Formal Methods in System Design, , 1–21 (1997)

Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs

BERNARD BOIGELOT* boigelot@montefiore.ulg.ac.be
Université de Liège, Institut Montefiore, B28, 4000 Liège Sart-Tilman, Belgium.

PATRICE GODEFROID god@bell-labs.com
Bell Laboratories, Lucent Technologies, 1000 E. Warrenville Road, Naperville, IL 60566, U.S.A.

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.

Keywords: symbolic verification, protocols, infinite state spaces, queues, model checking, testing

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., [13, 20, 24, 29]).

* "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. [21, 26]). 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 for constructing a *finite* and *exact* representation of the state space of such a communication protocol, even if its 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 [8]. 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 $\mathcal{M}_i$, $1 \leq i \leq |\mathcal{M}|$.

Formally, a protocol P is a tuple (C, c_0, Act, 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*, Act is a finite set of *actions*, Q is a finite set of unbounded FIFO queues, $M = \cup_{i=1}^{|Q|} M_i$ 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 $a \in Act$.

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* γ, or simply a *state*, of a protocol is an element of the set $C_1 \times \cdots \times C_{|\mathcal{M}|} \times M_1^* \times \cdots \times M_{|Q|}^*$. A global state $\gamma = (c(1), c(2), \ldots, c(|\mathcal{M}|), w(1), w(2), \ldots, w(|Q|))$ assigns to each finite-state machine $\mathcal{M}_i$ 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 γ. The *initial global state* of the system is $\gamma_0 = (c_0(1), c_0(2), \ldots, c_0(|\mathcal{M}|), \varepsilon, \ldots, \varepsilon)$, i.e., we assume that all queues are initially empty.

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

- if $(c_1, q_i!w, c_2) \in T$, then $(c_1(1), c_1(2), \ldots, c_1(|\mathcal{M}|), w'(1), w'(2), \ldots, w'(|Q|)) \xrightarrow{\tau} (c_2(1), c_2(2), \ldots, c_2(|\mathcal{M}|), w''(1), w''(2), \ldots, w''(|Q|))$ 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);

- if $(c_1, q_i?w, c_2) \in T$, then $(c_1(1), c_1(2), \ldots, c_1(|\mathcal{M}|), w'(1), w'(2), \ldots, w'(|Q|)) \xrightarrow{\tau} (c_2(1), c_2(2), \ldots, c_2(|\mathcal{M}|), w''(1), w''(2), \ldots, w''(|Q|))$ where $w'(i) = ww''(i)$ 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);

- if $(c_1, a, c_2) \in T$, then $(c_1(1), c_1(2), \ldots, c_1(|\mathcal{M}|), w'(1), w'(2), \ldots, w'(|Q|)) \xrightarrow{a} (c_2(1), c_2(2), \ldots, c_2(|\mathcal{M}|), w''(1), w''(2), \ldots, w''(|Q|))$ 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 γ' is said to be *reachable* from another global state γ if there exists a sequence of global transitions $(\gamma_{i-1}, a_i, \gamma_i)$, $1 \leq i \leq n$, such that $\gamma = \gamma_0 \xrightarrow{a_1} \gamma_1 \cdots \gamma_{n-1} \xrightarrow{a_n} \gamma_n = \gamma'$. The *global state space* of a system is the (possibly infinite) set of all states that are reachable from the initial global state γ_0.

EXAMPLE: As an example of communication protocol, consider the well-known Alternating-Bit Protocol [4]. This protocol can be modeled by two finite-state

