Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations
Ahmed Bouajjani *, Peter Habermehl
VERIMAG, Centre Équation, 2 av. de Vignate, 38610 Gières, France

Abstract

We address the verification problem of FIFO-channel systems. We apply the symbolic analysis principle to these systems. We represent their sets of configurations using structures called constrained queue-content decision diagrams (CQDDs) combining finite-state automata with linear arithmetical constraints on number of occurrences of symbols. We show that CQDDs allow forward and backward reachability analysis of systems with nonregular sets of configurations. Moreover, we prove that CQDDs allow to compute the exact effect of the repeated execution of any fixed cycle in the transition graph of a system. We use this fact to define a generic reachability analysis semi-algorithm parametrized by a set of cycles Θ. Given a set of configurations, this semi-algorithm performs a least fixpoint calculation to construct the set of its successors (or predecessors). At each step, this calculation is accelerated by considering the cycles in Θ as additional “meta-transitions” in the transition graph, generalizing the approach of Boigelot and Godefroid (CAV‘96, Lecture Notes in Computer Science, Vol. 1102, Springer, Berlin, 1996). ©1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Verification; Infinite-state systems; FIFO-channel systems; Symbolic reachability analysis; Non-regular sets; Automata; Linear constraints

1. Introduction

The analysis of the behaviour of a system is often reduced to solving some reachability problems on its formal model, which is usually some kind of a transition system. For example, verifying that all computations of a system are safe consists in checking whether the set of reachable states (the successors of the initial states) intersects with the set of dangerous states. Equivalently, we can check whether the set of initial states has an empty intersection with the set of states from which a dangerous state is reachable (the predecessors of the dangerous states). These two formulations of the safety verification problem correspond respectively to the forward and backward reachability
analysis algorithms. Therefore, the computation of the sets of successors or predecessors of a given set of states \( S \) is a fundamental component of almost every verification technique.

Let \( \text{post}(S) \) and \( \text{pre}(S) \) denote respectively the set of immediate successors and predecessors of a set \( S \) and let \( \text{post}^*(S) \) and \( \text{pre}^*(S) \) denote the set of all its successors and predecessors. Clearly, \( \text{post}^*(S) \) is the limit of the infinite non-decreasing sequence \( (X_i)_{i \geq 0} \) given by \( X_0 = S \) and \( X_{i+1} = X_i \cup \text{post}(X_i) \) for every \( i \geq 0 \). Similarly, \( \text{pre}^*(S) \) is the limit of the infinite sequence obtained by considering the \( \text{pre} \) function instead of \( \text{post} \).

From these definitions, one can derive straightforwardly iterative procedures for computing the sets \( \text{post}^*(S) \) and \( \text{pre}^*(S) \) that consist simply of computing the elements of the sequence of the \( X_i \)'s and checking for each index \( i \), whether \( X_{i+1} = X_i \), in which case, we know that the limit is reached.

For systems modelled as finite-state automata, such algorithms are guaranteed to terminate. However, for more general system modelled as automata augmented with variables ranging over unbounded data types (integers, reals, stacks, queues, etc.), the sets \( X_i \) are in general infinite and the sequence \( (X_i)_{i \geq 0} \) is not guaranteed to converge in a finite number of steps.

In order to verify such systems we need to find a class of finite structures that can represent the infinite sets of states we are interested in. Moreover, in order to apply forward/backward reachability analysis algorithms, this class of structures should at least be effectively closed under union, intersection, and the \( \text{post} \) and \( \text{pre} \) operators. Finally, since we have to compare two sets to detect convergence and we wish to check whether a set is empty, the inclusion and emptiness problems of this class of structures should be decidable.

For instance, in the case of systems manipulating integer or real valued variables like Petri nets (vector addition systems) or timed and hybrid automata, representation structures based on polyhedra or sets of linear constraints can be considered (e.g. [3, 7, 2, 16]). When manipulating sequential data structures like stacks or queues whose sets of states are vectors of words, automata-based representation structures can naturally be used [9, 5, 8, 14]. On such structures the \( \text{post}^* \) and \( \text{pre}^* \) operations can be implemented along with set-theoretic operations, membership and emptiness testing.

After solving the representation and calculation problems, we face the convergence problem of the sequence \( X_i \). In general, this sequence never reaches its limit and an exact acceleration of the computation of this limit must be considered, usually by defining another increasing sequence \( (Y_i)_{i \geq 0} \) such that for every \( i \geq 0 \), \( X_i \subseteq Y_i \), and \( Y_i = \bigcup_{j \geq 0} X_j \).

This approach has been used in [9, 8] to define model-checking algorithms for push-down systems and various linear and branching-time logics, using (alternating) finite-state automata as representation structures of sets of stack contents. In that case, the accelerated sequence \( (Y_i)_{i \geq 0} \) is guaranteed to converge.

In [5], automata-based structures called queue-content decision diagrams (QDDs) are used to represent queue contents of FIFO-channel systems (communicating finite-state
machines, or CFSM for short). However, contrary to the case of pushdown systems, the set of reachable states of a CFSM is in general not regular, and hence is not amenable to QDD representation. Actually, it is well known that CFSMs can simulate Turing machines and that they can generate any recursively enumerable set [10]. Moreover, there is no algorithm for constructing the set of reachable states even if we know that this set is regular [10, 12, 1]. Hence, for such systems, the best we can hope for are acceleration techniques that increase the chance of termination in most cases encountered in practice.

In [7], an acceleration technique has been introduced and applied to systems with integer variables. Later, this technique has been adapted to the case of FIFO-channel systems in [5]. It is based on the notion of meta-transition, which corresponds, roughly speaking, to the iterative execution of some cycle (of a special kind) in the control graph of the system. More precisely, given a cycle \( \theta \), the acceleration consists in adding to each \( X_{i+1} \) the set of states \( post^*_\theta(X_i) \) which corresponds to the set of all successors of \( X_i \) after repeating \( \theta \) as much as possible. Then, the problem is to determine for which cycles \( \theta \) the considered class of representation structures is effectively closed under \( post^*_\theta \). In [5], only cycles \( \theta \) of very restricted forms are considered in order to guarantee that the \( post^*_\theta \) image of a regular set is also regular. In [6] necessary and sufficient conditions on the form of the cycles are given for this.

In this paper, we also consider CFSMs and propose a generalization of the approach adopted in [5, 6] by allowing exact acceleration using any cycle in the transition graph of the system. The difficulty comes from the fact that the set of states reachable via a cycle is in general nonregular, and we need to go beyond finite automata in order to represent such sets. The structure we propose, constrained QDD (CQDD), is based on a combination of (a subclass of) finite-state automata augmented with linear arithmetical constraints on the number of times their transitions are used in accepting runs. We use arithmetical constraints because the iterative execution of a cycle introduces constraints on the number of occurrences of some symbols (messages) in different channels. Consider for instance a cycle where the system sends one symbol to each of two different channels. Then, the set of reachable configurations by the execution of this cycle is the set of pairs of sequences having the same length.

We show that CQDDs satisfy the desirable properties of a representation structure. Moreover, and this constitutes our main technical result, we prove that the class of CQDD representable sets of states is effectively closed under the function \( post^*_\theta \) for every cycle \( \theta \). We prove also that the class consisting of reverse images of CQDD representable sets is effectively closed under the function \( pre^*_\theta \) for every cycle \( \theta \). These results allow to define a generic forward/backward reachability analysis semi-algorithm which is parametrized by a set of cycles in the transition graph of the system. When it terminates, our algorithm returns the exact set of successors (or predecessors) of a given CQDD representable (or CQDD reverse representable) set of states. A variety of analysis algorithms can be derived from our algorithm by determining adequate strategies for choosing the set of cycles to be considered for acceleration. The
algorithm of Boigelot and Godefroid [5] can be seen as a particular instance of our algorithm.

1.1. Related work

There are several works on symbolic analysis of CFSMs. The idea of using regular (or recognizable) sets to represent sets of reachable configurations is probably due to Pachl [17]. However, he does not provide effective procedures for computing these representations. The idea of using regular sets together with acceleration techniques based on computing effects of control cycles (loop-first search) is introduced in [5] and extended in [6]. There, finite-state automata (QDDs) are used as representation structures. A similar approach is adopted in [13] where regular expressions are used as representation structures. The semi-algorithm proposed in that work can use more control cycles than the ones proposed in [5, 6], but only an upper approximation of the set of reachable configurations is computed.

In [11, 19] a model-checking semi-algorithm is proposed for CFSMs based on a finite representation of the state-graph by means of graph grammars. This approach is different from ours since it is based on a finite representation of the (infinite) state-graph of the system rather than on a representation of the set of reachable states. Other authors [1, 12] analyse CFSMs assuming that they have lossy or unreliable channels (queues); we do not employ such simplifying assumptions in this work. Finally, [18] proposed (terminating) algorithms which generate upper approximations of the set of reachable states. This is different from our approach which constructs the exact set of reachable states using a fixpoint calculation and which "helps" the calculation to terminate using exact accelerations.

1.2. Outline

The rest of the paper is organized as follows: In Section 2 we define the CFSMs. In Section 3, we introduce the CQDDs and give the results concerning their closure properties and their decision problems. In Section 4, we show that CQDDs are suitable for representing and manipulating sets of configurations of CFSMs. In Section 5 we present our main result saying that CQDDs allow to compute the effect of any cycle in the control graph of a CFSM. In Section 6 we present our generic forward and backward reachability analysis algorithm. We give concluding remarks in Section 7.

2. Communicating finite-state machines

We consider a generalization of communicating finite-state machines (CFSM) defined in [4]. A CFSM is a finite-state machine which can send and receive messages over a finite set of unbounded FIFO queues. In the usual definition of CFSMs, a transition either appends a message to the end of a queue or removes a message from the head of a queue. We generalize this by allowing simultaneously appending and removing messages from several queues.
Definition 2.1. A communicating finite-state machine (CFSM for short) is a tuple 
\((S,K,\Sigma,T)\) where

- \(S\) is a finite set of control states,
- \(K\) is a finite set of unbounded FIFO queues,
- \(\Sigma\) is a finite set of messages,
- \(T\) is a finite set of transitions.

Each transition is of the form \((s_1,op,s_2)\), where \(s_1\) and \(s_2\) \(\in\) \(S\), and \(op\) is a finite set of queue operations of the form \(\kappa_i!w\) or \(\kappa_i?w\) with \(\kappa_i\in K\) and \(w\in \Sigma^*\) such that for each queue \(\kappa_i\) there is at most one label \(\kappa_i!w\) or \(\kappa_i?w\) in \(op\).

Intuitively, the queue operation \(\kappa_i!w\) corresponds to adding the word \(w\) to the right of queue \(i\) and \(\kappa_i?w\) corresponds to removing \(w\) from the left of queue \(i\).

We introduce the notion of configuration of a CFSM.

Definition 2.2. Let \(\mathcal{M} = (S,K,\Sigma,T)\) be a CFSM. A configuration of \(\mathcal{M}\) is a tuple 
\(\gamma = (s,w)\) where \(s\) is a control state in \(S\), and \(w = (w_1,\ldots,w_{|K|})\) is a \(|K|\)-dim multi-word (i.e., a tuple in \((\Sigma^*)^{|K|}\)), each \(w_i\) being the contents of the queue \(\kappa_i\), for \(i\in \{1,\ldots,|K|\}\).

We denote by \(Conf\) the set of all configurations of \(\mathcal{M}\), i.e., \(Conf = S \times (\Sigma^*)^{|K|}\).

We define a global transition relation between configurations of a CFSM.

Definition 2.3. Let \(\gamma = (s,w_1,\ldots,w_{|K|})\) and \(\gamma' = (s',w'_1,\ldots,w'_{|K|})\) be two configurations of a CFSM \(\mathcal{M} = (S,K,\Sigma,T)\), and let \(op\) be a set of queue operations. Then, we have \(\gamma \xrightarrow{op} \gamma'\) if and only if there exists a transition \((s,op,s')\in T\) such that, for every \(i\in \{1,\ldots,|K|\}\),

- if \(\kappa_i?w\in op\) then \(w'_i = w_i\),
- if \(\kappa_i!w\in op\) then \(w'_i = w_iw\),
- otherwise \(w'_i = w_i\).

Definition 2.4. Let \(\mathcal{M} = (S,K,\Sigma,T)\) be a CFSM. Given a transition \(\tau = (s,op,s')\in T\), we say that \(\tau\) is executable at \(\gamma = (s,w)\) if there exists \(\gamma' = (s',w')\) such that \(\gamma \xrightarrow{op} \gamma'\). In this case, \(\gamma'\) is the immediate successor of \(\gamma\) by \(\tau\), and \(\gamma\) is the immediate predecessor of \(\gamma'\) by \(\tau\). We denote by \(pre_\tau\) and \(post_\tau\) the functions in \(2^{Conf} \to 2^{Conf}\) such that, for every set of configurations \(C\), \(pre_\tau(C)\) (resp. \(post_\tau(C)\)) is the set of immediate predecessors (resp. successors) of the configurations in \(C\) by the transition \(\tau\). We denote respectively by \(pre\) and \(post\) the unions of the functions \(pre_\tau\) and \(post_\tau\), for all the transitions \(\tau\in T\). The functions \(pre^*\) and \(post^*\) are the reflexive transitive closures of \(pre\) and \(post\), respectively.

The function \(pre^*\) (resp. \(post^*\)) yields the set of all predecessors (resp. successors) of a given set of configurations.

Definition 2.5. We say that a sequence \(\theta\) of transitions in \(T\) is a cycle if it is of the form \((s_0,op_0,s_1)(s_1,op_1,s_2)\cdots(s_{n-1},op_n,s_n)\) and \(s_0 = s_n\) for some \(n>0\).
The notion of executability can be generalized in the obvious manner to sequences of transitions (in particular to cycles). The definitions of \( \text{pre}_r \) and \( \text{post}_r \) can also be straightforwardly generalized to sequences of transitions.

**Definition 2.6.** Let \( \theta = \tau_1 \cdots \tau_n \) be a sequence of transitions. Then, \( \text{pre}_\theta = \text{pre}_{\tau_1} \circ \cdots \circ \text{pre}_{\tau_n} \) and \( \text{post}_\theta = \text{post}_{\tau_1} \circ \cdots \circ \text{post}_{\tau_n} \), where \( \circ \) denotes the composition of functions. The functions \( \text{pre}_\theta^* \) and \( \text{post}_\theta^* \) are the reflexive transitive closures of \( \text{pre}_\theta \) and \( \text{post}_\theta \), respectively.

Given a set of configurations \( C \), \( \text{pre}_\theta^*(C) \) (resp. \( \text{post}_\theta^*(C) \)) is the set of predecessors (resp. successors) of configurations in \( C \) obtained by iterating an arbitrary number of times the sequence \( \theta \). Notice that this number of iterations is strictly greater than one only if \( \theta \) is a cycle.

The following notations and definitions are useful in the proofs of the results presented in Sections 3 and 4.

