A Blocking-Based Approach to Protocol Validation

Yu Lei, David Kung and Qizhi Ye

{ylei,kung,qiye}@cse.uta.edu

A Blocking-Based Approach to Protocol Validation

Yu Lei, David Kung, and Qizhi Ye
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019
{ylei,kung,qiye}@cse.uta.edu

ABSTRACT

One common approach to protocol validation is reachability analysis, which involves systematically exploring the state space of a protocol. The main challenge of reachability analysis is dealing with the state explosion problem. In this paper, we present a new reachability analysis approach, called blocking-based simultaneous reachability analysis, for protocol validation. This approach has the potential to significantly reduce the number of states that have to be explored but can still be used to detect several logical errors that are commonly found in a protocol.

1. INTRODUCTION

A protocol is often specified as a set of processes that interact with each other by exchanging messages over some communication channels. Protocol validation refers to the activity of verifying the absence of logical errors for a given protocol. Logical errors that are commonly found in a protocol include non-executable transitions, and invalid states such as deadlock, channel overflow, or unspecified reception states. One commonly used approach to protocol validation is reachability analysis, which involves systematically exploring the state space of a protocol. This approach can be easily automated, but it suffers from the well-known state explosion problem, i.e., the number of reachable states can be enormous for many protocols. To alleviate the state explosion problem, many approaches have been developed to reduce the number of states that have to be explored while still preserving important error detection capabilities. In this paper, we describe a new reachability analysis approach, called blocking-based simultaneous reachability analysis (BSRA), for protocol validation. In this approach, a protocol is modeled as a group of communicating finite state machines (CFSMs) in which each process is represented as a CFSM and each channel is represented as a First-In-First-Out (or FIFO) queue. Central to BSRA is the notion of a global blocking point, which refers to a global state in which every process either defines a receive transition or defines no transitions at all. Instead of exploring every global state, BSRA only explores a subset of global blocking points, which is usually a small portion of the entire state space. The simultaneous aspect of BSRA refers to the fact that during state exploration, multiple processes are allowed to proceed in an arbitrary order to derive the successor states of a global blocking point. We present a BSRA algorithm and show how to use this algorithm to detect logical errors.

The next section defines the CFSM model, and gives a general framework for reachability analysis. Section 3 presents an algorithm that implements BSRA. Section 4 shows how to use the BSRA algorithm to detect logical errors. Section 5 briefly overviews the related work. Section 6 provides the concluding remarks.

2. PRELIMINARIES

2.1 The CFSM Model

The CFSM model is a widely used notation for describing the behavior of network protocols. In this model, a protocol $\Pi$ consisting of $n$ processes is defined as a tuple $(P, C, M)$, where

- $P = \{P_i\}_{1 \leq i \leq n}$, where $P_i$ is the $i$-th process.
- $C = \{C_{ij}\}_{1 \leq i, j \leq n \land i \neq j}$, where $C_{ij}$ is the simplex, error-free channels from $P_i$ to $P_j$.
- $M = \{M_{ij}\}_{1 \leq i, j \leq n \land i \neq j}$, where $M_{ij}$ is the set of messages that can be sent from $P_i$ to $P_j$. It is assumed that $M_{ij} \cap M_{kl} = \emptyset$ if $(i, j) \neq (k, l)$, where $1 \leq k, l \leq n$.

We use $M^-_i$ to denote the set of messages that can be sent by process $P_i$, i.e., $M^-_i = \bigcup_{1 \leq j \leq n \land j \neq i} M_{ij}$, and $M^+_i$ is the set of messages that can be received by process $P_i$, i.e., $M^+_i = \bigcup_{1 \leq j \leq n \land j \neq i} M_{ji}$. We also use $M_i$ to denote $M^-_i \cup M^+_i$.

Each process $P_i$ is further defined as a tuple $(S_i, s^0_i, \delta_i)$, where (1) $S_i$ is a finite set of states; (2) $s^0_i \in S_i$ is the initial state; and (3) $\delta_i \subseteq S_i \times M_i \times S_i$ is a set of transitions. Let $t = (s, m, s') \in \delta_i$. We say that $t$ is send transition if $m \in M^-_i$, or a receive transition if $m \in M^+_i$. We refer to $s$, $m$, and $s'$ as the source state, message, and destination state of $t$, and denote them as $src(t)$, $msg(t)$, and $dest(t)$ respectively. We also use $chan(t)$ to denote the channel $msg(t)$ is sent to or received from, and $proc(t)$ the process in which $t$ is defined. We will designate $n$ as the number of processes in a protocol.

Let $\Pi = (P, C, M)$ be a protocol. A global state of $\Pi$ is a snapshot of $\Pi$ that consists of the local state $s_i$ of each process $P_i \in P$ and the content $w_{ij}$ of each channel $C_{ij} \in C$. Formally, $g = (s_1, \ldots, s_n; w_{12}, \ldots, w_{n(n-1)})$. Note that $w_{ij}$ is a finite sequence of messages and can be empty. We use $\epsilon$ to denote the empty sequence, $\cdot$ message concatenation, and $head(w_{ij})$ the first message in $w_{ij}$. The initial global state $g_0$ of $\Pi$ consists of the initial state for each process and the empty sequence for each channel, i.e., $g_0 = (s^0_1, \ldots, s^0_n, \epsilon, \ldots, \epsilon)$. 
Let $s$ be a local state of process $P_i$. Let $\text{trans}(s)$ be the set of transitions defined at $s$, i.e., $\text{trans}(s) = \{ t \in \delta_i | \text{src}(t) = s \}$. Let $g = (s_1, s_2, \ldots, s_{n-1})$ be a global state. Let $\text{trans}(g)$ be the set of transitions defined at $g$, i.e., $\text{trans}(g) = \bigcup_{1 \leq i \leq n} \text{trans}(s_i)$. Let $t \in \text{trans}(g)$. Then $t$ is executable if $t$ is a send transition, or $t$ is a receive transition and $\text{chan}(t)$ is not empty. Note that a send transition $t$ is considered to be executable even if $\text{chan}(t)$ is full. In this case, we will report a channel overflow as discussed later.

