Reachability Sets of Parametrized Rings As Regular Languages

Laurent Fribourg

LSV, E.N.S. Cachan & CNRS
61 av. Pdt Wilson, 94235 Cachan - France
fribourg@lsv.ens-cachan.fr

Hans Olsén

IDA, Linköping University
S-58183 Linköping - Sweden
hanol@ida.liu.se

Abstract

We present here a method for deriving a regular language that characterizes the set of reachable states of a given parametrized ring (made of N of identical components). The method basically proceeds in two steps: first one generates a regular language L by inductive inference from a finite sample of reachable states; second one formally checks that L characterizes the whole set of reachable states.

1 Introduction

During these last years, several kinds of methods have been explored in order to prove a property P about a ring of N identical finite-state processes irrespective of its size N. They are essentially three. The first is by *induction* (see, e.g., [20,19,13]), but often relies on human help for the introduction of appropriate 'lemmas' or 'invariants'. The second is by *reduction* to the verification problem for a fixed small size (e.g., N=2) (see, e.g., [10,17]), but works only for restrictive classes of rings. The third is by *abstraction* (see, e.g., [8,18,15]): an abstract model of the ring is provided depending on the property P to be proved, then an invariant property is generated from which P follows. In this kind of works, property P concerns the global state of the system. This state is an N-tuple $(q^1, ..., q^N)$, and viewed as a word $q^1...q^N$ defined on the alphabet Σ of states of the individual components. The property P is expressed as an inclusion relationship into a regular language. For example (see [8]), for expressing the property of mutual exclusion algorithm for processes on a ring, it suffices to state that the configurations of the ring belong to the regular language $n^*\ c\ n^*$ where c (resp. n) means that the component is (resp. is not) in the critical section. Such methods are *approximate* because they use abstraction, and are *backward-oriented* (or *top-down*) because they use the property P to be proved as a starting point.

Article published in Electronic Notes in Theoretical Computer Science (1997) 1–12

Here in contrast, we are interested by *exact* and *forward-oriented* (or *bottom-up* methods), i.e., computation methods which start from the initial state. Regular languages will be used here, not to express abstract properties of the system, but to characterize the exact set of reachable states. This fourth kind of method is originally due to Dijkstra [9]. Our aim is to *mechanize* Dijkstra's method. The main problem is to reproduce Dijkstra's "guess" of regular languages resulting from the iterative (unlimited) application of various system transitions. In order to achieve such a goal, we use classical automatic methods of inductive inference that proceed by generalization over finite samples of examples. We also provide a (mechanizable) equality test allowing to check that the result of the inductive inference step is correct.

2 Preliminaries

In the following the transitions of the concurrent system will be modelized as rewrite rules. We assume given an alphabet Σ of letters, and an alphabet $\mathcal{V}$ (disjoint from Σ) of variable symbols. The variable symbols are X, Y with possible indices. Words on $(\Sigma \cup \mathcal{V})^*$ will be denoted by lower case greek letters $\lambda, \mu, \rho, \ldots$ The empty word will be denoted ϵ. A transition will be characterized by a rewrite rule of the form $\lambda \rightarrow \rho$ where λ and ρ denotes words on $(\Sigma \cup \mathcal{V})^*$. A word μ rewrites to μ' via R iff there exists a rule $\lambda \rightarrow \rho$ in R such that $\mu = f(\lambda)$ and $\mu' = f(\rho)$ for some substitution f (i.e., some mapping from V to Σ^*). We denote by $\xrightarrow{R}$ the one-step reduction relation defined by the set of transitions R. We denote $\Delta_R(L)$ the language generated from L by applying $\xrightarrow{R}$, i.e. the set: $\{\mu' \mid \mu \xrightarrow{R} \mu' \text{ for some } \mu \in L\}$. We denote by $\xrightarrow{R^*}$ (resp. $\xrightarrow{\{R,id\}}$) the reflexive transitive closure (resp. reflexive closure) of $\xrightarrow{R}$. We denote by $\Delta_R^*(L)$ the language generated from L by applying $\xrightarrow{R^*}$, i.e. the set: $\{\mu' \mid \mu \xrightarrow{R^*} \mu'$ for some $\mu \in L\}$. The set $\Delta_R^*(L)$ is equivalently defined as $\bigcup_{k \geq 0} \Delta_R^k(L)$ where $\Delta_R^k(L)$ is defined by: $\Delta_R^0(L) = L$ and $\Delta_R^{k+1}(L) = \Delta_R(\Delta_R^k(L))$.

