The Power of QDDs
(DRAFT)

Bernard Boigelot[1*], Patrice Godefroid[2], Bernard Willems[1] and Pierre Wolper[1]

[1] Université de Liège
Institut Montefiore, B28
B-4000 Liège Sart-Tilman, Belgium
{boigelot,willems,pw}@montefiore.ulg.ac.be

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

Abstract. Queue-content Decision Diagrams (QDDs) are finite-automaton based data structures for representing (possibly infinite) sets of contents of a finite collection of unbounded FIFO queues. Their intended use is to serve as a symbolic representation of the possible queue contents that can occur in the state space of a protocol modeled by finite-state machines communicating through unbounded queues. This is done with the help of a loop-first search, a state-space exploration technique that attempts whenever possible to compute symbolically the effect of repeatedly executing a loop any number of times, making it possible to analyze protocols with infinite state spaces though without the guarantee of termination. This paper first solves a key problem concerning the use of QDDs in this context: it precisely characterizes when, and shows how, the operations required by a loop-first search can be applied to QDDs. Then, it addresses the problem of exploiting QDDs and loop-first searches to broaden the range of properties that can be checked from simple state reachability to temporal logic. Finally, a sufficient criterion for the termination of a loop-first search using QDDs is given.

Contact Author: Bernard Boigelot
Phone: +32-4-3662970
Fax: +32-4-3662984
Email: boigelot@montefiore.ulg.ac.be

* "Aspirant" (Research Assistant) for the National Fund for Scientific Research (Belgium). The work of this author was done partly while visiting Bell Laboratories.

1 Introduction

Finite-state machines that communicate by exchanging messages via *unbounded* FIFO queues are a popular model for representing and reasoning about communication protocols. This model is also used to define the semantics of standardized protocol specification languages such as SDL and Estelle (e.g., see [Tur93]). Indeed, unbounded queues provide a useful abstraction that simplifies the semantics of specification languages, and frees the protocol designer from implementation details related to buffering policies and limitations. In contrast, while unboundedness can simplify the modeling of protocols, it seems to complicate their verification. Indeed, it is well known that most interesting verification problems, such as deadlock detection, are undecidable for this class of systems [BZ83], since one unbounded queue is sufficient to simulate the tape of a Turing machine.

Recently [BG96], it has been argued that this contradiction might not be inherent: in practice, most verification problems may very well turn out to be decidable for a subclass containing most "real" protocols. After all, protocols are not designed randomly, precise design principles are used to enforce some "regularity" in their temporal behavior. Moreover, by using appropriate verification techniques for dealing with infinite state spaces, one might actually be able to verify properties of such systems more *efficiently* than verifying properties of systems with finite but very large state spaces.

This observation motivated the development in [BG96] of a new verification framework for communication protocols modeled by finite-state machines communicating via unbounded queues. Specifically, an algorithm is presented for constructing a *finite* and *exact* representation of the state space of such a set of communicating finite-state machines, even if this state space is infinite. The 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 is done by using *meta-transitions*: given a loop that appears in the protocol description, a meta-transition is a transition that generates all global states that can be reached after repeated executions of the body of the loop. A new data structure named *Queue-content Decision Diagram* (QDD) is introduced for representing (possibly infinite) sets of contents for a finite collection of queues. From this symbolic representation, it is then straightforward to verify 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, this algorithm may not terminate its search, due to the general undecidability result recalled above. However, in practice, properties of several simple communication protocols with infinite state spaces have been verified successfully with this method [BG96].

In this paper, we build upon this work, and extend these previous results in several ways. First, we very significantly extend the class of meta-transitions that can be handled in the context of QDDs. Indeed, in [BG96], algorithms are only given for three simple, but frequent, types of meta-transitions. Here we precisely characterize the class of meta-transitions that preserve recognizability and give a generic algorithm for computing the effect of any recognizability-preserving meta-transition. For systems with one queue, we show that the iteration of *any* sequence of queue operations preserves the recognizability of the queue content. For systems with more than one queue, we show that recognizability is preserved by the iteration of a sequence of operations if and only if this sequence of operations does not have more than one projection that is "counting" messages in a sense we make precise later.

Thereafter, we turn to the problem of moving from state reachability to the verification of more complex properties and in particular temporal properties. We show how by using the technique of [VW86], the model checking [CES86] of linear-time temporal logic formulas [MP92] can be done with the help of QDDs. Basically, we reduce the model-checking problem to the nonemptiness of Büchi automata [Büc62] with queues, for which QDDs can be used to obtain a partial solution. Finally, we give an algorithmic criterion that is sufficient for ensuring the termination of a loop-first search in an infinite state space.

2 Protocols, QDDs and Loop-First Searches

Our goal is to explore the state space of protocols 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. The direct technique for generating the state space of such a system would be to start in the initial state and explore all possible transitions. Of course, since the message queues are unbounded, this search will a priori not terminate. So, we proceed differently. We first eliminate concurrency from the system without computing the effect of the operations on queues, i.e., we compute the global *control* state space of the system. We thus obtain a finite-state machine with queues. This is the structure to which we then apply the techniques that are the subject of this paper.

Formally, we thus start with protocols described by tuples of the form $P = (C, c_0, A, Q, M, T)$ where

- C is a finite set of control states,
- $c_0 \in C$ is an *initial control state*,
- A is a finite set of *local actions* (those not involving the queues),
- Q is a finite set of queues,
- M is a finite set of *messages* partitioned in as many sets M_i as there are queues $q_i \in Q$ (assuming that each queue uses a set of different messages is convenient and not restrictive),
- $T \subseteq C \times Op \times C$ where $Op = A \cup \{q_i!m \mid q_i \in Q \text{ and } m \in M_i\} \cup \{q_i?m \mid q_i \in Q \text{ and } m \in M_i\}$ is a finite set of transitions. A transition of the form $c_1 \xrightarrow{\alpha} c_2$ $(\alpha \in A)$ represents a change of the control state from c_1 to c_2 without any change to the content of the queues; a transition of the form $c_1 \xrightarrow{q_i!m} c_2$ represents a change of the control state from c_1 to c_2 while appending the message m to the end of the queue q_i; and a transition of the form $c_1 \xrightarrow{q_i?m} c_2$

represents a change of the control state from c_1 to c_2 while removing the message m from the head of the queue q_i.

A global state of such 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 $P = (C, c_0, A, Q, M, T)$ with $|Q| = n$ is an element of the set $\mathcal{S} = C \times M_1^* \times \cdots \times M_n^*$, i.e. is of the form $\gamma = (c, w(1), w(2), \ldots, w(n))$ where, for $1 \leq j \leq n$, $w(j) \in M_j^*$. The *initial global state* of the system is $\gamma_0 = (c_0, \varepsilon, \ldots, \varepsilon)$, i.e., we assume that all queues are initially empty (ε represents the empty word).