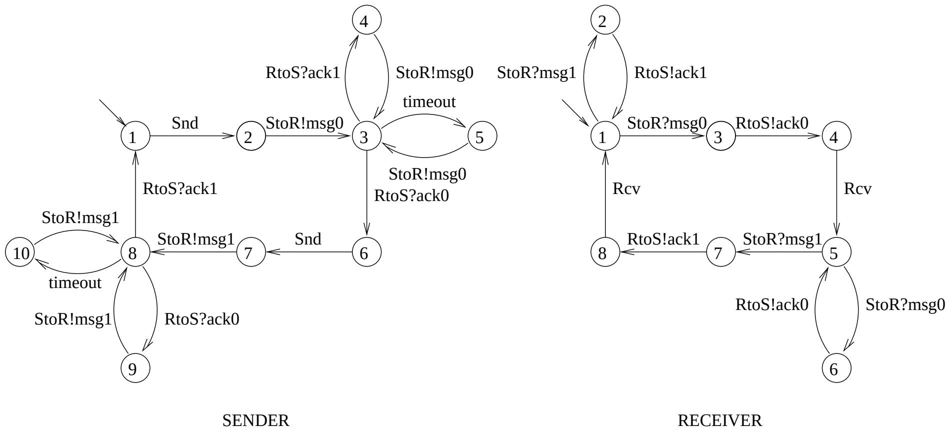

Figure 1. Alternating-Bit Protocol

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, Act, Q, M, T) where $C = C_{Sender} \times C_{Receiver}$, where $C_{Sender} = \{1,2,3,4,5,6,7,8,9,10\}$ and $C_{Receiver} = \{1,2,3,4,5,6,7,8\}$; $c_0 = (1,1)$; $Act = \{Snd, Rcv, timeout\}$; $Q = \{StoR, RtoS\}$; $M = M_{StoR} \cup M_{RtoS}$, where $M_{StoR} = \{msg0, msg1\}$ and $M_{RtoS} = \{ack0, ack1\}$; and T contains the transitions $((s_1,r_1), op, (s_2,r_2))$ where 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 *Rcv* 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* [7] 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 Σ is an alphabet (finite set of symbols), S is a finite set of states, $\Delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$ is a transition relation (ε 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) $w = a_1 a_2 \ldots a_n$ of symbols in Σ 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 1. A QDD A for a protocol P is a deterministic finite-state automaton (M, S, Δ, 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 1, 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 1 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 [19]). 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 Δ" to mean that transitions (s_{i-1}, a_i, s_i), $1 \leq i \leq n$, such that $w = a_1 a_2 \ldots a_n$, $s_0 = s$, $s_n = s'$, and $s_i, 1 \leq i < n$, are new (fresh) states, are added to Δ.

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

SEND(queue_id i, word w, QDD (M, S, Δ, s_0, F)) {

For all states $s \in S$ such that

$\exists w' \in (\cup_{j=1}^{i} M_j)^* : s_0 \stackrel{w'}{\Rightarrow} s$,

do the following operations:

- Add a new state s' to S;
- For all transitions $t = (s, m, s'') \in \Delta$ such that $m \in M_j, j > i$:
  Replace t by (s', m, s'');
- For all transitions $t = (s'', m, s) \in \Delta$ such that $m \in M_j, j > i$:
  Replace t by (s'', m, s');
- Add (s, w, s') to Δ;
- If $s \in F$, add s' to F, and remove s from F;

Return DETERMINIZE((M, S, Δ, s_0, F)).

}

RECEIVE(queue_id i, word w, QDD (M, S, Δ, s_0, F)) {

For all states $s \in S$ such that

$\exists w' \in (\cup_{j=1}^{i-1} M_j)^* : s_0 \stackrel{w'}{\Rightarrow} s$,

do the following operations:

- Add a new state s' to S;
- For all transitions $t = (s, m, s'') \in \Delta$ such that $m \in M_j, j \geq i$:
  Replace t by (s', m, s'');
- For all transitions $t = (s'', m, s) \in \Delta$ such that $m \in M_j, j \geq i$:
  Replace t by (s'', m, s');
- For all states $s'' \in S$ such that $s' \stackrel{w}{\Rightarrow} s''$:
  Add a transition (s, ε, s'') to Δ;
- If $s \in F$, add s' to F, and remove s from F;

Return DETERMINIZE((M, S, Δ, s_0, F)).

}

Figure 2. $q_i!w$ and $q_i?w$

associated with the control state c' reached after the execution of a transition (c, op, c') from the control state c, with $op \in \{q_i!w, q_i?w\}$. We have the following:

- $L_{q_i!w}(A) = \{w'' | \exists w' \in L(A) : w''|_{M_i} = w'|_{M_i} w \wedge \forall j \neq i : w''|_{M_j} = w'|_{M_j}\}$,
- $L_{q_i?w}(A) = \{w'' | \exists w' \in L(A) : w'|_{M_i} = ww''|_{M_i} \wedge \forall j \neq i : w''|_{M_j} = w'|_{M_j}\}$.

Algorithms for computing a QDD A' that accepts all possible queue-contents obtained after the execution of a transition of the form $q_i!w$ or $q_i?w$ on a QDD

$A = (M, S, \Delta, s_0, F)$ are given in Figure 2. The correctness of these algorithms is established by the following two theorems.

THEOREM 1 *Let A be a QDD, let A' denote the automaton returned by 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: See Appendix. ■

THEOREM 2 *Let A be a QDD, let A' denote the automaton returned by RECEIVE(i, w, A), and let $L(A')$ denote the language accepted by A'. Then A' is a QDD such that $L(A') = L_{q_i?w}(A)$.*

Proof: See Appendix. ■

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 $w \in M_i^*$ to the queue q_i of the system, for all $k \geq 0$. The operation $(q_i?w)^*$ denotes the union of all possible queue-contents obtained after receiving k sequences of messages $w \in M_i^*$ from the queue q_i of the system, for all $k \geq 0$. 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 $w_1 \in M_i^*$ from the queue q_i and sending k sequences of messages $w_2 \in M_j^*$ to the queue q_j, for all $k \geq 0$, and for $i \neq j$.