We say that a language L is *R-invariant* iff: $\Delta_R(L) \subset L$. We say that R is *noetherian* if there is no infinite sequence of words $\mu_1, \mu_2, \mu_3, \ldots$ of Σ^* such that $\mu_1 \xrightarrow{R} \mu_2 \xrightarrow{R} \mu_3 \xrightarrow{R} \ldots$ The symbol R^{-1} will denote the rewrite system obtained by exchanging the lefthand side and righthand side of the rules of R. (We have $\mu \xrightarrow{R} \mu'$ iff $\mu' \xrightarrow{R^{-1}} \mu$.) The concatenation symbol is '.', but will be often omitted in the following.

3 Parametrized Rings

We consider a system organized as a ring of N identical components, which are finite state automata. The set of states of each component is defined over a finite alphabet Σ. The global state of the ring will be characterized as a *word* over Σ^N. The i-th letter of the word corresponds to the state of the i-th component. There will be two types of transitions affecting the states of the system. The *internal transitions* are transitions which affect only one component at a time. The *external transitions* are transitions which affect two contiguous components at a time. (The 1st and N-th component are considered as contiguous due to the ring structure.) We are interested in characterizing the *reachability set* of the ring, that is the set of global states that are obtained by applying repeatedly any sequence of transitions, starting from a given initial state for the ring of length N. Actually, instead of reasoning on a ring of a determinate length N, we will consider the union of all the rings for all length $N = 2, 3, 4, \ldots$. Therefore we will reason on words of Σ^*, rather than on words of Σ^N. The initial state will be characterized not by a word of length N, but by a regular language L_{init} (e.g., of the form $a_0 b_0^+$). An internal transition will be characterized by a rewrite rule of the form $(XaY) \rightarrow (Xa'Y)$, where a (resp. a') denote the state of the modified component of the ring before (resp. after) application of the transition. An external transition will be characterized by a couple of rewrite rules $\langle r, r' \rangle$, where r is a rewrite rule of the form $(XabY) \rightarrow (Xa'b'Y)$ and r' is the rewrite rule $(bXa) \rightarrow (b'Xa')$. In r, letters a, b (resp. a', b') denote the states of two modified contiguous components of the ring before (resp. after) application of the transition. In r', the letters a, b (resp. a', b') denote the states of the last and first components before (resp. after) application of the transition. The set of rewrite rules associated to the concurrent system will be denoted by R. The reachability set of the system is thus formalized as $\Delta_R^*(L_{init})$.

Note 1 *An* array *of N identical components can be modelized similarly to a ring using internal and external transitions. The difference is that external transitions will simply consist of rules of the form: $(XabY) \rightarrow (Xa'b'Y)$. (There will be no additional rules of the form: $(bXa) \rightarrow (b'Xa')$ since the first and last elements of an array are not in communication.)*

4 Generating Regular Languages

A basic underlying claim here is that the reachability sets of the parametrized rings (or arrays) can be often expressed as regular languages. In order to construct effectively such regular languages, one will appeal to methods of *inductive inference* (or *generalization*) for regular languages. Typically, one proceeds as follows:

(1) generate a finite set $S^{<k}$ of words of length than k that correspond to the reachable states for rings of length less than k.
(2) use a generalization method for constructing a regular language L from $S^{<k}$.

In order to generate $S^{<k}$ in step 1, one can consider the set $L_{init}^{<k}$ of words of L_{init} of length less than k, and construct the set $\Delta_R^*(L_{init}^{<k})$. (This set is finite and made of words of length less than k since the rules of R are length-preserving.)

In step 2, as a generalization method, one can either use a general synthesis procedure by inductive inference (see [2,4]; *cf.* [3,5]), or one may apply a method more adapted to ring problems as developed in [12]. Let us briefly explain the latter method. One constructs the finite set $cont(S^{<k})$ of couples of letters that are contiguous in words of $S^{<k}$, i.e.:

$$cont(S^{<k}) = \{(a,b) \mid \mu.a.b.\mu' \in S \ \text{ for some } \mu, \mu' \in \Sigma^*\}$$

One constructs also the finite set $cont'(S^{<k})$ of couples of last and first letters of words of $S^{<k}$, i.e.:

$$cont'(S^{<k}) = \{(a,b) \mid b.\mu.a \in S \ \text{ for some } \mu \in \Sigma^*\}$$

One then constructs the language $L_{local}(S^{<k})$ of words where every couple of contiguous letters belongs to $cont(S^{<k})$, i.e.:

$$\lambda \in L_{local}(S^{<k}) \ \text{ iff } \quad \forall a,b \in \Sigma \ \forall \mu, \mu' \in \Sigma^* : \ \lambda = \mu.a.b.\mu' \Longrightarrow (a,b) \in cont(S^{<k}).$$

