Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs
(Extended Abstract)

Bernard Boigelot*  Patrice Godefroid
Université de Liège  Lucent Technologies – Bell Laboratories
Institut Montefiore, B28  1000 E. Warrenville Road
4000 Liège Sart-Tilman, Belgium  Naperville, IL 60566, U.S.A.
Email: boigelot@montefiore.ulg.ac.be  Email: god@bell-labs.com

Abstract

We study the verification of properties of communication protocols modeled by a finite set of finite-state machines that communicate by exchanging messages via unbounded FIFO queues. It is well-known that most interesting verification problems, such as deadlock detection, are undecidable for this class of systems. However, in practice, these verification problems may very well turn out to be decidable for a subclass containing most "real" protocols.

Motivated by this optimistic (and, we claim, realistic) observation, we present an algorithm that may construct a finite and exact representation of the state space of a communication protocol, even if this state space is infinite. Our algorithm performs a loop-first search in the state space of the protocol being analyzed. A loop-first search is a search technique that attempts to explore first the results of successive executions of loops in the protocol description (code). A new data structure named Queue-content Decision Diagram (QDD) is introduced for representing (possibly infinite) sets of queue-contents. Operations for manipulating QDDs during a loop-first search are presented.

A loop-first search using QDDs has been implemented, and experiments on several communication protocols with infinite state spaces have been performed. For these examples, our tool completed its search, and produced a finite symbolic representation for these infinite state spaces.

1 Introduction

State-space exploration is one of the most successful strategies for analyzing and verifying properties of finite-state concurrent reactive systems. It proceeds by exploring a global state graph representing the combined behavior of all concurrent components in the system. This is done by recursively exploring all successor states of all states encountered during the exploration, starting from a given initial state, by executing all enabled transitions in each state. The state graph that is explored is called the state space of the system. Many different types of properties of a system can be checked by exploring its state space: deadlocks, dead code, violations of user-specified assertions, etc. Moreover, the range of properties that state-space exploration techniques can verify has been substantially broadened during the last decade thanks to the development of model-checking methods for various temporal logics (e.g., [CES86, LP85, QS81, VW86]).

* "Aspirant" (Research Assistant) for the National Fund for Scientific Research (Belgium). The work of this author was done in part while visiting Bell Laboratories.
Verification by state-space exploration has been studied by many researchers (cf. [Lin89, Rud87]). The simplicity of the strategy lends itself to easy, and thus efficient, implementations. Moreover, verification by state-space exploration is fully automatic: no intervention of the designer is required. The main limit of state-space exploration verification techniques is the often excessive size of the state space. Obviously, this state-explosion problem is even more critical when the state space being explored is infinite.

In contrast with the last observation, we show in this paper that verification by state-space exploration is also possible for systems with infinite state spaces. Specifically, we consider communication protocols modeled by a finite set of finite-state machines that communicate by exchanging messages via unbounded FIFO queues. We present a state-space exploration algorithm that may construct a finite and exact representation of the state space of such a communication protocol, even if this state space is infinite. From this symbolic representation, it is then straightforward to verify many properties of the protocol, such as the absence of deadlocks, whether or not the number of messages stored in a queue is bounded, and the reachability of local and global states.

Of course, given an arbitrary protocol, our algorithm may not terminate its search. Indeed, it is well-known that unbounded queues can be used to simulate the tape of a Turing machine, and hence that most interesting verification problems are undecidable for this class of systems [BZ83]. However, in practice, these verification problems may very well turn out to be decidable for a subclass containing most "real" protocols. To support this claim, properties of several communication protocols with infinite state spaces have been verified successfully with the algorithm introduced in this paper.

In the next section, we formally define communication protocols. Our algorithm performs a loop-first search in the state space of the protocol being analyzed. A loop-first search is a search technique that attempts to explore first the results of successive executions of loops in the protocol description (code). This search technique is presented in Section 3. A new data structure, the Queue-content Decision Diagram (QDD), is introduced in Section 4 for representing (possibly infinite) sets of queue-contents. Operations for manipulating QDDs during a loop-first search are presented in Section 5. A loop-first search using QDDs has been implemented, and experiments on several communication protocols with infinite state spaces are reported in Section 6. This paper ends with a comparison between our contributions and related work.

