Unavoidable Configurations of Parameterized Rings of Processes

Marie Duflot[1], Laurent Fribourg[1] and Ulf Nilsson[2]

[1] LSV, CNRS & ENS de Cachan, 61 av. du Prés. Wilson, 94235 Cachan cedex, France
[2] IDA, Linköping University, 581 83 Linköping, Sweden
{duflot,fribourg}@lsv.ens-cachan.fr, ulfni@ida.liu.se

Abstract. Rewrite systems over words are often used for modeling distributed algorithms over linear networks (or rings) of N processes, where N is a parameter. Here we are interested in constructing a regular set of configurations $\mathcal{G}$ which is *unavoidable*, i.e., such that any infinite derivation intersects $\mathcal{G}$. We give some sufficient conditions of the rewrite system that allow us to construct an unavoidable set $\mathcal{G}$ using Caucal's algorithm of *prefix rewriting*. This construction is used to show the *convergence* property of distributed algorithms to closed subsets of configurations. The method is useful for proving the correctness of self-stabilizing algorithms and the liveness property of termination detection algorithms. It has been implemented, and successfully applied to several significant examples, treated in a uniform mechanical way for the first time.

1 Introduction

We consider distributed algorithms over linear or circular arrays with a parametric number N of machines. All machines (except perhaps the bottom and the top ones) are identical finite-state automata, which communicate by reading the state of their neighbors. A global system configuration is the concatenation of all the local states of the machines. For brevity, we consider only moves that simultaneously modify the local states a and b of *two* contiguous machines. (Moves modifying just one or more than two machines can be treated similarly.) The moves are described uniformly by rewrite rules of the form $ab \to a'b'$. We want to prove *convergence* of the system to a restricted set $\mathcal{L}$ of configurations, i.e., proving that every infinite sequence of reductions reaches $\mathcal{L}$.

As shown in [3], using a theoretical result of Dershowitz, the problem can be solved by exploring the branches of an infinite tree, representing repeated steps of reduction and "narrowing" (i.e. unification coupled with reduction). In order to make the exploration feasible, the infinite tree has to be folded into a finite, oriented graph. But this presupposes a manual process of generalization to infer a regular language from a finite set of words (see [3]). In this paper we address the problem of generating such a graph in an automatic and exact way (without over-approximation, as in [3]). More precisely, we observe that if only narrowing steps are taken into account, each step can be viewed either as a form of prefix or suffix rewriting, in which case Caucal's algorithm for generating regular languages using prefix rewriting can be applied to generate the graph automatically. We also discuss sufficient conditions under which we may disregard reduction steps altogether without loss of completeness.

The method has been implemented, and all examples of convergence of self-stabilizing algorithms treated manually in [3], have been processed mechanically. As a further application of the new method, we also demonstrate the verification of liveness properties of *termination detection* algorithms.

Comparison with related work. As mentioned above, we focus on the problem of proving convergence properties of distributed algorithms. In the classical framework, such proofs are typically done by exhibiting a well-founded measure on the set of configurations that strictly decreases after each step of reduction (or a bounded number of them) as long as the set $\mathcal{L}$ has not been reached. These measures are usually very subtle and require a deep knowledge of the analyzed algorithm in order to be discovered by hand (e.g., [17,25,14]). Our method, in contrast, does not require such specific knowledge, and can be mechanized.

We use several basic ideas that already appear in the literature. The idea of viewing distributed algorithms over linear networks of machines as rewrite systems and sets of configurations as regular languages, has been used before (see e.g., [1,16,19,20,22,26]). However, interest is generally restricted to showing that the *reachability set* of a distributed algorithm, i.e. the set of all configurations reached by a *finite* derivation, is regular. In contrast we focus on unavoidable sets, i.e., sets such that any *infinite* derivation traverses them. That is, we want to prove liveness properties rather than safety properties. To our knowledge, only two recent works investigate the verification of liveness properties for parameterized rings [7,24]. In [7] Bouajjani et al. prove liveness properties (or, more generally, ω-regular properties) of distributed algorithms by using strategies for guessing automatically extended forms of reachability sets in the framework of Büchi automata. Pnueli and Shahar [24] use related techniques of "acceleration" but in the framework of second-order monadic logic. In [7], the method basically relies on the fact that every rewrite system $\mathcal{S}$ modeling a studied algorithm is Noetherian (i.e., it does not contain infinite derivations). This is never the case in the examples treated here. We only assume that a strict subset of $\mathcal{S}$ (obtained by removing the rules affecting the top machine) is Noetherian, which is a crucial difference. In [24], the method must take into account an unbounded number of fairness assumptions, several for each process, and these requirements are also parameterized, often making the computation diverge (see [24], p. 340). In contrast, our method assumes only the weak fairness assumption that the top machine is affected at least once in every infinite derivation. The techniques that we use here are also very different from the ones used in [7,24], and involve first-order rewriting operations such as narrowing.