**Notation 2.1.** We denote by \( \varepsilon \) the empty word. Let \( u = (u_1, \ldots, u_n) \) and \( v = (v_1, \ldots, v_n) \) be two \( n \)-dim multi-words. Then, we denote by \( uv \) the multi-word \( (u_1v_1, \ldots, u_nv_n) \).

**Notation 2.2.** Let \( u \) and \( v \) two \( n \)-dim multi-words. If there exists a \( n \)-dim multi-word \( w' \) such that \( uw' = v \) then we denote \( w' \) by \( u^{-1}v \).

**Definition 2.7.** Let \( \tau = (s, op, s') \) be a transition of a CFSM \( (S, K, \Sigma, T) \). Then, \( \text{out}(\tau) \) denotes the multi-word \( (w_1, \ldots, w_{|K|}) \in (\Sigma^*)^{|K|} \) such that, for every \( i \in \{1, \ldots, |K|\} \), \( w_i = w \) if \( \kappa_i w \in op \), otherwise \( w_i = \varepsilon \). We denote by \( \text{in}(\tau) \) the multi-word in \( (\Sigma^*)^{|K|} \) such that the \( i \)-th component of \( \text{in}(\tau) \) is \( w \) if \( \kappa_i w \in op \), otherwise \( \varepsilon \).

Intuitively, \( \text{out}(\tau) \) is the output of the system into the queues by executing \( \tau \), whereas \( \text{in}(\tau) \) is the input of the system from the queues. These notions can be generalized straightforwardly to sequences of transitions.

**Definition 2.8.** Let \( \theta = \tau_1 \cdots \tau_n \) be a sequence of transitions. Then, \( \text{out}(\theta) = \text{out}(\tau_1) \circ \text{out}(\tau_2) \cdots \text{out}(\tau_n) \) and \( \text{in}(\theta) = \text{in}(\tau_1) \circ \text{in}(\tau_2) \cdots \text{in}(\tau_n) \).

### 3. Representation structures

In this section we introduce representation structures for sets of contents of queues. These structures, called constrained queue-content decision diagrams (CQDD), consist of a combination of finite-state automata (restricted deterministic simple automata) with linear constraints on the number of times transitions in these automata are taken. These constraints are expressed in Presburger arithmetics. This combination allows to represent nonregular sets of contents of queues. We start by giving the definition of these structures, and then we prove closure properties concerning the class of languages they define and show that their emptiness and inclusion problems are decidable.
3.1. Preliminary definitions

3.1.1. Presburger arithmetics

Presburger arithmetics is the first-order logic of natural numbers with addition, subtraction and the usual ordering. Let \( x \) range over an enumerable set of variables, and consider the set of terms given by

\[
\begin{align*}
 t := & 0 | 1 | x | t - t | t + t \\
\end{align*}
\]

Then, the set of \textit{Presburger formulas} is defined by

\[
\begin{align*}
 f := & t \leq t | f \lor f | \exists x. f \\
\end{align*}
\]

Classical abbreviations can be used like integer constants (\( k \)), equality (=), boolean connectives as conjunction (\( \land \)), implication (\( \Rightarrow \)) and equivalence (\( \Leftrightarrow \)) as well as universal quantification (\( \forall \)).

We say that \( f \) is a Presburger formula over a set of variables \( X = \{x_1, \ldots, x_n\} \), and we write \( f(X) \), if the set of free variables in \( f \) is precisely \( X \).

The semantics of Presburger formulas is defined in the standard way. Given a formula \( f \) with free variables \( X = \{x_1, \ldots, x_n\} \), and a valuation \( v : X \rightarrow \mathbb{N} \), we say that \( v \) satisfies \( f \), and write \( v \models f \), if the evaluation of \( f \) under \( v \) is true. We say that a formula \( f \) is valid if every valuation satisfies \( f \).

3.1.2. Finite-state automata

A finite-state automaton over \( \Sigma \) is a tuple \( A = (Q, q_0, \rightarrow, F) \) where \( Q \) is a finite set of states, \( q_0 \) is the initial state, \( \rightarrow \subseteq Q \times \Sigma \times Q \) is a transition relation (we write \( q \xrightarrow{a} q' \) instead of \( (q, a, q') \in \rightarrow \)), and \( F \subseteq Q \) is a set of final states.

Given a word \( w = a_0 \ldots a_t \in \Sigma^* \), a run of \( A \) over \( w \) is a sequence of transitions \( \rho = (s_0, a_0, s_1) \ldots (s_\ell, a_\ell, s_{\ell + 1}) \in \rightarrow^{\ell + 1} \) such that \( s_0 = q_0 \). The run \( \rho \) is accepting if \( s_{\ell + 1} \in F \). The language accepted by \( A \), denoted by \( L(A) \), is the set of words \( w \in \Sigma^* \) such that there is an accepting run of \( A \) over \( w \).

A finite-state automaton is deterministic, if for every state \( s \in S \) and every symbol \( a \in \Sigma \), there is at most one state \( s' \) such that \( (s, a, s') \in \rightarrow \).

3.2. Simple automata

We introduce hereafter the classes of simple automata and restricted simple automata. These classes are used in the definition of the representation structures we consider.

\textbf{Definition 3.1.} A simple automaton over \( \Sigma \) (SA) is a finite-state automaton \( A = (Q, q_0, \rightarrow, \{q_m\}) \), a finite set of indices \( I \subseteq \mathbb{N}^2 \) and a set \( \Sigma' \subseteq \Sigma \) where

- \( \forall (i, j) \in I. \ 0 \leq i \leq j \leq m \),
- \( \forall (i, j) \in I. \ \forall (i', j') \in I. \ (i', j') \neq (i, j) \Rightarrow (j < i' \lor j' < i) \),
- \( Q \) is a finite set of states such that \( Q = \{q_0, q_1, \ldots, q_m\} \cup \bigcup_{(i,j) \in I} P_{i,j} \) with \( P_{i,j} = \{p_{i,j}', \ldots, p_{i,j}^{(j-i)}\} \),
• $q_0$ is the initial state,
• $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions defined as the smallest set such that:
  1. $\forall i \in \{0, \ldots, m-1\}, \exists a \in \Sigma. q_i \xrightarrow{a} q_{i+1}$.
  2. $\forall (i,j) \in I$, if $P_{i,j} \neq \emptyset$ then
     $\exists a \in \Sigma. q_j \xrightarrow{a} p_{i,j}$,
  3. $\forall a \in \Sigma. p_{i,j}^k \xrightarrow{a} p_{i,j}^{k+1}$,
  4. $\forall a \in \Sigma. q_{j-1} \xrightarrow{a} q_j$.
• $q_m$ is the (unique) final state.

Example 3.2. A simple automaton

![](image)

The set of indices associated with the automaton is $I = \{(0,1),(2,2),(4,4)\}$ and $\Sigma' = \{a,b\}$. The automaton accepts the language $a(aa)^*b(ab)^*ab(a+b)^*$.

Notice that the outdegree of the states $q_i$'s, except maybe $q_m$, is at most 2 whereas the outdegree of the states in the $P_{i,j}$ is always 1.

Definition 3.3. Let $(i,j) \in I$. The loop defined by $(i,j)$ is the sequence of transitions

$(q_i, b_{i+1}, q_{i+1}) \cdots (q_{j-1}, b_j, q_j) (q_j, a_0, p_{i,j}^1) (p_{i,j}^1, a_1, p_{i,j}^2) \cdots (p_{i,j}^l, a_l, q_i)$. We say that $q_i$ is the root of the loop, and that the loop forms the word $b_{i+1} \ldots b_j a_0 \ldots a_l$. A simple loop is a loop defined by a pair of the form $(i,i)$.

Each state different from $q_m$ belongs to at most one loop. The state $q_m$ has a particular status since it may be the root of several loops, but in this case all these loops are self-loops.

Now, we introduce a subclass of simple automata, called restricted simple automata, by giving the state $q_m$ the same status as the other $q_i$'s.

Definition 3.4. A restricted simple automaton (RSA) is a simple automaton such that points (3) and (4) in Definition 3.1 are replaced by

$(3') \forall (i,j) \in I$, if $P_{i,j} = \emptyset$ then $\exists a \in \Sigma$ with $q_j \xrightarrow{a} q_i$.

Notice that for RSAs the set $\Sigma'$ used in the definition of SAs is not needed.
Example 3.5. A restricted simple automaton:

![Diagram of an automaton]

The set of indices associated with the automaton is \( I = \{(0,1), (2,2), (3,4)\} \). The automaton accepts the language \( a(aa)^*b(ab)^*ab(bab)^* \).

It can be seen that RSAs accept languages over \( \Sigma \) which are definable by regular expressions of the form \( u_1v_1^*u_2v_2^* \cdots u_mv_m^*u_{m+1} \) where the \( u_i \)'s and the \( v_i \)'s are words over \( \Sigma \) such that only \( u_1 \) and \( u_{m+1} \) may be empty. SAs accept words of the form \( u_1v_1^*u_2v_2^* \cdots u_m(a_1 + \cdots + a_r)^* \) where the \( a_i \)'s are symbols in \( \Sigma \).

Notation 3.1. We write DSA (resp. DRSA) for deterministic SA (resp. RSA).

Notice that in simple automata, as well as in restricted simple automata, nondeterministic choices may occur only at the states \( q_i \).

Finally, we define the characteristic formula of a simple automaton. This formula characterizes the set of valuations corresponding to all accepting runs of \( A \). Let us first introduce some notations.

Notation 3.2. Let \( A = (Q, q_0, \rightarrow, \{q_m\}) \) be a simple automaton. Let \( X_A \) be the set of variables \( \{x_t : t \in \rightarrow\} \). Given a run \( \rho \) of \( A \), we denote by \( \nu_\rho \) the valuation of the variables in \( X_A \) such that, for every \( x_t \in X_A \), \( \nu_\rho(x_t) = |\rho_t| \), where \( |\rho_t| \) is the number of times the transition \( t \) appears in the run \( \rho \). For each \( q \in Q \), we denote by \( q^+ \) (resp. \( q^- \)) the set of transitions of the form \( (q', a, q) \) (resp. \( (q, a, q') \)) for some \( q' \in Q \) and \( a \in \Sigma \).

Definition 3.6. Let \( A \) be a simple automaton. The characteristic formula of \( A \), denoted by \( [A] \), is the Presburger formula given by

\[
\left( \bigwedge_{q \in Q \setminus \{q_0, q_m\}} \sum_{t \in q^+} x_t = \sum_{t \in q^-} x_t \right) \land 
\left( 1 + \sum_{t \in q_0^+} x_t = \sum_{t \in q_0^-} x_t \right) \land 
\left( \sum_{t \in q_m^+} x_t = 1 + \sum_{t \in q_m^-} x_t \right)
\]

(1)

It can be easily seen that the following fact holds:

Proposition 3.7. For every valuation \( \nu \) of the variables in \( X_A \), \( \nu \) satisfies \( [A] \) if and only if there exists an accepting run \( \rho \) of \( A \) such that \( \nu = \nu_\rho \).
The formula \([A]\) corresponds to a system of flow equations expressing the fact that the number of times a run takes a transition leading to a state is equal to the number of times this run takes a transition leaving that state, except for the initial and the final states for which there is a difference of one.

Clearly, if \(A\) has an accepting run \(\rho\), then \(v_\rho\) satisfies \([A]\). The other direction holds due to the fact that the structure of simple automata is a sequence of loops. In a more general case, where loops may appear in different branches of the automaton, it would be necessary to express in addition the fact that a solution of the flow equations corresponds to a same path from the initial to the final state.

3.3. Constrained queue-content decision diagrams

We define in this section the constrained queue-content decision diagrams. These structures are obtained by combining deterministic restricted simple automata with linear arithmetical constraints expressed in Presburger arithmetics. They are particular cases of the more general classes of constrained simple automata and deterministic constrained simple automata. Let us define first these two classes.

**Definition 3.8.** For any \(n \geq 1\), an \(n\)-dim constrained simple automaton (CSA) is a set of accepting components \(\mathcal{C} = \{\langle \mathcal{A}_1, f_1 \rangle, \ldots, \langle \mathcal{A}_m, f_m \rangle \}\) where, for every \(i \in \{1, \ldots, m\}\),

- \(\mathcal{A}_i\) is a tuple of \(n\) simple automata \((A_1, \ldots, A_n)\) over \(\Sigma\),
- \(f_i\) is a Presburger formula over a set of variables \(V_i\) containing the set \(X_i = \{x_t: t \in T_i\}\), where \(T_i\) is the set of all the transitions of the automata in \(\mathcal{A}_i\) i.e., \(T_i = \bigcup_{j=1}^{n} \rightarrow_j\).

The CSA \(\mathcal{C}\) accepts an \(n\)-dim multi-language; i.e., a set of tuples of \(n\) words. For every \(i \in \{1, \ldots, m\}\), the multi-language of the accepting component \(\langle \mathcal{A}_i, f_i \rangle\), denoted by \(L(\langle \mathcal{A}_i, f_i \rangle)\) is the set of tuples of words \((w_1, \ldots, w_n) \in (\Sigma^*)^n\) for which there are accepting runs \((\rho_1, \ldots, \rho_n)\) of the automata \((A_1, \ldots, A_n)\) respectively, such that \(\exists (V_i \setminus X_i)\cdot f_i\) is satisfied by the valuation \((v_1, \ldots, v_n)\) (i.e., the valuation associating with each variable \(x_t\) the integer \(|\rho_1 \ldots \rho_n|_t\), where \(t \in T_i\). The multi-language of the CSA \(\mathcal{C}\), denoted by \(L(\mathcal{C})\), is the union \(\bigcup_{i=1}^{m} L(\langle \mathcal{A}_i, f_i \rangle)\).

Intuitively, for each accepting component \(\langle \mathcal{A}_i, f_i \rangle\) the formula \(f_i\) puts constraints on the multi-words accepted by the vector of automata \(\mathcal{A}_i\). These constraints concern the numbers of times transitions of \(\mathcal{A}_i\) are used to accept the multi-words.

**Notation 3.3.** We use \(tt\) as an abbreviation of the formula \(\bigwedge_{t \in T} x_t \geq 0\), i.e., the formula which imposes no constraints.

**Definition 3.9.** An \(n\)-dim CDSA is an \(n\)-dim CSA such that all its automata are deterministic (DSAs). An \(n\)-dim constrained queue-content decision diagram (CQDD) is an \(n\)-dim CDSA such that all its automata are restricted (DRSAs).

**Notation 3.4.** For every \(n \geq 1\), we denote by \(n\text{-CQDD}\) (resp. \(n\text{-CDSA}\)) the class of all \(n\)-dim \(n\text{-CQDDs}\) (resp. \(n\)-dim \(n\text{-CDSSAs}\)).
Definition 3.10. We say that an $n$-dim multi-language $\mathcal{L}$ is CQDD (resp. CDSA) definable if there exists an $n$-dim CQDD (resp. CDSA) $\mathcal{C}$ such that $L(\mathcal{C}) = \mathcal{L}$. An $n$-dim multi-language $\mathcal{L}$ is CQDD (resp. CDSA) reverse definable if its reverse image, denoted by $\mathcal{L}^R$, is CQDD (resp. CDSA) definable.