Likewise, one constructs the language $L'_{local}(S^{<k})$ of words whose last and first letters form a couple belonging to $cont'(S^{<k})$, i.e.:

$$\lambda \in L'_{local}(S^{<k}) \ \text{ iff } \quad \forall a,b \in \Sigma \ \forall \mu, \mu' \in \Sigma^* : \ \lambda = b.\mu.a \Longrightarrow (a,b) \in cont'(S^{<k}).$$

It can be seen that $L_{local}(S^{<k})$ and $L'_{local}(S^{<k})$ are regular languages. Finally one considers a third regular language L_{global} that is specific of the ring system under consideration. A typical example is a regular language expressing a mutual exclusion property of the form $n^*.c.n^*$ (*cf.* introduction). The language L synthesized by our method is the intersection $L_{local}(S^{<k}) \cap L'_{local}(S^{<k}) \cap L_{global}$. An example of application of this method is given in section 7.

We will suppose in the following that we are given a procedure, called $guess$, that, from R and a regular language L_{init} as inputs, generates regular languages as candidates for $\Delta_R^*(L_{init})$. The (nondeterministic) output of such a procedure will be denoted $guess(L_{init}, R)$. Once a regular language has been generated, we will perform an equality test in order to formally verify that the language coincides with the reachability set of the system. If it is not the case, then the procedure $guess$ is called again until the output language satisfy the equality test. This is explained in the next section.

5 Testing Regular Languages

Our test is based on the following property:

Proposition 1 *Let L and L' be two sets of words over Σ^*, and R a rewrite system. Suppose that R^{-1} is noetherian. Then:*

$$L' = \Delta_R^*(L) \quad \textit{iff} \quad L' = \Delta_R(L') \cup L.$$

PROOF.

Let us show $L' = \Delta_R^*(L) \Longrightarrow \Delta_R(L') \cup L \subset L'$.
Since, by definition, $\Delta_R^*(L)$ is a superset of L, it suffices to show: $L' = \Delta_R^*(L) \Longrightarrow \Delta_R(L') \subset L'$. So suppose $L' = \Delta_R^*(L)$ and $x \in \Delta_R(L')$, and let us show that $x \in L'$. We have: $x \in \Delta_R(L')$ iff $x' \xrightarrow{R} x$ for some $x' \in L' = \Delta_R^*(L)$. Since $x' \in \Delta_R^*(L)$, we have: $x' \in \Delta_R^k(L)$ for some $k \geq 0$. Therefore: $x \in \Delta_R^{k+1}(L)$. Hence $x \in \Delta_R^*(L)$, i.e. $x \in L'$, q.e.d.

Let us show now $L' = \Delta_R^*(L) \Longrightarrow L' \subset \Delta_R(L') \cup L$.
Suppose $L' = \Delta_R^*(L)$ and $x \in L'$, and let us show that $x \in \Delta_R(L') \cup L$. Since $x \in \Delta_R^*(L)$, we have: $x \in \Delta_R^k(L)$ for some $k \geq 0$. Either $k = 0$ or $k > 0$. In case $k = 0$, we have: $x \in \Delta_R^0(L)$, i.e.: $x \in L$. In case $k > 0$, we have: $x' \xrightarrow{R} x$ for some $x' \in \Delta_R^{k-1}(L)$. Therefore $x' \in \Delta_R^*(L) = L'$. It follows: $x \in \Delta(L')$. In both cases, we have: $x \in \Delta(L') \cup L$, q.e.d.

We have thus shown: $L' = \Delta_R^*(L) \Longrightarrow L' = \Delta_R(L') \cup L$. This means that $\Delta_R^*(L)$ is a solution in L' of equation $L' = \Delta_R(L') \cup L$. Let us now show that this solution is unique, i.e.: $L'_1 = \Delta_R(L'_1) \cup L \wedge L'_2 = \Delta_R(L'_2) \cup L$ imply $L'_1 = L'_2$. By symmetry, it suffices to show that $L'_1 = \Delta_R(L'_1) \cup L \wedge L'_2 = \Delta_R(L'_2) \cup L$ imply $L'_1 \subset L'_2$, i.e. the following proposition $\mathit{prop}(x)$:

$$L'_1 = \Delta_R(L'_1) \cup L \wedge L'_2 = \Delta_R(L'_2) \cup L \wedge x \in L'_1 \quad \Longrightarrow \quad x \in L'_2.$$

Let us prove $\mathit{prop}(x)$ by noetherian induction on x. Suppose that $x \in L'_1$, and let us show that $x \in L'_2$. Since $x \in L'_1$ and $L'_1 = \Delta_R(L'_1) \cup L$, then either x belongs to L or to $\Delta_R(L'_1)$. If x belongs to L, then x belongs to $L'_2 (= \Delta_R(L'_2) \cup L)$, and we are done. If x belongs to $\Delta_R(L'_1)$, then $x' \xrightarrow{R} x$ for some $x' \in L'_1$. By induction hypothesis (since R^{-1} is noetherian), we have $\mathit{prop}(x')$, therefore: $x' \in L'_2$. It follows that x belongs to $\Delta_R(L'_2)$, hence to L'_2 (because $L'_2 = \Delta_R(L'_2) \cup L$). This achieves the proof of $\mathit{prop}(x)$.