2 Communicating Finite-State Machines

Consider a protocol modeled by a finite set $\mathcal{M}$ of finite-state machines that communicate with each other by sending and receiving messages via a finite set $Q$ of unbounded FIFO queues, modeling communication channels. Let $M_i$ denote the set of messages that can be stored in queue $q_i$, $1 \leq i \leq |Q|$. For notational convenience, let us assume that the sets $M_i$ are pairwise disjoint. Let $C_i$ denote the finite set of states of machine $M_i$, $1 \leq i \leq |\mathcal{M}|$.

Formally, a protocol $P$ is a tuple $(C, c_0, A, Q, M, T)$ where $C = C_1 \times \cdots \times C_{|\mathcal{M}|}$ is a finite set of control states, $c_0 \in C$ is an initial control state, $A$ is a finite set of actions, $Q$ is a finite set of unbounded FIFO queues, $M = \bigcup_{i=1}^{|\mathcal{M}|} M_i$ is a finite set of messages, and $T$ is a finite set of transitions, each of which is a triple of the form $(c_1, op, c_2)$ where $c_1$ and $c_2$ are control states, and $op$ is a label of one of the forms $q_i!w$, where $q_i \in Q$ and $w \in M_i^*$, $q_i?w$, where $q_i \in Q$ and $w \in M_i^*$, or $a$, where $a \in A$.

A transition of the form $(c_1, q_i!w, c_2)$ represents a change of the control state from $c_1$ to $c_2$ while appending the messages composing $w$ to the end of queue $q_i$. A transition of the form $(c_1, q_i?w, c_2)$ represents a change of the control state from $c_1$ to $c_2$ while removing the messages composing $w$ from the head of queue $q_i$.

A global state of a protocol is composed of a control state and a queue-content. A queue-content
associates with each queue \( q_i \) a sequence of messages from \( M_i \). Formally, a global state \( \gamma \), or simply a state, of a protocol is an element of the set \( C_1 \times \cdots \times C_{|M|} \times M_1^* \times \cdots \times M^*_{|Q|} \). A global state \( \gamma = (c(1), c(2), \ldots, c(|M|), w(1), w(2), \ldots, w(|Q|)) \) assigns to each finite-state machine \( M_i \) a “local” (control) state \( c(i) \in C_i \), and associates with each queue \( q_j \) a sequence of messages \( w(j) \in M_j^* \) which represents the content of \( q_j \) in the global state \( \gamma \). The initial global state of the system is \( \gamma_0 = (c_0(1), c_0(2), \ldots, c_0(|M|), \varepsilon, \ldots, \varepsilon) \), i.e., we assume that all queues are initially empty.

A global transition relation \( \rightarrow \) is a set of triples \((\gamma, a, \gamma')\), where \( \gamma \) and \( \gamma' \) are global states, and \( a \in A \cup \{\tau\} \). Let \( \gamma \xrightarrow{a} \gamma' \) denote \((\gamma, a, \gamma') \in \rightarrow \). Relation \( \rightarrow \) is defined as follows:

1. if \((c_1, q_i, w, c_2) \in T, \) then \((c_1(1), c_1(2), \ldots, c_1(|M|), w(1), w(2), \ldots, w(|Q|)) \xrightarrow{T} (c_2(1), c_2(2), \ldots, c_2(|M|), w''(1), w''(2), \ldots, w''(|Q|)) \) where \( w''(i) = w'(i)w \) and \( w''(j) = w'(j) \), \( j \neq i \) (the control state changes from \( c_1 \) to \( c_2 \) and \( w \) is appended to the end of queue \( q_i \));