CQDDs allow to define nonregular multi-languages. For instance, consider the context-sensitive language $L_1 = \{a^n b^m d^n b^m : n, m \geq 1\}$. To define $L_1$, we use the automaton $A_1$ represented by the following picture:

![Diagram of automaton A1](image)

Then, $L_1$ is defined by the 1-dim CQDD $\{(A_1, f_1)\}$ where $f_1$ is given by:

$$x_{(q_1, a, q_1)} = x_{(q_3, a, q_3)} \land x_{(q_2, b, q_2)} = x_{(q_4, b, q_4)}.$$

Now, let $L_2 = \{(a^n b^m d^n a^m, c^m d^n a^m) : n, m \geq 1\}$ be a 2-dim multi-language. To define this multi-language, we use two automata, the automaton $A_1$ above and the automaton $A_2$ given by the following picture:

![Diagram of automaton A2](image)

Then, $L_2$ is defined by the 2-dim CQDD $\{(A_1, A_2, f_2)\}$ where $f_2$ is given by:

$$x_{(q_1, a, q_1)} = n \land x_{(q_3, a, q_3)} = n \land x_{(q_2, d, q_2)} = n \land x_{(q_4, b, q_4)} = m \land x_{(q_1, c, q_1)} = n \land x_{(q_3, a, q_3)} = m.$$

Hence, as it can be seen from these examples, CQDDs allow to express nonregular multi-languages involving constraints on number of occurrences of symbols at some positions that may be in a same word (as in $L_1$), or even in different words (as in $L_2$). This allows to represent sets of queue contents such that there are counting constraints relating the contents of different queues.

3.4. Basic operations

In this section, we address the question whether the class of CQDD definable sets is closed under the boolean operations. We show that it is closed under union and intersection, but the complementation of a CQDD set is a CDSA set. However, the
intersection of a CQDD set with a CDSA is a CQDD set. This means that the inter-
section of a CQDD set with the complement of another CQDD set is always a CQDD
set.

Moreover, we show that the class of CQDD sets is closed under concatenation and left-quotient. These two operations are essential in the constructions of the successors and predecessors of sets of configurations represented by means of CQDDs.

The proofs of these results are rather involved because all the operations modify the structure of the automata while the arithmetical constraints in CQDDs depend strongly on this structure since they are expressed on the transitions of the automata. To simplify the presentation of the proofs, they always begin with the case of 1-dim CQDDs with a single accepting component. Then, we give the generalization to n dimensional CQDDs. The generalization to several accepting components is straightforward.

3.4.1. Closure under intersection and union

**Proposition 3.11.** For every \( n \geq 1 \), the intersection of an \( n \)-CQDD with an \( n \)-CDSA is an \( n \)-CQDD.

**Proof.** Let \( \mathcal{C}_1 = \{ (A_1, f_1(V_1)) \} \) be a CQDD and \( \mathcal{C}_2 = \{ (A_2, f_2(V_2)) \} \) be a CDSA where, for \( i \in \{1, 2\} \), \( V_i \) is a superset of \( X_i = \{ x_t: t \in \rightarrow_i \} \).

The product of two finite-state automata is defined as usual. Let us denote by \( A_1 \times A_2 \) the automaton constructed using the product of \( A_1 \) and \( A_2 \) and which accepts the language \( L(A_1) \cap L(A_2) \) (its unique final state is the pair of the final states of \( A_1 \) and \( A_2 \)). It is easy to see that \( A_1 \times A_2 \) is a deterministic restricted simple automaton (by cutting all the useless states and transitions). Each transition in \( A_1 \times A_2 \) is composed of a transition \( t_1 \) in \( A_1 \) and a transition \( t_2 \) in \( A_2 \), and let us denote by \( (t_1, t_2) \) this composed transition. Then, it is easy to check that \( L(\mathcal{C}_1) \cap L(\mathcal{C}_2) \) is accepted by the CQDD:

\[
\mathcal{C}_1 \times \mathcal{C}_2 = \{ (A_1 \times A_2, f_{1,2}(X_{1,2} \cup V_1 \cup V_2)) \},
\]

where \( X_{1,2} = \{ x_{(t_1,t_2)}: t_1 \in \rightarrow_1 \land t_2 \in \rightarrow_2 \} \) and the formula \( f_{1,2} \) is given by

\[
f_{1} \land f_{2} \land \bigwedge_{t_1 \in \rightarrow_1} \left( x_{t_1} = \sum_{t_2 \in \rightarrow_2} x_{(t_1,t_2)} \right) \land \bigwedge_{t_2 \in \rightarrow_2} \left( x_{t_2} = \sum_{t_1 \in \rightarrow_1} x_{(t_1,t_2)} \right).
\]

**Generalization to n dimensions:** Let \( \mathcal{C}_1 = \{ ((A_1, \ldots, A_i^n), f_i(V_i)) \} \) and \( \mathcal{C}_2 = \{ ((A_2, \ldots, A_2^n), f_2(V_2)) \} \) where for \( i \in \{1, 2\} \), \( V_i \) is a superset of \( X_i = \{ x_t: t \in \bigcup_{j=1}^n \rightarrow_j \} \). Then, \( L(\mathcal{C}_1) \cap L(\mathcal{C}_2) \) is accepted by the CQDD:

\[
\mathcal{C}_1 \times \mathcal{C}_2 = \{ ((A_1 \times A_2^1, \ldots, A_1^n \times A_2^n), f_{1,2}(X_{1,2} \cup V_1 \cup V_2)) \},
\]
where \( X_{1,2} = \bigcup_{j=1}^{n} \{ x(t_i, t'_i) : t_i \in \rightarrow i \land t'_i \in \rightarrow j \} \) and the formula \( f_{1,2} \) is given by

\[
f_{1,2} = \bigwedge_{j=1}^{n} \left( \bigwedge_{t_i \in \rightarrow i} x(t_i, t'_i) = \sum x(t_i, t'_i) \land \bigwedge_{t'_i \in \rightarrow j} x(t_i, t'_i) = \sum x(t'_i, t_i) \right). \]

The closure under union is obvious and \( n \)-CQDDs are special cases of \( n \)-CDSAs. Hence, we obtain the following proposition:

**Proposition 3.12.** For every \( n \geq 1 \), the class \( n \)-CQDD is closed under union and intersection.

### 3.4.2. Closure under complement

To prove closure under complement we need a simple technical lemma:

**Lemma 3.13.** Let \( A \) be a DSAS \( A \). Then, there exist a \( k \in \mathbb{N} \) and DSASs \( A_1, \ldots, A_k \) such that \( \Sigma^* \setminus L(A) = \bigcup_{i=1}^{k} L(A_k) \).

**Proof.** We use the classical construction for complementation of automata. To obtain a union of DSAS we modify it a little by adding a state for each symbol of the alphabet and each state which has no outgoing transition with this symbol. Let \( A = (Q, q_0, \rightarrow, \{ q_m \}) \). We construct the automaton \( A^c = (Q^c, q_0^c, \rightarrow^c, F) \) which recognizes \( \Sigma^* \setminus L(A) \) as follows:

- \( Q^c = Q \cup \{ q^c_{a,b} \mid q \in Q \wedge a \in \Sigma \wedge \exists q' \in Q : q \rightarrow a q' \} \),
- \( q_0^c = q_0 \),
- \( \rightarrow^c \) is the smallest set which satisfies
  - \( \rightarrow \subseteq \rightarrow^c \)
  - \( \forall b \in \Sigma : q^c_{a,b} \rightarrow^c q^c_{a,b} \)
  - \( \forall q \in Q^c : q \rightarrow^{c^*} q_{q,a} \)
- \( F = Q^c \setminus \{ q_m \} \)

Finally we obtain for each state in \( F \) an automaton \( A_i \) which recognizes the language accepted by this accepting state. This automaton is perhaps not a DSA, but we can cut all the transitions which lead to states from which the final state is not reachable. We obtain a DSA. \( \square \)

**Example 3.14.** Let \( \Sigma = \{ a, b \} \) and \( A \) be given by

- \( a \)
- \( b \)
- \( s \)
- \( p \)
- \( r \)
Then, an automaton which accepts $\Sigma^* \setminus L(A)$ is given by

![Automaton Diagram]

This automaton can be decomposed easily into five deterministic simple automata.

**Proposition 3.15.** For every $n \geq 1$, the complement of an $n$-CQDD is an $n$-CDSA.

**Proof.** Let us consider the one-dimensional CQDD $\mathcal{G} = \{ \langle A, f(V) \rangle \}$ where $V$ is a superset of $X = \{ x_t: t \in \mathbb{X} \}$. With Lemma 3.13 the complement of a deterministic restricted simple automaton is a union of deterministic simple automata, i.e., $\overline{L(A)}$ can be written as $\bigcup_{i=1}^{m} L(A_i)$ where every $A_i$ is a deterministic simple automaton. Let $\rightarrow_i$ be the transition relation of $A_i$ and let $X_i = \{ x_t: t \in \rightarrow_i \}$. Then, clearly, the language $L(\{ \mathcal{G} \})$ is accepted by the CDSA:

$$\overline{\mathcal{G}} = \{ \langle A_i, \rightarrow_i \rangle: 1 \leq i \leq m \} \cup \{ \langle A, \neg \exists(V \setminus X). f \rangle \}.$$

Here it is important that the automaton $A$ is deterministic, i.e., every word accepted by $A$ has exactly one run on $A$.

**Generalization to $n$ dimensions:** A multi-word is not in the language of a given $n$-CQDD, if at least one component of a multi-word is not accepted by the automaton for this component, or all the components of the multi-word are accepted but they do not satisfy the constraints. Formally, let $\mathcal{G} = \{ \langle A^1, \ldots, A^n, f(V) \rangle \}$ where $V$ is a superset of $X = \{ x_t: t \in \bigcup_{j=1}^{n} \rightarrow_j \}$. $L(A^j)$ can be written as $\bigcup_{i=1}^{m} L(A^j_i)$ where each $A^j_i$ is a DSA. Let $\rightarrow_j^i$ be the transition relation of $A^j_i$ and let $X_i = \{ x_t: t \in \bigcup_{j=1}^{n} \rightarrow_j^i \}$. Then, the language $L(\{ \mathcal{G} \})$ is accepted by the CDSA:

$$\overline{\mathcal{G}} = \{ \langle B^1, B^2, \ldots, B^n, \rightarrow \rangle: (\forall k \in \{ 1, \ldots, n \} \cdot (B^k = A^k \lor \exists j \in \{ 1, \ldots, m_j \}. B^k = A^j_i))$$

$$\land \exists k \in \{ 1, \ldots, n \}. (B^k \neq A^k) \} \cup \{ \langle A^1, \ldots, A^n, \neg \exists(V \setminus X). f \rangle \}.$$

$\square$
3.4.3. Closure under concatenation

In this section we prove that CQDDs are closed under concatenation.

**Definition 3.16.** Let $L_1$ and $L_2$ be two $n$-dim multi-languages. The concatenation of $L_1$ and $L_2$, denoted by $L_1 \cdot L_2$, is the set $\{w \in (\Sigma^*)^n : \exists u \in L_1 \cdot \exists v \in L_2 \cdot w = uv\}$.

The concatenation of two deterministic restricted simple automata $A_1$ and $A_2$ can give a non-deterministic simple automaton. We can determinize this automaton and then the problem is that we have to define a constraint which relates the variables associated to transitions of this automaton with variables associated with transitions of $A_1$ respectively $A_2$. Let us illustrate this problem with a simple example. Consider the CQDDs $\mathcal{G}_1 = \{(A_1, f_1(V_1))\}$ and $\mathcal{G}_2 = \{(A_2, f_2(V_2))\}$ where, for $i \in \{1, 2\}$, $A_i = (Q_i, q_i^{\text{init}}, \tau, q_i^{\text{fin}})$, and $V_i$ is a superset of $X_i = \{x_i : t \in \tau\}$. Let $A_1$ be the automaton

- $A_1$ is a simple automaton with states 0, 1, 2, and transitions labeled with symbols a and b.

- $A_2$ is a simple automaton with states 3, 4, 5, 6, and transitions labeled with symbols a, b, and c.

If we concatenate the two automata (by identifying the final state of $A_1$ with the initial state of $A_2$) and determinize with the usual subset construction we obtain the following automaton $A_d$ where each transition is a subset of transitions of $A_1$ and $A_2$. Notice, that several subsets (for example $\{2, 6\}$) appear twice:
Let \( X_d = \{ x_t : t \in \mathbb{N} \} \). To define \( L(C_{g1}) \cdot L(C_{g2}) \) we have to associate with \( A_d \) a formula \( f_d(V_d) \) (where \( V_d \) is a superset of \( X_d \)). Given a word \( w \in L(A_d) \) this formula should express the fact that there exist \( w_1 \in L(A_1) \) and \( w_2 \in L(A_2) \) such that \( w = w_1 \cdot w_2 \) and the valuation induced by the run of \( w_1 \) (resp. \( w_2 \)) on \( A_1 \) (resp. \( A_2 \)) satisfies \( f_1 \) (resp. \( f_2 \)). Hence, the formula \( f_d \) should relate the variables in \( X_d \) with the variables in \( X_1 \) and \( X_2 \). We need a systematic way to define this relation. This is difficult in general because

- a word \( w \in A_d \) can perhaps be split in different ways into two words \( w_1 \) and \( w_2 \), with \( w = w_1 \cdot w_2 \), \( w_1 \in L(A_1) \) and \( w_2 \in L(A_2) \),
- transitions of \( A_d \) can be labelled with the same subset of transitions of \( A_1 \) and \( A_2 \),
- a transition of \( A_1 \) or \( A_2 \) can appear several times in transitions of \( A_d \).

In the following we give several technical lemmas which show essentially that we can transform the automata \( A_1 \) and \( A_2 \), such that the derived automaton \( A_d \) has a structure which allows to define the correspondence between \( X_1 \), \( X_2 \) and \( X_d \) in an easy way.

**Lemma 3.17.** Let \( \mathcal{C} = \{ (A, f(V)) \} \) be a CQDD such that the final state of \( A \) is in a loop. Then, there exists a CQDD \( \mathcal{C}' = \{ (A', f'(V')) \} \) with \( L(\mathcal{C}) = L(\mathcal{C}') \) and the final state of \( A' \) is in a simple loop.