Let $t$ be an executable transition at a global state $g$. If $t$ is a send transition, its execution moves $\text{proc}(t)$ from $\text{src}(t)$ to $\text{dest}(t)$ and append a message to the tail of $\text{chan}(t)$. If $t$ is a receive transition, its execution moves $\text{proc}(t)$ from $\text{src}(t)$ to $\text{dest}(t)$ and removes the first message in $\text{chan}(t)$. The global state $g'$ reached by executing $t$ is an immediate successor of $g$, denoted as $g \rightarrow g'$ or $g \rightarrow g'$ if $t$ is not of interest.

A transition sequence $t_1t_2\ldots t_q$ is executable at a global state $g$ if there exists a sequence of global states $g_1, g_2, \ldots, g_q$ such that $g \rightarrow g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_q$. We will write $g \rightarrow^* g'$ if the intermediate states are not of interest or simply $g \rightarrow g'$ if the transitions are not of interest.

Note that $\rightarrow$ defines a binary relation between global states, and $\rightarrow^*$ is the transitive closure of $\rightarrow$. Let $g_0$ be the initial global state of a protocol $\Pi$. A global state $g$ of $\Pi$ is a reachable global state if $g_0 \rightarrow^* g$. We will only be interested in reachable global states and thus will refer to a reachable global state simply as a global state.

Now we are ready to define the logical errors.

- A transition $t$ of a protocol $\Pi$ is a non-executable transition if $t$ is not executable at any global state $g$ of $\Pi$.
- A global state $g$ of $\Pi$ is a deadlock state if $g$ has no executable transition and every channel is empty.
- A global state $g$ of $\Pi$ is a channel overflow state if there exists a send transition $t$ such that $t$ is executable at $g$ but its channel $\text{chan}(t)$ is full.
- A global state $g = (s_1, \ldots, s_n; w_{12}, \ldots, w_{n(n-1)})$ of $\Pi$ is an unspecified reception state if there exists a channel $C_{ij}$ such that no receive transition $t$ is executable at $g$ has $\text{msg}(t) = \text{head}(w_{ij})$.

Figure 1 shows an example CFSM system that consists of three processes $P_1$, $P_2$, and $P_3$.

![Figure 1: An example CFSM system](image)

### 2.2 A Reachability Analysis Framework

Fig. 2 shows a general framework for reachability analysis. In the framework, $\text{open}$ keeps the set of states that are reached but are yet to be explored, and $\text{explored}$ keeps the set of states that have already been explored. A global state $g$ is explored by deriving the set $\text{successor}(g)$ of successor states of $g$, which is the step marked with (**). Different approaches may initialize $\text{open}$ with different global states (marked with (*) and may derive different successor states. A straightforward approach is to initialize $\text{open}$ with the initial global state $g_0$ and derive every immediate successor of each global state $g$, i.e., $\text{successor}(g) = \{ g' | g \rightarrow g' \}$. Doing so leads to full reachability analysis, i.e., exploring every global state that could be possibly reached by a protocol. Note that $\text{open}$ can be implemented as a stack or queue, which results in a depth-first or breadth-first search respectively.

The global states explored by reachability analysis can be organized into a graph, called a reachability graph. In such a graph, each node represents a global state, and there exists an edge from node $n_1$ to node $n_2$ if $n_2$ is a successor state of $n_1$. Fig. 3 shows the full reachability graph of the example CFSM system, i.e., the graph constructed by full reachability analysis. Note that in Fig. 3, we show in each global state the local states of $P_1$, $P_2$, $P_3$ and the contents of channels $C_{13}$, $C_{12}$, and $C_{32}$, in the given order. The contents of the other channels are not shown as they are inactive.

### 3. BSRA

In this section, we describe a new reachability analysis approach, called blocking-based simultaneous reachability analysis (or BSRA), which only explores a portion of the state space of a protocol. We will show how to use BSRA to detect logical errors in a protocol in the next section.

#### 3.1 Basic Concepts

In this section we define some important concepts in BSRA. Let $\Pi = (P, C, M)$ be a protocol. A local blocking point of $\Pi$ is a local state that has one or more outgoing receive
transitions, or has no outgoing transitions at all. A global blocking point of \( \Pi \) is a global state in which every local state is a local blocking point. As an example, in Fig. 1, state 3 of \( P_1 \) is a local blocking point because it has no outgoing transitions, state 1 of \( P_3 \) is a local blocking point because it has two outgoing receive transitions \( t_2 \) and \( t_5 \), and state 1 of \( P_3 \) is a local blocking point because it has one outgoing receive transition \( t_1 \). It follows that in Fig. 3, \( q_2 \) is a global blocking point. We will refer to a global/local blocking point as a blocking point if the context is clear.