We can then define the *global transition relation* of a protocol $P = (C, c_0, A, Q, M, T)$. It is the set $\mathcal{G}$ of triples (γ, a, γ'), where γ and γ' are global states and $a \in A \cup \{\tau\}$, defined as follows (we write $\gamma \xrightarrow{a}_G \gamma'$ to denote the fact that $(\gamma, a, \gamma') \in \mathcal{G}$):

- if $c_1 \xrightarrow{q_i!m} c_2 \in T$, then $(c_1, w(1), w(2), \ldots, w(i), \ldots, w(n)) \xrightarrow{\tau}_G (c_2, w(1), w(2), \ldots, w(i)m, \ldots, w(n))$ (the control state changes from c_1 to c_2 and m is appended to the content of queue q_i);
- if $c_1 \xrightarrow{q_i?m} c_2 \in T$, then $(c_1, w(1), w(2), \ldots, mw'(i), \ldots, w(n)) \xrightarrow{\tau}_G (c_2, w(1), w(2), \ldots, w'(i), \ldots, w(n))$ (the control state changes from c_1 to c_2 and m is removed from the head of queue q_i);
- if $c_1 \xrightarrow{a} c_2 \in T$, then $(c_1, w(1), w(2), \ldots, w(n)) \xrightarrow{a}_G (c_2, w(1), w(2), \ldots, w(n))$ (the control state changes from c_1 to c_2, the queue content is unchanged).

A global state γ' is said to be *reachable* from another global state γ if there exists a sequence of global transitions $\gamma_{i-1} \xrightarrow{a_i}_G \gamma_i$, $1 \leq i \leq k$, such that $\gamma = \gamma_0 \xrightarrow{a_1}_G \gamma_1 \cdots \gamma_{k-1} \xrightarrow{a_k}_G \gamma_k = \gamma'$. 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.

The approach we use to explore, in a finite amount of time, the infinite state space of a protocol with unbounded queues is based on the two following tools:

- A finite representation for potentially infinite sets of possible queue contents, and
- A technique for generating a potentially infinite set of reachable states in one step.

A solution to the first of these problems, the *queue-content decision diagram*, was introduced in [BG96], and a solution to the second one, the *loop-first search*, was introduced in [BW94, BG96]. Let us describe them.

For a protocol $P = (C, c_0, A, Q, M, T)$, a set of queue contents is a set of vectors $(w(1), \ldots, w(n))$ where, for $1 \leq i \leq n$, $w(i) \in M_i^*$. The idea of QDDs is to represent a queue content by the concatenation of the corresponding queue contents taken in an arbitrary but fixed order, and to represent a set of queue contents by a finite automaton accepting the concatenated form representations of its elements. Note that since we have assumed that different queues use distinct alphabet messages, concatenating queue contents leads to a nonambiguous representation of queue contents.

A finite-state automaton on finite words is a tuple $\mathcal{A} = (S, \Sigma, \Delta, S_0, F)$, where S is a finite set of states, Σ is an alphabet (finite set of symbols), $\Delta \subseteq S \times \Sigma \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, 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) $w = a_1 a_2 \dots a_k$ of symbols in Σ is *accepted* by the automaton $\mathcal{A}$ if there exists a sequence of states $\sigma = s_0 \dots s_k$ such that $\forall 1 \leq i \leq k$: $(s_{i-1}, a_i, s_i) \in \Delta$, $s_0 \in S_0$, and $s_k \in F$. The set of words accepted by $\mathcal{A}$ is called the *language accepted by* $\mathcal{A}$, and is denoted by $L(\mathcal{A})$. Let us define the *projection* $w|_{M_i}$ of a word w on a subset M_i of the alphabet on which w is defined as the subsequence of w obtained by removing all symbols in w that are not in M_i. We can now define QDDs[3].

Definition 1. Given a protocol $P = (C, c_0, A, Q, M, T)$ with $|Q| = n$ and an ordering $q_1, \dots, q_n$ of the elements of Q, a QDD for P is a finite-state automaton $\mathcal{A} = (S, M, \Delta, S_0, F)$ on finite words such that

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

It is clear that, the alphabets of queues being disjoint, a word w accepted by a QDD represents a unique queue content, namely the content of q_i is $w|_{M_i}$. A QDD $\mathcal{A}$ thus indeed represents a set of queue contents, those corresponding to the words in $L(\mathcal{A})$. It is natural to ask the following two questions. Does the set of queue contents representable by a QDD depend on the order chosen for the queues, and what is a characterization of the sets of queue contents representable by QDDs? We answer these questions in terms of recognizable languages of word vectors.

Definition 2. A subset of $\prod_{1 \leq i \leq n} M_i^*$, is *recognizable* if it is a finite union of Cartesian products of regular languages, i.e., if it is of the form

$$\bigcup_{1 \leq j \leq k} \prod_{1 \leq i \leq n} L_{ij},$$

where each L_{ij} is a regular subset of M_i^*.

The expressive power of QDDs is independent of the order chosen for concatenating the queue contents and coincides exactly with the recognizable languages.

Theorem 3. *Independently of the order chosen for concatenating queue contents, the languages of queue contents representable by QDDs coincide exactly with the recognizable languages.*

Proof sketch Let $\mathcal{A} = (S, M, \Delta, S_0, F)$ be a QDD. By definition, for every $w \in L(\mathcal{A})$, we have $w = w|_{M_1} w|_{M_2} \dots w|_{M_n}$. We define the automata $\mathcal{A}_i = (S, M_i, \Delta_i, S_i, F_i)$, $1 \leq i \leq n$, as follows:

[3] The definition of QDDs that appears in [BG96] has been extended here to non-deterministic automata with sets of initial states, for the sake of generality.

- $\Delta_i = \Delta \cap (S \times M_i \times S)$, $1 \leq i \leq n$,
- F_i is the set of states in S that are reachable from an initial state in S_0 by reading only symbols in $M_1 \cup M_2 \cup \cdots \cup M_i$,
- $S_i = S_0$ if $i = 1$, and $S_i = F_{i-1}$ if $i > 1$.

Given $n+1$ states $s_1 \in S_1, s_2 \in S_2, \ldots s_n \in S_n, s_{n+1} \in F_n \cap F$, the automaton $\mathcal{A}_{s_1,\ldots,s_{n+1}}$ is defined as the product $\mathcal{A}'_1 \times \ldots \times \mathcal{A}'_n$ where $\mathcal{A}'_i$ is a copy of the automaton $\mathcal{A}_i$ with only s_i as initial state and s_{i+1} as final state. The language accepted by $\mathcal{A}_{s_1,\ldots,s_{n+1}}$ is a product of regular subsets of all the $(M_i)^*$. Since $L(\mathcal{A})$ is the (finite) union of all the possible $L(\mathcal{A}_{s_1,\ldots,s_{n+1}})$, $L(\mathcal{A})$ is recognizable.

The other direction of the theorem is immediate since regular languages are closed under concatenation and finite union. □

To exploit QDDs for exploring the state space of a protocol, one groups together global states with the same control component and one represents the corresponding set of queue contents by a QDD. Of course, if the queue contents represented by the QDDs are extended by one element at a time, the QDDs will always only represent finite sets and thus will be of very limited usefulness. What is needed is a technique for generating a whole set of reachable states in one step. This is the purpose of the *meta-transitions* introduced in [BW94].

Given a control state c of a protocol and a loop (cycle) $\ell = c \stackrel{op_1}{\to} \cdots \stackrel{op_k}{\to} c$ in the control graph from c to c, a meta-transition for ℓ is a transition $(c \stackrel{op_1}{\to} \cdots \stackrel{op_k}{\to} c)^*$, which we denote by $c \stackrel{(op_1;\ldots;op_k)^*}{\longrightarrow} c$, that generates all the global states that can be reached after repeated executions of the sequence of transitions composing ℓ. For example, in a system with one queue, executing the meta-transition $c \stackrel{(q!m)^*}{\to} c$ from the state (c, ε) generates the set of states $\{(c, m^k) \mid k \geq 0\}$.

