Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations

Ahmed Bouajjani [1] 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 CQDD's combining finite-state automata with linear arithmetical constraints on number of occurrences of symbols. We show that CQDD's allow forward and backward reachability analysis of systems with nonregular sets of configurations. Moreover, we prove that CQDD's 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 [5].

Key words: 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

[1] Corresponding author. E-mail: Ahmed.Bouajjani@imag.fr

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

From these definitions, one can derive straightforwardly iterative procedures for computing the sets $post^*(S)$ and $pre^*(S)$ 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 modeled as finite-state automata, such algorithms are guaranteed to terminate. However, for more general system modeled 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 *post* and *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 $post^*$ and 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 $Y_i \subseteq \bigcup_{i \geq 0} X_i$.

This approach has been used in [9,8] to define model-checking algorithms for pushdown 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 QDD's (queue-content decision diagrams) 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 CFSM's 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 θ, 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 θ as much as possible. Then, the problem is to determine for which cycles θ the considered class of representation structures is effectively closed under $post_\theta^*$. In [5], only cycles θ 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 CFSM's 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, CQDD (constrained QDD), 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 CQDD's 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 θ. We prove also that the class consisting of reverse images of CQDD representable sets is effectively closed under the function pre^*_θ for every cycle θ. 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 [5] can be seen as a particular instance of our algorithm.

Related work: There are several works on symbolic analysis of CFSM's. 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 (QDD's) 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 CFSM's 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] analyze CFSM's 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.

Outline: The rest of the paper is organized as follows: In Section 2 we define the CFSM's. In Section 3, we introduce the CQDD's and give the results concerning their closure properties and their decision problems. In Section 4, we show that CQDD's are suitable for representing and manipulating sets of configurations of CFSM's. In Section 5 we present our main result saying that CQDD's 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 CFSM's, 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, Σ, T) *where*

- S *is a finite set of* control states*,*
- K *is a finite set of unbounded FIFO queues,*
- Σ *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* κ_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* $\gamma = (s, \vec{w})$ *where* s *is a control state in* S*, and* $\vec{w} = (w_1, \ldots, w_{|K|})$ *is a* $|K|$*-dim multi-word (i.e., a tuple in* $(\Sigma^*)^{|K|}$*), each* w_i *being the contents of the queue* κ_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* $ww'_i = w_i$,
- *if* $\kappa_i!w \in op$ *then* $w'_i = w_i w$,
- *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* τ *is* executable *at* $\gamma = (s, \vec{w})$ *if there exists* $\gamma' = (s', \vec{w}')$ *such that* $\gamma \xrightarrow{op} \gamma'$. *In this case,* γ' *is the* immediate successor *of* γ *by* τ, *and* γ *is the* immediate predecessor *of* γ' *by* τ. *We denote by* pre_τ *and* $post_\tau$ *the functions in* $2^{Conf} \rightarrow 2^{Conf}$ *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* τ. *We denote respectively by* pre *and* $post$ *the unions of the functions* pre_τ *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* θ *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 pre_τ and $post_\tau$ can also be straightforwardly generalized to sequences of transitions.

Definition 2.6 *Let* $\theta = \tau_1 \ldots \tau_n$ *be a sequence of transitions. Then,* $pre_\theta = pre_{\tau_1} \circ \ldots \circ pre_{\tau_n}$ *and* $post_\theta = post_{\tau_n} \circ \ldots \circ post_{\tau_1}$, *where* $\circ$ *denotes the composition of functions. The functions* pre^*_θ *and* $post^*_\theta$ *are the reflexive transitive closures of* pre_θ *and* $post_\theta$ *respectively.*

Given a set of configurations C, $pre^*_\theta(C)$ (resp. $post^*_\theta(C)$) is the set of predecessors (resp. successors) of configurations in C obtained by iterating an arbitrary number of times the sequence θ. Notice that this number of iterations is strictly greater than one only if θ 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* ϵ *the* empty word. *Let* $\vec{u} = (u_1, \ldots, u_n)$ *and* $\vec{v} = (v_1, \ldots, v_n)$ *be two n-dim multi-words. Then, we denote by* $\vec{u}\vec{v}$ *the multi-word* $(u_1 v_1, \ldots, u_n v_n)$.

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

Definition 2.7 *Let $\tau = (s, op, s')$ be a transition of a CFSM (S, K, Σ, T). Then, $out(\tau)$ 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 $w_i = \epsilon$. We denote by $in(\tau)$ the multi-word in $(\Sigma^*)^{|K|}$ such that the i^{th} component of $in(\tau)$ is w if $\kappa_i?w \in op$, otherwise ϵ.*

Intuitively, $out(\tau)$ is the *output* of the system into the queues by executing τ, whereas $in(\tau)$ 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, $out(\theta) = out(\tau_1)out(\tau_2) \cdots out(\tau_n)$ and $in(\theta) = in(\tau_1)in(\tau_2) \cdots in(\tau_n)$.*

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:

$$t ::= 0 \mid 1 \mid x \mid t - t \mid t + t$$

Then, the set of *Presburger formulas* is defined by:

$$f ::= t \leq t \mid \neg f \mid f \vee f \mid \exists x.\ f$$

Classical abbreviations can be used like integer constants (k), equality ($=$), boolean connectives as conjunction ($\wedge$), 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* $\nu : X \to \mathbb{N}$, we say that ν *satisfies* f, and write $\nu \models f$, if the evaluation of f under ν 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 Σ is a tuple $A = (Q, q_0, \to, F)$ where Q is a finite set of states, q_0 is the initial state, $\to \subseteq Q \times \Sigma \times Q$ is a transition relation (we write $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \to$), and $F \subseteq Q$ is a set of final states.

Given a word $w = a_0 \ldots a_\ell \in \Sigma^*$, a *run* of A over w is a sequence of transitions $\rho = (s_0, a_0, s_1) \ldots (s_\ell, a_\ell, s_{\ell+1}) \in \to^{\ell+1}$ such that $s_0 = q_0$. The run ρ 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 $(s, a, s') \in \to$.

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.

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

- $\forall (i, j) \in I.\ 0 \leq i \leq j \leq m$,
- $\forall (i, j) \in I.\ \forall (i', j') \in I.\ ((i', j') \neq (i, j) \Rightarrow (j < i' \vee j' < i))$,
- *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 $P_{i,j} = \{p^1_{i,j}, \ldots, p^{\ell_i}_{i,j}\}$,*
- *q_0 is the initial state,*
- *$\to \subseteq Q \times \Sigma \times Q$ 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*
  - $\exists! a \in \Sigma.\ q_j \xrightarrow{a} p^1_{i,j}$,
  - $\forall k \in \{1, \ldots, \ell_i - 1\}.\ \exists! a \in \Sigma.\ p^k_{i,j} \xrightarrow{a} p^{k+1}_{i,j}$,

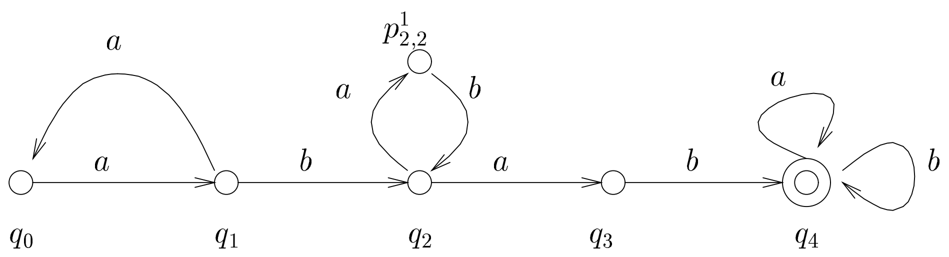
· $\exists ! a \in \Sigma.\ p_{i,j}^{\ell_i} \xrightarrow{a} q_i$

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

(4) If $(m,m) \in I$ *and* $P_{m,m} = \emptyset$, *then* $\forall\ a \in \Sigma'.\ q_m \xrightarrow{a} q_m$

- q_m *is the (unique) final state.*

Example 3.2 *A simple automaton:*



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* $(q_i, b_{i+1}, q_{i+1}) \cdots (q_{j-1}, b_j, q_j)(q_j, a_0, p^1_{i,j})(p^1_{i,j}, a_1, p^2_{i,j}) \cdots (p^{\ell_i}_{i,j}, a_{\ell_i}, q_i)$. *We say that* q_i *is the* root *of the loop, and that the loop* forms *the word* $b_{i+1} \dots b_j a_0 \dots a_{\ell_i}$. *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.