The blocking frontier, denoted as \( \text{frontier}(g) \), of a global state \( g \) is a set of blocking points that can be reached from \( g \) by only executing zero or more send transitions. If \( g \) itself is a blocking point, then \( g \) is included in \( \text{frontier}(g) \). As an example, in Figure 3, \( \text{frontier}(q_0) = \{ q_2 \} \), because \( q_2 \) is the only blocking point that can be reached from \( q_0 \) by executing send transitions only, and \( \text{frontier}(q_2) = \{ q_2 \} \), because \( q_2 \) is itself a blocking point and it can reach no other blocking points by executing send transitions only.

A simultaneous receive transition set \( R \) at a global state \( g \) is a set of executable receive transitions at \( g \) such that it has at most one receive transition for each process, i.e., \( \forall t_1, t_2 \in R : \text{proc}(t_1) \neq \text{proc}(t_2) \). A linearization of \( R \) can be obtained by putting all the transitions in \( R \) into a total order. It is easy to see that every linearization of \( R \) is an executable transition sequence at \( g \) and leads to the same global state. Therefore, the transitions in a simultaneous receive transition set can be executed in any order, i.e., simultaneously.

Let \( \Pi = (P, C, M) \) be a protocol. A simultaneous send sequence set at a global state \( g \) is a set of send transition sequences \( \omega_1, \ldots, \omega_n \), where \( \omega_i \) is an executable send transition sequence of \( P_i \in P \) at \( g \). Let \( \sigma \_i \) denote the projection of a transition sequence \( \sigma \) onto \( P_i \), i.e., the transition sequence obtained by removing from \( \sigma \) all transitions \( t \) with \( \text{proc}(t) \neq P_i \). Then, a linearization of \( Q = \{ \omega_1, \ldots, \omega_n \} \) is a sequence \( \sigma \) of transitions such that \( \sigma \_i = \omega_i \). It is easy to see that every linearization of \( Q \) is an executable transition sequence at \( g \) and leads to the same global state. Therefore, the send sequences in a simultaneous send sequence set can also be executed simultaneously.

For simplicity, we will refer to a simultaneous receive transition set as a simultaneous transition set and a simultaneous send sequence set as a simultaneous sequence set.

### 3.2 The BSRA Algorithm

In this section we present an algorithm for implementing BSRA. We first show how to compute the set \( \text{succ}(b) \) of successor states for a given blocking point \( b \), which is the core component of the algorithm. Let \( \Pi = (P, C, M) \) be a protocol. Let \( b \) be a blocking point. The computation of \( \text{succ}(b) \) consists of two main steps. The first step is to generate a set of simultaneous transition sets at \( b \), each of which can be executed simultaneously to derive an intermediate global state. The second step is to compute the blocking frontier \( \text{frontier}(g) \) of each intermediate global state \( g \) derived in the first step. This is accomplished by generating a set of simultaneous sequence sets at \( g \), each of which can be executed simultaneously to derive a blocking point in \( \text{frontier}(g) \). The set \( \text{succ}(b) \) of successor states of \( b \) consists of the union of the blocking frontiers of the intermediate global states.

Before we elaborate the two steps, we define the cross product, denoted as \( S_1 \times S_2 \times \ldots \times S_n \), of sets \( S_1, S_2, \ldots, \) and \( S_n \) as follows: \( S_1 \times S_2 \times \ldots \times S_n = \{ (c_1, c_2, \ldots, c_n) \mid c_i \in S_i, \ 1 \leq i \leq n \} \). Note that unlike a Cartesian product, which contains a set of ordered tuples, a cross product contains a set of unordered sets.

#### 3.2.1 Generating Simultaneous Transition Sets

This step is to generate a set of simultaneous transition sets for a given blocking point \( b \). We first introduce the notion of a maximal simultaneous transition set. A simultaneous transition set \( R \) at \( b \) is maximal if there does not exist a receive transition \( r \) such that \( R \cup \{ r \} \) is also a simultaneous transition set at \( g \). Note that intermediate states reached during the simultaneous execution of the transitions in a simultaneous transition set are not explored. Therefore, it is desired to generate maximal simultaneous transition sets such that the number of intermediate states that will be skipped can be maximized.

Maximal simultaneous transition sets at \( b \) can be computed as follows. First, for each process \( P_i \), we find the set \( R_i \) of receive transitions that are executable at \( b \). Next, we compute the cross product \( R \) of those non-empty sets \( R_i \). Each set in \( R \) is a maximal simultaneous transition set at \( b \). As an example, in Figure 3, the only maximal simultaneous transition set at \( q_2 \) is \( \{ t_3, t_4 \} \). This is because at \( q_2 \), \( R_1 = \emptyset \), \( R_2 = \{ t_2 \} \), and \( R_3 = \{ t_1 \} \).

However, generating only maximal simultaneous transition sets is insufficient for detecting logical errors. This is due to the existence of potentially executable transitions (PETs). A receive transition \( r \) defined at a global state \( g \) is a PET if \( \text{chan}(r) \) is empty at \( g \). Note that since \( \text{chan}(r) \) is empty, \( r \) is not executable at \( g \). However, \( r \) may become executable at a global state \( g' \) that is reachable from \( g \). This can happen if one of the transitions leading from \( g \) to \( g' \) sends a message to \( \text{chan}(r) \), and \( \text{proc}(r) \) does not execute any transition during the transfer from \( g \) to \( g' \). It is possible that the execution of \( r \) at \( g' \) leads to some path that may contain a logical error. In order to detect the error, the
execution of proc(r) needs to be delayed until g' is reached. Therefore, given a simultaneous transition set R at b, if R contains a (receive) transition r from a process that has one or more PETs at b, then another simultaneous transition set R' needs to be generated such that R' does not include r, which means that proc(r) is delayed in R'. This can be done as follows. Let R_1 be the set of executable receive transitions of process P_i at b. If P_i has one or more PETs at b, we add to R_1 a special transition ε. Then, we compute the cross product R of those non-empty R_i's. Given a simultaneous transition set \( R \in R \), if \( R \) contains only ε, we ignore \( R \); otherwise, we delete ε from R.