2. if \((c_1, q_i, w, c_2) \in T, \) then \((c_1(1), c_1(2), \ldots, c_1(|M|), w(1), w(2), \ldots, w(|Q|)) \xrightarrow{T} (c_2(1), c_2(2), \ldots, c_2(|M|), w''(1), w''(2), \ldots, w''(|Q|)) \) where \( w''(i) = uw''(i) \) and \( w''(j) = w'(j) \), \( j \neq i \) (the control state changes from \( c_1 \) to \( c_2 \) and \( w \) is removed from the head of queue \( q_i \));

3. if \((c_1, c_2) \in T, \) then \((c_1(1), c_1(2), \ldots, c_1(|M|), w(1), w(2), \ldots, w(|Q|)) \xrightarrow{T} (c_2(1), c_2(2), \ldots, c_2(|M|), w''(1), w''(2), \ldots, w''(|Q|)) \) with \( w''(i) = w'(i), \) for all \( 1 \leq i \leq |Q| \) (the control state changes from \( c_1 \) to \( c_2 \) while the action \( a \) is performed).

A global state \( \gamma' \) is said to be reachable from another global state \( \gamma \) if there exists a sequence of global transitions \((\gamma_{n-1}, a_i, \gamma_n)\), \( 1 \leq i \leq n \), such that \( \gamma = \gamma_0 \xrightarrow{a_1} \gamma_1 \cdots \xrightarrow{a_n} \gamma_n = \gamma' \). The global state space of a system is the (possibly infinite) set of all states that are reachable from the initial global state \( \gamma_0 \).

**Example 1** As an example of communication protocol, consider the well-known Alternating-Bit Protocol [BSW69]. This protocol can be modeled by two finite-state machines Sender and Receiver that communicate via two unbounded FIFO queues \( StoR \) (used to transmit messages from the Sender to the Receiver) and \( RtoS \) (used to transmit acknowledgments from the Receiver to the Sender).

Precisely, the Alternating-Bit Protocol is modeled by the protocol \((C, c_0, A, Q, M, T)\) where \( C = C_{\text{Sender}} \times C_{\text{Receiver}}, \) where \( C_{\text{Sender}} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \) and \( C_{\text{Receiver}} = \{1, 2, 3, 4, 5, 6, 7, 8\}; \) \( c_0 = \{1, 1\} \); \( A = \{\text{Snd}, \text{Rev}, \text{timeout}\}; \) \( Q = \{\text{StoR}, \text{RtoS}\}; \) \( M = M_{\text{StoR}} \cup M_{\text{RtoS}}, \) where \( M_{\text{StoR}} = \{\text{msg}0, \text{msg}1\} \) and \( M_{\text{RtoS}} = \{\text{ack}0, \text{ack}1\}; \) and \( T \) contains the transitions \((s_1, r_1), op, (s_2, r_2)\) where

![Figure 1: Alternating-Bit Protocol](image-url)
either $r_1 = r_2$ and $(s_1, op, s_2)$ is a transition in the Sender machine of Figure 1, or $s_1 = s_2$ and $(r_1, op, r_2)$ is a transition in the Receiver machine of Figure 1. The action $Snd$ models a request to the Sender, coming from a higher-level application, to transmit data to the Receiver side. The actual data that are transmitted are not modeled, only message numbers $msg0$ and $msg1$ are transmitted over the queues. Similarly, the action $Rev$ models the transmission of data received by the Receiver to a higher-level application. The actions labeled by timeout model the expiration of timeouts.

3 Loop-First Search

All state-space exploration techniques are based on a common principle: they spread the reachability information along the transitions of the system to be analyzed. The exploration process starts with the initial global state of the system, and tries at every step to enlarge its current set of reachable states by propagating these states through transitions. The process terminates when a stable set is reached.

In order to use the above state-space exploration paradigm for verifying properties of systems with infinite state spaces, two basic problems need to be solved: one needs a representation for infinite sets of states, as well as a search technique that can explore an infinite number of states in a finite amount of time.

In the context of the verification of communication protocols as defined in the previous section, our solution to the first problem is to represent the control part explicitly and the queue-contents "symbolically". Specifically, we will use special data structures for representing (possibly infinite) sets of queue-contents associated with reachable control states.