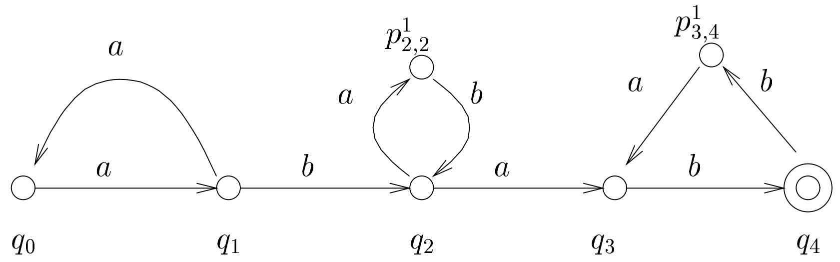
Now, we introduce a subclass of simple automata, called *restricted simple automata*, by giving the state q_m the same status as the other q_i's.

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 RSA's the set Σ' used in the definition of SA's is not needed.

Example 3.5 *A restricted simple 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 RSA's accept languages over Σ 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 Σ such that only u_1 and u_{m+1} may be empty. SA's accept words of the form $u_1v_1^*u_2v_2^*\cdots u_m(a_1+\ldots+a_\ell)^*$ where the a_i's are symbols in Σ.

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* ρ *of* A, *we denote by* ν_ρ *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* ρ. *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:*

$$\Big(\bigwedge_{q\in Q\setminus\{q_0,q_m\}} \sum_{t\in q^+} x_t = \sum_{t\in q^-} x_t\Big) \wedge$$
$$\Big(1+\sum_{t\in q_0^+} x_t = \sum_{t\in q_0^-} x_t\Big) \wedge \Big(\sum_{t\in q_m^+} x_t = 1+\sum_{t\in q_m^-} x_t\Big) \qquad (1)$$

It can be easily seen that the following fact holds:

Proposition 3.7 *For every valuation ν of the variables in X_A, ν satisfies $[A]$ if and only if there exists an accepting run ρ 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 ρ, then ν_ρ 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 $(A^i_1, \ldots, A^i_n)$ over Σ,*
- *f_i is a Presburger formula over a set of variables V_i containing the set $X_i = \{x_t : t \in \mathcal{T}_i\}$, where $\mathcal{T}_i$ is the set of all the transitions of the automata in $\mathcal{A}_i$ i.e., $\mathcal{T}_i = \bigcup_{j=1}^n \rightarrow^i_j$.*

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 $(A^1_i, \ldots, A^n_i)$ respectively, such that $\exists (V_i \setminus X_i).\ f_i$ is satisfied by the valuation $(\nu_{\rho_1}, \ldots, \nu_{\rho_n})$ (i.e., the valuation associating with each variable x_t the integer $|\rho_1 \ldots \rho_n|_t$, where $t \in \mathcal{T}_i$). 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.

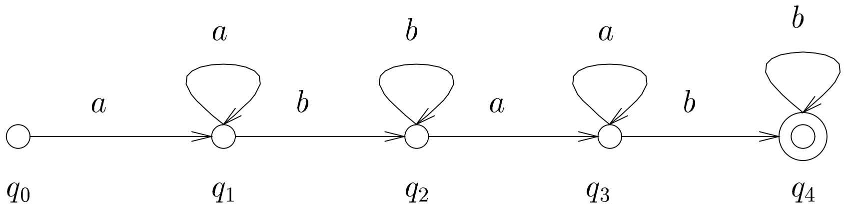
Notation 3.3 *We use tt as an abbreviation of the formula $\bigwedge_{t \in \mathcal{T}_i} x_t \geq 0$; i.e., the formula which imposes no constraints.*

Definition 3.9 *An n-dim CDSA is an n-dim CSA such that all its automata are deterministic (DSA's). An n-dim* Constrained Queue-content Decision Diagram (CQDD) *is an n-dim CDSA such that all its automata are restricted (DRSA's).*

Notation 3.4 *For every $n \geq 1$, we denote by n-CQDD (resp. n-CDSA) the class of all n-dim CQDD's (resp. n-dim CDSA's).*

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.

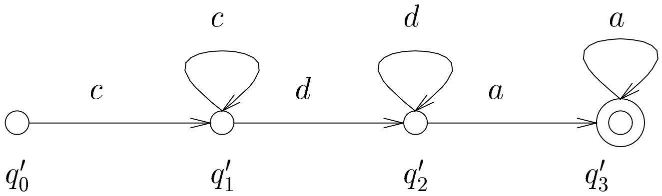
CQDD's allow to define nonregular multi-languages. For instance, consider the context-sensitive language $L_1 = \{a^n b^m a^n b^m : n, m \geq 1\}$. To define L_1, we use the automaton A_1 represented by the following picture:



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

$$x_{(q_1,a,q_1)} = x_{(q_3,a,q_3)} \;\;\wedge\;\; x_{(q_2,b,q_2)} = x_{(q_4,b,q_4)}$$

Now, let $L_2 = \{(a^n b^m a^n b^m, c^m d^n a^m) : n, m \geq 1\}$ 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:



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

$$x_{(q_1,a,q_1)} = n \ \wedge \ x_{(q_3,a,q_3)} = n \ \wedge \ x_{(q'_2,d,q'_2)} = n \ \wedge$$
$$x_{(q_2,b,q_2)} = m \ \wedge \ x_{(q_4,b,q_4)} = m \ \wedge \ x_{(q'_1,c,q'_1)} = m \ \wedge \ x_{(q'_3,a,q'_3)} = m$$

Hence, as it can be seen from these examples, CQDD's 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 intersection 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 CQDD's.

The proofs of these results are rather involved because all the operations modify the structure of the automata while the arithmetical constraints in CQDD's 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 CQDD's with a single accepting component. Then, we give the generalization to n dimensional CQDD's. 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 $\mathcal{C}_1 = \{\langle A_1, f_1(V_1) \rangle\}$ a CQDD and $\mathcal{C}_2 = \{\langle A_2, f_2(V_2) \rangle\}$ 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

$A_1 \times A_2$ 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 by a transition t_1 in A_1 and a transition t_2 in A_2, let us denote $\langle t_1, t_2 \rangle$ 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 $X_{1,2} = \{x_{\langle t_1, t_2 \rangle} : t_1 \in \rightarrow_1 \ \wedge t_2 \in \rightarrow_2\}$ and the formula $f_{1,2}$ is given by:

$$f_1 \wedge f_2 \wedge \bigwedge_{t_1 \in \rightarrow_1} (x_{t_1} = \sum_{t_2 \in \rightarrow_2} x_{\langle t_1, t_2 \rangle}) \wedge \bigwedge_{t_2 \in \rightarrow_2} (x_{t_2} = \sum_{t_1 \in \rightarrow_1} x_{\langle t_1, t_2 \rangle})$$

Generalization to n dimensions: Let $\mathcal{C}_1 = \{\langle (A_1^1, \ldots, A_1^n), f_1(V_1) \rangle\}$ and $\mathcal{C}_2 = \{\langle (A_2^1, \ldots, A_2^n), f_2(V_2) \rangle\}$ where for $i \in \{1, 2\}$, V_i is a superset of $X_i = \{x_t : t \in \bigcup_{j=1}^n \rightarrow_i^j\}$. Then, $L(\mathcal{C}_1) \cap L(\mathcal{C}_2)$ is accepted by the CQDD:

$$\mathcal{C}_1 \times \mathcal{C}_2 = \{\langle (A_1^1 \times A_2^1, \ldots, A_1^n \times A_2^n), f_{1,2}(X_{1,2} \cup V_1 \cup V_2) \}$$

where $X_{1,2} = \bigcup_{j=1}^n \{x_{\langle t_1^j, t_2^j \rangle} : t_1^j \in \rightarrow_1^j \ \wedge t_2^j \in \rightarrow_2^j\}$ and the formula $f_{1,2}$ is given by:

$$f_1 \wedge f_2 \wedge \bigwedge_{j=1}^{n} (\bigwedge_{t_1^j \in \rightarrow_1^j} x_{t_1^j} = \sum_{t_2^j \in \rightarrow_2^j} x_{\langle t_1^j, t_2^j \rangle} \wedge \bigwedge_{t_2^j \in \rightarrow_2^j} x_{t_2^j} = \sum_{t_1^j \in \rightarrow_1^j} x_{\langle t_1^j, t_2^j \rangle})$$