We have thus shown the uniqueness of the solution in L' of equation $L' = \Delta_R(L') \cup L$. Since $\Delta_R^*(L)$ is a solution of such an equation, we have: $L' = \Delta_R^*(L)$ iff $L' = \Delta_R(L') \cup L$.

□

Proposition 1 holds even if L and L' are not regular languages. The important point is that, when L and L' are known regular languages, equation $L' = \Delta_R(L') \cup L$ can be mechanically verified. This is because one can construct a *transducer* whose output language is $\Delta_R(L')$ when L' is given as an input language (see [14]). Therefore $\Delta_R(L')$ is a regular language, and the equation $L' = \Delta_R(L') \cup L$ is a just an equality between regular languages. Testing this equality, one will be able to detect that a (candidate) regular language L' output by *guess* is actually the sought solution (viz., $L' = \Delta_R^*(L)$).

Note 2 *Suppose that R^{-1} is noetherian and $\Delta^*(L)$ regular. Then, from proposition 1 and the fact that regular languages are enumerable, it follows that the regular language characterizing $\Delta^*(L)$ is* effectively constructible*: it suffices to start enumerating the set of all the regular languages $L_1, L_2, ...$ and stop for i such that $\Delta_R(L_i) = L_i \cup L$.*

Suppose now that the inverse system R^{-1} is not noetherian. We are then led to consider a subset R' of R such that R'^{-1} is noetherian, as well as its complementary part $R'' \equiv R \backslash R'$. In order to generate $\Delta_R^*(L)$, we will interleave synthesis of languages corresponding to the application of $\xrightarrow{R'^*}$ on the one hand, and generation of languages corresponding to one-step application of $\xrightarrow{R''}$ on the other hand. More precisely: we start with a language L_0 set to L_{init}, then construct $L'_0 \equiv \Delta_{R''}(L_0) \cup L_0$ (L'_0 is regular because L_0 is regular and $\Delta_{R''}(L_0)$ can be seen as an output of a transducer); we then synthesize L_1 such that $L_1 = \Delta_{R'}^*(L'_0)$, and iterate the process until one gets a language L_i that is R-invariant. This can be recapitulated as follows:

$$L_{init} \equiv L_0 \xrightarrow{\{R'',id\}} L'_0 \xrightarrow{R'^*} L_1 \xrightarrow{\{R'',id\}} L'_1 \xrightarrow{R'^*} L_2 \xrightarrow{\{R'',id\}} ...$$

This procedure can be seen as a form of *alternate* and *accelerated* bottom-up computation: it is "alternate" because it interleaves application of R''- and R'-rules, and is "accelerated" because one constructs in one step the result of applying the reflexive-transitive closure $\xrightarrow{R'^*}$ of $\xrightarrow{R'}$. As inputs the procedure has the initial language L_{init}, and the partition $R' \uplus R''$ of R. As output, it gives a regular language equal to $\Delta_R^*(L_{init})$. We call such a procedure *AABUP* (for 'Alternate Accelerated Bottom-UP Evaluation'). It is defined formally as:

$i := 0, L_i := L_{init}$
while L_i non invariant by $R \equiv R' \cup R''$:
do
    $L'_i := \Delta_{R''}(L_i) \cup L_i$
  repeat $L'' := guess(L'_i, R')$ **until** $\Delta_{R'}(L'') = L'' \cup L'_i$
  $L_{i+1} := L''$
  $i := i + 1$
od
return L_i

Note that the test of (non)invariance of L_i by R in the while-loop reduces to a test of (non)invariance of L_i by R'' (except at the initialisation for i=0) because L_i is equal to $\Delta^*_{R'}(L_{i-1})$ by construction, and is thus invariant by R'.

Note also that procedure *AABUP* is not guaranteed to terminate. There are two sources of nontermination. First, the repeat-loop does not terminate when, for some i, $\Delta^*_{R'}(L'_i)$ is not regular. Second, the while-loop does not terminate when, for no $i > 0$, L_i is R''-invariant.