As an example, consider g_2 again in Figure 3. Earlier we have shown that the only maximal simultaneous transition set at g_2 is \( \{ t_3, t_7 \} \). Observe that t_3 is a PET at g_2, since chan(t_3), which is L_3, is empty at g_2. Therefore, we need to add ε to the set R_1 of executable transitions of P_2 at g_2, i.e., \( R_2 = \{ t_3, ε \} \). Recall that R_1 = ∅ and R_2 = \{ t_1 \}. Thus, we obtain two simultaneous sets \( \{ t_3, t_7 \} \) and \( \{ t_1 \} \). Note that the execution of P_2 is delayed in the set \( \{ t_1 \} \). Also note that if the set \( \{ t_1 \} \) is not generated, t_3 would never be exercised.

### 3.2.2 Computing Blocking Frontiers

Let \( g = (s_1, \ldots, s_n; \omega_1, \ldots, \omega_n) \) be an intermediate global state reached by executing a simultaneous transition set from a global blocking point. In order to compute the blocking frontier of \( g \), we first find the set \( Q \) of send transition sequences that are executable at \( g \) for each process \( P_i \). Then, we compute the cross product \( Q = Q_1 \otimes Q_2 \otimes \cdots \otimes Q_n \). Each simultaneous sequence set \( Q = (\omega_1, \ldots, \omega_n) \in Q \) can be executed simultaneously to derive a blocking point in \( \text{frontier}(g) \).

The main challenge in this step is finding \( Q \) for each process \( P_i \). Recall that \( P_i \) is modeled as a CFMS, which can be visually represented as a graph. In the graph, each node represents a local state of \( P_i \). There exists an edge \( e \) from node \( s \) to node \( s' \) if there exists a transition \( t \) such that \( \text{src}(t) = s \) and \( \text{dest}(t) = s' \). We are only interested in edges that represent send transitions, which we refer to as send edges. To compute \( Q \), we conduct a depth-first traversal of the graph that starts from the local state \( s_i \) of \( P_i \) at \( g \) and only follows send edges. During the traversal, the sequence of \( Q \) of transitions leading to the current state \( s \) from \( s_i \) is kept on a stack. If \( s \) is a local blocking point, then we add \( Q \) to \( Q \). In particular, if the starting state \( s_i \) is a local blocking point, we add an empty sequence to \( Q \). Note that in case that a cycle is detected during the traversal, we will report a channel overflow error, since the cycle consists of send transitions only and can be repeated infinitely. In the rest of this paper, we assume that there exists no send transition cycle in any process.

We stress that the computation of \( Q \) only depends on the static structure of \( P_i \). Thus, such computation needs to be performed at most once for each local state \( s \), and can be done prior to the analysis or lazily during the analysis.

Fig. 4 shows a function, called \( \text{Compute\_Succ} \), for computing the set of successor states for a given blocking point. Note that each successor state computed by \( \text{Compute\_Succ} \) is also a blocking point. Our BSRA algorithm implements the reachability analysis framework in Fig. 2 by (1) initializing open with the blocking points in the frontier \( \text{frontier}(g_0) \) of the initial global state \( g_0 \); and (2) using function \( \text{Compute\_Succ} \) to derive the successor states of each blocking

---

4. DETECTION OF LOGICAL ERRORS

In this section, we discuss how to use the BSRA algorithm to detect logical errors in a protocol. Theorems 1 and 2 state that the BSRA algorithm guarantees the detection of non-executable transitions and deadlocks. In Sections 4.1 and 4.2, we show how to slightly modify the BSRA algorithm to detect channel overflows and unspecified receptions, respectively. Note that the formal proofs for all the theorems are provided in the supplementary material.

**Theorem 1.** The BSRA algorithm reports a transition \( t \) of a protocol \( \Pi \) as a non-executable transition iff \( t \) is a non-executable transition of \( \Pi \).

**Theorem 2.** The BSRA algorithm reports a deadlock of a protocol \( \Pi \) iff \( \Pi \) has a deadlock.

4.1 Channel Overflow

We use an example to illustrate that the BSRA algorithm does not guarantee the detection of channel overflow. Figure 6 shows the example system (left) and its BSRG constructed by the BSRA algorithm (right). Note that in the
BSRG, each global state only shows the contents of channels $C_{31}$ (from $P_3$ to $P_1$) and $C_{12}$ (from $P_1$ to $P_2$), in the given order. The contents for the other channels are not shown because these channels are not active. Assume that the capacity of $C_{12}$ is 1. Then, the BSRRA algorithm reports no channel overflow for $C_{12}$. However, a careful examination would reveal that there does exist a channel overflow state for $C_{12}$, namely $o = (4,1,2,3,\epsilon,m_{1} \cdot m_{3})$.