**Proof.** Let \( I \) be the set of indices associated with \( A = (Q, q_0, \rightarrow, \{ q_m \}) \). If the last loop of \( A \) is not simple, then there exists \( (i, m) \in I \) such that the states \( q_i, \ldots, q_m \) are in the last loop. An accepting run of \( A \) visits at least once these states. Let \( A'' \) be the automaton constructed from \( A \) by unfolding the last loop up to \( q_m \), i.e. the states \( q_i, \ldots, q_{m-1} \) are copied and the transition relation modified such that the first visit of these states is outside the new loop and the others inside. Now, the final state of \( A'' \) is in a simple loop. Obviously \( L(A) = L(A'') \). Let \( \mathcal{C}' = \{ (A', f'(V')) \} = \mathcal{C} \times \{ (A'', tt) \} \). Since we impose no constraint on \( A'' \), \( L(\mathcal{C}) = L(\mathcal{C}') \). Notice that the final state of \( A' \) is in a simple loop and that the product modifies the constraint \( f \). \( \square \)

We show that each CQDD can be decomposed into a finite union of CQDDs whose automata have an arbitrary large last loop.

**Lemma 3.18.** Let \( \mathcal{C} = \{ (A, f(V)) \} \) be a CQDD such that the final state of \( A \) is in a loop. Let \( w \) be the word formed by this loop. Then, \( \forall n \in \mathbb{N} \), \( \exists i \in \mathbb{N} \) with \( 0 \leq i < n \) there exist CQDDs \( \mathcal{C}_i = \{ (A_i, f_i(V_i)) \} \) such that the last loop of \( A_i \) is simple, contains the final state and forms the word \( w^i \). Furthermore, \( L(\mathcal{C}) = \bigcup_{i=0}^{n-1} L(\mathcal{C}_i) \).

**Proof.** Let \( \mathcal{C} = \{ (A, f(V)) \} \) be a CQDD with \( A = (Q, q_0, \rightarrow, \{ q_m \}) \) and a set of indices \( I \) such that the final state is in a loop which forms the word \( w \). With Lemma 3.17 we can suppose that this loop is simple. Let \( n \in \mathbb{N} \). Starting from \( A \) we construct \( n \) automata \( A_i' \) such that \( L(A) = \bigcup_{i=0}^{n-1} L(A_i') \) and such that the last loop of the \( A_i' \)'s forms the word \( w^n \). The automaton \( A_i' \) is obtained from \( A \) by unfolding the last loop \( i \) times and the
Lemma 3.19. Let $\mathcal{C} = \{ (A, f(V)) \}$ be a CQDD such that there exist $k_1, k_2 \in \mathbb{N}$ with $k_2 > k_1$ and $w \in \Sigma^+$ such that the first $k_1 |w|$ states of $A$ are not in a loop and form the word $w^{k_1}$ and the first loop of $A$ has $k_2 |w|$ states and forms the word $w^{k_2}$. Then there exists a CQDD $\mathcal{C}' = \{ (A', f'(V')) \}$ such that $L(\mathcal{C}) = L(\mathcal{C}')$ and the initial state of $A'$ is in a loop with size $k_2 |w|$ and forms the word $w^{k_2}$.

Proof. Let $A = (Q, q_0, \rightarrow, \{ q_m \})$ and let $I$ be the associated set of indices. Intuitively, we have to fold the sequence formed by the first $k_1 |w|$ states of $A$ on its first loop. Let $c_1 = k_1 |w|$ and $c_2 = k_2 |w|$. The automaton $A$ consists of three parts: First, $c_1$ states which are not in a loop. Second, a loop with $c_2$ states and third, the rest of the automaton (which can be empty). Either there is at most one transition $t$ from a state $q$ of the second part to the third or the second part contains the final state $q$. In these two cases, let $w'$ be the word formed by the transitions in $A$ from the initial state to $q$. For example,

Here we have $w' = w^5$. Let $t_1, \ldots, t_{c_1}$ be the first $c_1$ transitions of $A$ and $t'_{c_1}, \ldots, t'_{c_2}$ the $c_2$ transitions of the first loop of $A$ and $\mathcal{F}$ the set of all other transitions. $\mathcal{F}$ can be empty.

We construct an automaton $A'$ in the following way: The first part contains the initial state in a loop of size $c_2$ which forms the word $w^{k_2}$. Let $q'$ be the state obtained by unrolling $w'$ on this loop starting from the initial state. Since $|w'| < c_1 + c_2 < 2c_2$ the run of $w'$ on the loop passes at most once through the initial state.

- If the state $q$ of $A$ defined above is the final state of $A$, then the final state of $A'$ is $q'$.
- If there is a transition $t$ from the state $q$ in $A$ to the third part of $A$, then from $q'$ there is the same transition and the second part of the automaton $A'$ is a copy of part 3 of $A$. For example,
It is clear that $A'$ is a DRSA. Now, we have to define the formula $f'(V')$: A transition in the first loop of $A'$ corresponds either to a transition in the first part of $A$ or to a transition in the second part (the first loop) of $A$. Furthermore, we have to express the fact that the number of executions of transitions of $A$ corresponds to an accepting run in $A$ and that they satisfy $f$. This fact can be expressed by the constraint $f'(V')$.

Let $t'_1, \ldots, t'_2$ be the transitions of the first loop of $A'$ and $\mathcal{T}'$ the set of the other transitions. $f'(V')$ is given by

$$
f(V) \land [A] \land \bigwedge_{i=1}^{c_1} (x_{t'_i} = x_{t_i} + x_{t_i \mod c_2}) \land \bigwedge_{i=c_1+1}^{c_2} (x_{t'_i} = x_{t_i \mod c_2}) \land \bigwedge_{t \in \mathcal{T}} x_t = x_{t'}.
$$

In the example, we have among other constraints $x_{t'_i} = x_{t_i} + x_{t_i \mod 13}$.

Notice, that the constraint $f'$ can impose the fact, that the first loop of $A'$ has to be taken at least once. This corresponds to the case, where the unrolling of $w'$ on the loop passes the initial state (see example above).

It is easy to see that $L(\mathcal{C}) = L(\mathcal{C}')$. \[\]

Let $w \in \Sigma^+$. We show that each CQDD which satisfies a certain condition can be split into two sets. One, where for each automaton $A$ in the CQDD, $w^k$ is a prefix of $L(A)$ for all $k \in \mathbb{N}$ and a second, where for each automaton $A$ in the CQDD $w^k$ is a prefix of $L(A)$ for only finitely many $k \in \mathbb{N}$.

**Notation 3.5.** Let $L \subseteq \Sigma^*$. We denote by $\text{Pref}(L)$ the set of prefixes of words in $L$.

**Lemma 3.20.** Let $\mathcal{C} = \{ (A, f(V)) \}$ be a CQDD with $A = (Q, q_0, \rightarrow, \{ q_m \})$. Let $k_1, k_2 \in \mathbb{N}$ with $k_2 > k_1$ and $w \in \Sigma^+$ such that there exists $q_1 \in Q$ with $q_0 \xrightarrow{w^{k_1}} q_1 \xrightarrow{w^{k_2}} q_1$. Then there exist two CQDD $\mathcal{C}'$ and $\mathcal{C}''$ with $L(\mathcal{C}) = L(\mathcal{C}') \cup L(\mathcal{C}'')$ such that

- for all automata $A'$ in $\mathcal{C}'$ we have
  - the initial state is in a loop,
  - the loop with the initial state has size $k_2 | w |$ and forms the word $w^{k_2}$.
- for all automata $A''$ in $\mathcal{C}''$ we have $w^{k_1} \not\in \text{Pref}(L(A''))$.

**Proof.** First, notice that the language $\Sigma^*$ can be written as $L \cup \overline{L}$ where $L = w^{k_1} \cdot \overline{(w^{k_2})^*}$. Then, it is easy to see that $L$ can be expressed as the union of languages of deterministic simple automata $A_1', \ldots, A_n'$ such that the first $k_1 | w |$ states of the automata are not in a loop and form the word $w^{k_1}$. Furthermore, the first loop has size $k_2 | w |$ and forms the word $w^{k_2}$. Then, the product of two automata $A$ and $A'_i$ is an automaton such that the first $k_1 | w |$ states of the automaton are not in a loop and form the word $w^{k_1}$. Furthermore, the first loop has size $k_2 | w |$ and forms the word $w^{k_2}$. Then, with Lemma 3.19 we can transform the CQDD $\mathcal{C}' = \mathcal{C} \times \{ (A_i', tt) : 1 \leq i \leq n' \}$ such that the automata have the appropriate form. $\overline{L}$ can be written as a union of languages of deterministic simple automata $A_1'', \ldots, A_n''$. Then, $\mathcal{C}''$ is given by $\mathcal{C} \times \{ (A_i'', tt) : 1 \leq i \leq n'' \}$. \[\]
Lemma 3.21. Let $\mathcal{C} = \{\langle A, f(V) \rangle \}$ be a CQDD and $k \in \mathbb{N}$. Then there exists a CQDD $\mathcal{C}'$ such that $L(\mathcal{C}) = L(\mathcal{C}')$ and all the automata in $\mathcal{C}'$ have either less than $k$ states or the first $k$ states are not in a loop.

Proof. It is easy to see that $\Sigma^*$ can be written as a finite union of languages of deterministic simple automata $A_1, \ldots, A_n$ such that the automata $A_i$ have either less than $k$ states or the first $k$ states are not in a loop. Then, $\mathcal{C}'$ is given by $\bigcup_{i=1}^n (\mathcal{C} \times \{ \langle A_i, f_i \rangle \})$.

Now, we are ready to prove the following fact:

Proposition 3.22. For every $n \geq 1$, the class $n$-CQDD is closed under concatenation.

Proof. Consider the CQDDs $\mathcal{C}_1 = \{\langle A_1, f_1(V_1) \rangle \}$ and $\mathcal{C}_2 = \{\langle A_2, f_2(V_2) \rangle \}$ where, for $i \in \{1, 2\}$, $A_i = (Q_i, q_i^{\text{init}}, \rightarrow_i, q_i^{\text{fin}})$, and $V_i$ is a superset of $X_i = \{x_i: t \rightarrow_i \}$. We show that we can construct a CQDD which accepts $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$. If the final state of $A_1$ is not in the loop, we just glue the two automata (this gives a deterministic automata) and take the conjunction of the two formulae. If the final state is in a loop, the automaton which we obtain by gluing $A_1$ and $A_2$ together could be non-deterministic. Determinization modifies its structure and we have to redefine the constraints. In the following we use the preceding lemmas to suppose that $\mathcal{C}_1$ and $\mathcal{C}_2$ have a special form. Let $w$ be the word given by the loop containing the final state of $A_1$. We consider two cases:

Case 1: $\exists k \in \mathbb{N}, \forall k', k w k' \not\in \text{Pref}(L(A_2))$

Notice that this condition can be easily checked on the automaton $A_2$. Then, we can suppose with Lemma 3.18 that

1. the last loop of $A_1$ is simple, forms the word $w^k$ and has size $k|w|$.

Furthermore we can suppose with Lemma 3.21 that

2. either $A_2$ has no loop and less than $k|w|$ states or the first $k|w|$ states are not in a loop.

Now, the concatenation of the two DRSA $A_1$ and $A_2$ is a RSA (it suffices to glue the state $q_1^{\text{fin}}$ with the state $q_2^{\text{init}}$). Let $A_1 \cdot A_2$ be the so obtained automaton. Since CQDD's are defined using deterministic automata, we have to determinize $A_1 \cdot A_2$. For that, we use the standard subset construction. We call the automaton obtained by this construction $A_d$. Each state of $A_d$ is a subset of states of $A_1$ and $A_2$. The transitions of $A_d$ are tagged by subsets of $\rightarrow_1 \cup \rightarrow_2$. Because of conditions (1) and (2) $A_d$ consists of three parts:

- Part 1: A copy of $A_1$ without the last loop. Here all the transitions are tagged by one transition of $A_1$.

- Part 2: Each state of this part is either a set with one state of $A_1$ or a subset of $A_1$ and one of the first $k|w|$ states of $A_2$. This part forms a loop with root $\{q_1^{\text{init}}\}$. Because of $w^k \not\in \text{Pref}(L(A_2))$ there is at least one transition in the loop tagged only by one transition of $A_1$, because at some point $A_2$ will "disagree" with the last loop of $A_1$.

- Part 3: A copy of the rest of $A_2$ (can be empty if $A_2$ has no loop). Here all the transitions are tagged by one transition of $A_2$. 

There is exactly one transition from part 1 to part 2, as well as from part 2 to part 3. Hence, the automaton $A_d$ is a DRSA. We illustrate this with a schematic example: Let $A_1$ be

\[
\text{Rest of } A_1
\]

and $A_2$:

\[
\text{Part 1} \quad \text{Part 2} \quad \text{Part 3}
\]

The remaining problem is to define a formula for this automaton. Let $\rightarrow_d$ be the set of transitions of $A_d$ and $X_d = \{x_t: t \in \rightarrow_d\}$. For a transition $t \in \rightarrow_d$ let us denote by $\hat{\lambda}(t)$ the associated set of transitions in $A_1$ and $A_2$. Because of conditions (1) and (2) each transition of $A_1$ and $A_2$ appears at most once in a transition of $A_d$. Therefore, an execution of a transition $t$ in $A_d$ corresponds to the execution of one of the transitions in $\hat{\lambda}(t)$. This fact imposes a constraint expressed by the formula $f_d(V_1 \cup V_2 \cup X_d)$:

\[
\bigwedge_{t \in \rightarrow_d} \left( x_t = \sum_{t' \in \hat{\lambda}(t)} x_{t'} \right).
\]

In addition to that, we have to express the fact, that the number of executions of the transitions in $A_1$ and $A_2$ corresponds to an accepting run in $A_1$ and $A_2$ and that they must satisfy $f_1 \land f_2$. Then, the constraints are given by the formula $f_{\text{set}}(V_1 \cup V_2 \cup X_d)$:

\[
[A_1] \land [A_2] \land f_1 \land f_2 \land f_d(V_1 \cup V_2 \cup X_d)
\]
and it is easy to check that the language \( L(\mathcal{G}_1) \cdot L(\mathcal{G}_2) \) is accepted by the CQDD:

\[
\mathcal{G}_1 \cdot \mathcal{G}_2 = \{ (A_d, f_{det}) \}.
\]

**Case 2:** \( \forall n \in \mathbb{N}: w^n \in \text{Pref}(L(A_2)) \)

Again, this condition can be easily checked on the automaton \( A_2 \). Since \( A_2 \) is a deterministic simple automaton, there exist necessarily \( k_1, k_2 \in \mathbb{N} \) with \( k_2 > k_1 \) and \( q_1 \in Q_2 \) such that \( q_{\text{init}} \overset{w^{k_1}}{\rightarrow} q_1 \overset{w^{k_2}}{\rightarrow} q_1 \) (in order to guarantee \( k_2 > k_1 \), the path \( q_1 \overset{w^{k_2}}{\rightarrow} q_1 \) could correspond to repetitions of a cycle). Because of Lemma 3.20 we can partition \( \mathcal{G}_2 \) into two CQDDs:

- \( \mathcal{G}_2'' \) where for all automata \( A'' \) we have \( w^{k_1} \not\in \text{Pref}(L(A'')) \). For these CQDD's we can reason as in Case 1.
- \( \mathcal{G}_2' \) where all automata involved have a special form.