Let us finally point out an optimization. It happens sometimes that some transitions of R are *unnecessary* (or *redundant*) because the rest of the transitions suffice to generate the whole set of reachable states. One can easily prove (*a posteriori*) the redundancy of a subset of transitions, say U, of R by constructing the reachability set S via $R \setminus U$, then testing the invariance of S via U afterwards.

In the two subsequent sections, we apply the above procedure for computing the reachability set of an array example given in [14], and a ring example given by Dijkstra [9].

6 MUX Array

The MUX example [14] is as follows. Each element of Σ (i.e., each state of an array component) is a couple of the form $[q_1, q_t]$ where: q_1 is 0,1 or 2 (0 stands for 'waiting', 1 for 'idle', 2 for 'in critical section'), q_t is 0 or 1 (0 stands for 'empty', 1 for 'with token').

The set R of transitions is: $\{r_1, r_2, r_3, r_4, r_5, r_6\}$. Transitions r_1, r_2, r_3, r_4, r_5 are internal while r_6 is external. They are defined as follows:

$r_1 :\ (X[0,0]Y) \rightarrow (X[1,0]Y)$
$r_2 :\ (X[0,1]Y) \rightarrow (X[1,1]Y)$
$r_3 :\ (X[2,0]Y) \rightarrow (X[0,0]Y)$
$r_4 :\ (X[2,1]Y) \rightarrow (X[0,1]Y)$
$r_5 :\ (X[1,1]Y) \rightarrow (X[2,1]Y)$
$r_6 :\ (X[0,1][1,0]Y) \rightarrow (X[0,0][1,1]Y)$

The language of initial states L_{init} is: $[0,1][0,0]^+$. In the following, we will use an expression of the form, say, $[\{0,1\}, q_t][0,0]^+$ as an abbreviation for $[0,q_t][0,0]^+ \cup [1,q_t][0,0]^+$.

As for R' we take the set $\{r_1, r_3, r_4, r_5, r_6\}$. As for R'', we take the set $\{r_2\}$. It is not difficult to see that R'^{-1} is noetherian. (But r_2 introduces nontermination because the sequence $(r_2 r_5 r_4)^{-1}$ creates a cycle.) We have:

$L_0 = L_{init} \equiv [0,1][0,0]^+$.

$L'_0 \equiv \Delta_{R''}(L_0) \cup L_0 = [\{0,1\},1][0,0]^+$.

As for L_1, one synthesizes (using, e.g., inductive inference method of [5]):

$L_1 = [\{0,1,2\},1][\{0,1\},0]^+ \cup [\{0,1\},0]^+[\{0,1,2\},1][\{0,1\},0]^*$

In order to verify $L_1 = \Delta^*_{R'}(L'_0)$, one tests whether $L_1 = \Delta_{R'}(L_1) \cup L'_0$. We have:

$$\begin{aligned}\Delta_{R'}(L_1) = {} & [\{0,1\},0]^*[1,1][\{0,1\},0]^*[1,0][\{0,1\},0]^* \\ & \cup[\{0,1\},0]^*[1,0][\{0,1\},0]^*[1,1][\{0,1\},0]^* \\ & \cup[\{0,1\},0]^*[\{0,2\},1][\{0,1\},0]^+ \\ & \cup[\{0,1\},0]^+[\{0,2\},1][\{0,1\},0]^*\end{aligned}$$

So $L_1 = \Delta_{R'}(L_1) \uplus [1,1][0,0]^+$. Hence $L_1 = \Delta_{R'}(L_1) \cup L'_0$ (because $[1,1][0,0]^+ \subset L'_0 \subset L_1$). Therefore $L_1 = \Delta^*_{R'}(L'_0)$. Besides L_1 is R-invariant, i.e. $\Delta_R(L_1) \subset L_1$, because $R = R' \cup R''$, $\Delta_{R'}(L_1) \subset L_1$ (as seen above) and $\Delta_{R''}(L_1) = [1,1][\{0,1\},0]^+ \cup [\{0,1\},0]^+[1,1][\{0,1\},0]^* \subset L_1$. Therefore: $L_1 = \Delta^*_R(L_{init})$.

As a recapitulation, we have: $L_{init} \equiv L_0 \xrightarrow{\{R'',id\}} L'_0 \xrightarrow{R'^*} L_1 = \Delta^*_R(L_{init})$.

7 Dijkstra's Ring

Dijkstra's ring [9] can be modelized as follows. Each element of Σ (i.e., each state of a ring component) is a triple of the form $[q_1, q_t, q_b]$ where: q_1 is 0,1 or 2 (0 stands for 'waiting', 1 for 'idle', 2 for 'in critical section'), q_t is 0 or 1 (0 stands for 'empty', 1 for 'with token'), q_b is 0 or 1 (0 stands for 'white', 1 for 'black').