In the following, we show how to slightly modify the BSRRA algorithm such that the new algorithm, namely BSRRA.CO, guarantees the detection of channel overflow for a given channel. To detect channel overflow for all the channels in a protocol, the BSRRA.CO algorithm needs to be applied once for each of these channels. Let $\Pi$ be a protocol and $C$ a channel of $\Pi$. In essence, the problem of detecting channel overflow for $C$ is to find the maximum number of messages that may be present in $C$ at a given time and then compare it with the capacity of $C$. Therefore, it is desirable to delay the reception of messages from $C$ as much as possible. This can be accomplished as follows. For each simultaneous transition set $T$ constructed by the BSRRA algorithm at a blocking point $b$, if $T$ contains at least two transitions and one of them is a receive transition $t$ with $chan(t) = C$, then the BSRRA.CO algorithm tries to delay the execution of $t$ by creating an additional simultaneous transition set $T'$ that does not include $t$, i.e., $T = T - t$. For example, in Fig. 6, the BSRRA.CO algorithm will also create an additional simultaneous transition set $\gamma = \{t_2, \epsilon, \epsilon\}$ at $b_0$. Let $g$ be the state reached from $b_0$ by executing $\gamma$. Then, it is easy to see that the channel overflow state $o$ is in the blocking frontier of $g$ and thus will be detected by the BSRRA.CO algorithm.

**Theorem 3.** Let $\Pi$ be a protocol. Assume that the BSRRA.CO algorithm is applied to a channel $C$ of $\Pi$. The algorithm reports a channel overflow iff $\Pi$ has a channel overflow state for $C$.

**4.2 Unspecified Reception**

Again we use an example to show that a direct application of the BSRRA algorithm does not guarantee the detection of unspecified receptions. Figure 6 shows an example CFMSM system (left) and its BSRRA constructed by the BSRRA algorithm (right). In the BSRRA, we show in each global state the contents of channels $C_{21}$ (from $P_2$ to $P_1$) and $C_{12}$ (from $P_1$ to $P_2$) in the given order. Note that none of the states $g_0, b_0$, and $b_1$ is an unspecified reception state, and thus the BSRRA algorithm reports no unspecified reception. However, the example system does have two unspecified reception states, namely $u_1 = (2,1,\epsilon,m_1)$ and $u_2 = (1,2,m_2,\epsilon)$.

**Figure 7: An unspecified reception example**

Let $g = (s_1,\ldots,s_n;w_{12},\ldots,w_{n(n-1)})$ be an unspecified reception state of a protocol $\Pi$. Then, at $g$, there must exist a channel $C_{ij}$ (from process $P_i$ to $P_j$) such that no receive transition at the local state $s_j$ of $P_j$ can receive the first message in $w_{ij}$. Let $s = s_i$ and $m = head(w_{ij})$. We refer to $(s,m)$ as an unspecified reception pair. For example in Fig. 7, the pair consisting of state 1 in $P_2$ and message $m_1$ is an unspecified reception pair. Note that an unspecified reception state is caused by an unspecified reception pair, whereas an unspecified reception pair may cause many unspecified reception states.

In the following, we show how to augment a protocol $\Pi$ such that applying the BSRRA algorithm to the augmented protocol guarantees the detection of every unspecified reception pair in $\Pi$. Observe that the BSRRA algorithm only checks global blocking points against potential logical errors. Therefore, in order to detect an unspecified reception pair $(s,m)$, $s$ must appear in at least one global blocking point.
This can only happen if $s$ is a local blocking point. Moreover, in order to detect every unspecified pair involving $s$, $s$ also needs to have at least one PET. (A formal justification for having at least one PET is provided in the proof of Theorem 4 in the supplementary material.) This suggests the following augmentation: For each local state $s$, we add a special receive transition $t$ such that (1) $src(t) = dest(t) = s$; (2) $msg(t) = \tau$, where $\tau$ is a special message that is not sent by any process in $\Pi$; and (3) $chan(t)$ is a special channel that is always empty. Note that $t$ will be considered as a PET at any global state containing $s$ but it will never become executable. Thus, the addition of $t$ only influences the behavior of the BSRA algorithm but does not change the behavior of $\Pi$. For example, in Fig. 7, after the above augmentation, both $u_1$ and $u_2$ will become a blocking point and thus will be detected.

**Theorem 4.** Let $\Pi$ be a protocol. Let $\Pi'$ be the protocol obtained by augmenting $\Pi$. Applying the BSRA algorithm to $\Pi'$ reports every unspecified reception pair in $\Pi$.

5. RELATED WORK

Many state reduction techniques have been proposed in the communities of both protocol validation and model checking. Among those techniques, the most successful ones are perhaps partial order reduction (or POR) [1] [2] [6] [7] and simultaneous reachability analysis (or SRA) [5] [8] [9] [3]. Both POR and SRA are based on the observation that many protocol properties do not distinguish the ordering of independent events. Therefore, in order to verify those properties, it is sufficient to only explore one of the possibly many interleavings of each partial order. What is different between POR and SRA is that they use different approaches to avoid exploring redundant interleavings of the same partial order. In POR, a selective search is used to avoid exploring redundant interleavings. At each global state, a subset of executable transitions is identified and only these transitions are explored to derive successor states. This is in contrast to full reachability analysis which explores every executable transition at a global state. In SRA, redundant interleavings are avoided by allowing multiple processes to proceed simultaneously. At each global state, a set of simultaneously executable transition sets is identified. Each such set contains at most one transition from each process and can be executed in an arbitrary order to derive a successor state. The other interleavings of such a set are not explored.