Plan of the paper. Section 2 recalls some definitions about string rewrite systems. Section 3 explains the basic operation of narrowing as a form of prefix rewriting. Section 4 explains how to construct regular unavoidable languages using a closure assumption on the set of narrowings. Section 5 gives a syntactic criterion ensuring the closure of this set. Section 6 explains how to use unavoidable languages for proving the correctness property of self-stabilizing algorithms and the liveness property of termination detection algorithms. The implementation and experimental results are given in Section 7. Section 8 concludes.

2 String Rewriting for Linear Networks of Machines

We use string rewrite systems to model the moves that affect the local states of the component machines. Local states of machines belong to a finite alphabet Σ. Each word $a_1 \cdots a_N$ (where N is a parameter) constitutes a global configuration of the system where machine i is in state a_i ($1 \le i \le N$). In the following, each move involves exactly two contiguous machines. The moves are described by rewrite rules. Special rules have to be considered for the bottom and top machines. In this framework, we model the actions of a distributed algorithm as a set $\mathcal{S}$ of rewrite rules. We will use first-order variables W, X, Y to represent in a symbolic way a generic set of states for a portion of contiguous machines. We first adapt some basic definitions from (string) rewrite systems [5,11] to our framework.

A *ground word* is an element of Σ^*, with ε denoting the empty word. An *open word* is an expression of the form uWv where u, v are elements of $\Sigma^+ (= \Sigma^* - \{\varepsilon\})$. Henceforth, *word* refers to an open or ground word, which is written t, u or v (possibly adorned). Letters are denoted by a, b, c or d (possibly adorned). A rewrite system $\mathcal{S}$ is a set of length-preserving rules, divided into three subsets $Top_{\mathcal{S}}$, $Bottom_{\mathcal{S}}$ and $Middle_{\mathcal{S}}$: *bottom rules* in $Bottom_{\mathcal{S}}$ are applied to the two leftmost letters of words; *top rules* in $Top_{\mathcal{S}}$ are applied to the leftmost and rightmost letters of words; the other rules in $Middle_{\mathcal{S}}$ are called *middle rules*. More precisely, let a, b, a' and b' be letters of Σ, and X, Y variables. Then:

- $Bottom_{\mathcal{S}}$ consists of rules of the form $abX \to a'b'X$
- $Middle_{\mathcal{S}}$ consists of rules of the form $XabY \to Xa'b'Y$ (with $X \neq \varepsilon$)
- $Top_{\mathcal{S}}$ consists of rules of the form $bXa \to b'Xa'$

A top rule makes the leftmost and rightmost components of the array communicate: the system should be considered a ring of machines.[1]

Example. Consider the alphabet $\Sigma = \{0, 1, 2\}$ and the rewrite rules $\mathcal{BD} = \{B_1, M_1, M_4, T_4\}$, which is a simplified version of Beauquier-Debas' original system [4] (see the example of Sect. 6.1 for the full system):

$$\begin{aligned} B_1 &: \ 12X \to 21X \\ M_1 &: X10Y \to X01Y \text{ with } X \neq \varepsilon \\ M_4 &: X02Y \to X20Y \text{ with } X \neq \varepsilon \\ T_4 &: \ 2X1 \to 1X2 \end{aligned}$$

Here B_1 is a bottom rule, T_4 a top rule, M_1 and M_4 middle rules.

[1] One can consider also top rules of the form $Xab \to Xa'b'$ where such communication does not exist. In this case $Y \neq \varepsilon$ should be added to middle rules.