□

The closure under union is obvious and n-CQDD's are special cases of n-CDSA's. 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:

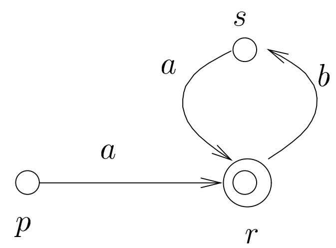
Lemma 3.13 *Let A be a DRSA A. Then, there exist a $k \in \mathbb{N}$ and DSA's $A_1, \ldots, A_k$ such that $\Sigma^* \setminus L(A) = \bigcup_{i=1}^k L(A_k)$.*

PROOF. We use the classical construction for complementation of automata. To obtain a union of DSA 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:

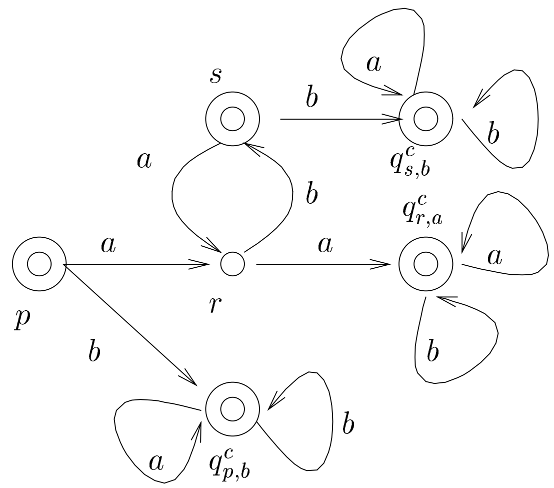
- $Q^c = Q \cup \{q^c_{q,a} | q \in Q \wedge a \in \Sigma \wedge \nexists q' \in Q. q \xrightarrow{a} q'\}$,
- $q_0^c = q_0$,
- $\rightarrow^c$ is the smallest set which satisfies:
  - $\rightarrow^c \supseteq \rightarrow$
  - $\forall b \in \Sigma.\ q^c_{q,a} \xrightarrow{b}{}^c\ q^c_{q,a}$
  - $\forall q^c_{q,a} \in Q^c.\ q \xrightarrow{a}{}^c\ q^c_{q,a}$
- $F = Q^c \setminus \{q_m\}$

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 given by:*



Then, an automaton which accepts $\Sigma^* \setminus L(A)$ is given by:



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 $\mathcal{C} = \{\langle A, f(V)\rangle\}$ where V is a superset of $X = \{x_t : t \in \rightarrow\}$. 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 $\overline{L(\{\mathcal{C}\})}$ is accepted by the CDSA:

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

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 $\mathcal{C} = \{\langle (A^1, \ldots, A^n), f(V)\rangle\}$ where V is a superset of $X = \{x_t : t \in \bigcup_{j=1}^{n} \rightarrow^j\}$. $\overline{L(A^j)}$ can be written as $\bigcup_{i=1}^{m_j} L(A_i^j)$ where each A_i^j is a DSA. Let $\rightarrow_i^j$ be the transition relation of A_i^j and let $X_i = \{x_t : t \in \bigcup_{j=1}^{n} \rightarrow_i^j\}$. Then, the language $\overline{L(\{\mathcal{C}\})}$ is accepted by the CDSA:

$$\begin{aligned}
\overline{\mathcal{C}} = \{\langle (B^1, B^2, \ldots, B^n), tt\rangle :& \\
&(\forall k \in \{1, \ldots, n\}.\ (B^k = A^k \vee \exists j \in \{1, \ldots, m_j\}.B^k = A_k^j)) \\
&\wedge \exists k \in \{1, \ldots, n\}(B^k \neq A^k)\} \\
\cup\ \{\langle (A^1, \ldots, A^n), \neg\exists(V \setminus X).\ f\rangle\}&
\end{aligned}$$

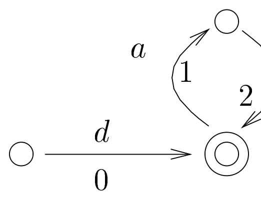
□

3.4.3 Closure under concatenation

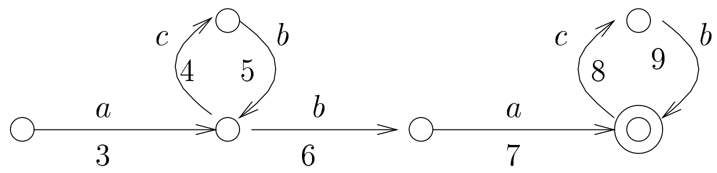
In this section we prove that CQDD's are closed under concatenation.

Definition 3.16 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two n-dim multi-languages. The* concatenation *of $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted by $\mathcal{L}_1 \cdot \mathcal{L}_2$, is the set $\{\vec{w} \in (\Sigma^*)^n : \exists \vec{u} \in \mathcal{L}_1.\ \exists \vec{v} \in \mathcal{L}_2.\ \vec{w} = \vec{u}\vec{v}\}$.*

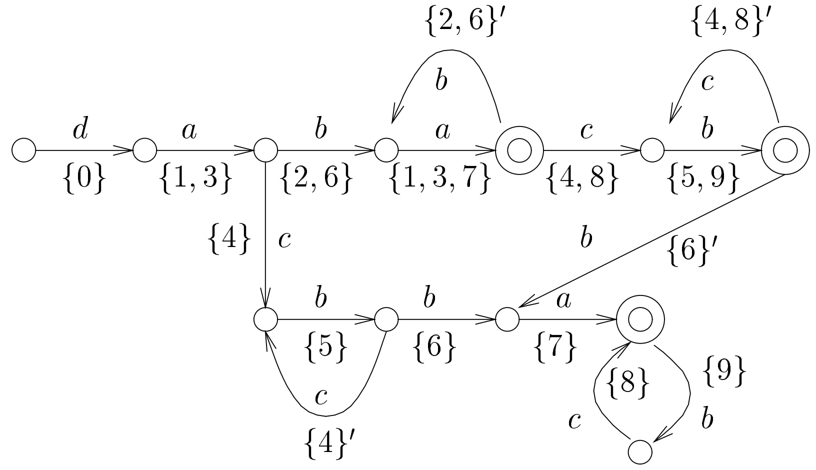
The concatenation of two deterministic restricted simple automata A_1 and A_2 can give a non-deterministic simple automaton. We can determinize this automaton and then the problem is that we have to define a constraint which relates the variables associated to transitions of this automaton with variables associated with transitions of A_1 respectively A_2. Let us illustrate this problem with a simple example. Consider the CQDD's $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ and $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ where, for $i \in \{1,2\}$, $A_i = (Q_i, q^i_{init}, \rightarrow_i, q^i_{fin})$, and V_i is a superset of $X_i = \{x_t : t \in \rightarrow_i\}$. Let A_1 be the automaton



and A_2 the automaton



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 $X_d = \{x_t : t \in \rightarrow_d\}$. To define $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ we have to associate to 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 $w = w_1.w_2$ 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 $w = w_1.w_2$, $w_1 \in L(A_1)$ 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 $\mathcal{C} = \{\langle A, f(V)\rangle\}$ be a CQDD such that the final state of A is in a loop. Then, there exists a CQDD $\mathcal{C}' = \{\langle A', f'(V')\rangle\}$ 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 to $A = (Q, q_0, \rightarrow, \{q_m\})$. 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 $\mathcal{C}' = \{\langle A', f'(V')\rangle\} = \mathcal{C} \times \{\langle A'', tt\rangle\}$. 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. $\square$

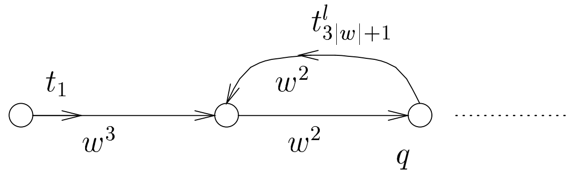
We show that each CQDD can be decomposed into a finite union of CQDD's whose automata have an arbitrary large last loop.