To solve the second problem, we will use these data structures for simultaneously exploring (possibly infinite) sets of global states rather than individual global states. This may make it possible to reach a stable representation of the set of reachable global states, even if this set is infinite. In order to simultaneously generate sets of reachable states from a single reachable state, meta-transitions [BW94] can be used. Given a loop that appears in the protocol description and a control state $c$ in that loop, a meta-transition is a transition that generates all global states that can be reached after repeated executions of the body of the loop. By definition, all these global states have the same control state $c$.

The classical enumerative state-space exploration algorithm can then be rewritten in such a way that it works with sets of global states, i.e., pairs of the form (control state, data structure), rather than with individual states. Initially, the search starts from an initial global state. At each step during the search, whenever meta-transitions are executable, they are explored first, which is a heuristic aimed at generating many reachable states as quickly as possible. This is why we call such a search a loop-first search. The search terminates if the representation of the set of reachable states stabilizes. This happens when, for every control state, every new deducible queue-content is included in the current set of queue-contents associated with that control state. At this moment, the final set of pairs (control state, data structure) represents exactly the state space of the protocol being analyzed.

In order to apply the verification method described above, we need to define a data structure for representing (possibly infinite) sets of queue-contents, and algorithms for manipulating these data structures. Specifically, whenever a transition or a meta-transition is executed from a pair (control state, data structure) during a loop-first search, the new pair (control state, data structure) obtained after the execution of this (meta-)transition has to be determined. Therefore, from any given such data structure, one needs to be able to compute a new data structure representing the effect of sending messages to a queue $(q_i, w)$ and receiving messages from a queue $(q_i, ?w)$, as well as the result of executing frequent types of meta-transitions, such as repeatedly sending messages on a queue $(q_i, w^*)$, repeatedly receiving messages from a queue $(q_i, ?w^*)$, and repeatedly receiving the
sequence of messages $w_1$ from a queue $q_i$ followed by sending another sequence of messages $w_2$ on another queue $q_j$, $i \neq j$, $((q_i; w_1; q_j; w_2)')$. Finally, basic operations on sets are also needed, such as checking if a set of queue-contents is included in another set, and computing the union of two sets of queue-contents.

4 Queue-content Decision Diagrams

Queue-content Decision Diagrams (QDDs) are data structures that satisfy all the constraints listed in the previous section. A QDD is a special type of finite-state automaton on finite words. A finite-state automaton on finite words is a tuple $A = (\Sigma, S, \Delta, s_0, F)$, where $\Sigma$ is an alphabet (finite set of symbols), $S$ is a finite set of states, $\Delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$ is a transition relation ($\epsilon$ denotes the empty word), $s_0 \in S$ is the initial state, and $F \subseteq S$ is a set of accepting states. A transition $(s, a, s')$ is said to be labeled by $a$. A finite sequence (word) $w = a_1a_2 \ldots a_n$ of symbols in $\Sigma$ is accepted by the automaton $A$ if there exists a sequence of states $\sigma = s_0 \ldots s_n$ such that $\forall 1 \leq i \leq n : (s_{i-1}, a_i, s_i) \in \Delta$, and $s_n \in F$. The set of words accepted by $A$ is called the language accepted by $A$, and is denoted by $L(A)$. Let us define the projection $w|_{M_i}$ of a word $w$ on a set $M_i$ as the subsequence of $w$ obtained by removing all symbols in $w$ that are not in $M_i$. An automaton is said to be deterministic if it does not contain any transition labeled by the empty word, and if for each state, all the outgoing transitions are labeled by different symbols.

Precisely, QDDs are defined as follows.

**Definition 2** A QDD $A$ for a protocol $P$ is a deterministic finite-state automaton $(M, S, \Delta, s_0, F)$ on finite words such that

$$\forall w \in L(A) : w = w|_{M_1}w|_{M_2} \ldots w|_{M_n}.$$ 

A QDD is associated with each control state reached during a loop-first search, and represents a set of possible queue-contents for this control state. Each word $w$ accepted by a QDD defines one queue-content $w|_{M_i}$ for each queue $q_i$ in the protocol.