Like SRA, BSRA also allows multiple processes to proceed simultaneously to derive a successor state. However, BSRA significantly differs from SRA, which does not have the notions of local/global blocking points. In SRA, a process can execute at most one transition from a global state to one of its successor states. This is in contrast to BSRA in which a process may execute a sequence of transitions from a global blocking point to one of its successor blocking points. Executing more transitions allows more intermediate states to be skipped. Therefore, BSRA has a higher potential to reduce the number of states that have to be explored.

6. CONCLUSION

In this paper, we have presented a new reachability analysis approach, namely BSRA, for protocol validation. We have shown that even though BSRA only explores a set of global blocking points, it can still be used to detect logical errors including non-executable transition, deadlock, channel overflow and unspecified reception. Since it is often the case that only a small number of global states are global blocking points, BSRA has the potential to significantly reduce the number of states that have to be explored for protocol validation.

In [4], we reported a BSRA-based algorithm for deadlock detection of asynchronous message-passing programs that are modeled as extended finite state machines (EFSMs). Our empirical results in [4] indicate that BSRA can achieve significantly better state reduction rates than POR and SRA for deadlock detection. Currently, we are developing a protocol validation tool based on BSRA. We plan to use this tool to conduct an empirical evaluation of the effectiveness of BSRA for protocol validation. In particular, we want to compare BSRA to POR and SRA in terms of their state reduction rates for protocol validation.

7. REFERENCES


8. SUPPLEMENTARY MATERIAL

This material is provided only for the review purpose and includes the formal proofs for Theorems 1, 2, 3, and 4.

We first introduce some notations. Let \( \Pi \) be a protocol. Let \( BSRG(\Pi) \) be the BSRG of \( \Pi \) constructed by the BSR-ALGORITHM. Let \( BSRG(CO(\Pi, C)) \), where \( C \) is a channel of \( \Pi \), be the BSRG of \( \Pi \) constructed by the BSR-ALGORITHM for channel \( C \). Let \( BSRGUSR(\Pi) \) be the BSRG of \( \Pi \) constructed by applying the BSR-ALGORITHM to the augmented version of \( \Pi \) for detecting unspecified reception. We will augment the above BSRGs by adding the initial global state \( g_0 \) of \( \Pi \) and adding an edge from \( g_0 \) to each of the initial blocking points. (Recall that BSR starts from \( \text{frontier}(g_0) \) instead of \( g_0 \).) We will use \( BSRG^*(\Pi), BSRG^*(\Pi, C), \) and \( BSRGUSR^*(\Pi) \) to denote the augmented versions of the above graphs, respectively. Note that \( g_0 \) is the single root of those augmented BSRGs.

Let \( n \) and \( n' \) be two nodes in a BSRG. We write \( n \succeq n' \) if there exists an edge \( e \) from \( n \) to \( n' \). If \( n \) is the initial global state \( g_0 \) (and \( n' \) is an initial blocking point), then we call \( e \) an initial edge and write \( e = Q \), where \( Q \) is a simultaneous sequence set at \( g_0 \), indicating that \( n' \) can be reached by executing \( Q \) from \( g_0 \). Otherwise, \( e \) is a regular edge, and we write \( e = (R, Q) \), where \( R \) is a simultaneous transition set and \( Q \) is a simultaneous sequence set, indicating that \( b' \) can be reached from \( b \) by executing \( R \) followed by \( Q \). We write \( n \prec n' \) if there exists a path \( H = e_1e_2 \ldots e_m \), where \( m \geq 1 \), from \( n \) to \( n' \) such that \( n = n_1 \prec n_2 \prec \cdots \prec n_m = n' \). We use greek letters, such as \( \sigma, \gamma, \alpha, \) and \( \beta \), as well as their subscripted forms, to represent a transition sequence that may be empty. We use \( [\sigma] \) to denote the length of a sequence \( \sigma \). We call a non-empty transition sequence as a r-starting sequence if the first transition in the sequence is a receive transition. Let \( R \) be a simultaneous transition set and \( Q \) a simultaneous sequence set. We use \( L(R) \) and \( L(Q) \) to represent the linearizations of \( R \) and \( Q \), respectively. For an initial edge \( e = Q \), we write \( L(e) = L(Q) \); for a regular edge \( e = (R, Q) \), we write \( L(e) = \{\alpha \beta | \alpha \in L(R) \land \beta \in L(Q)\} \). Let \( H = e_1e_2 \ldots e_m \) be a path. We write \( L(H) = \{\beta_1 \cdot \beta_2 \cdots \beta_m | \beta_i \in L(e_i)\} \). We will designate \( n \) as the number of processes in a protocol. We will use \( i, j, k \) to range from \( 1 \) to \( n \), unless otherwise specified.

**Lemma 1.** Let \( \Pi = (P, C, M) \) be a protocol. Let \( b \) be a blocking point in \( BSRG^*(\Pi) \). Let \( g \) be a global state of \( \Pi \) such that \( b \prec g \), where \( \sigma \downarrow \) is either empty or a r-starting sequence. Then, there must exist a blocking point \( b' \) in \( BSRG^*(\Pi) \) such that \( (1) \ b \prec b' \) and \( (2) \) given \( \gamma \in L(H), \gamma \downarrow = \sigma \downarrow \cdot \alpha \downarrow \cdot \gamma \downarrow \) (i.e., \( \sigma \downarrow \alpha \downarrow \gamma \downarrow \) is a prefix of \( \gamma \downarrow \)).

**Proof.** Let \( \sigma \downarrow = \beta_1 \cdot \beta_2 \cdot \cdots \beta_{q-1} \beta_q \cdot \gamma \downarrow \) be a send transition sequence with \( \beta_q \cdot \gamma \downarrow \) being a receive transition. Then, \( \sigma \downarrow \) is a prefix of \( \gamma \downarrow \).