Using QDDs and meta-transitions added to the set T of transitions of the protocol, the classical enumerative state-space exploration algorithm can be rewritten in such a way that it works with sets of global states, i.e., pairs of the form (control state, QDD), 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 such a search is called 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, QDD) represents *exactly* the set of reachable states of the protocol being analyzed.

So, the problem we are left with is the following. Given a set of global states represented by a pair $(c, \mathcal{A})$, where c is a control state and $\mathcal{A}$ a QDD, and given a meta-transition, compute the QDD $\mathcal{A}'$ representing the set of queue contents that can be reached by executing this meta-transition from $(c, \mathcal{A})$. In [BG96], this problem was solved for three particular types of meta-transitions : repeatedly sending messages on a queue $((q_i!m_1;\ldots;q_i!m_k)^*)$, repeatedly receiving messages from a queue $((q_i?m_1;\ldots;q_i?m_k)^*)$, and repeatedly receiving a sequence of messages from a queue q_i followed by sending another sequence of messages on another queue q_j, $i \neq j$, $((q_i?m_1; \ldots; q_i?m_{k_i}; q_j!m'_1; \ldots; q_j!m'_{k_j})^*)$. In the next section, we extend this result by characterizing precisely the set of meta-transitions that preserve recognizability (and hence representability by QDDs) and by providing a generic algorithm for computing the effect of the execution of any meta-transition in this class.

3 Operations on QDDs

In this section, we establish results about computing the image of a QDD under the repeated execution of a sequence of queue operations. The following notations are used. If op_1, op_2, ..., op_p are queue operations ($q_i!m$ or $q_i?m$), then $\sigma = op_1; op_2; \ldots; op_p$ is a *sequence* of operations. The effect of a sequence of operations $\sigma = op_1; op_2; \ldots; op_p$ on a queue content w is $\sigma(w) = op_p(\cdots op_2(op_1(w))\cdots)$. The effect of a sequence of operations σ on a language L of queue contents is $\sigma(L) = \{\sigma(w) \mid w \in L\}$. The effect of the *Kleene closure* σ^* of such a sequence is defined as

$$\sigma^*(L) = \bigcup_{k \geq 0} \sigma^k(L).$$

If σ is a sequence of operations, then the number of queue operations composing σ is denoted $|\sigma|$. We denote by $\sigma_!$ (resp. $\sigma_?$) the subsequence of σ consisting of all the send (resp. receive) operations. Finally, we write $\mu(\sigma)$ to represent the word obtained from σ by extracting the message symbols from the queue operations, i.e. replacing each $q_i!m$ and $q_i?m$ by m.

3.1 Systems with one FIFO Queue

If the system being analyzed has only one FIFO queue q, then the QDDs used for its state-space exploration by a loop-first search are nothing but ordinary finite automata on finite words over the message alphabet of q. It then follows that the notions of recognizable and regular languages of queue contents are equivalent. Our first result states that in this particular case, the Kleene closure of every sequence of elementary queue operations preserves the regularity of languages of queue contents. Precisely, we have the following theorem.

Theorem 4. *Let q be a FIFO queue, M be the message alphabet of q, and σ be a sequence of queue operations on q. For every regular language of queue contents $L \subseteq M^*$, the languages $\sigma(L)$ and $\sigma^*(L)$ are regular.*

Proof sketch The proof is constructive and works with a QDD representation of L. The algorithm embodying the construction together with a brief description of how it operates can be found in Appendix A.1. □

In a system with one queue, we can thus exploit the meta-transitions corresponding to the repetition of any arbitrary sequence of queue operations.

3.2 Systems with more than one FIFO Queue

In this case, one cannot hope to obtain a result similar to Theorem 4, since iterating simple sequences like $q_1!m_1; q_2!m_2$ can trivially generate non-recognizable languages. (For instance, if σ denotes this sequence, then we have $\sigma^*(\{\varepsilon\}) = \{m_1^k m_2^k \mid k \in \mathbf{N}\}$.) For this class of systems, the first step is thus to characterize precisely the sequences of operations whose Kleene closure preserves the recognizability of languages of queue contents (and therefore the possibility of representing them by QDDs). Our characterization is based on the following notion.

Definition 5 *Let σ_i be a sequence of operations concerning only one queue q_i, and Σ_i be the message alphabet of q_i. The sequence σ_i is* counting *if and only if it satisfies one of the following conditions:*

1. $|\Sigma_i| > 1$ *and* $|\sigma_{i!}| > 0$,
2. $|\Sigma_i| = 1$ *and* $|\sigma_{i!}| > |\sigma_{i?}|$.

Intuitively, a sequence σ of operations that satisfies the previous definition is called "counting" since in that case, there are languages L for which the number k of applications of such a sequence σ on L can be determined by examining the language $\sigma^k(L)$, for any $k \geq 0$ (which implies that $\sigma^k(L) \neq \sigma^\ell(L), \forall k \neq \ell$).

Let σ be a sequence of operations, and q_i be a FIFO queue. The *projection* $\sigma|_i$ of σ on q_i is defined as the subsequence obtained by deleting from σ the operations on queues other than q_i. The following result states that a sequence always preserves the recognizability of languages of queue contents if and only if at most one of its projections is counting.

Theorem 6. *Consider a protocol with $n > 0$ FIFO queues. A sequence of operations σ is such that $\sigma^*(L)$ is recognizable for every recognizable language L of queue contents if and only if there do not exist $1 \leq i < j \leq n$, such that $\sigma|_i$ and $\sigma|_j$ are counting sequences.*

Proof sketch The proof of the sufficient condition is constructive and the corresponding algorithm described in terms of QDDs is given in Appendix A.1. The proof of the necessary condition is omitted due to space limitations. □

In a loop-first search, the effect of any meta-transition satisfying Theorem 6 can thus be computed and represented by a QDD.

4 Properties

In [BG96], only state reachability properties are considered. These properties can be decided on the sole basis of the exact representation of the set of reachable states produced by a loop-first search (when it terminates). Here, we investigate whether more elaborate properties can be checked with a loop-first search and QDDs. Specifically, we discuss the verification of temporal properties. One might argue that checking such properties, expressed for instance as linear-time temporal logic formulas, is problematic using our approach. Indeed, a loop-first search using QDDs computes a representation of the set of reachable states, not of the transitions between these. This contrasts with the approach of Quemener and Jéron [QJ95, QJ96], which produces a graph grammar representing jointly the set of reachable states and the transition relation between these states. In this section, we show that representing symbolically the transition relation is not required. More precisely, we show that temporal properties can be verified using QDDs.