Reduction consists in replacing an instance of a rule's left-hand side by the corresponding instance of the rule's right-hand side. Formally, given two ground terms t and t', we say that *reduction* applies from t to t' via middle rule $M : XabY \to Xa'b'Y$ if $t = uabv$ and $t' = ua'b'v$ for some strings $u \in \Sigma^+, v \in \Sigma^*$. This is written $t \to_M t'$. The notion of reduction via M also applies when t is an open word of the form t_1Wt_2. Reduction of t then consists in replacing ab with $a'b'$, either in the left part t_1 or in the right part t_2 of t. The notion of reduction of a ground/open word t is defined similarly for a bottom rule B (resp. top rule T) instead of a middle rule M. In the case of a bottom reduction (resp. top reduction), the replaced letters a and b of t occur in the first and second position (resp. last and first ones). We say that reduction applies from t to t' via $\mathcal{S}$, and write $t \to_{\mathcal{S}} t'$, if $t \to_R t'$ for some rule $R \in \mathcal{S}$. The reduction of a set of words J via $\mathcal{S}$ is defined by $Reduce_{\mathcal{S}}(J) = \{t' \mid \exists t \in J \ \wedge \ t \to_{\mathcal{S}} t'\}$. The set J is said to be *closed* under a rewrite system $\mathcal{S}$ if $Reduce_{\mathcal{S}}(J) \subseteq J$.

Example. The set $\mathcal{L} = 20^*10^* \cup 10^*20^*$ is closed under $\mathcal{BD}$.

The mapping $Reduce_{\mathcal{S}}$ can be seen as a *transducer* (see, e.g., [20]) and $Reduce_{\mathcal{S}}(J)$ is regular if J is regular. As usual, $\to^*_{\mathcal{S}}$ denotes the reflexive-transitive closure of $\to_{\mathcal{S}}$. It is well-known that the set generated from J by iterated application of $\to_{\mathcal{S}}$ (i.e, by application of $\to^*_{\mathcal{S}}$), is generally not regular, even if J is regular. Nevertheless, Caucal has shown that iterated *prefix* reduction (at the beginning of a word) and iterated *suffix* reduction (at the end of a word), behave as transducers. Furthermore, Caucal has designed an efficient algorithm, polynomial in time for the size of $\mathcal{S}$, for these computations [8]. In this paper, we use prefix and suffix strategies of reduction in order to apply Caucal's algorithm.

A *ground derivation* is a (possibly infinite) sequence of reductions over ground words. A set of rules $\mathcal{S}$ is said to be *Noetherian* if any ground derivation using rules of $\mathcal{S}$ is finite.

Example. For the system $\mathcal{BD}$ defined above, we have $t_1 = 2\underline{10}0 \to_{M_1} t_2 = 20\underline{10} \to_{M_1} t_3 = \underline{2}00\underline{1} \to_{T_4} t_4 = 10\underline{02} \to_{M_4} t_5 = 1\underline{02}0 \to_{M_4} t_6 = \underline{12}00 \to_{B_1} t_1 = 2100$. (The letters replaced at each step of reduction are underlined for the sake of clarity.) We have thus an infinite (cyclic) ground derivation. Hence $\mathcal{BD}$ is *non*-Noetherian. On the other hand, $\mathcal{BD} - T_4$ is Noetherian. Consider the well-founded measure ψ that associates $\sum_{i \in S_1}(N-i) + \sum_{j \in S_2} j$ with every word $a_1 \cdots a_N$, where S_1 (resp. S_2) is the set of positions i (resp. j) such that $a_i = 1$ (resp. $a_j = 2$). It is easy to see that application of rule M_1 (or M_4) makes ψ decrease by 1, while B_1 makes ψ decrease by 2.

Note that testing the closure of a regular set J under $\mathcal{S}$ is decidable: it consists in checking the inclusion of two regular sets.

For each middle rule M (resp. bottom rule B) it is convenient to define two variants of M (resp. one variant of B) as follows:

Definition 1. Given a middle rule $M : XabY \to Xa'b'Y$ and a bottom rule $B : abY \to a'b'Y$,

– the *prefix extension* of M, written M_{pre}, is: $X{\odot}bY \to X{\odot}a'b'Y$,
– the *suffix extension* of M, written M_{suf}, is: $Xa{\odot}Y \to Xa'b'{\odot}Y$,
– the *suffix extension* of B, written B_{suf}, is: $a{\odot}Y \to a'b'{\odot}Y$,

where $\odot$ is a new constant added to Σ.