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 $op \in \{(q_i!w)^*, (q_i?w)^*, (q_i?w_1; q_j!w_2)^*\}$. We have the following:

- $L_{(q_i!w)^*}(A) = \{w'' | \exists w' \in L(A), k \geq 0 : w''|_{M_i} = w'|_{M_i} w^k \wedge \forall j \neq i : w''|_{M_j} = w'|_{M_j}\}$,
- $L_{(q_i?w)^*}(A) = \{w'' | \exists w' \in L(A), k \geq 0 : w'|_{M_i} = w^k w''|_{M_i} \wedge \forall j \neq i : w''|_{M_j} = w'|_{M_j}\}$,
- $L_{(q_i?w_1;q_j!w_2)^*}(A) = \{w'' | \exists w' \in L(A), k \geq 0 : w'|_{M_i} = w_1^k w''|_{M_i} \wedge w''|_{M_j} = w'|_{M_j} w_2^k \wedge \forall l \notin \{i, j\} : w''|_{M_l} = w'|_{M_l}\}$.

Algorithms for computing a QDD A' that accepts all possible queue-contents obtained after the execution of a meta-transition of the form $(q_i!w)^*$, $(q_i?w)^*$, 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 3 *Let A be a QDD, let A' denote the automaton returned by SEND-STAR(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 Appendix. ■

SEND-STAR(queue_id i, word w, QDD (M, S, Δ, s_0, F)) {

For all states $s \in S$ such that

$\exists w' \in (\cup_{j=1}^{i} M_j)^* : s_0 \overset{w'}{\Rightarrow} s$,

do the following operations:

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

Return DETERMINIZE((M, S, Δ, s_0, F)).

}

RECEIVE-STAR(queue_id i, word w, QDD (M, S, Δ, s_0, F)) {

For all states $s \in S$ such that

$\exists w' \in (\cup_{j=1}^{i-1} M_j)^* : s_0 \overset{w'}{\Rightarrow} s$,

do the following operations:

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

Return DETERMINIZE((M, S, Δ, s_0, F)).

}

Figure 3. $(q_i!w)^*$ and $(q_i?w)^*$

RECEIVE-SEND-STAR(queue_id i, word w_1, queue_id j, word w_2, QDD (M, S, Δ, s_0, F)) {

Let n be the greatest integer such that

$\exists s_1, \ldots s_{n+1} \in S : s_1 \overset{w_1}{\Rightarrow} s_2 \overset{w_1}{\Rightarrow} \cdots \overset{w_1}{\Rightarrow} s_{n+1}$,

with $\forall 1 \leq k < l \leq n+1 : s_k \neq s_l$;

Let A_0 denote the QDD (M, S, Δ, s_0, F);

For all k, $1 \leq k \leq n+1$, compute $A_k =$ SEND(j, w_2, RECEIVE(i, w_1, A_{k-1}));

If $L(A_{n+1}) = \emptyset$:

- Return DETERMINIZE($\cup_{k=0}^{n} A_k$);

If $L(A_{n+1}) \neq \emptyset$:

- Let $p = 1$;
- While $L(A_{n+1}) \neq L$(RECEIVE(i, w_1^p, A_{n+1})):
  $p := p + 1$;
- For all k, $2 \leq k \leq p$, compute $A_{n+k} =$ SEND(j, w_2, RECEIVE(i, w_1, A_{n+k-1}));
- Compute $A_{n+p+1} =$SEND-STAR(j, w_2^p, DETERMINIZE($\cup_{k=n+1}^{n+p} A_k$));
- Return DETERMINIZE($\cup_{k=0}^{n} A_k \cup A_{n+p+1}$).

}

Figure 4. $(q_i?w_1; q_j!w_2)^*$

THEOREM 4 *Let A be a QDD, let A' denote the automaton returned by RECEIVE-STAR(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 Appendix. ■

LEMMA 1 *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 $L(A_{n+1}) = L(RECEIVE(i, w_1^p, A_{n+1}))$.*

Proof: See Appendix. ■

THEOREM 5 *Let A be a QDD, let A' denote the automaton returned by RECEIVE-SEND-STAR(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 Appendix. ■

It is worth noticing that, as a corollary of the last theorem, we have that 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 . Meta-transitions are added to the protocol description for loops that match either $(q_i!w)^*$, $(q_i?w)^*$, or $(q_i?w_1; q_j!w_2)^*$. Precisely, the meta-transitions $(3, (RtoS?ack1; StoR!msg0)^*, 3)$, $(3, (StoR!msg0)^*, 3)$, $(8, (RtoS?ack0; StoR!msg1)^*, 8)$ and $(8, (StoR!msg1)^*, 8)$ are added to the set of transitions of *Sender*, and the meta-transitions $(1, (StoR?msg1; RtoS!ack1)^*, 1)$ and $(5, (StoR?msg0; RtoS!ack0)^*, 5)$ are added to the set of transitions of *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 $(M, \{s_0\}, \{\}, s_0, \{s_0\})$, which corresponds to the queue-content ε 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 SPARC10 workstation. The number of (meta-)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(i, w, A). We also performed experiments on several simple sliding-window protocols [28], 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.

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 [1, 2, 12, 11, 14, 16, 17, 18, 25, 27]. 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 [22], 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 [11], 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. In [6], the class of meta-transitions that preserve recognizability is precisely characterized, and a generic algorithm for computing the effect of the execution of any meta-transition in this class is given. Moreover, it is shown how the range of properties that can be checked with loop-first searches and QDDs can be broadened from simple state reachability to temporal properties. A sufficient criterion for the termination of a loop-first search using QDDs is also presented.