The idea is the following. Let $P = (C, c_0, A, Q, \Sigma, T)$ be a protocol as defined in Section 2. Furthermore, assume we are given a labeling function $\Lambda : C \rightarrow 2^{\mathcal{P}}$ associating to each control state in C a finite set of atomic propositions in $\mathcal{P}$, and a property $\Pi \subseteq (2^{\mathcal{P}})^\omega$ defined by a linear-time temporal logic (LTL) formula [MP92], or by a Büchi automaton [Büc62]. The problem is to check if every run of P from the initial configuration $(c_0, \varepsilon, \ldots, \varepsilon)$ satisfies Π with respect to Λ. To do this, using the required constructions from [VW94, SVW87, Saf88], we build a Büchi automaton $\mathcal{B}_{\overline{\Pi}}$ that accepts the complement of Π, and we compute the product $\mathcal{B}_{\mathcal{P},\overline{\Pi}} = P \times \mathcal{B}_{\overline{\Pi}}$. The result is a protocol enhanced by a set of accepting states, which is defined as the Cartesian product of the control states of P by the accepting states of $\mathcal{B}_{\overline{\Pi}}$. We call such a machine a *Büchi automaton with queues* (as it is indeed a Büchi automaton whose transitions may be labeled by queue operations). The property Π is satisfied by every run of P if and only if the set of accepting runs of $\mathcal{B}_{\mathcal{P},\overline{\Pi}}$ is empty. In other words, we have reduced the model-checking problem for ω-regular properties to testing the emptiness of the language accepted by a Büchi automaton with queues.

This last problem is undecidable, since LTL model-checking is undecidable for protocols as defined in Section 2 [AJ94]. Nonetheless, a partial decision procedure can be obtained as follows. Let $\mathcal{B}$ be a Büchi automaton with queues. An accepting run of $\mathcal{B}$ is a run containing an infinite number of occurrences of some accepting control state c_i (remember that there is a finite number of such states), the queue contents at each visit to this state being allowed to vary. Since it is impossible to check all the runs of $\mathcal{B}$, our partial decision procedure will only search for runs containing an infinite number of occurrences of c_i produced by the infinite execution of a sequence of transitions forming a cycle from c_i to c_i. This amounts to performing a reachability analysis (determining the reachable accepting states) followed by a cycle analysis (determining the infinitely iterable sequences of transitions). The former can be done by a loop-first search, and the latter by testing the cycles corresponding to meta-transitions for infinite iterability. This check is possible thanks to the following result.

Theorem 7. *Let σ be any sequence of queue operations, and let $ITERABLE(\sigma)$ denote the set of all the queue contents w from which σ can be infinitely executed, i.e., such that $\sigma^k(\{w\})$ is non-empty for every $k \geq 0$. Then $ITERABLE(\sigma)$ is recognizable. Moreover, there exists an algorithm for computing a QDD representing $ITERABLE(\sigma)$, given σ.*

Proof sketch The algorithm is given in Appendix A.2. Intuitively, this algorithm is based on the observation that a sequence σ of queue operations can be infinitely iterated if and only if all the sequences $\sigma|_i$ of operations can be infinitely iterated. Therefore, the set of queue contents from which σ can be infinitely iterated is the Cartesian product of the sets of queue contents for which each $\sigma|_i$ can be infinitely iterated. □

During a loop-first search in a Büchi automaton with queues, whenever an accepting control state is reached, the algorithm defined in the previous theorem can be used to test whether this control state can be visited infinitely often by repeatedly executing a sequence of operations forming a cycle in that state. Of course, the technique we have just described can fail to detect nonemptiness of the Büchi automaton with queues. In verification terms, this means that we could fail to detect that the protocol does not satisfy the property. So, what we propose here is not a verification algorithm, but a powerful technique for detecting errors in a protocol, which is often considered to be the most valuable role of model checking. Furthermore, in the next section we show that, under some additional conditions, we can obtain a stronger result.

5 Termination

There is no necessary and sufficient criterion for characterizing the class of communication protocols for which a loop-first search with QDDs terminates. Indeed, it is easy to reduce the halting problem for finite-state machines communicating via unbounded queues (and hence the halting problem for Turing machines) to the decidability of termination for loop-first searches with QDDs.

However sufficient conditions can be obtained. Indeed, we now present an algorithmic criterion on a protocol that is *sufficient* for guaranteeing the termination of a loop-first search in the state space of that protocol, provided that the search is performed in a breadth-first order. The criterion is based on the following definition.

Definition 8. Let σ_1 and σ_2 be two sequences of elementary queue operations. The sequence σ_1 *precedes favorably* σ_2, which we note $\sigma_1 \lhd \sigma_2$, if and only if for every recognizable language L of queue contents, we have $(\sigma_2;\sigma_1)(L) \subseteq (\sigma_1;\sigma_2)(L)$.

In other words, if $\sigma_1 \lhd \sigma_2$, then the sequence $\sigma_1;\sigma_2$ always generates at least all the states generated by the sequence $\sigma_2;\sigma_1$.

In what follows, a *simple cycle* is a cycle that does not contain any occurrence of another cycle. Precisely, it is a sequence of transitions

$$c = c_1 \stackrel{\text{op1}}{\to} c_2 \stackrel{\text{op2}}{\to} c_3 \stackrel{\text{op3}}{\to} \cdots c_k = c$$

such that for all $1 \leq i \neq j < k$, $c_i \neq c_j$. Following the usual definition, a subset of the nodes of a graph is a *strongly connected component* if for every nodes n and n' in the subset, there exists a directed path from n to n'. We are now ready to state the sufficient termination criterion.

Theorem 9. *Let $P = (C, c_0, A, Q, \Sigma, T)$ be a protocol such that :*

– *For each simple cycle* $c_i \xrightarrow{\sigma} c_i$ *in the control graph of* P*, the meta-transition* $c_i \xrightarrow{\sigma^*} c_i$ *is in* T*. (This requires that* σ^* *preserves the recognizability of languages of queue contents.)*
– *For each meta-transition* $c_i \xrightarrow{\sigma^*} c_i$ *and transition* $c_j \xrightarrow{\sigma'} c_k$ *in* T *such that* c_i*,* c_j *and* c_k *belong to the same strongly connected component of the control graph* (C, T) *of* P*, and* $c_j \xrightarrow{\sigma'} c_k$ *is part of*
  - *a simple cycle* $c_i \xrightarrow{\sigma''} c_i$ *such that* $\sigma'' \neq \sigma$*, or*
  - *a simple cycle that does not visit* c_i*,*

  we have $\sigma \triangleleft \sigma'$*.*

A loop-first search of P *performed in a breadth-first order terminates.*

Proof sketch The idea is to show that every reachable global state is reachable by an exploration path whose length is bounded by the number of meta-transitions times the number of transitions in P. An exploration path is a finite sequence of transitions and meta-transitions. We first prove that multiple occurrences of the same meta-transition can be deleted from any exploration path. Indeed, suppose that this path is of the form

$$\cdots c \xrightarrow{\sigma^*} c \xrightarrow{\sigma_1} c' \xrightarrow{\sigma_2} c'' \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_k} c \xrightarrow{\sigma^*} c \cdots,$$

where $c, c', c'', \ldots$ are control states, σ^* is a meta-transition, and each σ_i is either a transition or a meta-transition. Every σ_i appears in an occurrence of the cycle $c \xrightarrow{\sigma} c$ in the path, or is such that $\sigma \triangleleft \sigma_i$, since σ_i links nodes in the same strongly connected component as c. It follows that every recognizable language L of queue contents is such that

$$(\sigma^*; \sigma_1; \ldots \sigma_k; \sigma^*)(L) \subseteq (\sigma^*; \sigma_1; \ldots \sigma_k)(L).$$

It follows that the second occurrence of the meta-transition $c \xrightarrow{\sigma^*} c$ can be deleted from the exploration path. This shows that any reachable configuration can be reached by a path containing a bounded number of meta-transitions. It remains to be proven that the number of transitions can also be bounded. Indeed, suppose that the path contains a simple cycle

$$c \xrightarrow{\sigma_1} c_1 \xrightarrow{\sigma_2} c_2 \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_i} c.$$

This cycle can be replaced by a meta-transition on c (whose existence is required by assumption). Deleting repeatedly the simple cycles and redundant meta-transitions, one finally obtains an exploration path of length bounded by the number of meta-transitions times the number of transitions in P. □

The criterion can be algorithmically decided thanks to the following result.