By Definition 2, a total order $<$ is implicitly defined on the set $Q$ of all queues $q_i$ in the protocol such that, for all QDDs for this protocol, transitions labeled by messages in $M_i$ always appear before transitions labeled by messages in $M_j$ if $i < j$. Therefore, for all QDDs for a protocol, a given queue-content can only be represented by one unique word. In other words, Definition 2 implicitly defines a “canonical” representation for each possible queue-content. Note that this does not imply that QDDs are canonical representations for sets of queue-contents.

5 Operations on QDDs

Standard algorithms on finite-state automata on finite words can be used for checking if the language accepted by a QDD is included in the language accepted by another QDD, for computing the union of QDDs, etc. (e.g., see [LP81]). In what follows, $A_1 \cup A_2$ will denote an automaton that accepts the language $L(A_1) \cup L(A_2)$, while DETERMINIZE$(A)$ will denote a deterministic automaton that accepts the language $L(A)$. We will write “Add $(s, w, s')$ to $\Delta$” to mean that transitions $(s_{i-1}, a_i, s_i)$, $1 \leq i \leq n$, such that $w = a_1a_2 \ldots a_n$, $s_0 = s$, $s_n = s'$, and $s_i, 1 \leq i < n$, are new (fresh) states, are added to $\Delta$. 

We now describe how to perform the other basic operations on QDDs listed in Section 3.

Let $A$ be the QDD associated with a given control state $c$. Let $L(A)$ denote the language accepted by $A$, and let $L_{op}(A)$ denote the language that has to be associated with the control state $c'$ reached
Theorem 3 Let $A$ be a QDD, let $A'$ denote the automaton returned by $\text{SEND}(i, w, A)$, and let $L(A')$ denote the language accepted by $A'$. Then $A'$ is a QDD such that $L(A') = L_{q_i, w}(A)$.

Proof Proofs are omitted here due to space limitations. See the full paper. □

Theorem 4 Let $A$ be a QDD, let $A'$ denote the automaton returned by $\text{RECEIVE}(i, w, A)$, and let $L(A')$ denote the language accepted by $A'$. Then $A'$ is a QDD such that $L(A') = L_{q_i, w}(A)$.
SEND-STAR(queue Id i, word w, QDD (M, S, Δ, s₀, F))

For all states s ∈ S such that
\[ \exists w' ∈ (\bigcup_{j=1}^i M_j)^* : s₀ w' → s, \]
do the following operations:

- Add two new states \( s' \) and \( s'' \) to \( S \);
- For all transitions \( t = (s, m, s') \in Δ \) such that \( m ∈ M_j, j > i \):
  Replace \( t \) by \( (s', m, s'') \);
- For all transitions \( t = (s'', m, s) \in Δ \) such that \( m ∈ M_j, j > i \):
  Replace \( t \) by \( (s'', m, s') \);
- Add \( (s, ε, s') \), \( (s', ε, s'') \) and \( (s', w, s') \) to \( Δ \);
- If \( s ∈ F \), add \( s'' \) to \( F \);

Return DETERMINIZE((M, S, Δ, s₀, F)).

RECEIVE-STAR(queue Id i, word w, QDD (M, S, Δ, s₀, F))

For all states s ∈ S such that
\[ \exists w' ∈ (\bigcup_{j=1}^i M_j)^* : s₀ w' → s, \]
do the following operations:

- Add a new state \( s' \) to \( S \);
- For all transitions \( t = (s, m, s') \in Δ \) such that \( m ∈ M_j, j ≥ i \):
  Replace \( t \) by \( (s', m, s') \);
- For all transitions \( t = (s, m, s') \in Δ \) such that \( m ∈ M_j, j ≥ i \):
  Replace \( t \) by \( (s', m, s') \);
- For all states \( s'' \in S \) such that \( \exists w' ∈ \{w\}^* : s' w' → s'' \):
  Add a transition \( (s, ε, s'') \) to \( Δ \);
- If \( s ∈ F \), add \( s'' \) to \( F \);

Return DETERMINIZE((M, S, Δ, s₀, F)).