We can suppose that

1. the initial state of \( A_2 \) is in a loop which forms the word \( w^{k_2} \) and has size \( k_2|w| \).
   Furthermore we can suppose (Lemma 3.18) that
2. the last loop of \( A_1 \) is simple, forms the word \( w^{k_2} \) and has size \( k_2|w| \).

As in case 1, we construct the automaton \( A_d \) and define \( -\rightarrow_d, X_d \) and \( \lambda(t) \). \( A_d \) consists of three parts:

- **Part 1:** A copy of \( A_1 \) without the last loop. All the transitions are tagged by one transition of \( A_1 \).
- **Part 2:** consists of the last loop of \( A_1 \) together with the first loop of \( A_2 \) (the two loops have exactly the same form). Each state is a pair with one state of the last loop \( A_1 \) and one state of the first loop of \( A_2 \).
- **Part 3:** consists of the rest of \( A_2 \) (may be empty if the final state of \( A_2 \) is in the first loop).

There is exactly one transition from part 1 to part 2 as well as from part 2 to part 3 (if it is not empty). Hence, \( A_d \) is a DRSA. We illustrate this with a schematic example:

Let \( A_1 \) be

\[
\begin{array}{c}
\text{Rest of } A_1 \\
\end{array}
\]

and \( A_2 \):

\[
\begin{array}{c}
\text{Rest of } A_1 \\
\end{array}
\]
Then $A_d$ has the form

\[ \begin{aligned} \text{Part 1} & \quad \text{Part 2} & \quad \text{Part 3} \\ \vdots \end{aligned} \]

Each transition of $A_1$ and $A_2$ appears at most once in a transition of $A_d$. As in case 1 we define $f_d(V_1 \cup V_2 \cup X_d)$:

\[ \bigwedge_{t \in \rightarrow_d} \left( x_t = \sum_{t' \in \Delta(t)} x_{t'} \right) \]

and $f_{\text{det}}(V_1 \cup V_2 \cup X_d)$:

\[ [A_1] \land [A_2] \land f_1 \land f_2 \land f_d(V_1 \cup V_2 \cup X_d). \]

Then, the language $L(c_{g1}) \cdot L(c_{g2})$ is accepted by the CQDD:

\[ \mathcal{C}_1 \cdot \mathcal{C}_2 = \{(A_d, f_{\text{det}})\}. \]

**Generalization to n dimensions:** We can suppose for each dimension that the two automaton $A_1$ and $A_2$ have the special form required. Then we construct one accepting component. The constraint of this component is a conjunction of the constraints for each dimension.

3.4.4. Closure under left-quotient

**Definition 3.23.** Let $L_1$ and $L_2$ be n-dim multi-languages. The left-quotient of $L_1$ by $L_2$, denoted by $L_2^{-1} \cdot L_1$, is the set $\{w \in (\Sigma^*)^n : \exists w' \in L_2, w' w \in L_1\}$; i.e., the set of multi-words allowing to extend elements of $L_2$ to elements of $L_1$.

**Proposition 3.24.** For every $n \geq 1$, the class n-CQDD is closed under left-quotient.

**Proof.** Consider the CQDDs $\mathcal{C}_1 = \{(A_1, f_1(V_1))\}$ and $\mathcal{C}_2 = \{(A_2, f_2(V_2))\}$ where, for $i \in \{1, 2\}$, $A_i = (Q_i, q_i^{\text{init}}, \rightarrow_i, q_i^{\text{fin}})$, and $V_i$ is a superset of $X_i = \{x_t: t \in \rightarrow_i\}$. We show that there is a CQDD which accepts $L(\mathcal{C}_2)^{-1} \cdot L(\mathcal{C}_1)$.

We start by considering the languages $L(A_2) \cap \text{Pref}(L(A_1))$ and $L(A_2)^{-1} \cdot L(A_1)$. For that, we construct the product of the automata $A_1$ and $A_2$. Let $F$ be the set of states $q \in Q_1$ such that $(q, q_{\text{fin}}^2)$ is a state in $A_1 \times A_2$.

Then, for each $q \in F$, consider the automaton $A_{1,2}^{\text{q}}$ which consists of a copy of $A_1 \times A_2$ with $(q_0^1, q_0^2)$ as initial state and $(q, q_{\text{fin}}^2)$ as final state, and let $A_{1}^{\text{q}}$ be a copy
of $A_1$ such that the initial state is $q$. These automata are all simple (by cutting all the non-reachable states and transitions).

Clearly, we have

$$\bigcup_{q \in F} L(A_{1,2}^q) = L(A_2) \cap \text{Pref}(L(A_1))$$

and

$$\bigcup_{q \in F} L(A_1^q) = L(A_2)^{-1} \cdot L(A_1).$$

Now, to obtain the language $L(\mathcal{C}_2)^{-1} \cdot L(\mathcal{C}_1)$, we have to define the constraints on the words accepted by the automata $A_1^2$. Let $\mathcal{F}(A_1^2)$ be the set of transitions of $A_1^2$.

We have $\mathcal{F}(A_2^q) \subseteq \rightarrow_1$. We have to express the fact that the accepted words must be suffixes of words in $L(\mathcal{C}_1)$ and prolongations of words in $L(\mathcal{C}_2) \cap \text{Pref}(L(\mathcal{C}_1))$. Hence, we only accept among elements of $L(A_1^q)$, words $w$ (let $\rho_w$ the accepting run of $w$ in $A_1^q$) such that there exists a word $w'$ (the removed prefix) having an accepting run $\rho_{w'}$ in $A_2^q$ satisfying $f_2$ (i.e., $\rho_{w'}$ satisfies $f_2 \land [A_{1,2}^q]$), and such that $\rho_w \rho_{w'}$ (which is an accepting run of $w'w$ in $A_1$) satisfies the formula $f_1$. The only problem comes from the fact that the transitions of the accepting run of $w'w$ in $A_1$ which are in the loop with the state $q$ can be part of $A_{1,2}^q$ or $A_1^q$.

To express the constraints, we need to rename the variables associated to the transitions of $A_1$ and $A_2$. Let $Y_i = \{y_t: t \in \rightarrow_i\}$, and let $U_i = (V_i \setminus X_i) \cup Y_i$, for $i \in \{1, 2\}$. Then, let $X_q = \{t: t \in \mathcal{F}(A_1^q) \text{ and } t \text{ is in the loop containing } q\}$ and $X_q' = \{t: t \in \mathcal{F}(A_2^q)\}$. Notice, that $X_q$ is empty, if $q$ is not in a loop.

The language $L(\mathcal{C}_2)^{-1} \cdot L(\mathcal{C}_1)$ is accepted by the CQDD:

$$\mathcal{C}_2^q \cdot \mathcal{C}_1 = \{(A_1^q, f_{\text{quot}}^q((X_1^q \setminus X_q) \cup U_1 \cup U_2 \cup X_{1,2})): q \in F\}$$

where $X_{1,2} = \{x_{(t_1, t_2)}: t_1 \in \rightarrow_1 \land t_2 \in \rightarrow_2\}$ and the formula $f_{\text{quot}}^q$ is given by

$$f_1' \land f_2' \land [A_{1,2}^q] \land \bigwedge_{t_1 \in \rightarrow_1} y_{t_1} = \sum_{t_2 \in \rightarrow_2} x_{(t_1, t_2)} \land \bigwedge_{t_2 \in \rightarrow_2} y_{t_2} = \sum_{t_1 \in \rightarrow_1} x_{(t_1, t_2)},$$

where the formulas $f_1'$ and $f_2'$ are defined by

$$f_1' = f_1[y_{t_1}/x_t: x_t \in X_1 \setminus X_q][(y_t + x_t)/x_t: x_t \in X_q],$$

$$f_2'= f_2[y_{t_1}/x_t: x_t \in X_2].$$

**Generalization to n dimensions:** We construct the set of automata $A_{1,2}^q$ and $A_1^q$ for each dimension. For every combination of these automata we obtain one accepting component by modifying the constraints simultaneously for each dimension.
3.5. Emptiness, membership, and inclusion problems

We show that the emptiness problem is decidable for the whole class of CSAs, and hence for CQDDs in particular.

**Proposition 3.25.** The emptiness problem is decidable for CSAs.

**Proof.** Consider the CSA $\mathcal{C} = \{\langle A, f \rangle\}$. Clearly, $L(\mathcal{C}) \neq \emptyset$ if and only if the Presburger formula $[A] \land f$ is satisfiable. The generalization to $n$-dimensional CQDDs is obvious. □

Notice that Proposition 3.25 can be generalized to more general structures where any finite-state automata are used instead of simple automata. This is due to the fact that one can associate to every finite-state automaton a characteristic formula using Parikh’s theorem.

Finally, by Propositions 3.11, 3.15, and 3.25, we deduce the following fact:

**Corollary 3.26.** For every $n \geq 1$, the membership problem as well as the inclusion problem are decidable for $n$-dim CQDDs.

4. Manipulating sets of configurations of CFSMs

We show in this section how CQDDs can be used to represent sets of configurations of CFSMs. Then, we show that the class of CQDD representable sets of configurations is effectively closed under union, intersection, and the successor function post, and that furthermore, this class has decidable emptiness and inclusion problems. This means that CQDD can be used as a representation structure in a forward analysis algorithm. We also show that CQDDs can be used for backward reachability analysis. For that, it suffices to represent the reverse images of the queue contents.

4.1. Representing sets of configurations

Let $M = (S, K, \Sigma, \gamma, T)$ be a CFSM. Every set of configurations $C \subseteq Conf$ can be written as a union $\bigcup_{s \in S} \{s\} \times \mathcal{L}_s$ where the $\mathcal{L}_s$’s are $|K|$-dim multi-languages ($\mathcal{L}_s$ represents the set of contents of the queues at the control state $s$).

**Definition 4.1.** We say that a set of configurations $C = \bigcup_{s \in S} \{s\} \times \mathcal{L}_s$ is CQDD representable (resp. reverse representable) if for every $s \in S$, the multi-language $\mathcal{L}_s$ is CQDD definable (resp. reverse definable).

**Example 4.2.** Let us consider the system $M$ which is represented by the following picture: 

![Diagram](image)
The set of configurations of $\mathcal{M}$ reachable from the configuration $(s_0, e, e, e)$ is given by
\[
\{s_0\} \times \{(a^n, (ba)^m, a^m) : n, m \geq 0\} \cup \{s_1\} \times \{(a^n, (ba)^m b, a^m) : n, m \geq 0\}
\]
and is clearly CQDD representable.

4.2. Basic operations on sets of configurations

In the sequel, we present results allowing to manipulate and to reason about sets of configurations that are CQDD representable or reverse representable.

First of all, it is straightforward to deduce from Propositions 3.12 and 3.25, and Corollary 3.26, the following fact:

**Theorem 4.3.** The class of CQDD representable (resp. reverse representable) sets of configurations is effectively closed under union and intersection, and has decidable emptiness, membership and inclusion problems.

Now, we show that the class of CQDD representable (resp. reverse representable) sets of configurations is effectively closed under the post (resp. pre) function.

Let $z = (s, op, s') \in T$. Then, it is easy to see that if there is a successor of a configuration $(s, w)$ by $z$ it is of the form $(s', (in(z)^{-1} \cdot w) \cdot out(z))$. Then, the closure of the class of CQDD representable sets under the post function follows directly from the closure of this class under left-quotient and concatenation.

**Theorem 4.4.** For every CQDD representable set of configurations $C$, the set of configurations post($C$) is CQDD representable and effectively constructible.

**Proof.** Let $(S, K, \Sigma, T)$ be a CFSM. Then, since CQDDs are closed under union (Proposition 3.12), and since the post function distributes w.r.t. union, it suffices to show that, for every $s \in S$, for every $z = (s, op, s') \in T$, and for every $|K|$-dim CQDD $\mathcal{C}$, we can effectively construct a CQDD $\mathcal{C}'$ such that $\{s\} \times L(\mathcal{C}') = post(\{s\} \times L(\mathcal{C}))$. Indeed, it is very easy to see that the two multi-languages $\mathcal{J} = \{in(\tau)\}$ and $\mathcal{C} = \{out(\tau)\}$ are CQDD representable. Then, using Propositions 3.24 and 3.22, we can construct a CQDD $\mathcal{C}'$ such that $L(\mathcal{C}') = (\mathcal{J}^{-1} \cdot L(\mathcal{C})) \cdot \mathcal{C}$. $\square$

Let $z = (s, op, s') \in T$. The predecessor of a configuration $(s', w)$ by $\tau$ is $(s, in(\tau) \cdot w \cdot out(\tau)^{-1})$. Then, if we represent a configuration by taking the reverse of each queue contents, it can be seen that the predecessor of $(s', w^R)$ is $(s, (out(\tau)^R)^{-1} \cdot w^R \cdot in(\tau)^R)$. Hence, the closure of the class of CQDD reverse representable sets under the pre function is also a consequence of its closure under left-quotient and concatenation.

**Theorem 4.5.** For every CQDD reverse representable set of configurations $C$, the set of configurations pre($C$) is CQDD reverse representable and effectively constructible.
Proof. Let \((S,K,\Sigma,T)\) be a CFSM. Again, since CQDD's are closed under union, and since the \(pre\) function distributes w.r.t. union, it suffices to show that, for every \(s' \in S\), for every \(\tau = (s', op, s) \in T\), and for every \(|K|\)-dim CQDD \(\mathcal{C}\), we can effectively construct a CQDD \(\mathcal{C}'\) such that \(\{s'\} \times L(\mathcal{C}') = pre^\tau(\{s\} \times L(\mathcal{C}))\). Indeed, let \(\mathcal{J} = \{\text{in}(\tau)^R\}\) and \(\mathcal{E} = \{\text{out}(\tau)^R\}\). These two multi-languages are obviously CQDD representable. Then, a CQDD \(\mathcal{C}'\) such that \(L(\mathcal{C}') = (\mathcal{E}^{-1} \cdot L(\mathcal{C})) \cdot \mathcal{J}\) can be constructed using Propositions 3.24 and 3.22. □

5. Computing the effect of a control cycle

We present in this section our main technical result. We show that, given a CFSM \(\mathcal{M}\), for every cycle \(\theta\) in the transition graph of \(\mathcal{M}\), the class of CQDD representable (resp. reverse representable) sets of configurations is effectively closed under the \(post^\theta\) function (resp. the \(pre^\theta\) function).

5.1. Main result

Theorem 5.1. For every CQDD representable set of configurations \(C\), and every cycle \(\theta\), the set of configurations \(post^\theta(C)\) is CQDD representable and effectively constructible.

Proof. We give hereafter an informal presentation of the main steps of the proof. The full details of the proof are given in Section 5.2.

Let \((S,K,\Sigma,T)\) be a CFSM. We show that for every \(s \in S\), and for every cycle \(\theta\) starting from \(s\), and for every \(|K|\)-dim CQDD \(\mathcal{C}\), we can effectively construct a CQDD \(\mathcal{C}'\) with \(\{s\} \times L(\mathcal{C}') = post^\theta(\{s\} \times L(\mathcal{C}))\).