Theorem 10. *There exists an algorithm for deciding if two sequences* σ_1 *and* σ_2 *of queue operations are such that* $\sigma_1 \triangleleft \sigma_2$*.*

Proof sketch The algorithm is given in Appendix A.3. □

The sufficient termination criterion can also be applied to the model-checking method presented in the previous section.

Theorem 11. *Testing the emptiness of a Büchi automaton with queues that satisfies the criterion of Theorem 9 is decidable.*

Proof sketch The central part of the proof is to show that when it satisfies the conditions of Theorem 9, a Büchi automaton with queues has an accepting computation that visits an accepting state infinitely often by only repeatedly executing a simple loop. □

Though the conditions of Theorem 9 might seem difficult to fulfill, there are common practical situations under which they are immediately satisfied. For instance, if all strongly connected components of the control graph consist of a single cycle whose iteration preserves the recognizability of queue contents.

6 Example

Consider the communication protocol composed of the two state machines shown in Figure 1, and of the unbounded FIFO queues q_{in}, q_{out}, and q_{temp}.

The *consumer* state machine takes as input from q_{in} a sequence of messages m_0 and m_1 and, when it escapes falling into a deadlock, produces as output on q_{out} a sequence of messages m_2 whose length is equal to the smallest of n_0 and n_1, where n_0 (resp. n_1) is the number of messages m_0 (resp. m_1) received from q_{in}. This state machine uses internally the queue q_{temp} for storing intermediate results. The *producer* state machine sends on q_{in} a sequence of messages m_0 followed by a sequence of messages m_1.

Precisely, the protocol is the tuple (C, c_0, A, Q, M, T), where $C = C_{\text{consumer}} \times C_{\text{producer}}$, $C_{\text{consumer}} = C_{\text{producer}} = \{1, 2, 3\}$, $c_0 = (1, 1)$, $A = \emptyset$, $Q = \{q_{\text{in}}, q_{\text{out}}, q_{\text{temp}}\}$, $M = M_{q_{\text{in}}} \cup M_{q_{\text{out}}} \cup M_{q_{\text{temp}}}$, $M_{q_{\text{in}}} = \{m_0, m_1\}$, $M_{q_{\text{out}}} = \{m_2\}$, $M_{q_{\text{temp}}} = \{m'_0, m'_1\}$, and T contains the transitions of the form $((c_1, c'_1), op, (c_2, c'_2))$, where either $c_1 = c_2$ and (c'_1, op, c'_2) is a transition in the consumer state machine, or $c'_1 = c'_2$ and (c_1, op, c_2) is a transition in the producer state machine.

For this example, every simple cycle in the control graph satisfies the condition stated in Theorem 6, and thus can be associated with a meta-transition. A loop-first search performed with those meta-transitions terminates. The set of reachable states obtained at the end of the loop-first search is given at Figure 2, as a finite union of tuples of the form $(c_{\text{consumer}}, c_{\text{producer}}, E_{\text{in}}, E_{\text{out}}, E_{\text{temp}})$, where $(c_{\text{consumer}}, c_{\text{producer}})$ are control states, and E_{in}, E_{out} and E_{temp} regular expressions denoting the corresponding sets of reachable queue contents.

It is worth emphasizing that the search would not stop if meta-transitions like $(1, 2) \xrightarrow{(q_{\text{in}}?m_1; q_{\text{temp}}?m'_0; q_{\text{out}}!m_2)^*} (1, 2)$ were not added to the system. This was not possible with the algorithms presented in [BG96], which could not compute the effect of such meta-transitions.

7 Conclusions and Comparison with Other Work

The analysis of finite-state machines communicating through unbounded FIFO queues is a problem that has been studied for many years. However, our QDD and loop-first search approach is quite different from most earlier attempts to solve this problem. Indeed, rather than trying to find a cute but maybe useless subproblem that is decidable, we rely (except in cases satisfying the conditions of Theorem 9) on a "best effort" approach. From a theoretical point of view, this might seem unsatisfactory, but from a practical point of view, this is not at all troublesome. Indeed, what matters is that the QDD based loop-first search does often terminate. Having no guarantee of success is no more problematic than knowing that a theoretically terminating finite-state search might fail due to the excessive resources that it can require. Another advantage of our method is that it operates very much like a traditional state-space search. This makes it feasible to incorporate it into a state-space search verification tools such as SPIN [Hol91] and, furthermore, to combine it with other search efficiency enhancing techniques such as partial-order methods [Val92, Pel94, God96] or techniques for handling other unbounded data types such as integers [BW94].

From a more abstract point of view, the use we make of QDDs has forced us to develop a new set of algorithms operating on finite automata. These result illustrates the power of finite automata as a representation and hence the power of QDDs. In fact, as a formalism for representing sets of state during a state-space search, finite automata (and hence QDDs) have all the required properties. They have a simple semantics, are easy to manipulate and are closed under all the usual operations. Note that more expressive representations such as rational relations [Pac87] or context-free grammars [LP81] could not be used as a more powerful substitute for QDDs since language inclusion is not decidable for these formalisms. Actually, we view finite-automata representations of state sets as a cornerstone within the set of techniques usable for exploring infinite state spaces.

A possible extension of our work would be to broaden the set of basic operations, i.e., the set formed by the two operations "send" and "receive", that can be performed on queues. For instance, extending this basic set with another operation "non-deterministic send" $q!(m_1 \text{ or } m_2)$, which (non-deterministically) appends either a message m_1 or a message m_2 at the tail of a queue q, makes it possible to construct languages such as $(m_1 \cup m_2)^*$, that are representable by QDDs but could not be generated otherwise.

In [QJ96], another semi-algorithm for the verification of communicating finite-state machines is introduced. This semi-algorithm may generate a graph grammar that represents jointly the set of reachable states and the transition relation of an infinite state space. Since transitions are preserved with this representation, it is then possible (when the semi-algorithm terminates) to check temporal properties such as CTL formulas [QJ95]. In contrast, our QDDs are simpler than the graph grammars of Quemener and Jéron, while still sufficient to check LTL formulas as discussed in Section 4. Since our semi-algorithm attempts to preserve less information in the symbolic representation it produces than the one of [QJ95] and [QJ96], one might expect it to terminate more often.

Interesting future work is to formally validate this claim. Due to the lack of experimental data, it is not known how the two methods compare in practice.