Figure 3: \((q_i!w)^*\) and \((q_i?w)^*\)

Proof  See the full paper. ■

We now consider the meta-transitions discussed in Section 3. The operation \((q_i!w)^*\) denotes the union of all possible queue-contents obtained after sending \( k \) sequences of messages \( w ∈ M_i^* \) to the queue \( q_i \) of the system, for all \( k ≥ 0 \). The operation \((q_i?w)^*\) denotes the union of all possible queue-contents obtained after receiving \( k \) sequences of messages \( w ∈ M_i^* \) from the queue \( q_i \) of the system, for all \( k ≥ 0 \). The operation \((q_i?w_1; q_j?w_2)^*\) denotes the union of all possible queue-contents obtained after receiving \( k \) sequences of messages \( w_1 ∈ M_j^* \) from the queue \( q_j \) and sending \( k \) sequences of messages \( w_2 ∈ M_j^* \) to the queue \( q_j \), for all \( k ≥ 0 \), and for \( i ≠ j \).

Let \( A \) be the QDD associated with a given control state \( c \). Let \( L(A) \) denote the language accepted by \( A \), and let \( L_{op}(A) \) denote the language that has to be associated with the control state \( c \) reached after the execution of a meta-transition \((c, op, c)\) with \( op ∈ \{(q_i!w)^*, (q_i?w)^*, (q_i?w_1; q_j?w_2)^*\} \). We have the following:

- \( L_{(q_i!w)^*}(A) = \{ w'' | \exists w' ∈ L(A), k ≥ 0 : w''|M_i = w'|M_i w^k | ∀ j ≠ i : w''|M_j = w'|M_j \} \),
- \( L_{(q_i?w)^*}(A) = \{ w'' | \exists w' ∈ L(A), k ≥ 0 : w'|M_i = w^k w'' | M_i | ∀ j ≠ i : w''|M_j = w'|M_j \} \),
RECEIVE-SEND-STAR\(_{(\text{queue}, i j), \text{word} w_1, \text{queue}, i j, \text{word} w_2, \text{QDD} (M, S, \Delta, s_0, F)}\) \\
\{ \\
\text{Let } n \text{ be the greatest integer such that} \\
\exists s_1, \ldots, s_{n+1} \in S : s_1 \not\rightarrow s_2 \not\rightarrow \cdots \not\rightarrow s_{n+1}, \\
\text{with } \forall 1 \leq k < l \leq n + 1 : s_k \neq s_l; \\
\text{Let } A_0 \text{ denote the QDD } (M, S, \Delta, s_0, F); \\
\text{For all } k, 1 \leq k \leq n + 1, \text{ compute } A_k = \text{SEND}(j, w_2, \text{RECEIVE}(i, w_1, A_{k-1})); \\
\text{If } L(A_{n+1}) = \emptyset: \\
\quad \bullet \text{ Return } \text{DETERMINIZE}(\cup_{k=0}^{n} A_k); \\
\text{If } L(A_{n+1}) \neq \emptyset: \\
\quad \bullet \text{ Let } p = 1; \\
\quad \bullet \text{ While } L(A_{n+1}) \neq L(\text{RECEIVE}(i, w_1^p, A_{n+1})): \\
\quad \quad p := p + 1; \\
\quad \bullet \text{ For all } k, 2 \leq k \leq p, \text{ compute } A_{n+k} = \text{SEND}(j, w_2, \text{RECEIVE}(i, w_1, A_{n+k-1})); \\
\quad \bullet \text{ Compute } A_{n+p+1} = \text{RECEIVE-STAR}(j, w_1^{p+1}, \text{DETERMINIZE}(\cup_{k=n+1}^{n+p+1} A_k)); \\
\quad \bullet \text{ Return } \text{DETERMINIZE}(\cup_{k=n+1}^{n+p+1} A_k). \\
\}

Figure 4: \((q_i ; w_1 ; q_j ; w_2)^*\)

\[ L(\# (q_i ; w_1 ; q_j ; w_2)) (A) = \{ w' | \exists w' \in L(A), k \geq 0 : w' | M_i = w_1^k w' | M_i \land w' | M_j = w_1^k w_j \land \forall l \not\in \{i, j\} : w' | M_l = w' | M_l \}. \]

Algorithms for computing a QDD \(A'\) that accepts all possible queue-contents obtained after the execution of a meta-transition of the form \((q_i ; w)^*\), \((q_i ; w_1)^*\), or \((q_i ; w_1 ; q_j ; w_2)^*\) on a QDD \(A = (M, S, \Delta, s_0, F)\) are given in Figures 3 and 4. The correctness of these algorithms is established by the following theorems.