The set R of transitions is: $\{r_1, r_2, r_3, r_4, r'_4, r_5, r'_5, r_6, r'_6, r_7, r_8, r'_8\}$. Transitions r_1, r_2, r_3 and r_7 are internal while couples $\langle r_4, r'_4\rangle$, $\langle r_5, r'_5\rangle$, $\langle r_6, r'_6\rangle$ and $\langle r_8, r'_8\rangle$ correspond to external transitions. They are defined as follows:

$r_1:\ (X[1,0,1]Y) \to (X[0,0,1]Y)$
$r_2:\ (X[0,1,q_b]Y) \to (X[2,1,q_b]Y)$
$r_3:\ (X[2,q_t,q_b]Y) \to (X[1,q_t,q_b]Y)$
$r_4:\ (X[q_1,q_t,0][1,0,0]Y) \to (X[q_1,q_t,1][0,0,0]Y)$
$r'_4:\ ([1,0,0]X[q_1,q_t,0]) \to ([0,0,0]X[q_1,q_t,1])$
$r_5:\ (X[1,1,1][q_1,0,q_b]Y) \to (X[1,0,0][q_1,1,q_b]Y)$
$r'_5:\ ([q_1,0,q_b]X[1,1,1]) \to ([q_1,1,q_b]X[1,0,0])$
$r_6:\ (X[q_1,q_t,0][q'_1,0,1]Y) \to (X[q_1,q_t,1][q'_1,0,1]Y)$
$r'_6:\ ([q'_1,0,1]X[q_1,q_t,0]) \to ([q'_1,0,1]X[q_1,q_t,1])$
$r_7:\ (X[1,1,0]Y) \to (X[2,1,0]Y)$
$r_8:\ (X[q_1,q_t,1][1,0,0]Y) \to (X[q_1,q_t,1][0,0,0]Y)$
$r'_8:\ ([1,0,0]X[q_1,q_t,1]) \to ([0,0,0]X[q_1,q_t,1])$

where q_1, q'_1 stand 0,1 or 2, and q_b, q_t stand for 0 or 1.

The language of initial states L_{init} is: $[1,1,0][1,0,0]^+$.

As for R' we take the set $\{r_1, r_3, r_4, r'_4, r_5, r_6, r'_6\}$. As for R'', we take the set $\{r'_5\}$. As for redundant transitions, we take $\{r_7, r_8, r'_8\}$. It is not difficult to see that R'^{-1} is noetherian.

Initially $i = 0, L_0 = L_{init}$. The rule r'_5 cannot be applied to L_0, therefore $L'_0 = L_0$. In order to generate L_1, let us apply our synthesis method based on contiguity constraints (see section 4). As a sample $S^{<k}$, we take $\Delta^*_{R'}(L^{<5}_{init})$. The set $cont'(S^{<k})$ associated with $S^{<k}$ is made of the following couples:
$([1,0,0],[1,1,0])$, $([0,0,0],[1,1,0])$, $([0,0,0],[1,1,1])$, $([1,0,0],[1,1,1])$,
$([0,0,0],[1,0,0])$, $([1,0,0],[1,0,0])$, $([0,1,0],[1,0,0])$, $([0,0,0],[1,0,1])$,
$([1,0,0],[1,0,1])$, $([0,1,0],[1,0,1])$, $([0,0,1],[1,0,1])$, $([0,1,1],[1,0,1])$,
$([1,0,1],[1,0,1])$, $([0,0,1],[0,0,0])$, $([0,1,1],[0,0,0])$, $([1,0,1],[0,0,0])$,
$([0,1,1],[0,0,1])$, $([0,0,1],[0,0,1])$, $([1,0,1],[0,0,1])$, $([0,0,0],[0,0,1])$,
$([1,0,0],[0,0,1])$, $([0,1,0],[0,0,1])$, $([2,1,0],[1,0,0])$, $([2,1,0],[1,0,1])$,
$([2,1,0],[0,0,1])$, $([2,1,1],[1,0,1])$, $([2,1,1],[0,0,0])$, $([2,1,1],[0,0,1])$,
$([1,1,0],[1,0,0])$, $([1,1,0],[1,0,1])$, $([1,1,0],[0,0,1])$, $([1,1,1],[1,0,1])$,
$([1,1,1],[0,0,0])$, $([1,1,1],[0,0,1])$.