In [23], 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 [3] for the verification of real-time systems, where dense-time domains are represented by polyhedra. This idea also appeared in [7], where the values of integer variables are represented by periodic vector sets. These symbolic representations are quite different from QDDs.

For digital hardware verification [10], the most commonly used symbolic representation is certainly the Binary Decision Diagram (BDD) [9], which represents a boolean function (with a *finite* domain) as a directed acyclic graph. In [15], 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.

Acknowledgments

We wish to thank Michael Merritt, Mark Staskauskas and the anonymous referees for helpful comments on this paper. A preliminary version of this paper appeared in the proceedings of the 8th Conference on Computer Aided Verification [5].

References

1. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings of the 8th IEEE Symposium on Logic in Computer Science*, 1993.
2. P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. In *Proc. ICALP-94*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327. Springer-Verlag, 1994.
3. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.
4. K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmissions over half-duplex lines. *Communications of the ACM*, 2(5):260–261, 1969.
5. B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. In *Proc. 8th Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12, New Brunswick, August 1996. Springer-Verlag.
6. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The Power of QDDs. In *Proceedings of the Fourth International Static Analysis Symposium (SAS'97)*, Lecture Notes in Computer Science, Paris, September 1997. Springer-Verlag.

7. B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67, Stanford, June 1994. Springer-Verlag.
8. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, 1983.
9. R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
10. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, June 1990.
11. G. Cécé, A. Finkel, and S. Purushothaman. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(3):20–31, 1996.
12. A. Choquet and A. Finkel. Simulation of linear FIFO nets having a structured set of terminal markings. In *Proc. 8th European Workshop on Application and Theory of Petri Nets*, pages 95–112, Saragoza, 1987.
13. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.
14. A. Finkel. A new class of analyzable cfsms with unbounded FIFO channels. In *Proc. 8th IFIP WG 6.1 International Symposium on Protocol Specification, Testing, and Verification*, pages 1–12, Atlantic City, 1988. North-Holland.
15. P. Godefroid and D. E. Long. Symbolic Protocol Verification with Queue BDDs. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*, pages 198–206, New Brunswick, July 1996.
16. M. G. Gouda, E. M. Gurari, T. H. Lai, and L. E. Rosier. On deadlock detection in systems of communicating finite-state machines. *Computers and Artificial Intelligence*, 6(3):209–228, 1987.
17. T. Jeron. Testing for unboundedness of FIFO channels. In *Proc. STACS-91: Symposium on Theoretical Aspects of Computer Science*, volume 480 of *Lecture Notes in Computer Science*, pages 322–333, Hamburg, 1991. Springer-Verlag.
18. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.
19. H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, 1981.
20. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.
21. M.T. Liu. Protocol engineering. *Advances in Computing*, 29:79–195, 1989.
22. J. K. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proc. 7th IFIP WG 6.1 International Symposium on Protocol Specification, Testing, and Verification.* North-Holland, 1987.
23. W. Peng and S. Purushothaman. Data flow analysis of communicating finite state machines. *ACM Transactions on Programming Languages and Systems*, 13(3):399–442, 1991.
24. J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Int'l Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.
25. L. E. Royer and H. C. Yen. Boundedness, empty channel detection and synchronization for communicating finite automata. *Theoretical Computer Science*, 44:69–105, 1986.
26. H. Rudin. Network protocols and tools to help produce them. *Annual Review of Computer Science*, 2:291–316, 1987.
27. A. P. Sistla and L. D. Zuck. Automatic temporal verification of buffer systems. In *Proc. 3rd Workshop on Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 93–103, Aalborg, July 1991. Springer-Verlag.
28. A. Tanenbaum. *Computer Neworks.* Prentice Hall, 1989.
29. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

Appendix

Correctness Proofs

THEOREM 1 *Let A be a QDD, let A' denote the automaton returned by 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:

- Let us prove first that $L_{q_i!w}(A) \subseteq L(A')$. Let u be a word in $L(A)$. Hence, there exists a path $\pi = s_0 \xrightarrow{m_0} s_1 \xrightarrow{m_1} \cdots \xrightarrow{m_{n-1}} s_n$ in A accepting u. Since $u = u|_{M_1} \cdots u|_{M_N}$, π contains exactly one state s_l such that $\forall k < l : m_k \in \cup_{j=1}^{i} M_j$ and $\forall k \geq l : m_k \in \cup_{j=i+1}^{N} M_j$. Therefore, s_l is a state "s" that satisfies the condition in line 3 of the algorithm, and the algorithm replaces the transition (s_l, m_l, s_{l+1}) (if any) by (s', m_l, s_{l+1}), where the state s' is a new state added by the algorithm. Moreover, the algorithm also adds (s_l, w, s') to the set of transitions of A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), before being determinized, the resulting automaton contains the path $\pi' = s_0 \xrightarrow{m_0} \cdots \xrightarrow{m_{l-1}} s_l \xrightarrow{w} s' \xrightarrow{m_l} s'_{l+1} \xrightarrow{m_{l+1}} \cdots \xrightarrow{m_{n-1}} s'_n$. In the case where m_{l-1} is the last transition of π, s_l is accepting in A, and is replaced by s' in the set of accepting states of A'. In any case, the path π' is accepting, and the automaton A' accepts $u|_{M_1} \cdots u|_{M_i} w u|_{M_{i+1}} \cdots u|_{M_N}$.

