increase/decrease their values and test whether or not their values are zero, then we can simulate a two counter automaton by a protocol which transmits messages successively without using any communication control symbols. Consequently, we shall get the above result by reducing the problem to the well-known undecidable one of two counter automata [6]. However, there are instances where it can be verified by tracing a protocol directly whether or not it progresses correctly.

IV. 1 Progressiveness of abstract protocols

In Stenning's abstract protocol (for short, APₘ), there is a state variable N in the receiver which shows that (i) all messages with sequence numbers less than N have been accepted by the sink process in the receiver and (ii) the message with sequence number N has not been received correctly, and also there is a state variable L in the sender whose value represents the value N in the receiver. Therefore, if it occurs that a message with sequence number L cannot be transmitted indefinitely, then the protocol will not progress correctly. In APₘ, it is not guaranteed that L is chosen as a timeout sequence number when the occurrence of timeout is detected. That is, it may happen that APₘ does not progress. Hence, we consider a new abstract protocol (for short, APₙ) of which the state variables and the predicates are the same as those of APₘ, except that the predicate (13) is replaced by the following predicate (23) that chooses the smallest timeout sequence number.

\[ Q_{\text{get the smallest timeout no.}} : \exists w \{ T(S(w) \geq \tau \land \forall j (S(w) > T(j) < \tau) \land Tn' = S(w) - 1 \land Ps' = 3) \} \quad (23) \]

The predicate (23) does not satisfy the condition (C1) in Pro-
position 1. However, the set of values of sequence numbers which can be assigned to \( T_n' \) by the predicate (13) properly contains those which are assigned to \( T_n' \) by the predicate (23), then the predicate which are invariant in \( \text{AP}_s \) are also invariant in \( \text{AP}_N \). Since no actions except action “tick” can be executed during the transmission of a sequence of messages, it can be easily proved by tracing the behaviour of the sender that the following predicates (24) ~ (26) are strongly inductive.

\[
\forall j \{ (j > H \land j < L) \supset T(j) = 0 \} \tag{24}
\]

\[
\forall j \{ H \geq j \geq L \supset T(j) > 0 \} \tag{25}
\]

\[
\forall i \forall j \{ (T_n < j < i \leq H \supset T(i) \leq T(j)) \land (L \leq j < i \leq T_n \supset T(i) \leq T(j)) \}\tag{26}
\]

The predicate (26) shows that the values of timers monotonously decrease in \([L, T_n]\) and \((T_n, H]\) during retransmission. Hence, the next predicate (27) holds in \( \text{AP}_s \) by (26) except during retransmission.

\[
\forall i \forall j \{ L \leq j < i \leq H \supset T(i) \leq T(j) \}\tag{27}
\]

Since (24) ~ (27) are invariant in \( \text{AP}_s \) they are also invariant in \( \text{AP}_N \). When the action corresponded to the predicate (23) is executed in \( \text{AP}_N \) it is shown by (27) that timers monotonously decrease between \( L \) and \( H \), then the predicate (23) is nothing but to choose \( L - 1 \) as a starting sequence number of retransmissions (retransmission starts at \( L \) by the predicate (14)), that is, the predicate (23) is equivalent to the predicate (28).

\[
Q_{\text{get timeout No.}} : T_n' = L - 1 \land Ps' = 3 \tag{28}
\]

The difference between an abstract protocol where only the predicate (23) is replaced by the predicate (28) and Bochmann's abstract protocol [2] is as follows: the latter begins to retransmit from \( L \)
unconditionally, the former checks the value of timers at first then
decides whether or not retransmission should be executed. In this
way, messages from $L$ to the value of the largest sequence number
which has been transmitted are retransmitted in $AP_N$ if a timeout
occurs, then we can conclude that the protocol will progress correctly
if no transmission errors occur. Furthermore, we can define the
similar protocol as $AP_N$ by using a single timer instead of $T(\cdot)$.

V. Conclusions

In this paper, we have formalized a class CS of concurrent
program systems which use array variables and a class PS of pred-
icates, where it is decidable whether or not a given predicate in PS
satisfies a certain condition which is a sufficient condition for the
predicate to be inductive (and hence invariant) in a concurrent
program system of CS. Moreover, we have shown that these results
can be applied to verify a communication protocol. It is an inter-
esting problem to investigate a class of concurrent program systems
in which the verification problem of dynamic properties is decidable.

References

[1] T. Araki, T. Taniguchi, T. Tokura, T. Kasami and M. Mori, Verifica-
tion and parameter selection in concurrent systems (in Japanese), Papers
of Tech. Group on Automaton and Languages, AL76–68, IECE of Japan,

[2] G. V. Bochmann, Logical verification and implementation of protocols,


[4] D. Hibert and P. Bernays, Grundlagen der matematik I, (Springer-
Verlag, 1968).