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

PROOF. Let $\mathcal{C} = \{\langle A, f(V)\rangle\}$ 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 last loop of A_i' forms the word w^n instead of w. The rest of the automaton remains unchanged. For each i with $0 \leq i < n$ the CQDD's $\mathcal{C}_i$ are given by $\mathcal{C} \times \{\langle A_i', tt\rangle\}$. $\square$

Lemma 3.19 *Let $\mathcal{C} = \{\langle A, f(V)\rangle\}$ 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 $\mathcal{C}' = \{\langle A', f'(V')\rangle\}$ 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}.*

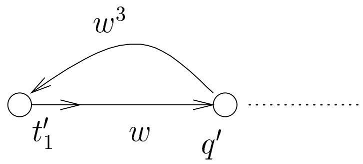
PROOF. Let $A = (Q, q_0, \rightarrow, \{q_m\})$ and let I be the associated set of indices. Intuitively, we have to fold the sequence formed by the first $k_1|w|$ states of A on its first loop. Let $c_1 = k_1|w|$ and $c_2 = k_2|w|$. The automaton A consists of three parts: First, c_1 states which are 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 $t^l_1, \ldots, t^l_{c_2}$ the c_2 transitions of the first loop of A and $\mathcal{T}$ the set of all other transitions. $\mathcal{T}$ 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,



Part 1 Part 2

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 facts can be expressed by the constraint $f'(V')$. Let $t'_1, \ldots, t'_{c_2}$ be the transitions of the first loop of A' and $\mathcal{T}'$ the set of the other transitions. $f'(V')$ is given by:

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

In the example, we have among other constraints $x_{t'_1} = x_{t_1} + x_{t^l_{3|w|+1}}$.

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 $L(\mathcal{C}) = L(\mathcal{C}')$. $\square$

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 $Pref(L)$ the set of prefixes of words in L.*

Lemma 3.20 *Let $\mathcal{C} = \{\langle A, f(V)\rangle\}$ 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 $q_0 \overset{w^{k_1}}{\rightarrow} q_1 \overset{w^{k_2}}{\rightarrow} q_1$. 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 $w^{k_1} \notin Pref(L(A''))$.*

PROOF. First, notice that the language Σ^* can be written as $L \cup \overline{L}$ where $L = w^{k_1}.(w^{k_2})^*.\Sigma^*$. 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 $\mathcal{C}' = \mathcal{C} \times \{\langle A'_i, tt\rangle : 1 \leq i \leq n'\}$ 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 $\mathcal{C} \times \{\langle A''_i, tt\rangle : 1 \leq i \leq n''\}$. □

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 Σ^* 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 $\bigcup_{i=1}^{n}(\mathcal{C} \times \{\langle A_i, tt\rangle\})$. □

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 CQDD's $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ and $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ where, for $i \in \{1, 2\}$, $A_i = (Q_i, q^i_{init}, \rightarrow_i, q^i_{fin})$, and V_i is a superset of $X_i = \{x_t : t \in \rightarrow_i\}$. 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: $\exists k \in \mathbb{N}.\ \forall k' \geq k.\ w^{k'} \notin Pref(L(A_2))$

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 $k|w|$.

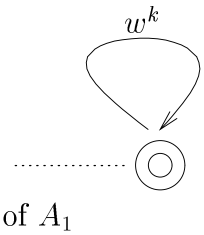
Furthermore we can suppose with Lemma 3.21 that

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

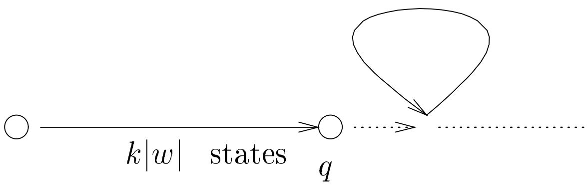
Now, the concatenation of the two DRSA A_1 and A_2 is a RSA (it suffices to glue the state q^1_{fin} with the state q^2_{init}). Let $A_1 \cdot A_2$ be the so obtained automaton. Since CQDD's are defined using deterministic automata, we have to determinize $A_1 \cdot A_2$. For that, we use the standard 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 3 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 state of A_1 and one of the first $k|w|$ states of A_2. This part forms a loop with root $\{q^2_{init}\}$. Because of $w^k \notin Pref(L(A_2))$ 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.

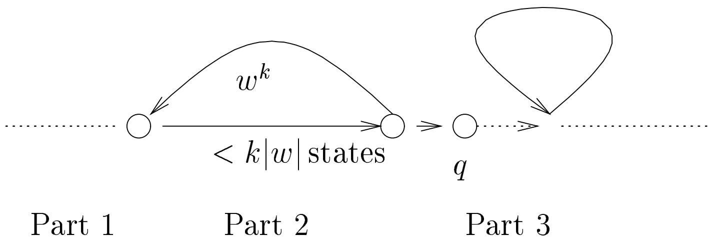
There is exactly one transition from part 1 to part 2, as well as from part 2 to part 3. Hence, the automaton A_d is a DRSA. We illustrate this with a schematic example: Let A_1 be



and A_2:



Then A_d has the form:



The remaining problem is to define a formula for this automaton. Let $\rightarrow_d$ be the set of transitions of A_d and $X_d = \{x_t : t \in \rightarrow_{A_d}\}$. For a transition $t \in \rightarrow_d$ let us denote by $\lambda(t)$ 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 $\lambda(t)$. This fact imposes a constraint expressed by the formula $f_d(V_1 \cup V_2 \cup X_d)$:

$$\bigwedge_{t \in \rightarrow_d} (x_t = \sum_{t' \in \lambda(t)} x_{t'})$$

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 $f_1 \wedge f_2$. Then, the constraints are given by the formula $f_{det}(V_1 \cup V_2 \cup X_d)$:

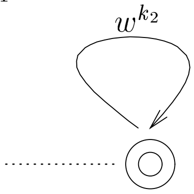
$$[A_1] \wedge [A_2] \wedge f_1 \wedge f_2 \wedge f_d(V_1 \cup V_2 \cup X_d)$$

and it is easy to check that the language $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ is accepted by the CQDD:

$$\mathcal{C}_1 \cdot \mathcal{C}_2 = \{\langle A_d, f_{det} \rangle\}$$

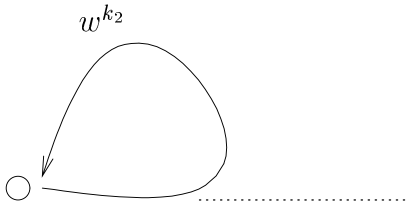
- Case 2: $\forall n \in \mathbb{N} : w^n \in Pref(L(A_2))$
  Again, this condition can be easily checked on the automaton A_2. Since A_2 is a deterministic simple automaton, there exist necessarily $k_1, k_2 \in \mathbb{N}$ with $k_2 > k_1$ and $q_1 \in Q_2$ such that $q^2_{init} \overset{w^{k_1}}{\rightarrow} q_1 \overset{w^{k_2}}{\rightarrow} q_1$ (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 $\mathcal{C}_2$ into two CQDD's:
  · $\mathcal{C}''_2$ where for all automata A'' we have $w^{k_1} \notin Pref(L(A''))$. For these CQDD's we can reason as in Case 1.
  · $\mathcal{C}'_2$ 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