**Theorem 5** Let \(A\) be a QDD, let \(A'\) denote the automaton returned by SEND-STAR\((\bar{i}, w, A)\), and let \(L(A')\) denote the language accepted by \(A'\). Then \(A'\) is a QDD such that \(L(A') = L_{(q_i ; w)^*}(A)\).

**Proof** See the full paper. ■

**Theorem 6** Let \(A\) be a QDD, let \(A'\) denote the automaton returned by RECEIVE-STAR\((\bar{i}, w, A)\), and let \(L(A')\) denote the language accepted by \(A'\). Then \(A'\) is a QDD such that \(L(A') = L_{(q_i ; w)^*}(A)\).

**Proof** See the full paper. ■

**Lemma 7** Let \(n\) and \(A_{n+1}\) be as defined in the algorithm RECEIVE-SEND-STAR\((i, w_1, j, w_2, A)\), with \(i \neq j\). If the language accepted by \(A_{n+1}\) is not empty, then there exists \(p\) such that \(0 < p \leq (n + 1)!\), and \(L(A_{n+1}) = L(\text{RECEIVE}(\bar{i}, w_1^p, A_{n+1}))\).

**Proof** See the full paper. ■

**Theorem 8** Let \(A\) be a QDD, let \(A'\) denote the automaton returned by RECEIVE-SEND-STAR\((\bar{i}, w_1, j, w_2, A)\), with \(i \neq j\), and let \(L(A')\) denote the language accepted by \(A'\). Then \(A'\) is a QDD such that \(L(A') = L_{(q_i ; w_1 ; q_j ; w_2)^*}(A)\).

**Proof** See the full paper. ■

It is worth noticing that, as a corollary of the last theorem, the language \(L_{(q_i ; w_1 ; q_j ; w_2)^*}(A)\) is regular.
6 Experimental Results

Consider again the Alternating-Bit protocol of Example 1. Meta-transitions are added to the protocol description for loops that match either \((q_i; w_i)^*\), \((q_i; w_i)^*\), or \((q_i; w_1; q_i; w_2)^*\). Precisely, the meta-transitions \((3, (RtoS; ack 1; StoR; msg 0)^*; 3), (3, (StoR; msg 0)^*; 3), (8, (RtoS; ack 0; StoR; msg 1)^*; 8), (8, (StoR; msg 1)^*; 8)\) are added to the set of transitions of the Sender, while the meta-transitions \((1, (StoR; msg 1; RtoS; ack 1)^*; 1)\) and \((5, (StoR; msg 0; RtoS; ack 0)^*; 5)\) are added to the set of transitions of the Receiver.

We have implemented (in C) a “QDD-package” containing an implementation of the algorithms for manipulating QDDs described in the previous section, and we have combined it with a loop-first search. Starting with the control state \((1, 1)\) and the QDD \((M, \{s_0\}, \emptyset, \emptyset, \{s_0\})\), which corresponds to the queue-content \(\epsilon\) for both queues \(StoR\) and \(RtoS\), the execution of the loop-first search for the Alternating-Bit protocol terminates after 5.9 seconds of computation on a SPARC 10 workstation. The number of transitions executed is 331. The largest QDD constructed during the search contains 21 states, and 52 control states are reachable from the initial state.

Many properties can be checked on the symbolic representation of the state space of the protocol obtained at the end of the search. For instance, it is then straightforward to prove that the protocol does not contain any deadlocks, that there are reachable control states where the number of messages in a queue is unbounded, that messages are always delivered in the correct order, etc.