- Now, we show that $L(A') \subseteq L_{q_i!w}(A)$. Let u be a word in $L(A')$, and let A'' be the automaton obtained before the determinization operation giving A'. Since the automata A' and A'' accept the same language, $u \in L(A'')$ and there exists a path π of A'' accepting u. Let us show that π is of the form $\pi_1(s, w, s')\pi_2$, where π_1 is composed only of transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$ while π_2 is composed only of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$.

  Since π_1 leads from s_0 to s and is composed of transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$, state s satisfies the condition on line 3 of the algorithm, and hence cannot be accepting in A''. Moreover, there exists a transition (s, w, s') in A'' added by the algorithm (at line 10). Since the algorithm does not perform any modification on transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$, all the transitions of π_1 are transitions in A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), there exists in A a path π'_2 from s corresponding to the same sequence of transitions as in π_2. Since the last state of π_2 is accepting in A'', the last state of π'_2 is accepting in A. Thus, the word v such that $\forall k \neq i : v|_{M_k} = u|_{M_k}$ and $v|_{M_i} w = u|_{M_i}$, is accepted by A.

■

THEOREM 2 *Let A be a QDD, let A' denote the automaton returned by RECEIVE(i, w, A), and let $L(A')$ denote the language accepted by A'. Then A' is a QDD such that $L(A') = L_{q_i?w}(A)$.*

Proof:

- Let us prove first that $L_{q_i?w}(A) \subseteq L(A')$. Let $u = u_1 w u_2$ be a word in $L(A)$, with u_1 (resp. u_2) only composed of symbols in $\cup_{j=1}^{i-1} M_j$ (resp. $\cup_{j=i}^{N} M_j$). There exists a path $\pi = s_0 \stackrel{m_0}{\rightarrow} s_1 \stackrel{m_1}{\rightarrow} \cdots \stackrel{m_{n-1}}{\rightarrow} s_n$ in A accepting u. Since $u = u|_{M_1} \cdots u|_{M_N}$, π contains exactly one state s_l such that $\forall k < l : m_k \in \cup_{j=1}^{i-1} M_j$ and $\forall k \geq l : m_k \in \cup_{j=i}^{N} M_j$. Therefore, s_l is a state "s" that satisfies the condition in line 3 of the algorithm, and the algorithm replaces the transition (s_l, m_l, s_{l+1}) by (s', m_l, s_{l+1}), where the state s' is a new state added by the algorithm. Moreover, it follows from the definition of u that π contains exactly one state s_p such that $s_l \stackrel{w}{\Rightarrow} s_p$. The state s_p satisfies the condition at line 10 of the algorithm, hence the algorithm adds (s, ε, s_p) to the set of transitions of A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), before being determinized, the resulting automaton contains the path $\pi' = s_0 \stackrel{m_0}{\rightarrow} \cdots \stackrel{m_{l-1}}{\rightarrow} s_l \stackrel{\varepsilon}{\rightarrow} s_p \stackrel{m_p}{\rightarrow} s'_{p+1} \cdots \stackrel{m_{n-1}}{\rightarrow} s'_n$. Since the path π' is accepting, the automaton A' accepts the word $u_1 u_2$.

- Now, we show that $L(A') \subseteq L_{q_i?w}(A)$. Let u be a word in $L(A')$, and let A'' be the automaton obtained before the determinization operation giving A'. Since the automata A' and A'' accept the same language, $u \in L(A'')$ and there exists a path π of A'' accepting u. Let us show that π is of the form $\pi_1 \pi_2$, where π_1 is composed only of transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$ while π_2 is composed only of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$.

  Since π_1 leads from s_0 to s and is composed of transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$, state s satisfies the condition on line 3 of the algorithm, and hence cannot be accepting in A''. Moreover, the only outgoing transition from s not labeled by a symbol in $\cup_{j=1}^{i-1} M_j$ can only be a transition (s, ε, s'') added by the algorithm (at line 11), with $s \stackrel{w}{\Rightarrow} s''$ in A. Since the algorithm does not perform any modification on transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$, all the transitions of π_1 are transitions in A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), there exists in A a path π'_2 from s'' corresponding to the same sequence of transitions as in π_2. Since the last state of π_2 is accepting in A'', the last state of π'_2 is accepting in A. Thus, the word v such that $\forall k \neq i : v|_{M_k} = u|_{M_k}$ and $v|_{M_i} = wu|_{M_i}$, is accepted by A.