**Lemma 2.** Let \( \Pi = (P, C, M) \) be a protocol. Let \( g_0 \) be the initial global state. Let \( g \) be a reachable global state such that \( g_0 \Downarrow g \). Then, there must exist a blocking point \( b \) in \( BSRG^*(\Pi) \) such that \( (1) \ b \prec b \) and \( (2) \) given \( \gamma \in L(H), \gamma \downarrow = \sigma \downarrow \cdot \alpha \downarrow \cdot \gamma \downarrow \) (i.e., \( \sigma \downarrow \alpha \downarrow \gamma \downarrow \) is a prefix of \( \gamma \downarrow \)).

**Proof.** Let \( \sigma \downarrow = \beta_1 \cdot \beta_2 \cdot \cdots \beta_{q-1} \beta_q \cdot \gamma \downarrow \) be a send transition sequence with \( \beta_q \cdot \gamma \downarrow \) being a receive transition. Then, \( \sigma \downarrow \) is either empty or a r-starting sequence. Also observe that transitions in \( Q \) but not in \( \sigma \) does not affect the executability of \( \sigma \). Therefore, \( \sigma \downarrow \) must be an executable transition sequence from \( \Pi \). By Lemma 1, there must exist a blocking point \( b \) in \( BSRG^*(\Pi) \) such that \( (1) \ b \prec b \) and \( (2) \) given \( \gamma \in L(H), \gamma \downarrow = \sigma \downarrow \cdot \alpha \downarrow \cdot \gamma \downarrow \). Then, \( g_0 \Downarrow b \), and given \( \gamma \in L(H) \), \( \gamma \downarrow = \sigma \downarrow \cdot \alpha \downarrow \cdot \gamma \downarrow \). This concludes our proof.

**Theorem 1.** The BSR-ALGORITHM reports a transition \( t \) of a protocol \( \Pi \) as a non-executable transition if \( \Pi \) is a non-executable transition of \( \Pi \).
Proof. The Only if part is trivially true. In the following, we only show the if part. It suffices to show that any transition that is executable will be executed by the BSRA algorithm. Let $g$ be a global state such that $g_0 \xrightarrow{a} g$. Let $t$ be a transition that is executable at a global state $g$. Let $g'$ be the state reached by executing $t$ at $g$. Let $\sigma' = \sigma \cdot t$.

Then, $g_0 \xrightarrow{\sigma'} g'$. Therefore, by Lemma 2, there must exist a blocking point $b$ in $BSRG^{*}(I)$ such that $g_0 \xrightarrow{\sigma'} g'$.

Then, $b \in L(H)$. Therefore, all the transitions in $\sigma'$ including $t$ will be executed by the BSRA algorithm. This concludes our proof.

Theorem 2. The BSRA algorithm reports a deadlock of a protocol $\Pi$ iff $\Pi$ has a deadlock.

Proof. Again we will only show the if part. Let $g_0$ be the initial global state of $\Pi$. Let $d$ be a deadlock state of $\Pi$ such that $g_0 \xrightarrow{\sigma} d$, where $|\sigma| \geq 0$. Then, by Lemma 2, there must exist a blocking point $b$ in $BSRG^{*}(I)$ such that $g_0 \xrightarrow{\sigma} b$. Let $\gamma \in L(H)$. Then, $\gamma \downarrow = \sigma \cdot \alpha_1$. Note that since $d$ is a deadlock state, $\alpha_1$ must be empty. This means that $d$ must be same as $b$ and thus will be reported. This concludes our proof.

Lemma 3. Let $\Pi = (P, C, M)$ be a protocol. Let $C_{ij}$ be a channel of $\Pi$. Let $b$ be a blocking point in $BSRG^{*}(I), C_{ij})$. Let $g$ be a channel overflow state for $C_{ij}$ such that $g_0 \xrightarrow{\sigma} g$. Then, there must exist a blocking point $b$ in $BSRG^{*}(I, C_{ij})$ such that (1) $b \xrightarrow{\sigma} b'$; and (2) given $\gamma \in L(H)$, $\gamma \downarrow = \sigma \cdot \alpha_k$, where $\alpha_k$ is a sequence of send and/or receive transitions if $k \neq j$ and a sequence of send transitions only if $k = j$.

Proof. The proof is the same as the proof for Lemma 1, except that it uses mappings $f'$ and $h'$ that are slightly different from mappings $f$ and $h$. Mapping $f'$ is defined such that $f'(P_i) = f(P_i)$ if $i \neq j$ and $f'(P_j) = \varepsilon$ if $i = j$. Then, $R' = \{f'(P_i) | f'(P_i) \neq \varepsilon\}$. Then, $R'$ is generated as a simultaneous transition set at $b$ by the BSRA algorithm. Recall that the BSRA algorithm generates additional simultaneous transition sets to delay the execution of a receive transition from $C_{ij}$. The rest of the proof is not affected. Note that with $f'$, $H$ cannot include any extra receive transition for $P_j$, i.e., a transition that is not in $\sigma$. That is, $\alpha_k$ can contain only extra send transition. This concludes our proof.