In [FM96], a framework closely related to the one introduced in [BG96] is proposed. Instead of QDDs, a specific class of regular expressions, called "well-parenthesized regular expressions", are used for representing languages of queue contents. Such expressions are generated during a search in an (infinite) state space whenever a sequence of operations that can be infinitely repeated is detected using a criterion presented in [Jer91]. The semi-algorithm of [FM96] may or may not return an exact symbolic representation when it terminates. In contrast, the method that we have discussed here *always* returns an *exact* symbolic representation of the set of reachable states when it terminates. Moreover, "well-parenthesized" regular expressions are strictly less expressive than QDDs. Also, the set of operations considered in [FM96] is strictly contained in the set of operations we can deal with using the results of Section 3.

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.

References

[AJ94] 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.

[BG96] 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.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[BW94] 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.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, 1983.

[CES86] 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.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Technical report, LIFAC, Ecole Normale Supérieure de Cachan, April 1996.

[GL96] 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.

[God96] P. Godefroid Partial-order methods for the verification of concurrent systems – An approach to the state-explosion problem. In Volume 1032 of *Lecture Notes in Computer Science*, Springer-Verlag, 1996.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International Editions, 1991.

[Jer91] T. Jéron. 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.

[LP81] H. R. Lewis and C. H. Papadimitriou. *Elements of the theory of computation.* Prentice Hall, 1981.

[MP92] Z. Manna and A. Pnueli. *The Temporal logic of reactive and concurrent systems: Specification.* Springer-Verlag, 1992.

[Pac87] 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.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag.

[QJ95] Y.-M. Quemener and Th. Jéron. Model-checking of CTL on infinite Kripke structures defined by simple graph grammars. Technical Report 2563, INRIA, June 1995.

[QJ96] Y.-M. Quemener and Th. Jéron. Finitely representing infinite reachability graphs of CFSMs with graph grammars. Technical Report 994, INRIA, March 1996.

[Saf88] S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tur93] K. J. Turner et al. *Using Formal Description Techniques – An Introduction to Estelle, Lotos and SDL.* Wiley, 1993.

[Val92] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1992.

[VW86] 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.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

A Algorithms

In this appendix, the algorithms mentioned in the extended abstract are presented together with a brief description. Complete correctness proofs are omitted due to space limitations.

A.1 Algorithms for Section 3

Let $A = (S, M, \Delta, S_0, F)$ be a QDD and $\sigma = x_1 x_2 \cdots x_k$ be a sequence of operations concerning only one queue. Figure 3 gives an algorithm for computing a QDD accepting the language $\sigma(L(A))$. This algorithm proceeds by executing successively the elementary operations composing σ. The effect of a send operation is to add a new state and a new transition to the QDD (lines 3–6), whereas a receive operation is performed by modifying the set of initial states of the QDD (lines 8–9).

Figure 5 gives an algorithm for computing a QDD accepting $\sigma^*(L(A))$. In this algorithm, $\text{lcm}(x, y)$ denotes tha least common multiple of the integers x and y, and $\lceil z \rceil$ denotes the smallest integer that is greater or equal to the rational z. The algorithm uses the subroutine APPEND-LOOP given at Figure 4. The idea behind the algorithm is to simulate the effect of an unbounded number of executions of APPLY. We know that each send operation executed by APPLY adds a new state and a new transition to the QDD, whereas each receive operation modifies its set of initial states. The first part (up to line 9) of the algorithm APPLY-STAR of Figure 5 tries to detect some periodicity among the sets of initial states of the QDDs modified by the receive operations. The remaining part of the algorithm is aimed at detecting periodicity in the states and transitions that are added by the send operations. Once periodicity is detected, a cycle is added to the set of transitions of the QDD by APPEND-LOOP. The purpose of the cycle is to simulate the effect of subsequent executions of the operations that generated the periodicity. The complete description of the algorithm is lengthy and is omitted here.

Let $A = (S,\ M,\ \Delta,\ S_0,\ F)$ be a QDD and σ a sequence of operations concerning n queues, such that $\sigma(L)$ is recognizable for each recognizable language L. Figure 8 gives an algorithm for computing a QDD accepting the language $\sigma(L(A))$. This algorithm uses a subroutine that is given at Figure 7. In this subroutine, $w|_{<i}$ denotes the projection of the word w over the symbols belonging to the queue alphabets $M_1, M_2, \ldots, M_{i-1}$, and the operator "$\cdot$" denotes the concatenation of regular languages. Given a transformation defined on the set of contents of one queue q_i, the subroutine PERFORM applies this transformation to the contents of q_i without modifiying the content of the other queues. The algorithm N-APPLY proceeds by calling PERFORM with the successive projections of σ on each queue belonging to the system.

An algorithm computing a QDD accepting $\sigma^*(L(A))$ is given at Figure 9. In this algorithm, $L|_{\neq i}$ denotes the projection of the language L over the set of symbols that do not belong to the message alphabet M_i of the i-th queue of the system. Like in the case of N-APPLY, the algorithm proceeds by decomposing the application of σ^* into successive applications of the different $(\sigma|_i)^*$.

A.2 Algorithm for Section 4

Let σ be a sequence of queue operations (concerning any number of queues). Figure 11 gives an algorithm for computing ITERABLE(σ). It uses the subroutine ONE-ITERABLE, given at Figure 10. In these figures, $\text{pre}(w)$ denotes the set of strict prefixes of the word w, and regular expressions denote QDDs representing the corresponding recognizable languages, while $\gcd(x, y)$ denotes the greatest common divisor of the two integers x and y. The algorithm proceeds by projecting σ onto each queue belonging to the system.

A.3 Algorithms for Section 5

Let σ_1 and σ_2 be sequences of queue operations (concerning any number of queues). Figure 12 presents an algorithm for deciding if $\sigma_1 \lhd \sigma_2$. In this figure, the notation $\sigma(w)$ denotes the result of applying the sequence of queue operations σ concerning only one queue to the queue content w (if σ cannot be applied to w, then we have $\sigma(w) = \emptyset$). The algorithm proceeds by considering the successive projections of σ_1 and σ_2 on the different queues belonging to the system. We have $\sigma_1 \lhd \sigma_2$ if and only if we have $(\sigma_1|_i) \lhd (\sigma_2|_i)$ for each queue q_i.

Let $B = (C, c_0, A, Q, \Sigma, T, F)$ be a Büchi automaton with queues satisfying our sufficient criterion for the termination of a loop-first search. Figure 13 gives an algorithm for deciding whether it has an accepting run or not. The algorithm is based on the fact that if there exists an accepting run, then there exists such a run where a simple cycle is infinitely iterated. In this algorithm, LOOP-FIRST(B) denotes the representation of the set of reachable states returned by a loop-first search in the state space of B (this representation consists of a finite set of pairs (control state, QDD)).