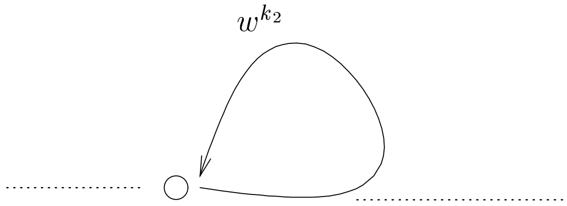


Rest of A_1

and A_2:



Then A_d has the form:



Part 1 Part 2 Part 3

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

$$\bigwedge_{t \in \rightarrow_d} (x_t = \sum_{t' \in \lambda(t)} x_{t'})$$

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

$$[A_1] \wedge [A_2] \wedge f_1 \wedge f_2 \wedge f_d(V_1 \cup V_2 \cup X_d)$$

Then, the language $L(\mathcal{C}_1) \cdot L(\mathcal{C}_2)$ is accepted by the CQDD:

$$\mathcal{C}_1 \cdot \mathcal{C}_2 = \{\langle A_d, f_{det} \rangle\}$$

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. $\square$

3.4.4 Closure under left-quotient

Definition 3.23 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be n-dim multi-languages. The* left-quotient *of $\mathcal{L}_1$ by $\mathcal{L}_2$, denoted by $\mathcal{L}_2^{-1} \cdot \mathcal{L}_1$, is the set $\{\vec{w} \in (\Sigma^*)^n : \exists \vec{w}' \in \mathcal{L}_2.\ \vec{w}'\vec{w} \in \mathcal{L}_1\}$; i.e., the set of multi-words allowing to extend elements of $\mathcal{L}_2$ to elements of $\mathcal{L}_1$.*

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

PROOF. Consider the CQDD's $\mathcal{C}_1 = \{\langle A_1, f_1(V_1)\rangle\}$ and $\mathcal{C}_2 = \{\langle A_2, f_2(V_2)\rangle\}$ where, for $i \in \{1,2\}$, $A_i = (Q_i, q^i_{init}, \rightarrow_i, q^i_{fin})$, 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 $L(A_2) \cap Pref(L(A_1))$ 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^2_{fin}) is a state in $A_1 \times A_2$.

Then, for each $q \in F$, consider the automaton $\overleftarrow{A^q_{1,2}}$ which consists of a copy of $A_1 \times A_2$ with (q^1_0, q^2_0) as initial state and (q, q^2_{fin}) as final state, and let $\overrightarrow{A^q_1}$ 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

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

and

$$\bigcup_{q \in F} L(\overrightarrow{A^q_1}) = L(A_2)^{-1} \cdot L(A_1)$$

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 $\overrightarrow{A^q_1}$. Let $\mathcal{T}(\overrightarrow{A^q_1})$ be the set of transitions of $\overrightarrow{A^q_1}$. We have $\mathcal{T}(\overrightarrow{A^q_1}) \subseteq \rightarrow_1$. 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 $L(\mathcal{C}_2) \cap Pref(L(\mathcal{C}_1))$. Hence, we only accept among elements of $L(\overrightarrow{A^q_1})$, words w (let ρ_w the accepting run of w in $\overrightarrow{A^q_1}$) such that there exists a word w' (the removed prefix) having an accepting run $\rho_{w'}$ in $\overleftarrow{A^q_{1,2}}$ satisfying f_2 (i.e., $\rho_{w'}$ satisfies $f_2 \wedge [\overleftarrow{A^q_{1,2}}]$), 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 $\overleftarrow{A^q_{1,2}}$ or $\overrightarrow{A^q_1}$.

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
$X_q = \{x_t : t \in \mathcal{T}(\overrightarrow{A^q_1})$ and t is in the loop containing $q\}$.
and
$X'_q = \{x_t : t \in \mathcal{T}(\overrightarrow{A^q_1})\}$. 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:

$$\mathcal{C}_2^{-1} \cdot \mathcal{C}_1 = \{\langle \overrightarrow{A^q_1}, f^q_{quot}((X'_q \setminus X_q) \cup U_1 \cup U_2 \cup X_{1,2})\rangle : q \in F\}$$

where $X_{1,2} = \{x_{\langle t_1,t_2\rangle} : t_1 \in\rightarrow_1 \ \wedge t_2 \in\rightarrow_2\}$ and the formula f^q_{quot} is given by:

$$f'_1 \wedge f'_2 \wedge [\overleftarrow{A^q_{1,2}}] \wedge \bigwedge_{t_1\in\rightarrow_1} y_{t_1} = \sum_{t_2\in\rightarrow_2} x_{\langle t_1,t_2\rangle} \wedge \bigwedge_{t_2\in\rightarrow_2} y_{t_2} = \sum_{t_1\in\rightarrow_1} x_{\langle t_1,t_2\rangle}$$

where the formulas f'_1 and f'_2 are defined by:

$$\begin{aligned} f'_1 &= f_1[y_t/x_t : x_t \in X_1 \setminus X'_q][(y_t + x_t)/x_t : x_t \in X_q] \\ f'_2 &= f_2[y_t/x_t : x_t \in X_2] \end{aligned}$$

Generalization to n dimensions: We construct the set of automata $\overleftarrow{A^q_{1,2}}$ and $\overrightarrow{A^q_1}$ 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 CSA's, and hence for CQDD's in particular.

Proposition 3.25 *The emptiness problem is decidable for CSA's.*

PROOF. Consider the CSA $\mathcal{C} = \{\langle A, f\rangle\}$. Clearly, $L(\mathcal{C}) \neq \emptyset$ if and only if the Presburger formula $[A] \wedge f$ is satisfiable. The generalization to n-dimensional CQDD's 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 CQDD's.*

4 Manipulating Sets of Configurations of CFSM's

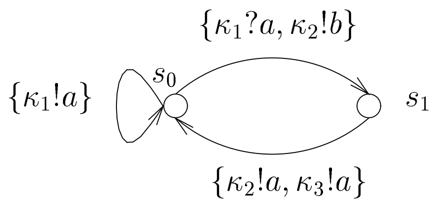
We show in this section how CQDD's can be used to represent sets of configurations of CFSM's. Then, we show that the class of CQDD representable sets of configurations is effectively closed under union, intersection, and the successor function *post*, and that furthermore, this class has decidable emptiness and inclusion problems. This means that CQDD can be used as a representation structure in a forward analysis algorithm. We also show that CQDD's 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 $\mathcal{M} = (S, K, \Sigma, T)$ 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 $\mathcal{M}$ which is represented by the following picture:*



The set of configurations of $\mathcal{M}$ reachable from the configuration $(s_0, \epsilon, \epsilon, \epsilon)$ is given by:

$$\begin{aligned} &\{s_0\} \times \{(a^n, (ba)^m, a^m) : n, m \geq 0\} \\ &\cup \{s_1\} \times \{(a^n, (ba)^m b, a^m) : n, m \geq 0\} \end{aligned} \tag{2}$$

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 $\tau = (s, op, s') \in T$. Then, it is easy to see that if there is a successor of a configuration $(s, \vec{w})$ by τ it is of the form $(s', (in(\tau)^{-1} \cdot \vec{w}) \cdot out(\tau))$. 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, Σ, T) be a CFSM. Then, since CQDD's 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 $\tau = (s, op, s') \in T$, and for every $|K|$-dim CQDD $\mathcal{C}$, we can effectively construct a CQDD $\mathcal{C}'$ such that $\{s'\} \times L(\mathcal{C}') = post_\tau(\{s\} \times L(\mathcal{C}))$. Indeed, it is very easy to see that the two multi-languages $\mathcal{I} = \{in(\tau)\}$ and $\mathcal{O} = \{out(\tau)\}$ are CQDD representable. Then, using Propositions 3.24 and 3.22, we can construct a CQDD $\mathcal{C}'$ such that $L(\mathcal{C}') = (\mathcal{I}^{-1} \cdot L(\mathcal{C})) \cdot \mathcal{O}$. $\square$

Let $\tau = (s, op, s') \in T$. The predecessor of a configuration $(s', \vec{w})$ by τ is $(s, in(\tau) \cdot \vec{w} \cdot out(\tau)^{-1})$. Then, if we represent a configuration by taking the reverse of each queue contents, it can be seen that the predecessor of $(s', \vec{w}^R)$ is $(s, (out(\tau)^R)^{-1} \cdot \vec{w}^R \cdot in(\tau)^R)$. 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, Σ, 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 $\{s'\} \times L(\mathcal{C}')^R = pre_\tau(\{s\} \times L(\mathcal{C})^R)$. Indeed, let $\mathcal{I} = \{in(\tau)^R\}$ and $\mathcal{O} = \{out(\tau)^R\}$. These two multi-languages are obviously CQDD representable. Then, a CQDD $\mathcal{C}'$ such that $L(\mathcal{C}') = (\mathcal{O}^{-1} \cdot L(\mathcal{C})) \cdot \mathcal{I}$ can be constructed using Propositions 3.24 and 3.22. $\square$

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 θ 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^*_θ function).