First of all, since \(post\) distributes w.r.t. union, it suffices to reason on CQDDs \(\mathcal{C}\) consisting of one accepting component \(\langle \mathcal{A}, f \rangle\). Let \(\mathcal{A} = (A_1, \ldots, A_{|K|})\).

The scheme of the proof is the following:

- We construct for each queue independently all the possible successor configurations after executing \(\theta\) a certain number of times. This number is represented by a variable \(x_\theta\). For that, we do not consider the constraints imposed by \(f\) on the transitions of the automata \(A_1, \ldots, A_{|K|}\) (these constraints are introduced later). So, for every \(i \in \{1, \ldots, |K|\}\), we calculate the effect of \(\theta\) after \(x_\theta\) iterations as if the system had only one queue \(k_i\) and the initial configurations were given by \(L(A_i)\). This yields a 1-dim CQDD such that each of its elements is a tuple \(\langle B, g \rangle\) where \(g\) has as free variables \(x_\theta\) as well as variables corresponding to the transitions of the initial automaton \(A_i\) and the transitions of the automaton \(B\). This means that \(g\) makes a link between the initial contents and the final ones and takes into account the fact that these final contents are obtained by iterating \(x_\theta\) times the cycle \(\theta\).
- The global result is then constructed as follows: we take from each CQDD corresponding to one of the queues a pair \(\langle B_i, g_i \rangle\), and we construct the \(|K|\)-dim accepting pair \(\langle (B_1, \ldots, B_{|K|}), h \rangle\), where \(h = \bigwedge_{i=1}^{|K|} g_i \land f\). The formula \(h\) links the original
contents of the queues with the final ones and imposes that the original ones satisfy the formula $f$. Moreover, the formula $h$ makes a link between the final contents from different queues, since each of the formulas $g_i$ depend on the same variable $x_0$, which means that these contents correspond to the same number of iterations of the cycle $\theta$.

The essential part of the proof is to show that the effect of a cycle on each queue can be effectively constructed as a 1-dim CQDD. The proof uses an idea of Jérón and Jard [15] which gives a sufficient condition for a sequence of transition to be able to be repeated perpetually. Some of the results presented in this section have been obtained independently in [13]. Let us fix a cycle $\theta$ and a queue $\kappa$, and let $\text{in}$ and $\text{out}$ denote the input and output of the system by $\theta$ on the queue $\kappa$. The proof consists of an analysis of the effect of $\theta$ on the contents of $\kappa$, depending on the input $\text{in}$, the output $\text{out}$, and the form of the initial contents. This analysis is done in a parameterized way since the effect must be computed not for one single content but for an infinite number of contents.

Let us start by the two simple cases where either $\text{in}$ or $\text{out}$ are the empty word. In the case where $\text{in} = e$, starting from a configuration $w$ the effect of repeating $x_\theta$ times the cycle $\theta$ is to concatenate the word $\text{out}^{x_\theta}$ to the right, which gives $w.\text{out}^{x_\theta}$. Clearly this effect can be constructed using the operation of concatenation between CQDDs. In the case where $\text{out} = e$, the effect of repeating $x_\theta$ times the cycle $\theta$ starting from a word $w$ is given by $(\text{in}^{x_\theta})^{-1}.w$. Thus, this effect can be constructed using the operation of left-quotient between CQDDs.

Now, we consider the most difficult case where both $\text{in}$ and $\text{out}$ are nonempty words. Consider a configuration of the form $\text{in}^k.x$ where $k$ is a positive integer, and $x$ is a word over $\Sigma$. Then, we know that $\theta$ can be iterated at least $k$ times. After $k$ iterations, the reached configuration is of the form $x.\text{out}^k$.

Then, the first question is whether $\theta$ can be executed further. For that, we should have necessarily either $\text{in}$ is a prefix of $x$, or $x$ is a prefix of $\text{in}$, because otherwise the system will not be able to consume $x$. It suffices to consider the case where $x$ is a prefix of $\text{in}$ because we can restrict our attention to contents of the form $\text{in}^k.x$ where $|x| < |\text{in}|$. Then, we know that the system will consume the word $x$ and will start consuming symbols of the word $\text{out}^k$ (from the left), and whenever $\text{in}$ can be consumed, $\text{out}$ is added to the right, and so on. Then, the question is how much iterations of $\theta$ can be performed that way. This depends of course on $\text{in}$, $\text{out}$, and $x$.

Let us start by the case where $|\text{in}| \leq |\text{out}|$. Then, the problem is to determine whether $\theta$ can be repeated forever or only a finite number of times, and what are the reached configurations in each case. We give conditions on the words $\text{in}$, $\text{out}$, and $x$ characterizing exactly the case when the cycle can be iterated an unbounded number of times. These conditions say that there must be a word $u$ such that $\text{out} = c.x.u$, where $c$ is the subword of $\text{in}$ such that $\text{in} = x.c$, and $c.x.u = u.c.x$. Moreover, we show that in this case, starting from a configuration $\text{in}^k.x$ the reached configuration after $x_\theta$ iterations of $\theta$ is $\text{in}^k.x.u^{x_\theta}$. This means that the effect of repeating $\theta$ is given by concatenating the language $\{u^n \mid n \geq 0\}$ such that $n = x_\theta$, which is clearly CQDD definable.
Let us consider an example. Let \( \text{in} = ba \), \( \text{out} = abab \), and \( x = b \). Then, we have \( c = a \) and \( u = ab \), and we have \( c.x.u = abab = u.c.x \). So, if we start for instance from the configuration \( \text{in}.x = bab \), the reachable configuration after \( n \) iterations of \( \theta \) is \( \text{in}.x.u^n = bab(ab)^n \). Indeed, we have

\[
\begin{align*}
\text{in}.x &= bab \\
\rightarrow x.\text{out} &= babab = \text{in}.x.u \\
\rightarrow x.\text{u}.\text{out} &= x.\text{out}.u &= bababab = \text{in}.x.u^2 \\
\rightarrow x.\text{u}^2.\text{out} &= x.\text{out}.u^2 &= babababab = \text{in}.x.u^3 \\
&\cdots
\end{align*}
\]

We show also that when the conditions are not satisfied, there is a bound \( k' \) which is defined in terms of \( \text{in} \), \( \text{out} \) and \( x \), such that, for every contents of the form \( x.\text{out}^k \), the cycle \( \theta \) can be executed at most \( k' \) times.

Now, consider the case where \( |\text{in}| > |\text{out}| \). Then, we know that for each single configuration, the number of iterations is finite. However, we are reasoning about infinite sets of configurations and we have to define the set of reachable configurations for all of them. Moreover, we have to take into account the fact that the output of the system could be reused as input. This makes the definition of the number of possible iterations of the cycle difficult. Let us consider an example. Let \( \text{in} = abab \), \( \text{out} = ba \), and \( x = a \). Then, starting from \( \text{in}^5.x = (abab)^5.a \) we have the following sequence of reachable configurations:

\[
\begin{align*}
\text{in}^5.x &= (abab)^5.a \\
\rightarrow \text{in}^4.x.\text{out} &= (abab)^4.a.ba \\
\rightarrow \text{in}^3.x.\text{out}^2 &= (abab)^3.a.(ba)^2 \\
&\cdots \\
\rightarrow^* x.\text{out}^5 &= a.(ba)^5 &= (abab)^2.a.ba = \text{in}^2.x.\text{out} \\
&\cdots \\
\rightarrow^* x.\text{out}^3 &= a.(ba)^3 &= (abab).a.ba = \text{in}.x.\text{out} \\
\rightarrow x.\text{out}^2 &= ababa = \text{in}.x \\
\rightarrow x.\text{out} &= aba
\end{align*}
\]

So, after 5 iterations, the system consumes \( \text{in}^5 \) and produces \( \text{out}^5 \), which allows to reach the configuration \( x.\text{out}^5 \). Then, it turns out that \( x.\text{out}^5 \) is actually equal to \( \text{in}^2.x.\text{out} \). This allows two additional iterations leading to the configuration \( x.\text{out}^3 \). Since this configuration is equal to \( \text{in}.x \), the cycle can be executed once more.

We prove that, roughly speaking, under some conditions on \( \text{in} \), \( \text{out} \), and \( x \), all the reachable configurations from \( \text{in}^k.x \) that are of length greater than a certain bound \( k' \) (defined in terms of the lengths of \( \text{in} \) and \( \text{out} \)) are of the form \( \text{in}^p.x.\text{out}^q \), where \( p \) and \( q \) are related with \( x_0 \) and \( k \) by linear constraints. Moreover, we show that for every \( k \), there is a successor of \( \text{in}^k.x \) of this form with a length smaller than \( k' \). Hence, we can start by constructing the set of all successors of the form specified above (this
set is clearly CQDD representable), and then we can compute their successors after $k'$ iterations of $\theta$, and this allows to capture all the other reachable configurations since for each configuration the size of its successors is decreasing.

On the other hand, we prove that when these conditions on $in$, $out$, and $x$ are not satisfied, there exists a constant $k''$ such that, for every contents of the form $x.out^k$, the cycle $\theta$ can be executed at most $k''$ times. □

5.2. Detailed proof of Theorem 5.1

5.2.1. Analysis of the effect of a cycle

In this section we analyse the effect of the repeated execution of one cycle $\theta$ in the transition graph of the system on the contents of one queue. We suppose for this analysis that all the other queues stay unchanged. We consider the case, where the contents of the queue permits the execution of the cycle beyond the initial contents; i.e., the writing of messages at the end of the queue enables the execution of the cycle after the reading of all the initial contents. All the other cases are easy to analyse.

Let $\kappa_i$ be a queue. For each queue in the system the cycle $\theta$ defines a word $in_i \in \Sigma^*$ (the $i$th component of $in(\theta)$) and a word $out_i \in \Sigma^*$ (the $i$th component of $out(\theta)$). We consider two cases:

- The size of the queue contents increases or stays the same after execution of the cycle and one given contents.
- The size of the queue contents decreases after the execution of the cycle and one given contents.

Analysis of a cycle with increasing size of contents: In this case, we consider $in_i$ and $out_i$ of the form

$$|out_i| \geq |in_i|, \forall i \neq e \land in_i \neq e.$$  

Here the size of what is written into the queue is greater or equal of what is read from the queue. Given one queue contents, the length of this contents increases or stays the same by executing the cycle.

We show hereafter that it is possible to decide whether a cycle can be executed perpetually and we give the effect of its repeated executions.

It is easy to see, that to execute the cycle perpetually, the queue contents must have the form $in_i^k.x$ where $x$ is a prefix of $in_i$. Therefore, let $in_i = x.c$ with $x,c \in \Sigma^*$ and $c \neq e$.

Definition 5.2. We call the word $x$ inc-repeating if and only if

$$\exists u,c_1,c_2 \in \Sigma^* \exists k \geq 0.out_i = c.x.u, u = (c.x)^k.c_1 \land c.x = c_1.c_2$$  

and $c.x.u = u.c.x$  

Lemma 5.3. If the word $x$ is inc-repeating then $out_i.u = u.out_i$.

Proof. With (4) we have $out_i.u = c.x.u.u = u.c.x.u = u.out_i$. □
We can show that if the word \( x \) is inc-repeating, then the cycle can be executed perpetually and vice versa. Formally:

**Proposition 5.4.** The word \( x \) is inc-repeating if and only if starting from a contents \( \text{in}_{k}^i.x \) with \( k > 0 \) the cycle can be executed perpetually. In this case, the queue contents after \( n \) executions is given by \( \text{in}_{k}^i.x.u^n \).

**Proof.** "\( \Rightarrow \)" Let \( x \) be inc-repeating. We prove by induction that starting from a queue contents \( \text{in}_{k}^i.x \) with \( k > 0 \) the cycle can be executed perpetually and the contents after \( n \) executions is given by \( \text{in}_{k}^i.x.u^n \).

- **Induction basis:** Given \( \text{in}_{k}^i.x \), an execution of the cycle yields \( \text{in}_{k-1}^i.x.u_{\text{out}} \). Using (3) this is equal to \( \text{in}_{k-1}^i.x.c.x.u \). Because \( \text{in}_1 = x.c \) this is equal to \( \text{in}_{k-1}^i.x.u \).

- **Induction step:** \( \text{in}_{k}^i.x \) can be executed \( n \) times and yields \( \text{in}_{k}^i.x.u^n \). Then, another execution yields \( \text{in}_{k}^i.x.u^n \).out \). By Lemma 5.3 this is equal to \( \text{in}_{k-1}^i.x.u^n \).out \). By Lemma 5.3 this is equal to \( \text{in}_{k}^i.x.u^n \).

"\( \Leftarrow \):" Starting from a queue contents \( \text{in}_{k}^i.x \), \( k \) executions of the cycle give a contents \( x.out_i^k \). Furthermore, since the cycle can be executed perpetually, for all \( k' \in \mathbb{N} \) there exists a \( x' \in \Sigma^* \) such that

\[
x.out_i^{k+k'} = \text{in}_{k'}^i.x.x'.
\]

Since \( \text{in}_i = x.c \), Eq. (5) is equivalent to

\[
out_i^{k+k'} = (c.x)^{k-1}.c.x'.
\]

Now, Eq. (6) implies that \( out_i \) starts with at least one \( c.x \) and the rest has to be a prefix of \( c.x \). Formally,

\[
\exists u, c_1, c_2 \in \Sigma^* \exists k_1 \geq 0 \text{ out}_i = c.x.u \land u = (c.x)^{k_1}.c_1 \land c.x = c_1.c_2
\]

which is equivalent to condition (3). By using the equations of (7) in (6) we obtain

\[
(c.x.(c.x)^{k_1}c_1)^{k+k'} = (c.x)^{k-1}.c.x'
\]

for all \( k' \in \mathbb{N} \). Thus we have

\[
\underbrace{c.x.c.x\ldots c.x.c_1.c.x\ldots}^{k_1 \text{ times}} = \underbrace{c.x.c.x\ldots c.x.c_1.c_2\ldots}^{k \text{ times}}
\]

This implies \( c_1.c.x = c.x.c_1 \). The equality

\[
c.x.u = u.c.x
\]

follows easily from the definition of \( u \). \( \square \)

If condition (7) is not satisfied, then starting from \( x.out_i^k \) the cycle can be executed at most \( \max\{k_3 : \exists d \in \Sigma^*.\text{out}_i = (c.x)^{k_3}.d\} \) times. If condition (7) is satisfied, then to
obtain (9) we need $k' \geq k_1 + 3$. For the $x$'s which are not inc-repeating we obtain therefore the following corollary:

**Corollary 5.5.** If $x$ is not inc-repeating, then for every $k \geq 0$ starting from $x.out^k_i$ the cycle can be executed at most $k_2 = \max\{k_3 + 3: \exists d \in \Sigma^* . out_i = (c.x)^k d\}$ times.