As for L_1 we take: $L_{local}(S^{<k}) \cap L'_{local}(S^{<k}) \cap L_{global}$ where L_{global} is:[1]
$$([\{0,1,2\},0,\{0,1\}])^* \;.\; [\{0,1,2\},1,\{0,1\}] \;.\; ([\{0,1,2\},0,\{0,1\}])^*$$
Such a language L_1 can be expressed under the form: $L^1_1 \cup L^2_1$, where L^1_1 is defined by: $\Phi\ [1,0,0]^*\ [\{0,1,2\},1,0]$,
and L^2_1 by: $\{\epsilon,[0,0,0]\}\ \Phi\ [1,0,0]^*\ \Psi\ \Phi'\ \{\epsilon,[0,0,0]\ [1,0,0]^*\}$,
with the constraint that the last and first triples of every element of L^2_1 form a couple belonging to $cont'(S^{<k})$, and the following definitions for Φ, Φ', Ψ:
$$\Phi = (\ [1,0,0]^*\ \{[1,0,1],[0,0,1]\}^+\ [0,0,0]\ )^*$$
$$\Phi' = (\ [0,0,0]\ [1,0,0]^*\ \{[1,0,1],[0,0,1]\}^+\ )^*$$
$$\Psi = \ [\{0,1,2\},1,0]\ [1,0,0]^*\ \{[1,0,1],[0,0,1]\}^+ \cup\ [\{0,1,2\},1,1]\ \{[1,0,1],[0,0,1]\}^*$$
The expression Ψ corresponds to the process in possession of the token. (The second component of its first triple is equal to '1'.) The expression Φ (resp. Φ') corresponds to the processes located at the left (resp. right) of the process in possession of the token.

One can show: $L_1 = \Delta(L_1) \uplus [1,1,0][1,0,0]^+$. Hence: $L_1 = \Delta(L_1) \cup L'_0$ (because $[1,1,0][1,0,0]^+ \subset L'_0 \subset L_1$). Therefore $L_1 = \Delta^*_{R'}(L'_0)$.

We have to run once more the while-loop of $AABUP$ because L_1 is not let invariant by R'' (viz., r'_5). After (one-step) application of r'_5 to L_1, one generates a new language L'_1, which is defined as L_1, except that the couple of last and first triples in L^2_1 may be in addition $([1,0,0],[0,1,0])$. By reapplying the synthesis method, one generates a candidate language L_2 for $\Delta^*_{R'}(L'_1)$ that is defined as in L'_1, except that now the couple of last and first triples in L^2_1 may also be

[1] L_{global} expresses a mutual exclusion property: it says that one and only one triple $[q_1, q_t, q_b]$ (corresponding to the component with token) has a component q_t equal to 1.

$([1,0,0],[0,1,1])$, $([0,0,0],[0,1,0])$, $([0,0,0],[0,1,1])$, $([1,0,0],[2,1,0])$, $([1,0,0],[2,1,1])$, $([0,0,0],[2,1,0])$, $([0,0,0],[2,1,1])$.

Then one can show: $L_2 = \Delta(L_2) \uplus [0,1,0][1,0,0]^+$. Hence: $L_2 = \Delta(L_2) \cup L'_1$ (because $[0,1,0][1,0,0]^+ \subset L'_1 \subset L_2$). Therefore $L_2 = \Delta^*_{R'}(L'_1)$. Since furthermore L_2 is let invariant by R'' as well as the redundant transitions r_7, r_8, r'_8, we have: $L_2 = \Delta^*_R(L_{init})$. As a recapitulation, we have:

$$L_{init} \equiv L_0 \xrightarrow{\{R'',id\}} L'_0 \xrightarrow{R'*} L_1 \xrightarrow{\{R'',id\}} L'_1 \xrightarrow{R'*} L_2 = \Delta^*_R(L_{init}) \ .$$

8 Final Remarks

This work bears some resemblances with [14]. In both works parametrized rings (or arrays) are modelized using the same kind of rewrite systems. Also regular languages are used for characterizing the result of applying the reflexive-transitive closure $\xrightarrow{R*}$ of a reduction relation $\xrightarrow{R}$. A first difference is that, in [14], computations are done backwards instead of forwards as here. This difference is not essential: we can also proceed backwards here by simply changing R into R^{-1}, and changing L_{init} into $\neg P$, where P is the property to be proved (see, e.g., [16], p. 189). A more important difference is that, here, we use an "accelerated" form of computation by trying to guess in one step the result of applying $\xrightarrow{R*}$ (instead of iterating $\xrightarrow{R}$ until stabilization as in [14]). On the other hand in [14], the work is extended from *linear* structures such as rings or arrays to *tree* structures.

As a future work, we plan to implement procedure $AABUP$, and to identify some restricted subclasses of parametrized rings for which the procedure always terminates. Note besides that our method is not specific to parametrized rings but can be applied *a priori* to any kind of concurrent systems modelizable by rewrite systems. It would be interesting in particular to apply the method to systems whose reachability sets are known to be always regular such as pushdown automata [6,11], communicating automata with lossy channels [1] and quasi-stable channels [7].