Lemma 4. Let $\Pi = (P, C, M)$ be a protocol. Let $g_0$ be the initial global state. Let $g$ be a channel overflow state for channel $C_{ij}$ such that $g_0 \xrightarrow{\sigma} g$. Then, there must exist a blocking point $b$ in $BSRG^{*}(I, C_{ij})$ such that (1) $g_0 \xrightarrow{\sigma} b'$; and (2) given $\gamma \in L(H)$, $\gamma \downarrow = \sigma \cdot \alpha_k$, where $\alpha_k$ is a sequence of send and/or receive transitions if $k \neq j$ and a sequence of send transitions only if $k = j$.

Proof. The proof can be easily constructed as it shares exactly the same line of thoughts as the proof for Lemma 2.

Theorem 3. Let $\Pi$ be a protocol. Assume that the BSRA algorithm is applied to a channel $C$ of $\Pi$. The algorithm reports a channel overflow iff $\Pi$ has a channel overflow state for $C$.

Proof. Again we only show the if part. Let $C = C_{ij}$ of $\Pi$, where $1 \leq i, j \leq n$ and $i \neq j$. Let $g$ be a channel overlap state for a channel $C_{ij}$ such that $g_0 \xrightarrow{\sigma} g$. By Lemma 4, there exists a blocking point $b$ in $BSRG^{*}(I, C_{ij})$ such that $g_0 \xrightarrow{\sigma} b$. Let $\gamma \in L(H)$. Then, $\gamma \downarrow = \sigma \cdot \alpha_j$, where $\alpha_j$ is a sequence of send transitions. Since $\alpha_j$ contains no receive transitions, if $C_{ij}$ overflows at $g$, then it also overflows at $b$. Therefore, the BSRA algorithm will report a channel overflow.

Lemma 5. Let $\Pi = (P, C, M)$ be a protocol. Let $b$ be a blocking point in $BSRG^{*}(I, C_{ij})$. Let $g$ be an unspecified reception state of $I$ such that $b \xrightarrow{\sigma} g$, where $\sigma$ is either empty or a r-starting sequence. Assume that $g$ is due to an unspecified reception pair $(s, m)$ let proc$(s) = P_k$. Then, there must exist a blocking point $b'$ in $BSRG^{*}(I, C_{ij})$ such that (1) $b \xrightarrow{\sigma} b'$; and (2) given $\gamma \in L(H)$, $\gamma \downarrow = \sigma \cdot \alpha_i$, where $\alpha_i$ is a prefix of $\gamma \downarrow$ if $i \neq k$, and $\gamma \downarrow = \sigma \cdot \alpha_i$ if $i = k$.

Proof. The proof is the same as the proof for Lemma 1, except that it uses mappings $f'$ and $h'$ that are slightly different from mappings $f$ and $h$. Mapping $f'$ is defined such that $f'(P_i) = f(P_i)$ if $i \neq k$ and $f'(P_k) = \varepsilon$ if $i = k$. Mapping $g'$ is defined such that $h'(P_i) = h(P_i)$ if $i \neq k$ and $h'(P_k) = \varepsilon$ if $i = k$. Let $R' = \{f'(P_i) | f'(P_i) \neq \varepsilon\}$. Let $Q' = \{h'(P_i) | h'(P_i) \neq \varepsilon\}$. Let $\Pi'$ be the protocol obtained by augmenting $\Pi$ as described in Section 4.2. In $\Pi'$, every local state has a PET. Thus, $R'$ must be generated as a simultaneous transition set at $b$ by the BSRA algorithm. Also, every local state is a local blocking point in $\Pi'$, which means that $Q'$ is also generated as a simultaneous sequence set at $b$ by the BSRA algorithm. The rest of the proof is not affected. Note that with $f'$, $H$ cannot include any extra transition for $P_k$, i.e., a transition that is not in $\sigma$. This concludes our proof.

Lemma 6. Let $\Pi = (P, C, M)$ be a protocol. Let $g_0$ be the initial global state. Let $g$ be an unspecified reception state of $\Pi$ such that $g_0 \xrightarrow{\sigma} g$. Assume that $g$ is due to an unspecified reception pair $(s, m)$. Let proc$(s) = P_k$. Then, there must exist a blocking point $b$ in $BSRG^{*}(I, C_{ij})$ such that (1) $g_0 \xrightarrow{\sigma} b$; and (2) given $\gamma \in L(H)$, $\gamma \downarrow = \sigma \cdot \alpha_i$, where $\alpha_i$ is a prefix of $\gamma \downarrow$ if $i \neq k$, and $\gamma \downarrow = \sigma \cdot \alpha_i$ if $i = k$.

Proof. The proof can be easily constructed as it shares exactly the same line of thoughts as the proof for Lemmas 1 and 4.

Theorem 4. Let $\Pi$ be a protocol. Let $\Pi'$ be the protocol obtained by augmenting $\Pi$. Applying the BSRA algorithm to $\Pi'$ reports every unspecified reception pair in $\Pi$.

Proof. Again we only show the if part. Let $\Pi = (P, C, M)$ be a protocol. Let $g$ be an unspecified reception state caused by an unspecified reception pair $(s, m)$ such that $g_0 \xrightarrow{\sigma} g$. Let proc$(s) = P_k$. By Lemma 6, there exists a blocking point $b$ in $BSRG^{*}(I, C_{ij})$ such that $g_0 \xrightarrow{\sigma} b$. Let $\gamma \in L(H)$. Then, $\gamma \downarrow = \sigma \cdot \alpha_i$. This means that $P_k$ stays at local state $s$ at $b$. Therefore, if $g$ is an unspecified reception state caused by $(s, m)$, $b$ is also an unspecified reception state caused by $(s, m)$. This concludes our proof.