5.1 Main result

Theorem 5.1 *For every CQDD representable set of configurations C, and every cycle θ, 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, Σ, T) be a CFSM. We show that for every $s \in S$, and for every cycle θ 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 CQDD's $\mathcal{C}$ 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 θ a certain number of times. This number is represented by a variable x_θ. 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 θ after x_θ iterations as if the system had only one queue κ_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_θ 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_θ times the cycle θ.
- 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 $h = \bigwedge_{i=1}^{|K|} g_i \wedge f$. 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_θ, which means that these contents correspond to the same number of iterations of the cycle θ.

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 [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 θ and a queue κ, and let *in* and *out* denote the input and output of the system by θ on the queue κ. The proof consists of an analysis of the effect of θ on the contents of κ, depending on the input *in*, the output *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 *in* or *out* are the empty word. In the case where $in = \epsilon$, starting from a configuration w the effect of repeating x_θ times the cycle θ is to concatenate the word out^{x_θ} to the right, which gives $w.out^{x_\theta}$. Clearly this effect can be constructed using the operation of concatenation between CQDD's. In the case where $out = \epsilon$, the effect of repeating x_θ times the cycle θ starting from a word w is given by $(in^{x_\theta})^{-1}.w$. Thus, this effect can be constructed using the operation of left-quotient between CQDD's.

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

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

Let us start by the case where $|in| \leq |out|$. Then, the problem is to determine whether θ 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 in, 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 $out = c.x.u$, where c is the subword of in such that $in = x.c$, and $c.x.u = u.c.x$. Moreover, we show that in this case, starting from a configuration $in^k.x$ the reached configuration after x_θ iterations of θ is $in^k.x.u^{x_\theta}$. This means that the effect of repeating θ is given by concatenating the language $\{u^n | n \geq 0\}$ such that $n = x_\theta$, which is clearly CQDD definable.

Let us consider an example. Let $in = ba$, $out = abab$, 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 $in.x = bab$, the reachable configuration after n iterations of θ is $in.x.u^n = bab(ab)^n$. Indeed, we have

$$\begin{aligned}
& in.x = bab \\
& \quad \rightsquigarrow x.out = babab = in.x.u \\
& \quad\quad \rightsquigarrow x.u.out = x.out.u = bababab = in.x.u^2 \\
& \quad\quad\quad \rightsquigarrow x.u^2.out = x.out.u^2 = babababab = in.x.u^3 \rightsquigarrow \cdots
\end{aligned}$$

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

Now, consider the case where $|in| > |out|$. 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 $in = abab$, $out = ba$, and $x = a$. Then, starting from $in^5.x = (abab)^5.a$ we have the following sequence of reachable configurations:

$$
\begin{aligned}
& in^5.x = (abab)^5.a \\
& \quad \rightsquigarrow in^4.x.out = (abab)^4.a.ba \\
& \quad\quad \rightsquigarrow in^3.x.out^2 = (abab)^3.a.(ba)^2 \\
& \quad\quad\quad \ldots \\
& \quad\quad\quad\quad \rightsquigarrow^* x.out^5 = a.(ba)^5 = (abab)^2.a.ba = in^2.x.out \\
& \quad\quad\quad\quad\quad \ldots \\
& \quad\quad\quad\quad\quad\quad \rightsquigarrow^* x.out^3 = a.(ba)^3 = (abab).a.ba = in.x.out \\
& \quad\quad\quad\quad\quad\quad\quad \rightsquigarrow x.out^2 = ababa = in.x \\
& \quad\quad\quad\quad\quad\quad\quad\quad \rightsquigarrow x.out = aba
\end{aligned}
$$

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

We prove that, roughly speaking, under some conditions on in, out, and x, all the reachable configurations from $in^k.x$ that are of length greater than a certain bound k' (defined in terms of the lengths of in and out) are of the form $in^p.x.out^q$, where p and q are related with x_θ and k by linear constraints. Moreover, we show that for every k, there is a successor of $in^k.x$ 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 θ, 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 θ can be executed at most k'' times. $\square$

5.2 Detailed proof of Theorem 5.1

5.2.1 Analysis of the effect of a cycle

In this section we analyze the effect of the repeated execution of one cycle θ 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 analyze.

Let κ_i be a queue. For each queue in the system the cycle θ 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

$$|out_i| \geq |in_i|, out_i \neq \epsilon \ \wedge \ in_i \neq \epsilon$$

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 $c \neq \epsilon$.

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

$$\exists u, c_1, c_2 \in \Sigma^* \, \exists k_1 \geq 0. \ out_i = c.x.u, u = (c.x)^{k_1}.c_1 \ \wedge \ c.x = c_1.c_2 \tag{3}$$
$$\textit{and} \ \ c.x.u = u.c.x \tag{4}$$

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 $in_i^k.x$ with $k > 0$ the cycle can be executed perpetually. In this case, the queue contents after n executions is given by $in_i^k.x.u^n$.*

PROOF. "$\Rightarrow$": Let x be inc-repeating. We prove by induction that starting from a queue contents $in_i^k.x$ with $k > 0$ the cycle can be executed perpetually and the contents after n executions is given by $in_i^k.x.u^n$.

- Induction basis:
  Given $in_i^k.x$, an execution of the cycle yields $in_i^{k-1}.x.out_i$. Using (3) this is equal to $in_i^{k-1}.x.c.x.u$. Because $in_i = x.c$ this is equal to $in_i^k.x.u$.
- Induction step:
  $in_i^k.x$ can be executed n times and yields $in_i^k.x.u^n$. Then, another execution yields $in_i^{k-1}.x.u^n.out_i$. By Lemma 5.3 this is equal to $in_i^{k-1}.x.out_i.u^n = in_i^{k-1}.x.c.x.u.u^n = in_i^k.x.u^{n+1}$.

"$\Leftarrow$": Starting from a queue contents $in_i^k.x$, 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

$$x.out_i^{k+k'} = in_i^{k'}.x' \tag{5}$$

Since $in_i = x.c$, (5) is equivalent to

$$out_i^{k+k'} = (c.x)^{k'-1}.c.x' \tag{6}$$

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

$$\exists u, c_1, c_2 \in \Sigma^* \; \exists k_1 \geq 0 \;\; out_i = c.x.u \wedge u = (c.x)^{k_1}.c_1 \;\wedge\; c.x = c_1.c_2 \tag{7}$$

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

$$(c.x.(c.x)^{k_1}.c_1)^{k+k'} = (c.x)^{k'-1}.c.x' \tag{8}$$

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

$$c.x.\overbrace{c.x.\ldots.c.x}^{k_1 \text{ times}}.c_1.c.x \cdots = c.x.\overbrace{c.x.\ldots.c.x}^{k_1 \text{ times}}.c.x.c_1.c_2 \cdots$$

This implies $c_1.c.x = c.x.c_1$. The equality

$$c.x.u = u.c.x \tag{9}$$

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 $max\{k_3 : \exists\, d \in \Sigma^*.\ out_i = (c.x)^{k_3}.d\}$ 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 $x.out_i^k$ the cycle can be executed at most $k_2 = max\{k_3 + 3 : \exists\, d \in \Sigma^*.\ out_i = (c.x)^{k_3}.d\}$ times.*

Analysis of a cycle with decreasing size of contents

Here we consider in_i and out_i with the following conditions

$$|in_i| > |out_i|, out_i \neq \epsilon \ \wedge \ in_i \neq \epsilon \tag{10}$$

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 $c \neq \epsilon$.