Acknowledgement

We have benefited from numerous helpful discussions with Alberto Pettorossi and Maurizio Proietti on regular languages, and with Alain Finkel and Philippe Schnoebelen on concurrent systems. We are also grateful to David Lesens for having pointed out reference [14] to us during the INFINITY workshop.

References

[1] P. Abdulla and B. Jonsson. "Verifying Programs with Unreliable Channels", *Proc. 8th Annual IEEE Symp. of Logic in Computer Science*, 1993, pp. 160-170.

[2] D. Angluin. "Inference of Reversible Languages", *J. ACM* 29:3, 1982, pp. 741-765.

[3] D. Angluin and C. Smith. "Inductive Inference: Theory and Methods", *Computing Surveys* 15:3, 1984, pp. 237-269.

[4] A.W. Biermann. "Fundamental Mechanisms in Machine Learning and Inductive Inference", *Fundamentals of AI*, LNCS 232, Springer-Verlag, 1986, pp. 133-169.

[5] A.W. Biermann and J.A. Feldman. "On the Synthesis of Finite-State Machines from Samples of their Behaviours", *IEEE Trans. Comput.* C-21, 1972, pp. 592-597.

[6] A. Bouajjani, J. Esparza and O. Maler. "Reachability Analysis of Pushdown Automata: Application to Model-Checking", *CONCUR'97*, LNCS 1243, Springer-Verlag, 1997.

[7] G. Cécé and A. Finkel. "Programs with Quasi-Stable Channels Are Effectively Recognizable", *CAV'97*, LNCS 1254, Springer-Verlag, 1997, pp. 304-315.

[8] E.M. Clarke, O. Grumberg and S. Jha. "Verifying Parametrized Networks Using Abstraction and Regular Languages", *Proc. CONCUR'95*, LNCS 962, Springer-Verlag, 1995, pp. 395–407.

[9] E.W. Dijkstra. "Invariance and Non-Determinacy", in *Mathematical Logic and Programming Languages*, C.A.R. Hoare and J.C. Sheperdson (eds.), Prentice Hall International, 1985, pp. 157–165.

[10] E. Emerson and K.S. Namjoshi. "Reasoning about Rings", *Proc. 22nd ACM Symp. on Principles of Programming Languages*, San Francisco, 1995.

[11] A. Finkel, B. Willems and P. Wolper. "A Direct Symbolic Approach to Model Checking Pushdown Systems", Pre-proceedings of *Infinity'97*, UPMAIL Technical Report 148, July 1997, pp. 30-40.

[12] L. Fribourg and H. Olsén. "Reachability Sets of Parametrized Rings As Regular Languages", Pre-proceedings of *Infinity'97*, UPMAIL Technical Report 148, July 1997, pp. 115-138.

[13] S. Graf and H. Saidi. "Verifying Invariants Using Theorem Proving", *CAV'96*, LNCS 1102, Springer-Verlag, 1996, pp. 196-207.

[14] Y. Kesten, O. Maler, M. Marcus, A. Pnueli and E. Shahar. "Symbolic Model Checking with Rich Assertional Languages", *CAV'97*, LNCS 1254, Springer-Verlag, 1997, pp. 424-435.

[15] D. Lesens, N. Halbwachs and P. Raymond. "Automatic Verification of Parametrized Linear Networks of Processes", *Proc. 24th ACM Symp. on Principles of Programming Languages*, 1997, Paris, pp. 346-357.

[16] N. Halbwachs. "About Synchronous Programming and Abstract Interpretation ", *SAS'94*, LNCS, Springer-Verlag, 1994, pp. 179-192.

[17] J. Li, I. Suzuki and M. Yamashita. "Fair Petri Net Languages and Structural Induction for Rings of Processes". *Theoretical Computer Science 135*, 1994, pp. 377-404.

[18] C. Norris Ip and D.L. Dill. "Verifying Systems with Replicated Components in Murφ", *CAV'96*, LNCS 1102, Springer-Verlag, 1996, pp. 147-158.

[19] S. Rajan, N. Shankar and M.K. Srivas. "An Integration of Model Checking with Automated Proof Checking", *Proc. Computer-Aided Verification 95*, LNCS 939, Springer-Verlag, 1995, pp. 84–97.

[20] P. Wolper and V. Lovinfosse. "Verifying Properties of Large Sets of Processes with Network Invariants", *Intl. Workshop on Automatic Verification Methods for Finite State Systems*, LNCS 407, Springer-Verlag, 1989, pp. 68-80.