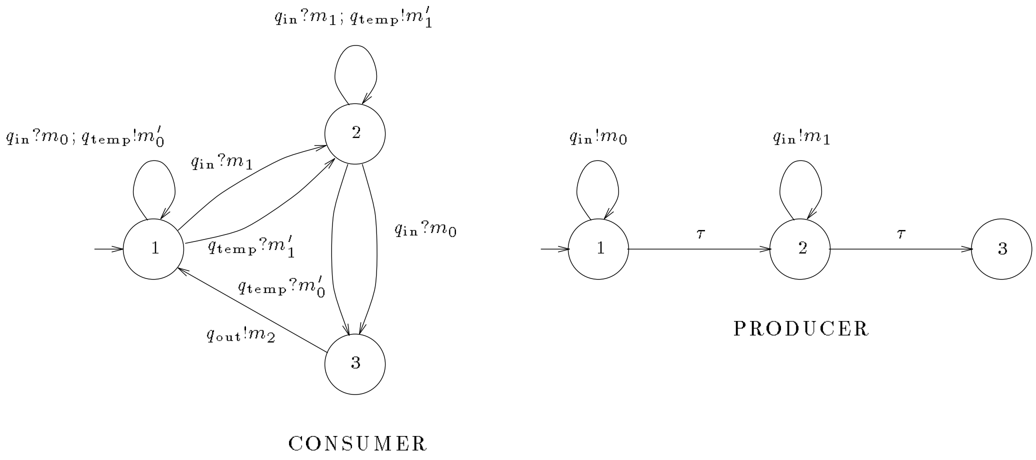

Fig. 1. Example of communication protocol.

$$
\begin{array}{l}
(1, 1, m_0^*, \varepsilon, (m_0')^*) \\
(1, 2, m_0^* m_1^*, \varepsilon, (m_0')^*) \\
(1, 2, m_1^*, m_2 m_2^*, (m_0')^*(m_1')^*) \\
(1, 3, m_0^* m_1^*, \varepsilon, (m_0')^*) \\
(1, 3, m_1^*, m_2 m_2^*, (m_0')^*(m_1')^*) \\
(2, 2, m_1^*, m_2^*, (m_0')^*(m_1')^*) \\
(2, 3, m_1^*, m_2^*, (m_0')^*(m_1')^*) \\
(3, 2, m_1^*, m_2^*, (m_0')^*(m_1')^*) \\
(3, 3, m_1^*, m_2^*, (m_0')^*(m_1')^*)
\end{array}
$$

Fig. 2. Reachable states of the example protocol.

APPLY(QDD (S, M, Δ, S_0, F), sequence $x_1 x_2 \cdots x_k$) :
- **for** $i := 1$ **to** k **do**
  - **if** x_i is a send operation **then**
    - $S' := S \cup \{s_i\}$ (with $s_i \notin S$);
    - $\Delta' := \Delta \cup \{(s, \mu(x_i), s_i) \mid s \in F\}$;
    - $F' := \{s_i\}$;
    - $(S, M, \Delta, S_0, F) := (S', M, \Delta', S_0, F')$
  - **else**
    - $S_0' := \{s' \in S \mid (\exists s \in S_0)((s, \mu(x_i), s') \in \Delta)\}$;
    - $(S, M, \Delta, S_0, F) := (S, M, \Delta, S_0', F)$;
- **return** (S, M, Δ, S_0, F).

Fig. 3. Algorithm for computing $\sigma(L)$, system with one queue.

APPEND-LOOP(QDD $(S,\ M,\ \Delta,\ S_0,\ F)$, word $a_1 a_2 \cdots a_k$) :

1: **if** $k = 0$ **then return** $(S,\ M,\ \Delta,\ S_0,\ F)$;

2: $S' := S \cup \{s_1,\ s_2,\ \ldots,\ s_k\}$ (with $\{s_1,\ s_2,\ \ldots,\ s_k\} \cap S = \emptyset$);

3: $\Delta' := \Delta \cup \{(s,\ a_1,\ s_1) \mid s \in F\} \cup \{(s_{i-1},\ a_i,\ s_i) \mid 1 < i \leq k\}$
$\cup \{(s_k,\ a_1,\ s_1)\}$;

4: $F' := F \cup \{s_k\}$;

5: **return** $(S',\ M,\ \Delta',\ S_0,\ F')$.

Fig. 4. Subroutine APPEND-LOOP.

APPLY-STAR(QDD (S, M, Δ, S_0, F), sequence σ) :

$A_0 := (S, M, \Delta, S_0, F)$;

if $|\sigma_?| = 0$ **then return** APPEND-LOOP$(A_0, \mu(\sigma_!))$;

$n := 0$;

$I_0 := S_0$;

repeat

  $n := n + 1$;

  $A_n := (S_n, M, \Delta_n, S_{0n}, F_n) :=$ APPLY(A_{n-1}, σ);

  $I_n := S_{0n} \cap S$

until there exists m such that $0 \leq m < n$ and $I_n = I_m$;

if $|\sigma_!| = 0$ **then return** $\bigcup_{0 \leq i < n} A_i$;

$p := \mathrm{lcm}(n - m, |\sigma_!|)$;

$k := \max(1, \left\lceil 2\frac{|\sigma_?|}{|\sigma_!|} - \frac{m}{p} \right\rceil)$;

$A_{m_1} := (S_{m_1}, M, \Delta_{m_1}, S_{0m_1}, F_{m_1}) :=$ APPLY(A_m, σ^{kp});

$A_{m_2} := (S_{m_2}, M, \Delta_{m_2}, S_{0m_2}, F_{m_2}) :=$ APPLY(A_{m_1}, σ^p);

$T_1 := \{s' \in S_{m_1} \mid (\exists s \in S, w \in M^*)((s, w, s') \in \Delta^*_{m_1} \wedge |w| \leq p|\sigma_?|)\}$;

$T_2 := \{s' \in S_{m_1} \mid (\exists s \in S, w \in M^*)((s, w, s') \in \Delta^*_{m_1} \wedge |w| \leq 2p|\sigma_?|)\}$;

$B_0 := (S_{m_1}, M, \Delta_{m_1}, S_{0m_1} \cap T_1, F_{m_1})$;

$B_1 :=$ APPEND-LOOP$(B_0, (\mu(\sigma_!))^p)$;

$C_0 := (S'_0, M, \Delta'_0, S'_{00}, F'_0) := (S_{m_1}, M, \Delta_{m_1}, S_{0m_1} \setminus T_1, F_{m_1})$;

$D_0 := (S_{m_2}, M, \Delta_{m_2}, S_{0m_2} \cap (T_2 \setminus T_1), F_{m_2})$;

if $|\sigma_!| \geq |\sigma_?|$ **then**

  $l := \frac{p}{|\sigma_!|}(|\sigma_!| - |\sigma_?|)$;

  $C_1 :=$ APPEND-LOOP$(C_0, (\mu(\sigma_!))^l)$;

  $D_1 :=$ APPEND-LOOP$(D_0, (\mu(\sigma_!))^l)$;

  $D_2 :=$ APPEND-LOOP$(D_1, (\mu(\sigma_!))^p)$;

  return $\bigcup_{0 \leq i < m+kp}$ APPLY$(A_0, \sigma^i) \cup \bigcup_{0 \leq i < p}$ APPLY$(B_1 \cup C_1 \cup D_2, \sigma^i)$

(...)

Fig. 5. Algorithm for computing $\sigma^*(L)$, system with one queue.