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

$$\exists\, c_1, c_2 \in \Sigma^*\ \exists\, k_1 > 0.\ c.x = out_i^{k_1}.c_1 \ \wedge \ c_1.c_2 = out_i \tag{11}$$

$$\textit{and} \ \ c_1.c_2 = c_2.c_1 \tag{12}$$

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

$$in_i^{|out_i|}.x = x.out_i^{|in_i|} \tag{13}$$

PROOF. (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

$$out_i^{|c_1|} = c_1^{|out_i|} \tag{14}$$

Thus, $in_i^{|out_i|}.x = (x.c)^{|out_i|}.x = x.(c.x)^{|out_i|} = x.(out_i^{k_1}.c_1)^{|out_i|}$
$= x.out_i^{k_1|out_i|}.c_1^{|out_i|} = x.out_i^{k_1|out_1|+|c_1|} = x.out_i^{|in_i|}$. □

A simple corollary of Lemma 5.7:

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

$$\forall\ d_1 > 0, in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|} \tag{15}$$

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 $x.out_i^k$ with $k > 0$ the cycle can be executed at most $k_2 = max\{k_3 + 3 : \exists d \in \Sigma^*.\ c.x = out_i^{k_3}.d\}$ 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 $x.out_i^k$. For each $k' < k$, the cycle can be executed k' times. This implies that there exists a $x' \in \Sigma^*$ with

$$x.out_i^{k+k'} = in_i^{k'}.x' \tag{16}$$

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

$$out_i^{k+k'} = (c.x)^{k'-1}.c.x' \tag{17}$$

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

$$\exists\ c_1, c_2 \in \Sigma^*\ \exists\ k_1 > 0\ \ c.x = out_i^{k_1}.c_1\ \wedge\ c_1.c_2 = out_i \tag{18}$$

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

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

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

$$\overbrace{c_1.c_2 \ldots c_1.c_2}^{k_1 \text{ times}}.c_1.c_2.c_1 \cdots = \overbrace{c_1.c_2 \ldots c_1.c_2}^{k_1 \text{ times}}.c_1.c_1.c_2 \cdots \tag{20}$$

Thus,

$$c_1.c_2 = c_2.c_1 \tag{21}$$

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

  $$\exists\, c_1, c_2 \in \Sigma^* \;\exists\, k_1 > 0.\; c.x = out_i^{k_1}.c_1 \;\wedge\; c_1.c_2 = out_i$$

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

  $$\forall\, d_1 > 0, in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|}$$

  Therefore the constant can not exist because $d_1|in_i|$ is not bounded and the equation $in_i^{d_1|out_i|}.x = x.out_i^{d_1|in_i|}$ shows that the cycle can be executed.

□

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

Proposition 5.10 *If the word x is dec-repeating, then for queue contents $in_i^k.x$ with $k \geq 4n_1$, if the queue contents of a successor after n' executions of the cycle has length $\geq 3n_1n_2 + |x|$, then it has the form*

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-d_3(n_2-n_1)+d_4}$$

where

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

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 $in_i^k.x = in_i^{d_1n_1+d_2}.x$ are given by $in_i^{d_1n_1+d_2-n'}.x.out_i^{n'}$. Since $n_1 + d_2 \geq n'$ and by Corollary 5.8 we have

  $$in_i^{d_1n_1+d_2-n'}.x.out_i^{n'} = in_i^{n_1+d_2-n'}.x.out_i^{(d_1-1)n_2+n'}$$

  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 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

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-d_3(n_2-n_1)+d_4}$$

Because the length of the obtained contents is greater than $3n_1n_2+|x| = 2n_1|in_i|+|x|+n_2|out_i|$, 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

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-d_3(n_2-n_1)+d_4} = \\ in_i^{2n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-(d_3+1)(n_2-n_1)-n_1+d_4}$$

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

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-(d_3+1)(n_2-n_1)+d_4}$$

□

5.2.2 Closure under $post_\theta^*$

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 $post_\theta^*$ for each cycle θ in the transition graph of the system. To prove this theorem it suffices to show the following proposition:

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

PROOF. Since *post* distributes w.r.t. union, it suffices to reason on CQDD's $\mathcal{C}$ consisting of one accepting component $\langle \mathcal{A}, f\rangle$. 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 θ on each queue κ_i after x_θ 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_θ for all the queues.

Let us fix a queue κ_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**: $out_i = \epsilon$ and $in_i = \epsilon$
  In this case there is neither reading nor writing of the queue. Therefore the contents of the queue does not change. Thus,
$$\mathcal{D}_i = \{\langle A_i, tt\rangle\}$$
- **Case 2**: $out_i = \epsilon$ and $in_i \neq \epsilon$
  In this case, there is nothing written into the queue, but the cycle reads from it. in_i can be read from the queue, if the queue contents starts with in_i. There can be several in_i in the queue contents. Therefore, for contents of the form $in_i^k.x \in L(A_i)$ for some $k \in \mathbb{N}$ and $x \in \Sigma^*$, $in_i^{k-x_\theta}.x$ is the new contents provided that $k \geq x_\theta$. Therefore,
$$\mathcal{D}_i = (\langle B_{in_i^*}, x_{t_B} = x_\theta\rangle)^{-1} \cdot \langle A_i, tt\rangle$$
- **Case 3**: $in_i = \epsilon$ and $out_i \neq \epsilon$
  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 $y \in L(A_i)$, $y.out_i^{x_\theta}$ is the new contents. Hence,
$$\mathcal{D}_i = \langle A_i, tt\rangle \cdot \langle B_{out_i^*}, x_{t_B} = x_\theta\rangle$$
- **Case 4**: $|out_i| \geq |in_i|, out_i \neq \epsilon$ and $in_i \neq \epsilon$
  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 $in_i^k.x$, where x is not a prefix of in_i and in_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 $in_i^{k-x_\theta}.x.out_i^{x_\theta}$ provided that $k \geq x_\theta$. Hence,
$$\mathcal{D}_i = (((\langle B_{in_i^*}, x_{t_B} = x_\theta\rangle)^{-1} \cdot \langle A_i, tt\rangle) \cdot \langle C_{out_i^*}, x_{t_C} = x_\theta\rangle$$
  $L(\mathcal{D}_i)$ 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 $in_i^k.x$, where x is a possibly empty prefix of in_i with $|x| < |in_i|$. In this case, the cycle can be executed at least k times and after that, the form of out_i determines if the cycle can still be executed. Let $c \in \Sigma^*$ such that $in_i = x.c$. k executions

of the cycle give a contents of the form $x.out_i^k$ 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 $in_i^k.x = (x.c)^k.x$ with $k > 0$, the cycle can be executed perpetually and after x_θ executions, the queue contents is given by $in_i^k.x.u^{x_\theta}$. 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 in_i. $\mathcal{E}_x$ is constructed as follows:
  If $u = \epsilon$ then
  $$\mathcal{D}_i = \{\langle A_i, tt\rangle\}$$
  otherwise, if the cycle can be executed once starting from x, then
  $$\mathcal{E}_x = (((\langle B_{in_i^*}, tt\rangle \cdot \langle A_x, tt\rangle) \times \langle A_i, tt\rangle) \cdot \langle C_{u^*}, x_{t_C} = x_\theta\rangle$$
  otherwise
  $$\mathcal{E}_x = (((\langle B_{in_i^*}, tt\rangle \cdot \langle A_{in_i.x}, tt\rangle) \times \langle A_i, tt\rangle) \cdot \langle C_{u^*}, x_{t_C} = x_\theta\rangle$$
- **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 $in_i^{k-n'}.x.out_i^{n'}$ with $k \geq n'$. With Corollary 5.5 there is a constant k_2 such that starting from a contents of the form $x.out_i^{n'}$ 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 $in_i^{k-n'}.x.out_i^{n'}$ 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 in_i. $\mathcal{E}_x$ is constructed as follows:
  First of all, let
  $$\mathcal{F}_x = ((((\langle B_{in_i^*}, x_{t_B} = n'\rangle) \cdot \langle A_x, tt\rangle)^{-1} \cdot \langle A_i, tt\rangle) \cdot \langle C_{out_i^*}, x_{t_C} = n'\rangle$$
  This gives us all the successors of the form $in_i^{k-n'}.x.out_i^{n'}$. For $n' = x_\theta$ 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 $x.out_i^{n'}$. 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 $\{s\} \times L(\mathcal{F}_x^j) = post_\theta^j(\{s\} \times L(\mathcal{F}_x))$. This can be done due to the fact that CQDD representable sets are closed under $post_\tau$ 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_θ of executions of the cycle. This number is given as $x_\theta = n' + j$. Thus,