*Analysis of a cycle with decreasing size of contents.* Here we consider $in_i$ and $out_i$ with the following conditions:

$$|in_i| > |out_i|, \quad out_i \neq \varepsilon \land in_i \neq \varepsilon$$

(10)

In this case, if we start to execute the cycle starting from a fixed contents, there are only a finite number of possible successors since the length of the queue contents decreases after each step. However, because we reason about infinite sets of configurations we have to define the set of all reachable configurations for all of them.

Given a queue contents $in_i^k.x$, $k$ executions of the cycle yields a contents $x.out_i^k$. We can show that there is a constant $k_2$ which depends only on $in_i$ and $out_i$ such that if starting from a contents of the form $x.out_i^k$ the cycle can be executed more than $k_2$ times, then the contents have a special form which can be captured by linear constraints and we can characterize all the successors.

For the rest of the section let $in_i = x.c$ with $x, c \in \Sigma^*$ and $c \neq \varepsilon$.

**Definition 5.6.** We call $x$ dec-repeating if and only if

$$\exists c_1, c_2 \in \Sigma^* . \exists k_1 > 0 . x = out_i^k . c_1 \land c_1 . c_2 = out_i$$

(11)

and

$$c_1 . c_2 = c_2 . c_1$$

(12)

**Lemma 5.7.** If $x$ is dec-repeating, then

$$\text{in}_i^{|out_i|} . x = x . \text{out}_i^{|in_i|}.$$  

(13)

**Proof.** Eq. (12) implies that $out_i . c_1 = c_1 . out_i$. Besides, for all $m \in \mathbb{N}$ we have $out_i^m = c_1^m . c_2^m$. By taking $m = |out_i|$ we obtain

$$\text{out}_i^{\text{out}_i} = c_1^{\text{|out}_i|}.$$  

(14)

Thus, $\text{in}_i^{\text{|out}_i|} . x = (x.c)^{|out_i|} . x = x.out_i^{k_1} . c_1 = x.out_i^{k_1} . c_1^{\text{|out}_i|} . x = x.out_i^{k_1} . c_1^{\text{|out}_i|} . x = x$.

(15)

A simple corollary of Lemma 5.7:

**Corollary 5.8.** If $x$ is dec-repeating, then

$$\forall d_i > 0, \text{in}_i^{d_i |out_i|} . x = x . \text{out}_i^{d_i |in_i|}.$$  

(16)
Proposition 5.9. The word $x$ is not dec-repeating if and only if there is a constant $k_2 \in \mathbb{N}$ such that starting from a contents $x.out_k^i$ with $k > 0$ the cycle can be executed at most $k_2 = \max\{k_3 + 3 : \exists d \in \Sigma^* . c.x = out_k^i . d\}$ times.

Proof. \(\Rightarrow\): Suppose that the constant does not exist. We show that this implies that $x$ is dec-repeating. Let us fix a queue contents $x.out_k^i$. For each $k' < k$, the cycle can be executed $k'$ times. This implies that there exists a $x' \in \Sigma^*$ with

$$x.out_{k'}^k = in_t^{k'} . x'. \quad (16)$$

If $k' > 0$, then (16) is equivalent to

$$out_{k'}^k = (c.x)^{k'-1} . c.x'. \quad (17)$$

If $k' > 1$, then because of (17) and $|in_t| > |out_t|$, we have

$$\exists c_1, c_2 \in \Sigma^* \exists k_1 > 0 . c.x = out_{k_1}^k . c_1 \land c_1 . c_2 = out_t. \quad (18)$$

This corresponds to condition (11). Using the equations (11) in (17) yields

$$(c_1 . c_2)^{k'} = ((c_1 . c_2)^{k_1} . c_1)^{k-1} . c.x'. \quad (19)$$

Now, if $k' \geq k_1 + 3$ then (19) implies

$$\underbrace{c_1 . c_2 \cdots c_1 . c_2} . \underbrace{c_1 . c_2 . c_1 \cdots} = \underbrace{c_1 . c_2 \cdots c_1 . c_2} . \underbrace{c_1 . c_1 . c_2 \cdots} . \quad (20)$$

Thus,

$$c_1 . c_2 = c_2 . c_1. \quad (21)$$

\(\Leftarrow\): Suppose that $x$ is dec-repeating. Then, we have

$$\exists c_1, c_2 \in \Sigma^* \exists k_1 > 0 . c.x = out_{k_1}^k . c_1 \land c_1 . c_2 = out_t$$

and $c_1 . c_2 = c_2 . c_1$. By Corollary 5.8, we have

$$\forall d > 0 . in_t^{d} . x = x . out_t^{d} . in_t.$$ 

Therefore the constant cannot exist because $d_1 |in_t|$ is not bounded and the equation $in_t^{d} . x = x . out_t^{d} . in_t$ shows that the cycle can be executed. \(\square\)

By the above lemma and Corollary 5.8, we can prove the following lemma, where $n_1 = |out_t|$ and $n_2 = |in_t|:

Proposition 5.10. If the word $x$ is dec-repeating, then for queue contents $in_t^{k_1} . x$ with $k \geq 4n_1$, if the queue contents of a successor after $n'$ executions of the cycle has length $\geq 3n_1 n_2 + |x|$, then it has the form

$$in_t^{n_1 + n_2 - d_4} . x . out_t^{d_4} . \underbrace{\cdots}_{n_1 + n_2 - n_3} + d_4,$$

where

- $k = d_1 n_1 + d_2$ with $0 \leq d_2 < n_1$,
• \( n' = d_3 n_1 + d_4 \) with \( 0 < d_4 \leq n_1 \),
• \( (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \geq 0 \).

**Proof.** Let us fix a \( k \) with \( k \geq 4n_1 \). We prove the lemma by induction on \( n' \). As induction basis we show that the lemma holds for all \( n' \leq n_1 \). Then, we show for all \( d_3 \in \mathbb{N} \) that if the lemma holds for \( n' \leq (d_3 + 1)n_1 \) then it holds for all \( n' \) with \((d_3 + 1)n_1 < n' \leq (d_3 + 2)n_1 \).

**Induction basis:** \( 1 \leq n' \leq n_1 \): In this case, \( d_4 = n' \), \( d_3 = 0 \) and all the successors of \( \text{in}_{i,x} = \text{in}_{i,n_1 + d_2,x} \) are given by \( \text{in}_{i,n_1 + d_2 - n',x} \text{. out}_{i}^{n'} \). Since \( n_1 + d_2 \geq n' \) and by Corollary 5.8 we have

\[
\text{in}_{i,n_1 + d_2 - n',x} \text{. out}_{i}^{n'} = \text{in}_{i,n_1 + d_2 - n',x} \text{. out}_{i}^{n_1 + d_4}
\]

which is the required form.

**Induction step:** Given a \( d_3 \in \mathbb{N} \), let \( n' \) such that \((d_3 + 1)n_1 < n' \leq (d_3 + 2)n_1 \). This implies that \( n' = (d_3 + 1)n_1 + d_4 \) with \( 0 < d_4 \leq n_1 \). Suppose that \( \text{in}_{i}^{k} \) has a successor after \( n' \) executions with length \( \geq 3n_1n_2 + |x| \). Then, the successor after \( n' - n_1 \) executions of the cycle has also length \( \geq 3n_1n_2 + |x| \) (the length decreases after an execution of the cycle). By the induction hypothesis the successors after \( n' - n_1 = d_3n_1 + d_4 \) steps are given by

\[
\text{in}_{i,n_1 + d_2 - d_4,x} \text{. out}_{i}^{n_1 + d_2 - n_4,x} \text{. out}_{i}^{n_1 + d_4}
\]

Because the length of the obtained contains is greater than \( 3n_1n_2 + |x| = 2n_1 |\text{in}_{i}^{k}| + |x| + n_2 |\text{out}_{i}^{k}| \), we have \( (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \geq n_2 \), and by Corollary 5.8 the successors after \( n' - n_1 \) steps are of the form

\[
\text{in}_{i,n_1 + d_2 - d_4,x} \text{. out}_{i}^{n_1 + d_2 - n_4,x} \text{. out}_{i}^{n_1 + d_4} = \text{in}_{i,n_1 + d_2 - d_4,x} \text{. out}_{i}^{n_1 + d_2 - n_4,x} \text{. out}_{i}^{n_1 + d_4}
\]

Then the successors after further \( n_1 \) steps (in total \( n' \) steps) are given by

\[
\text{in}_{i,n_1 + d_2 - d_4,x} \text{. out}_{i}^{n_1 + d_2 - d_4,x} \text{. out}_{i}^{n_1 + d_4}
\]

5.2.2. **Closure under post**

Using the results of the previous section we can show Theorem 5.1 which states that the class of CQDD representable configurations is closed under the function \( \text{post}_0^* \) for each cycle \( \theta \) in the transition graph of the system. To prove this theorem it suffices to show the following proposition:

**Proposition 5.11.** Let \((S,K,\Sigma,T)\) be a CFSM. Then, for every cycle \( \theta \) starting from some \( s \in S \), and for every \( |K| \)-dim CQDD \( \mathcal{C} \), we can effectively construct a CQDD \( \mathcal{C}' \) with \( \{s\} \times L(\mathcal{C}') = \text{post}_0^* \{s\} \times L(\mathcal{C}) \).

**Proof.** Since \( \text{post} \) distributes w.r.t. union, it suffices to reason on CQDD's \( \mathcal{C} \) consisting of one accepting component \( (\mathcal{A}, f) \). Let \( \mathcal{A} = (A_1, \ldots, A_{|K|}) \). As explained in the scheme of the proof in Section 5 it suffices to compute the effect of the cycle \( \theta \) on each queue
After \( x_0 \) executions starting from \( L(A_i) \). This effect is represented by a 1-dim CQDD \( \mathcal{D}_i \).

The global effect is constructed from these "local" effects by taking the same \( x_0 \) for all the queues.

Let us fix a queue \( \kappa_i \). In the following, we show how to construct \( \mathcal{D}_i \). For that, let us introduce some notations. Given a \( y \in \Sigma^* \) let \( A_y \) be the simple automaton which accepts the language \( \{ y \} \) and let \( B_y \) (resp. \( C_y \)) be the simple automaton which recognizes the language \( y^* \). \( B_y \) and \( C_y \) consist of exactly one loop. Let \( t_B \) (resp. \( t_C \)) be a transition of \( B_y \) (resp. \( C_y \)).

Now to construct \( \mathcal{D}_i \) we use the operations (left-quotient, concatenation, product) on CQDD's defined in Section 3.4. There are 5 cases to consider:

**Case 1:** \( \text{out}_i = \varepsilon \) and \( \text{ini}_i = \varepsilon \). In this case there is neither reading nor writing of the queue. Therefore the contents of the queue does not change. Thus,

\[
\mathcal{D}_i = \{ (A_i, tt) \}.
\]

**Case 2:** \( \text{out}_i = \varepsilon \) and \( \text{ini}_i \neq \varepsilon \). In this case, there is nothing written into the queue, but the cycle reads from it. \( \text{ini}_i \) can be read from the queue, if the queue contents starts with \( \text{ini}_i \). There can be several \( \text{ini}_i \) in the queue contents. Therefore, for contents of the form \( \text{ini}_i^k . x \in L(A_i) \) for some \( k \in \mathbb{N} \) and \( x \in \Sigma^* \), \( \text{ini}_i^{k-x_0} \cdot x \) is the new contents provided that \( k \geq x_0 \). Therefore,

\[
\mathcal{D}_i = (\langle B_{\text{ini}_i^k}^*, x_{\text{ini}_i^k} = x_0 \rangle)^{-1} \cdot (A_i, tt).
\]

**Case 3:** \( \text{ini}_i = \varepsilon \) and \( \text{out}_i \neq \varepsilon \). In this case, the cycle does not read from the queue, but it writes into it. A message can always be written into the queue. Thus, for every contents \( y \in L(A_i) \), \( y \cdot \text{out}_i^{x_0} \) is the new contents. Hence,

\[
\mathcal{D}_i = (A_i, tt) \cdot (B_{\text{out}_i^*}, x_{\text{out}_i^*} = x_0).
\]

**Case 4:** \( |\text{out}_i| \geq |\text{ini}_i| \), \( \text{out}_i \neq \varepsilon \) and \( \text{ini}_i \neq \varepsilon \). In this case, the size of what is written into the queue by the cycle is greater or equal than the size of what is read from the queue. We have analyzed in Section 5.2.1 the difficult part of this case. We have given with Proposition 5.4 a sufficient and necessary condition that a cycle can be executed perpetually starting from a given queue contents.

**Case 4.1:** We consider here words in \( L(A_i) \) which have the form \( \text{ini}_i^k \cdot x \), where \( x \) is not a prefix of \( \text{ini}_i \) and \( \text{ini}_i \) is not a prefix of \( x \). In this case, the cycle can be executed at most \( k \) times. Then, the reachable contents starting from these words are given by \( \text{ini}_i^{k-x} \cdot x \cdot \text{out}_i^{x_0} \) provided that \( k \geq x_0 \). Hence,

\[
\mathcal{D}_i = ((\langle B_{\text{ini}_i^k}^*, x_{\text{ini}_i^k} = x_0 \rangle)^{-1} \cdot (A_i, tt)) \cdot (C_{\text{out}_i^*}, x_{\text{out}_i^*} = x_0).
\]

\( L(\mathcal{D}_i) \) contains also successors of queue contents of the following case, but it is clear that all the words in \( L(\mathcal{D}_i) \) are valid successors.

**Case 4.2:** We consider here words in \( L(A_i) \) which have the form \( \text{ini}_i^k \cdot x \), where \( x \) is a possibly empty prefix of \( \text{ini}_i \) with \( |x| < |\text{ini}_i| \). In this case, the cycle can be executed
at least \( k \) times and after that, the form of \( \text{out}_i \) determines if the cycle can still be executed. Let \( c \in \Sigma^* \) such that \( \text{in}_i = x.c \). \( k \) executions of the cycle give a contents of the form \( x.\text{out}_i^k \) and the cycle could still be executed.

Case 4.2.1: The word \( x \) is inc-repeating. By using Proposition 5.4 we can see that starting from a queue contents \( \text{in}_i^k \cdot x = (x.c)^k \cdot x \) with \( k > 0 \), the cycle can be executed perpetually and after \( x_0 \) executions, the queue contents is given by \( \text{in}_i^k \cdot x.\text{out}_i^m \). Starting from a queue contents \( x \) we can execute the cycle perpetually if we can execute it once. This can easily be tested and depends on the order of adding and removing messages from the queue in the cycle.