■

THEOREM 3 *Let A be a QDD, let A' denote the automaton returned by SEND-STAR(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:

- Let us prove first that $L_{(q_i!w)^*}(A) \subseteq L(A')$. Let u be a word in $L(A)$. Hence, there exists a path $\pi = s_0 \stackrel{m_0}{\rightarrow} s_1 \stackrel{m_1}{\rightarrow} \cdots \stackrel{m_{n-1}}{\rightarrow} s_n$ in A accepting u. Since $u = u|_{M_1} \cdots u|_{M_N}$, π contains exactly one state s_l such that $\forall k < l : m_k \in \cup_{j=1}^{i} M_j$ and $\forall k \geq l : m_k \in \cup_{j=i+1}^{N} M_j$. Therefore, s_l is a state "s" that satisfies the condition in line 3 of the algorithm, and the algorithm replaces the transition (s_l, m_l, s_{l+1}) (if any) by by (s'', m_l, s_{l+1}), where the state s'' is a new state added by the algorithm. Moreover, the algorithm also creates another new state s' and adds the transitions (s_l, ε, s'), (s', ε, s'') and (s', w, s') to the set of transitions of A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are preserved in A' (a new state s'' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are mapped to transitions of same label to (resp. from) s''), before being determinized, the resulting automaton contains the path $\pi' = s_0 \stackrel{m_0}{\rightarrow} \cdots \stackrel{m_{l-1}}{\rightarrow} s_l \stackrel{\varepsilon}{\rightarrow} s' \stackrel{\varepsilon}{\rightarrow} s'' \stackrel{m_l}{\rightarrow} s'_{l+1} \stackrel{m_{l+1}}{\rightarrow} \cdots \stackrel{m_{n-1}}{\rightarrow} s'_n$. In the case where m_{l-1} is the last transition of π, s_l is accepting in A, and is replaced by s'' in the set of accepting states of A'. In any case, the path π' is accepting. Let $k \geq 0$ be an arbitrary integer. Since the set of transitions of A' contains the transition (s', w, s'), the path $\pi'_k = s_0 \stackrel{m_0}{\rightarrow} \cdots \stackrel{m_{l-1}}{\rightarrow} s_l \stackrel{\varepsilon}{\rightarrow} s' \stackrel{w^k}{\rightarrow} s' \stackrel{\varepsilon}{\rightarrow} s'' \stackrel{m_l}{\rightarrow} s'_{l+1} \stackrel{m_{l+1}}{\rightarrow} \cdots \stackrel{m_{n-1}}{\rightarrow} s'_n$ is also an accepting path of A'. It follows that for any $k \geq 0$, the automaton A' accepts $u|_{M_1} \cdots u|_{M_i} w^k u|_{M_{i+1}} \cdots u|_{M_N}$.

- Now, we show that $L(A') \subseteq L_{(q_i!w)^*}(A)$. Let u be a word in $L(A')$, and let A'' be the automaton obtained before the determinization operation giving A'. Since the automata A' and A'' accept the same language, $u \in L(A'')$ and there exists a path π of A'' accepting u. Let us show that π is of the form π_1 or $\pi_1(s, \varepsilon, s')(s', w, s')^k(s', \varepsilon, s'')\pi_2$, where π_1 is composed only of transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$, π_2 is composed only of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$, and $k \geq 0$.

  Since π_1 leads from s_0 to s and is composed of transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$, state s satisfies the condition on line 3 of the algorithm. Moreover, there exist transitions (s, ε, s'), (s', w, s') and (s', ε, s'') added by the algorithm (at line 10). Since the algorithm does not perform any modification on transitions labeled by a symbol in $\cup_{j=1}^{i} M_j$, all the transitions of π_1 are transitions in A. Moreover, if π_1 is an accepting path in A'', then it is also an accepting path in A. Hence, if $\pi = \pi_1$, then $u \in L(A)$. Let us assume now that $\pi \neq \pi_1$. Since all sequences of transitions labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are preserved in A' (a new state s'' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i+1}^{N} M_j$ are mapped to transitions of same label to (resp. from) s''), there exists in A a path π_2' from s corresponding to the same sequence of transitions as in π_2. Since the last state of π_2 is accepting in A'', the last state of π_2' is accepting in A. Thus, the word v such that $\forall j \neq i : v|_{M_j} = u|_{M_j}$ and $v|_{M_i} w^k = u|_{M_i}$, is accepted by A.

■

THEOREM 4 *Let A be a QDD, let A' denote the automaton returned by RECEIVE-STAR(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:

- Let us prove first that $L_{(q_i?w)^*}(A) \subseteq L(A')$. Let $u = u_1 w^k u_2$ be a word in $L(A)$, with u_1 (resp. u_2) only composed of symbols in $\cup_{j=1}^{i-1} M_j$ (resp. $\cup_{j=i}^{N} M_j$), and $k \geq 0$. There exists a path $\pi = s_0 \xrightarrow{m_0} s_1 \xrightarrow{m_1} \cdots \xrightarrow{m_{n-1}} s_n$ in A accepting u. Since $u = u|_{M_1} \cdots u|_{M_N}$, π contains exactly one state s_l such that $\forall k < l : m_k \in \cup_{j=1}^{i-1} M_j$ and $\forall k \geq l : m_k \in \cup_{j=i}^{N} M_j$. Therefore, s_l is a state "s" that satisfies the condition in line 3 of the algorithm, and the algorithm replaces the transition (s_l, m_l, s_{l+1}) by (s', m_l, s_{l+1}), where the state s' is a new state added by the algorithm. Moreover, it follows from the definition of u that π contains exactly one state s_p such that $s_l \stackrel{w^k}{\Rightarrow} s_p$. The state s_p satisfies the condition at line 10 of the algorithm, hence the algorithm adds (s, ε, s_p) to the set of transitions of A. Since all sequences of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), before being determinized, the resulting automaton contains the path $\pi' = s_0 \xrightarrow{m_0} \cdots \xrightarrow{m_{l-1}} s_l \xrightarrow{\varepsilon} s_p \xrightarrow{m_p} s'_{p+1} \cdots \xrightarrow{m_{n-1}} s'_n$. Since the path π' is accepting, the automaton A' accepts the word $u_1 u_2$.

- Now, we show that $L(A') \subseteq L_{q_i?w}(A)$. Let u be a word in $L(A')$, and let A'' be the automaton obtained before the determinization operation giving A'. Since the automata A' and A'' accept the same language, $u \in L(A'')$ and there exists a path π of A'' accepting u. Let us show that π is of the form π_1 or $\pi_1\pi_2$, where π_1 is composed only of transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$ while π_2 is composed only of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$.

Since π_1 leads from s_0 to s and is composed of transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$, state s satisfies the condition on line 3 of the algorithm. Moreover, the only outgoing transition from s not labeled by a symbol in $\cup_{j=1}^{i-1} M_j$ can only be a transition (s, ε, s'') added by the algorithm (at line 11), with $s \stackrel{w^k}{\Rightarrow} s''$ in A, and $k \geq 0$. Since the algorithm does not perform any modification on transitions labeled by a symbol in $\cup_{j=1}^{i-1} M_j$, all the transitions of π_1 are transitions in A. Moreover, if π_1 is an accepting path in A'', then it is also an accepting path in A. Hence, if $\pi = \pi_1$, then $u \in L(A)$. Let us assume now that $\pi \neq \pi_1$. Since all sequences of transitions labeled by a symbol in $\cup_{j=i}^{N} M_j$ are preserved in A' (a new state s' is associated to every intermediate state s that satisfies the condition in line 3, and all the incoming (outgoing) transitions to (resp. from) s labeled by a symbol in $\cup_{j=i}^{N} M_j$ are mapped to transitions of same label to (resp. from) s'), there exists in A a path π_2' from s'' corresponding to the same sequence of transitions as in π_2. Since the last state of π_2 is accepting in A'', the last state of π_2' is accepting in A. Thus, the word v such that $\forall j \neq i : v|_{M_j} = u|_{M_j}$ and $v|_{M_i} = w^k u|_{M_i}$, is accepted by A.

■

Lemma 1 *Let n and A_{n+1} be as defined by the algorithm computing the value of RECEIVE-SEND-STAR(i, w_1, j, w_2, A). If the language accepted by A_{n+1} is not empty, then there exists $p > 0$ such that A_{n+1} and RECEIVE(i, w_1^p, A_{n+1}) accept the same language.*

Proof:

- First, we prove that $L(A_{n+1}) \subseteq L(\text{RECEIVE}(i, w_1^p, A_{n+1}))$ for some $p > 0$. Let w be a word in $L(A_{n+1})$. For any such word, there exists $w' \in L(A)$ such that $w'|_{M_i} = w_1^{n+1}(w|_{M_i})$, $w|_{M_j} = (w'|_{M_j})w_2^{n+1}$, and $\forall k \notin \{i,j\} : w|_{M_k} = w'|_{M_k}$. Let $s_0, s_1, \ldots, s_x$ be the path of A accepting w'. It contains a subpath $s_y, s_{y+1}, \ldots, s_{y+n+1}$ such that $s_y \stackrel{w_1}{\Rightarrow} s_{y+1} \stackrel{w_1}{\Rightarrow} \cdots \stackrel{w_1}{\Rightarrow} s_{y+n+1}$ and $s_0 \stackrel{w''}{\Rightarrow} s_y$, with $w'' = w'|_{M_1} \cdots w'|_{M_{i-1}}$. By definition of n, this subpath contains a cycle of w_1, i.e., there exists z and l such that $y \leq z \leq y+n+1$, $1 \leq l \leq n+1$, and $s_z \stackrel{w_1^l}{\Rightarrow} s_z$. We thus have $\forall k \geq 0 : w'|_{M_1} \cdots w'|_{M_{i-1}} w_1^{kl} w'|_{M_i} \cdots w'|_{M_N} \in L(A)$. Choosing $k = (n+1)!/l$, we obtain $w'|_{M_1} \cdots w'|_{M_{i-1}} w_1^{(n+1)!} w'|_{M_i} \cdots w'|_{M_N} \in L(A)$, which implies $w|_{M_1} \cdots w|_{M_{i-1}} w_1^{(n+1)!} w|_{M_i} \cdots w|_{M_N} \in L(A_{n+1})$. By taking $p = (n+1)!$ and applying this result to all the words w in $L(A_{n+1})$, it follows that $L(A_{n+1}) \subseteq L(\text{RECEIVE}(i, w_1^p, A_{n+1}))$.