$$\mathcal{E}_x = \bigcup_{j=0}^{k_2} add(\mathcal{F}_x^j, x_\theta = n' + j)$$

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

- **Case 5**: $|in_i| > |out_i|, out_i \neq \epsilon$ and $in_i \neq \epsilon$

  For this case we use the results of Section 5.2.1. There are two subcases:
- **Case 5.1**:

  The contents have the form $in_i^k.x$, where x is not a prefix of in_i and in_i is not a prefix of x. Then, the set of reachable contents is given by $in_i^{k-x_\theta}.x.out_i^n$ with $k \geq x_\theta$. Hence, $\mathcal{D}_i$ is calculated like in case 4.1.
- **Case 5.2**:

  The contents have the form $in_i^k.x$, where x is a possibly empty prefix of in_i with $|x| < |in_i|$.
- **Case 5.2.1**: The word x is dec-repeating:

  Let $c \in \Sigma^*$ with $in_i = x.c$. We use Proposition 5.10. Let $n_1 = |out_i|$ and $n_2 = |in_i|$. For each queue contents $in_i^k.x$ with $k \geq 4n_1$, all its successors with length $\geq 3n_1n_2 + |x|$ after n' executions of the cycle are given by

$$in_i^{n_1+d_2-d_4}.x.out_i^{(d_1-1)n_2-d_3(n_2-n_1)+d_4} \tag{22}$$

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

Each $in_i^k.x$ 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 $in_i^k.x$, 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 θ.

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 in_i. $\mathcal{E}_x$ is constructed as follows:

We construct a formula to calculate the possible values of k in states of the form $in_i^k.x$. Let $\mathcal{F}_x = (\langle B_{in_i^*}, x_{t_B} = k\rangle \cdot \langle A_x, tt\rangle) \times \langle A_i, tt\rangle$. 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

$f_x \wedge n' = d_3 n_1 + d_4 \ \wedge \ k = d_1 n_1 + d_2 \ \wedge \ 0 < d_4 \leq n_1$

$\wedge \ 0 \leq d_2 < n_1 \ \wedge \ (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \geq 0$

Let

$\mathcal{G}_x = \langle B'_{in_i^*}, x_{t_{B'}} = n_1 + d_2 - d_4 \rangle \cdot \langle A_x, tt \rangle \cdot$

$\langle C_{out_i^*}, x_{t_C} = (d_1 - 1)n_2 - d_3(n_2 - n_1) + d_4 \wedge g_x \rangle$

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 $\mathcal{E}_x$.

- **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:

$$\forall \langle B_1, g_1 \rangle \in \mathcal{D}_1 \cdots \forall \langle B_{|K|}, g_{|K|} \rangle \in \mathcal{D}_{|K|}.\ \langle (B_1, \ldots, B_{|K|}), \bigwedge_{i=1}^{|K|} g_i \wedge f \rangle \in \mathcal{C}'$$

Theorem 5.12 *For every CQDD reverse representable set of configurations C, and every cycle θ, the set of configurations $pre_\theta^*(C)$ is CQDD reverse representable and effectively constructible.*

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

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. $D \cap post^*(I) = \emptyset$,
P2. $I \cap pre^*(D) = \emptyset$.

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., $post^*(C)$ and $pre^*(C)$. By definition, for $\phi \in \{post, pre\}$, we have:

$$\phi^*(C) = \bigcup_{i \geq 0} X_i$$

where

$$\begin{aligned} X_0 &= C \\ X_{i+1} &= X_i \cup \phi(X_i) \quad \text{for every } i \geq 0 \end{aligned}$$

In the case $\phi = post$ (resp. $\phi = pre$), if C is CQDD representable (resp. reverse representable), it can be deduced from Theorems 4.3, 4.4, and 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 CFSM's, 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 on 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 Θ, add at each step the set of successors (or predecessors) by each cycle in Θ. 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:

$$\begin{aligned} Y_0 &= C \\ Y_{i+1} &= Y_i \cup \phi(Y_i) \cup \textstyle\bigcup_{\theta \in \Theta} \phi_\theta^*(Y_i) \quad \text{for every } i \geq 0 \end{aligned}$$

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 Θ.

Therefore, using Theorems 4.3, 4.4, 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 (Θ, C):
  $Y := C$;
  repeat
    $Y' := Y$;
    $Y := Y \cup \phi(Y) \cup \bigcup_{\theta \in \Theta} \phi^*_\theta(Y)$
  until $Y' \subseteq Y$;
  return(Y)
end Reachability

A variety of reachability algorithms can be derived from the generic algorithm above by determining adequate strategies for choosing the set of cycles Θ (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 (QDD's) can be used in this case and allows to analyze some significant systems. But considering QDD's 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 Θ 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 (CQDD's) combining finite-state automata with counting constraints expressed in Presburger arithmetics. We have shown that these

[2] In the definition of QDD's, 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 DRSA's.

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 CQDD's 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, CQDD's 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, CQDD's are expressive enough for 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

[1] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127:91–101, 1996.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[3] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] G.V. Bochmann. Finite State Description of Communication Protocols. *Computer Networks*, 2, October 1978.

[5] B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. In *CAV'96*. Lecture Notes in

Computer Science, Vol. 1102, Springer-Verlag, 1996.

[6] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Static Analysis Symposium*. Lecture Notes in Computer Science, Vol. 1302, Springer-Verlag, 1997.

[7] B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In *CAV'94*. Lecture Notes in Computer Science, Vol. 818, Springer-Verlag 1994.

[8] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. Lecture Notes in Computer Science, Vol. 1243, Springer-Verlag, 1997.

[9] A. Bouajjani and O. Maler. Reachability Analysis of Pushdown Automata. In *Infinity'96*. Technical Report MIP-9614, Univ. Passau, 1996.

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

[11] O. Burkart and Y.M. Quemener. Model-Checking of Infinite Graphs Defined by Graph Grammars. In *Infinity'96*. Technical Report MIP-9614, Univ. Passau, 1996.

[12] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable Channels Are Easier to Verify Than Perfect Channels. *Information and Computation*, 124(1):20–31, 1996.

[13] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Internal report, LSV, ENS-Cachan, 1996.

[14] A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*. Technical Report Univ. Uppsala, 1997.

[15] Thierry Jéron and Claude Jard. Testing for Unboundedness of Fifo Channels. *Theoretical Computer Science*, 113(1):93–117, 1993.

[16] S. Melzer and J. Esparza. Checking System Properties via Integer Programming. In *ESOP'96*. Lecture Notes in Computer Science, Vol. 1058, Springer-Verlag, 1996.

[17] Jan Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Protocol Specification, Testing, and Verification VII*, pages 207–219. Elsevier Science Publishers (North-Holland), 1987.

[18] W. Peng and S. Puroshothaman. Data Flow Analysis of Communicating Finite State Machines. *ACM Transactions on Programming Languages and Systems*, 13(3):399–442, July 1991.

[19] Y.M. Quemener and T. Jéron. Finitely Representing Infinite Reachability Graphs of CFSMs with Graph Grammars. In *FORTE/PSTV'96*. Chapman and Hall, 1996.