(...)

else

    $a := 0$;

    while $S'_{0a} \neq \emptyset$ **do**

        $a := a + 1$;

        $C_a := (S'_a,\, M,\, \Delta'_a,\, S'_{0a},\, F'_a) := \mathrm{APPLY}(C_{a-1},\, \sigma^p)$;

    $b := 0$;

    $r_0 := m|\sigma_!| + (k+1)p|\sigma_!| - 3p|\sigma_?|$;

    repeat

        $b := b + 1$;

        if $r_{b-1} \geq 0$ **then**

            $D_b := \mathrm{APPLY}(D_{b-1},\, \sigma^p)$;

            $r_b := r_{b-1} + p|\sigma_!| - p|\sigma_?|$

        else

            $D_b := \mathrm{APPLY}(D_{b-1},\, (\sigma_!)^p)$;

            $r_b := r_{b-1} + p|\sigma_!|$

    until there exists c such that $0 \leq c < b$ and $r_c = r_b$;

    $E_0 := (S''_0,\, M,\, \Delta''_0,\, S''_{00},\, F''_0) := \bigcup_{0 \leq i < b} D_i$;

    $D := \text{APPEND-LOOP}(E_0,\, (\mu(\sigma_!))^p)$;

    $d := 0$;

    while $S''_{0d} \neq \emptyset$ **do**

        $d := d + 1$;

        $E_d := (S''_d,\, M,\, \Delta''_d,\, S''_{0d},\, F''_d) := \mathrm{APPLY}(E_{d-1},\, \sigma^p)$;

    return $\bigcup_{0 \leq i < m+kp} \mathrm{APPLY}(A_0,\, \sigma^i)$

        $\cup \bigcup_{0 \leq i < p} \mathrm{APPLY}(B_1 \cup \bigcup_{0 \leq j < a} C_j \cup D \cup \bigcup_{0 \leq j < d} E_j,\, \sigma^i)$.

Fig. 6. Algorithm for computing $\sigma^*(L)$, system with one queue (continued).

PERFORM(QDD (S, M, Δ, S_0, F), function FUNCT, integer i, sequence σ) :

1: $A := (S, M, \Delta, S_0, F)$;

2: $j := 0$;

3: **for each** $(s, s') \in S^2$ **such that**

$(\exists s_0 \in S_0, w \in L(A))((s_0, w|_{<i}, s) \in \Delta^* \wedge (s, w|_i, s') \in \Delta^*)$ **do**

4: $B_j := (S, M, \Delta, S_0, \{s\})$;

5: $C_j :=$ FUNCT($(S, M, \Delta, \{s\}, \{s'\}), \sigma$);

6: $D_j := (S, M, \Delta, \{s'\}, F)$;

7: $j := j + 1$;

8: **return** $\bigcup_{0 \leq k < j} (B_k \cdot C_k \cdot D_k)$.

Fig. 7. Subroutine PERFORM.

N-APPLY(QDD (S, M, Δ, S_0, F), sequence σ) :

1: $A_0 := (S, M, \Delta, S_0, F)$;

2: **for** $i := 1$ **to** n **do**

3: $A_i :=$ PERFORM(A_{i-1}, APPLY, i, $\sigma|_i$);

4: **return** A_n.

Fig. 8. Algorithm for computing $\sigma(L)$, system with n queues.

N-APPLY-STAR(QDD (S, M, Δ, S_0, F), sequence σ) :

$i := 1$;
repeat
  if $(|\Sigma_i| > 1$ **and** $|(\sigma|_i)_!| > 0)$ **or**
    $(|\Sigma_i| = 1$ **and** $|(\sigma|_i)_!| > |(\sigma|_i)_?|)$ **then goto** end;
  $i := i + 1$
until $i > n$;
end:
$q := 0$;
$A_0 := (S, M, \Delta, S_0, F)$;
$L_0 := L(A_0)|_{\neq i}$;
repeat
  $q := q + 1$;
  $A_q :=$ N-APPLY(A_{q-1}, σ);
  $L_q := L(A_q)|_{\neq i}$
until there exists p such that $0 \leq p < q$ and $L_p = L_q$;
if $i > n$ **then return** $\bigcup_{0 \leq k < q} A_k$;
$l := q - p$;
$B :=$ PERFORM$(A_q$, APPLY-STAR, i, $(\sigma|_i)^l)$;
return $\bigcup_{0 \leq k < q} A_k \cup \bigcup_{0 \leq j < l}$ N-APPLY(B, σ^j).

Fig. 9. Algorithm for computing $\sigma^*(L)$, system with n queues.

ONE-ITERABLE(sequence σ) :

if $|\sigma_!| < |\sigma_?|$ **then return** $\emptyset$;
if $|\sigma_?| = 0$ **then return** Σ^*;
if $\sigma = (\sigma_?; \sigma_!)$ **then**
    $g := \mathrm{gcd}(|\sigma_!|, |\sigma_?|)$;
    $U := \{u \in \Sigma^* \mid u \in \mathrm{pre}(\mu(\sigma_?)) \wedge u(\mu(\sigma_!))^{(|\sigma_?|/g)} = (\mu(\sigma_?))^{(|\sigma_!|/g)}\, u\}$;
    return $\bigcup_{u \in U} (\mu(\sigma_?))^+ u$
else
    $L_\sigma :=$ ONE-ITERABLE$(\sigma_?; \sigma_!)$;
    $L := L_\sigma$;
    for every $u \in \mathrm{pre}(\mu(\sigma_?))$ **do**
        if $|\sigma_!| = |\sigma_?|$ and $\sigma(u) = u$ **then** $L := L \cup \{u\}$;
        if $|\sigma_!| > |\sigma_?|$ **then**
            $l := \lceil (|\sigma_?| - |u|)/(|\sigma_!| - |\sigma_?|) \rceil$;
            if $\sigma^l \in L_\sigma$ **then** $L := L \cup \{u\}$;
    return L.

Fig. 10. Subroutine ONE-ITERABLE.

ITERABLE(sequence σ) :

for $i := 1$ **to** n **do**
    $L_i :=$ ONE-ITERABLE$(\sigma|_i)$;
return $\prod_{1 \le i \le n} L_i$.

Fig. 11. Algorithm for computing ITERABLE(σ), system with n queues.

PRECEDES?(sequences σ_1, σ_2) :
for $i := 1$ **to** n **do**
  $\sigma_a := (\sigma_1; \sigma_2)|_i$;
  $\sigma_b := (\sigma_2; \sigma_1)|_i$;
  if $\sigma_{a!} \neq \sigma_{b!} \vee \sigma_{a?} \neq \sigma_{b?}$ **then**
    return "$\sigma_1 \ntriangleleft \sigma_2$";
  for all $w \in \mathrm{pre}(\mu((\sigma_{b?})))$ **do**
    if $\sigma_a(w) \neq \sigma_b(w)$ **then**
      return "$\sigma_1 \ntriangleleft \sigma_2$";
return "$\sigma_1 \triangleleft \sigma_2$".

Fig. 12. Algorithm for deciding if $\sigma_1 \triangleleft \sigma_2$, system with n queues.

BUCHI-EMPTY?(Büchi automaton with queues $(C, c_0, A, Q, \Sigma, T, F)$) :
$S :=$ LOOP-FIRST($(C, c_0, A, Q, \Sigma, T)$);
for each $c_i \in F$ **do**
  $R := \{w \in \Sigma^* \mid (c_i, w) \in S\}$;
  for each meta-transition $c_i \xrightarrow{\sigma^*} c_i \in T$ **do**
    $I :=$ ITERABLE(σ);
    if $I \cap R \neq \emptyset$ **then**
      return "not empty";
return "empty".

Fig. 13. Test of emptiness for a Büchi automaton with queues.