Construction of \( \mathcal{D}_i \): \( \mathcal{D}_i \) is given by the finite union of the CQDD's \( \mathcal{E}_x \) where \( x \) is a prefix of \( \text{in}_i \). \( \mathcal{E}_x \) is constructed as follows:

If \( u = e \) then
\[ \mathcal{D}_i = \{ (\text{Ai}, \text{tt}) \} \]
otherwise, if the cycle can be executed once starting from \( x \), then
\[ \mathcal{E}_x = ((\langle \text{Bin}_i^*, \text{tt} \rangle \cdot (\text{Ax}, \text{tt})) \times (\text{Ai}, \text{tt})) \cdot (\text{C}_o^*, x_{tc} = x_0) \]
otherwise
\[ \mathcal{E}_x = ((\langle \text{Bin}_i^*, \text{tt} \rangle \cdot (\text{in}_i, \text{tt})) \times (\text{Ai}, \text{tt})) \cdot (\text{C}_o^*, x_{tc} = x_0). \]

Case 4.2.2: The word \( x \) is not inc-repeating. After \( n' \) executions of the cycle the set of reachable contents is given by \( \text{in}_i^{k-n'} \cdot x.\text{out}_i^{n'} \) with \( k \geq n' \). With Corollary 5.5 there is a constant \( k_2 \) such that starting from a contents of the form \( x.\text{out}_i^{n'} \) the cycle can be executed at most \( k_2 \) times. To capture all the possible successors of these contents it suffices to calculate the successors of \( \text{in}_i^{k-n'} \cdot x.\text{out}_i^{n'} \) with \( k \geq n' \) up to \( k_2 \) executions of the cycle.

Construction of \( \mathcal{D}_i \): \( \mathcal{D}_i \) is given by the finite union of the CQDD's \( \mathcal{E}_x \) where \( x \) is a prefix of \( \text{in}_i \). \( \mathcal{E}_x \) is constructed as follows:

First of all, let
\[ \mathcal{F}_x = (((\text{Bin}_i^*, x_{tb} = n')) \cdot (\text{Ax}, \text{tt}))^{-1} \cdot (\text{Ai}, \text{tt}) \cdot (\text{C}_o^*, x_{tc} = n'). \]

This gives us all the successors of the form \( \text{in}_i^{k-n'} \cdot x.\text{out}_i^{n'} \). For \( n' = x_0 \) this gives all the successors already generated in case 4.1. We know that the cycle could still be executed at most \( k_2 \) times from states of the form \( x.\text{out}_i^{n'} \). These states are included in \( \mathcal{F}_x \).

To obtain all the successors for up to \( k_2 \) executions of the cycle starting from states of \( \mathcal{F}_x \) we construct for every \( j \in \mathbb{N} \) with \( 0 \leq j \leq k_2 \), the CQDD \( \mathcal{F}_x^j \) such that \( \{s\} \times L(\mathcal{F}_x^j) = \text{post}_t^j(\{s\} \times L(\mathcal{F}_x)) \). This can be done due to the fact that CQDD representable sets are closed under \( \text{post}_t \) for every transition (see Theorem 4.4). Notice that \( \mathcal{F}_x = \mathcal{F}_x^0 \).
Moreover, we have to keep track of the overall number $x_0$ of executions of the cycle. This number is given as $x_0 = n' + j$. Thus,

$$\delta_x = \bigcup_{j=0}^{k_2} \text{add}(\mathcal{F}_x^j, x_0 = n' + j),$$

where $\text{add}(\mathcal{F}_x^j, x_0 = n' + j)$ replaces each formula $f$ appearing in the CQDD $\mathcal{F}_x^j$ by $f \land (x_0 = n' + j)$.

Case 5: $|\text{in}_i| > |\text{out}_i|$, $\text{out}_i \neq \epsilon$ and $\text{in}_i \neq \epsilon$. For this case we use the results of Section 5.2.1. There are two subcases:

Case 5.1: The contents have the form $\text{in}^k_i \cdot x$, where $x$ is not a prefix of $\text{in}_i$ and $\text{in}_i$ is not a prefix of $x$. Then, the set of reachable contents is given by $\text{in}^{k-x_0}_i \cdot x \cdot \text{out}^n_i$ with $k \geq x_0$. Hence, $\mathcal{D}_i$ is calculated like in case 4.1.

Case 5.2: The contents have the form $\text{in}^k_i \cdot x$, where $x$ is a possibly empty prefix of $\text{in}_i$ with $|x| < |\text{in}_i|$. Case 5.2.1: The word $x$ is dec-repeating. Let $c \in \Sigma^*$ with $\text{in}_i = x \cdot c$. We use Proposition 5.10. Let $n_1 = |\text{out}_i|$ and $n_2 = |\text{in}_i|$. For each queue contents $\text{in}^k_i \cdot x$ with $k \geq 4n_1$, all its successors with length $\geq 3n_1n_2 + |x|$ after $n'$ executions of the cycle are given by

$$\text{in}^{n_1+d_3-n_1}_i \cdot x \cdot \text{out}^{(d_1-1)n_2-d_3(n_2-n_1)+d_4}_i$$

where

- $k = d_1n_1 + d_2$ with $0 \leq d_2 < n_1$,
- $n' = d_3n_1 + d_4$ with $0 < d_4 \leq n_1$,
- $(d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \geq 0$.

Each $\text{in}^k_i \cdot x$ with $k \geq 4n_1$ has a successor $u_k$ of size smaller than $4n_1n_2 + |x|$ which is of the form (22), because at each step the length decreases by $n_2 - n_1$. Thus, there has to be a successor whose length is between $3n_1n_2 + |x|$ and $4n_1n_2 + |x|$. To capture the remaining successors of $\text{in}^k_i \cdot x$, it suffices to calculate the successors of contents of the form (22) up to $4n_1n_2 + |x|$ executions of the cycle. Indeed, by doing this we capture all the successors of $u_k$, because the length of the reached configurations decreases after each execution of $\theta$.

Construction of $\mathcal{D}_i$: $\mathcal{D}_i$ is given by the finite union of the CQDD's $\mathcal{D}_x$ where $x$ is a prefix of $\text{in}_i$. $\mathcal{D}_x$ is constructed as follows:

We construct a formula to calculate the possible values of $k$ in states of the form $\text{in}^k_i \cdot x$. Let $\mathcal{F}_x = (\langle B_{\text{in}_i}^+, x_0=k \rangle \cdot \langle A_x, tt \rangle) \times \langle A_i, tt \rangle$. Now let $f_x$ be the disjunction of each Presburger formula of each element of $\mathcal{F}_x$. Intuitively, $f_x$ provides us with constraints on $k$.

To calculate the possible set of successors let $g_x$ be the Presburger formula

$$f_x \land n' = d_2n_1 + d_4 \land k = d_1n_1 + d_2 \land 0 < d_4 \leq n_1$$

$$\land 0 \leq d_2 < n_1 \land (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \geq 0.$$
Let
\[ \mathcal{G}_x = (B^*_{in'}, x_{out'} = n_1 + d_2 - d_4) \cdot (A_x, tr) \cdot (C_{out'}, x_{in} = (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \land g_x). \]

This provides us with all the successors of the form (22). Let \( k_2 = 4n_1n_2 + |x| \). Then, to obtain the successors of up to \( k_2 \) executions of the cycle starting from states of \( \mathcal{G}_x \), we proceed as in case 4.2.2 and get \( \mathcal{G}'_x \).

Case 5.2.2: The word \( x \) is not dec-repeating. Similar to case 4.2.2.

Finally, we are able to give the resulting CQDD \( \mathcal{C}' \). \( \mathcal{C}' \) is the smallest set such that
\[ \forall (B_1, g_1) \in \mathcal{G}_1 \cdots \forall (B_{|K|}, g_{|K|}) \in \mathcal{G}_{|K|}, \left( \bigwedge_{i=1}^{|K|} g_i \land f \right) \in \mathcal{C}'. \]

**Theorem 5.12.** For every CQDD reverse representable set of configurations \( C \), and every cycle \( \theta \), the set of configurations \( \text{pre}_\theta^*(C) \) is CQDD reverse representable and effectively constructible.

**Proof.** Let CFSM \((S, K, \Sigma, T)\) be a CFSM. We show that for every \( s \in S \), and for every cycle \( \theta \) starting from \( s \), and for every \(|K|\)-dim CQDD \( \mathcal{C} \), we can effectively construct a CQDD \( \mathcal{C}' \) with \( \{s\} \times L(\mathcal{C})^R = \text{pre}_\theta^*((\{s\} \times L(\mathcal{C}))^R) \). The proof is actually the same as the one of Theorem 5.1 if we inverse the meaning of \( \text{in}(\theta) \) and \( \text{out}(\theta) \), and take their reverse images. \( \square \)

6. Reachability analysis

The basic (safety) verification problem consists in checking that a dangerous configuration can never be reached from an initial configuration. Thus, given a set of initial configurations \( I \) and a set of dangerous configurations \( D \), this problem can be formulated in the two following manners:

- **P1.** \( D \cap \text{post}^*(I) = \emptyset \),
- **P2.** \( I \cap \text{pre}^*(D) = \emptyset \).

The first formulation corresponds to a forward reachability analysis of the configuration space whereas the second one corresponds to a backward reachability analysis.

Hence, given a set of configurations \( C \), we wish to compute the set of its successors and predecessors; i.e., \( \text{post}^*(C) \) and \( \text{pre}^*(C) \). By definition, for \( \phi \in \{\text{post}, \text{pre}\} \), we have
\[ \phi^*(C) = \bigcup_{i \geq 0} X_i, \]
where
\[ X_0 = C, \]
\[ X_{i+1} = X_i \cup \phi(X_i) \quad \text{for every } i \geq 0. \]
In the case \( \phi = \text{post} \) (resp. \( \phi = \text{pre} \)), if \( C \) is CQDD representable (resp. reverse representable), it can be deduced from Theorems 4.3–4.5 that all the \( X_i \)'s are CQDD representable (resp. reverse representable). Hence, the equations above yields directly a semi-algorithm for calculating \( \phi^*(C) \) based on an iterative calculation of the \( X_i \)'s. Since the sequence of the \( X_i \)'s is increasing, their limit is reached if for some index \( i \) we have \( X_{i+1} \subseteq X_i \). In such a case the algorithm stops and returns \( X_i \). We can detect the occurrence of this case since the inclusion problem is decidable for CQDD representable (resp. reverse representable) set of configurations (by Theorem 4.3). If the set of initial and dangerous states are also CQDD representable (resp. reverse representable), then the problem P1 (resp. P2) above can be solved by Theorem 4.3.

Since the reachability problem is undecidable for CFSMs, an index \( i \) such that \( X_{i+1} \subseteq X_i \) does not exist in general, and the naive algorithm described above may never stop.

We propose a method to face this divergence problem which consists of an "exact acceleration" of the iterative calculation of the limit \( \phi^*(C) \) based on the following idea: Given a set of cycles in the transition graph of the system, say \( \Theta \), add at each step the set of successors (or predecessors) by each cycle in \( \Theta \). This operation is sound since all the added configurations belong to \( \phi^*(C) \), that is, \( \forall i \geq 0, Y_i \subseteq \phi^*(C) \). So, we compute \( \phi^*(C) \) as the limit of another increasing sequence of configurations \( (Y_i)_{i \geq 0} \) which is given by

\[
Y_0 = C, \\
Y_{i+1} = Y_i \cup \phi(Y_i) \cup \bigcup_{\theta \in \Theta} \phi_\theta(Y_i) \quad \text{for every } i \geq 0.
\]

Clearly, for every \( i \geq 0 \), we have \( X_i \subseteq Y_i \). Hence, the chance to reach the limit \( \phi^*(C) \) in a finite number of steps is greater (or at least equal) by considering the sequence of \( Y_i \)'s instead of the sequence of \( X_i \)'s, and this chance should increase with the size of \( \Theta \).

Therefore, using Theorems 4.3–4.5, 5.1, and 5.12, we obtain a generic reachability analysis semi-algorithm which computes (when it terminates) the exact set of successors (resp. predecessors) of a given CQDD representable (resp. reverse representable) set of configurations. This algorithm is given by:

**Reachability** \((\Theta, C)\):

\[
Y := C; \\
\text{repeat} \\
\quad Y' := Y; \\
\quad Y := Y \cup \phi(Y) \cup \bigcup_{\theta \in \Theta} \phi_\theta(Y) \\
\text{until } Y' \subseteq Y; \\
\text{return } (Y)
\]

A variety of reachability algorithms can be derived from the generic algorithm above by determining adequate strategies for choosing the set of cycles \( \Theta \) (for example, one reasonable strategy consists in considering elementary cycles).
For instance, the forward reachability analysis algorithm given in [5, 6] can be seen as particular instances of our algorithm.² There, the considered cycles are guaranteed to preserve regularity: starting from a regular set of configurations (finite-state automata recognizable), the set of reachable configurations by these cycles is also regular. Therefore, a representation structure based only on finite-state automata (QDDs) can be used in this case and allows to analyze some significant systems. But considering QDDs and only cycles of the form specified above or the cycles defined in [6] do not allow to reason about systems with nonregular sets of configurations like the system $\mathcal{M}$ given in Section 4. However, it is easy to see that our algorithm terminates and computes the exact set of configurations of the system $\mathcal{M}$ (given by 2) if we consider as $\Theta$ the set of the two elementary cycles $(s_0, \{\kappa_1!a\}, s_0)$, and $(s_0, \{\kappa_1?a, \kappa_2!b\}, s_1)(s_1, \{\kappa_2!a, \kappa_3!a\}, s_0)$.

7. Conclusion

We have applied the symbolic analysis principle to fifo-channel systems (communicating finite state machines). These systems have in general nonregular sets of configurations. We have proposed a representation structure for their sets of configurations (CQDDs) combining finite-state automata with counting constraints expressed in Presburger arithmetics. We have shown that these structures allow to compute the exact effect of the repeated execution of any cycle in the transition graph of a system. We use this fact to define a generic forward/backward reachability analysis algorithm which is parametrized by a set of cycles. This algorithm computes iteratively the set of successors (or predecessors) by considering these cycles as additional “meta-transitions” in the graph, following the approach adopted in [7, 5].

It can be seen that our reachability analysis procedure computes the fixpoint of a particular function on set of configurations. Actually, this procedure can be generalized to a model-checking procedure for any positive fixpoint formula constructed using disjunctions, conjunctions, and the successor (predecessor) function, starting from basic CQDD (reverse) representable sets.

In order to represent the reachability set by any cycle in the control graph of a CFSM, the expressive power of CQDDs is (almost) necessary. Indeed, the reachability set obtained by repeating a cycle is in general a set of multi-words $(w_1, \ldots, w_n)$ where the number of occurrences of symbols in different segments of the different words $w_i$ are related by linear constraints. However, the full power of Presburger arithmetics is not necessary to express these constraints (universal quantification is never used in these constraints).

On the other hand, CQDDs are defined using only a subclass of finite-state automata (restricted deterministic simple automata). So, these structures cannot for instance represent any regular set of configurations. However, CQDDs are expressive enough for

---

²In the definition of QDDs, any deterministic finite-state automata can be used. However, it can be checked that, starting from an initial configuration with empty queues, the constructed QDD is a union of DRSAs.
reachability analysis when the considered set of initial configurations is CQDD representable (usually, we start from some control state with empty queues), and when the "meta-transitions" used for acceleration are cycles.

References