- Now, we show that $L(\text{RECEIVE}(i, w_1^p, A_{n+1})) \subseteq L(A_{n+1})$ for the same $p = (n+1)!$. Let $w \in L(\text{RECEIVE}(i, w_1^p, A_{n+1}))$. There exists $w' \in L(A)$ such that $w'|_{M_i} = w_1^{p+n+1} w|_{M_i}$, $w|_{M_j} = w'|_{M_j} w_2^{n+1}$, and $\forall k \notin \{i,j\} : w|_{M_k} = w'|_{M_k}$. Let $s_0, s_1, \ldots, s_x$ be the path of A accepting w'. By definition of n, this path contains a subpath $s_y, s_{y+1}, \ldots, s_z$, with $z \geq y + p + n + 1$, such that $s_0 \stackrel{w''}{\Rightarrow} s_y$, with $w'' = w'|_{M_1} \cdots w'|_{M_{i-1}}$. This subpath contains k occurrences of a cycle accepting w_1^l, where k and l are such that $1 \leq l \leq n+1$ and $k.l \geq p$. By removing exactly p/l occurrences of this cycle from the path $s_0, s_1, \ldots, s_x$, we obtain a path of A accepting the word w''' such that $w'''|_{M_i} = w_1^{n+1} w|_{M_i}$, $w|_{M_j} = w'''|_{M_j} w_2^{n+1}$, and $\forall k \notin \{i,j\} : w|_{M_k} = w'''|_{M_k}$. Therefore, $w \in L(A_{n+1})$.

■

THEOREM 5 *If A is a QDD and A' is returned by RECEIVE-SEND-STAR(i, w_1, j, w_2, A), with $i \neq j$, then A' is a QDD such that $L(A') = (q_i?w_1; q_j!w_2)^*(L(A))$.*

Proof: Let n, p, $\{A_k\}$ be as defined by the algorithm computing RECEIVE-SEND-STAR. (The existence of p is proved by Lemma 1.) We have:

$$\begin{aligned}
&(q_i?w_1; q_j!w_2)^*(L(A)) \\
&\qquad = \bigcup_{k=0}^{\infty} (q_i?w_1; q_j!w_2)^k (L(A)) \\
&\qquad = \bigcup_{k=0}^{n} (q_i?w_1; q_j!w_2)^k (L(A)) \cup \bigcup_{k=n+1}^{\infty} (q_i?w_1; q_j!w_2)^k (L(A)) \\
&\qquad = \bigcup_{k=0}^{n} L(A_k) \cup \bigcup_{k=0}^{\infty} (q_i?w_1; q_j!w_2)^k (L(A_{n+1})).
\end{aligned}$$

- *If $L(A_{n+1}) = \emptyset$:* We have $(q_i?w_1; q_j!w_2)^*(L(A)) = \bigcup_{k=0}^{n} L(A_k) = L(A')$.

- *If $L(A_{n+1}) \neq \emptyset$:* Since $i \neq j$, the operations $q_i?m_1$ and $q_j!m_2$ commute. Hence, we have:

$$\begin{aligned}
&\bigcup_{k=0}^{\infty} (q_i?w_1; q_j!w_2)^k (L(A_{n+1})) \\
&\qquad = \bigcup_{i=0}^{\infty} \bigcup_{j=0}^{p-1} (q_i?w_1; q_j!w_2)^{pi+j} (L(A_{n+1})) \\
&\qquad = \bigcup_{i=0}^{\infty} \bigcup_{j=0}^{p-1} (q_j!w_2)^{pi} (q_i?w_1; q_j!w_2)^j (q_i?w_1)^{pi} (L(A_{n+1})).
\end{aligned}$$

By definition of p, $(q_i?w_1)^{pi}(L(A_{n+1})) = L(A_{n+1})$. Thus, we have:

$$\begin{aligned}
&\bigcup_{k=0}^{\infty} (q_i?w_1; q_j!w_2)^k (L(A_{n+1})) \\
&\qquad = \bigcup_{i=0}^{\infty} \bigcup_{j=0}^{p-1} (q_j!w_2)^{pi} (q_i?w_1; q_j!w_2)^j (L(A_{n+1}))
\end{aligned}$$

$$
\begin{aligned}
&= \bigcup_{i=0}^{\infty}(q_j!w_2)^{pi}\left(\bigcup_{j=0}^{p-1}(q_i?w_1;q_j!w_2)^j(L(A_{n+1}))\right) \\
&= (q_j!w_2^p)^*\left(\bigcup_{j=0}^{p-1}L(A_{n+j+1})\right) \\
&= L(A_{n+p+1}).
\end{aligned}
$$

Therefore, $(q_i?w_1;q_j!w_2)^*(L(A)) = \bigcup_{k=0}^{n} L(A_k) \cup L(A_{n+p+1})$.

■