Our tool has also been tested on several variants of the Alternating-Bit protocol, where the transitions labeled by “timeout” are removed from the protocol description, where the Sender/Receiver have various number of control states, etc. An interesting variant is the case where queues may lose messages (to model unreliable transmission media). In order to handle this case, it is sufficient to define one additional algorithm \(SEND\)-LOSSY(\(i, w, A\)), that merely returns \(A \cup SEND(\iow)\). We also performed experiments on several simple sliding-window protocols [Tan89], with various window sizes. For all these examples with infinite state spaces (more than 20 in total), our tool was able to successfully terminate its search within a few minutes of computation. This shows that, at least for this particular though important class of examples, our verification method is very useful and robust.

7 Comparison with Other Work and Conclusions

Although most verification problems are undecidable for arbitrary protocols modeled by communicating finite-state machines, decision procedures have been obtained for the verification of specific properties for limited sub-classes [KM69, RY86, GLR87, CFS87, Fins88, Jer91, SZ91, AJ93, AJ94, CFP96]. These sub-classes do not cover, e.g., the Alternating-Bit Protocol and the properties discussed in the previous section, which were easily verified using a loop-first search and QDDs.

Clearly, a necessary, but not sufficient, condition for the termination of our algorithm is that, for all reachable control states of the protocol, the language of queue-contents associated with that control state can be represented by a QDD. The class of protocols characterized by the above necessary condition is equivalent to the class of protocols for which, for each reachable control state of the protocol, the set of possible queue-contents can be described by a recognizable expression (i.e., a finite union of cartesian products of regular expressions). Indeed, it can be shown that any recognizable language can be represented by a QDD, and that any set of queue-contents represented by a QDD is a recognizable language.

In [Pac87], it is pointed out that several verification problems are decidable for the above class of protocols. However, no method is given for constructing a recognizable expression representing all possible queue-contents for each control state of the protocol. Actually, from [CFP96], it is easy to show that an algorithm for constructing such recognizable expressions, for any protocol in the class
defined above, cannot exist. In contrast, our contribution is to provide a practical algorithm which is able to compute such a representation for protocols in the above class, although not for all of them – this is impossible anyway.

In this paper, we have presented algorithms on QDDs for computing the effect of executing three frequent types of meta-transitions. These algorithms were sufficient for analyzing the protocols considered in the previous section. However, it is possible to design algorithms on QDDs for other types of meta-transitions as well. Interesting future work is to characterize precisely the set of meta-transitions that preserve recognizability and to provide a generic algorithm for computing the effect of the execution of any meta-transition in this class. These topics will be addressed in a forthcoming paper.

In [PP91], a verification method based on data-flow analysis is used to generate “flow equations” from the description of a set of communicating finite-state machines. By computing approximations of solutions for these equations, it is possible to show that the original system is free of certain types of errors. In contrast, our algorithm is able to produce an exact representation of the state space of the protocol being analyzed. This enables us not only to prove the absence of errors, but also to detect errors and to exhibit to the user sequences of transitions that lead to errors. Note that, obviously, approximations could also be used in our framework, e.g., for simplifying QDDs when they become too complex, or when the search does not seem to stop. For the examples we have considered so far, no approximations were necessary.

The idea of representing states partly explicitly (control part) and partly symbolically (data part) already appeared in [ACD93] for the verification of real-time systems, where dense-time domains are represented by polyhedra. This idea also appeared in [BW94], where the values of integer variables are represented by periodic vector sets. These symbolic representations are quite different from QDDs.

For digital hardware verification [BCM+90], the most commonly used symbolic representation is certainly the Binary Decision Diagram (BDD) [Bry92], which represents a boolean function (with a finite domain) as a directed acyclic graph. In [GL96], it is shown how QDDs can be combined with BDDs to improve the efficiency of classical BDD-based symbolic model-checking methods for verifying properties of communication protocols with large finite state spaces.

8 Acknowledgments

We wish to thank Michael Merritt and Mark Staskauskas for helpful comments on a preliminary version of this paper.

References