Let t be an open word t_1Wt_2, and denote by $t_\odot$ the word $t_1{\odot}t_2$ obtained from t by instantiating W with $\odot$. When t_2 starts with b, reduction via M_{pre} applies to $t_\odot$. Such a reduction affects a prefix of ${\odot}t_2$, and is hence a *prefix reduction* of the right part of $t_\odot$. Likewise, reduction via M_{suf} (resp. B_{suf}) affects a suffix of $t_1\odot$, and is hence a *suffix reduction* of the left part of $t_\odot$. The rule extensions will allow us to consider the operation of *narrowing*, defined hereafter, as a form of prefix or suffix reduction, and thus to apply Caucal's algorithm.

3 Narrowing

Next we define *narrowing* (e.g., [11]) over open words t_1Wt_2, with $t_1, t_2 \in \Sigma^+$. In this context, narrowing can be seen as a form of replacement of W with Wb, aW or ε, followed by reduction of the resulting word, as explained below.

3.1 Right, Left and Bottom Narrowing

Definition 2. Consider a middle rule $M: \ XabY \to Xa'b'Y$, a bottom rule $B: \ abX \to a'b'X$, and two open words t, t'.

– We say that *right narrowing* applies from t to t' via M if $t = uWbv$ and $t' = uWa'b'v$, for some $u \in \Sigma^+, v \in \Sigma^*$. This is written $t \rightsquigarrow_M^{right} t'$.
– We say that *left narrowing* applies from t to t' via M if $t = uaWv$ and $t' = ua'b'Wv$, for some $u, v \in \Sigma^+$. This is written $t \rightsquigarrow_M^{left} t'$.
– We say that *bottom narrowing* applies from t to t' via B if $t = aWv$ and $t' = a'b'Wv$, for some $v \in \Sigma^+$. This is written $t \rightsquigarrow_B^{bot} t'$.

Right narrowing from t to t' via M corresponds to prefix reduction from $t_\odot$ to $t'_\odot$ via M_{pre} (since $uWbv \rightsquigarrow_M^{right} uWa'b'v$ iff $u{\odot}bv \to_{M_{pre}} u{\odot}a'b'v$). Likewise, left narrowing via M (resp. bottom narrowing via B) corresponds to suffix reduction via M_{suf} (resp. B_{suf}). We have indeed: $uaWv \rightsquigarrow_M^{left} ua'b'Wv$ iff $ua{\odot}v \to_{M_{suf}} ua'b'{\odot}v$ (resp. $aWv \rightsquigarrow_B^{bot} a'b'Wv$ iff $a{\odot}v \to_{B_{suf}} a'b'{\odot}v$).

Example. Using $M_4 : X02Y \to X20Y$, we have $01W200 \rightsquigarrow_{M_4}^{right} 01W2000$. This narrowing is mimicked by the reduction $01{\odot}200 \to_{M_{4pre}} 01{\odot}2000$, using the prefix extension $M_{4pre}: \ X{\odot}2Y \to X{\odot}20Y$.

We write $t \rightsquigarrow_{\mathcal{S}}^{right} t'$ if $t \rightsquigarrow_M^{right} t'$ for some rule $M \in \mathcal{S}$, and similarly for left and bottom narrowing. We will abbreviate $\rightsquigarrow_{\mathcal{S}}^{bot} \cup \rightsquigarrow_{\mathcal{S}}^{left} \cup \rightsquigarrow_{\mathcal{S}}^{right}$ by $\rightsquigarrow_{\mathcal{S}}$. Given a set J of open words, we write:

$$Narrow_{\mathcal{S}}^*(J) = \{t' \mid \exists t \in J \wedge t \rightsquigarrow_{\mathcal{S}}^* t'\}$$
$$(Narrow_{\mathcal{S}} + Reduce_{\mathcal{S}})^*(J) = \{t' \mid \exists t \in J \wedge t \ (\rightsquigarrow_{\mathcal{S}} \cup \to_{\mathcal{S}})^* \ t'\}.$$

The first set is obtained by iterated application of narrowing only; the second one by interleaving narrowing and reductions via $\mathcal{S}$. Let $I_{\mathcal{S}}$ be the initial set of $\mathcal{S}$; i.e., the set of open words $b'Wa'$ corresponding to all right-hand sides of top rules $bXa \to b'Xa'$ of $\mathcal{S}$ (with W replacing X). We write:

$$\begin{aligned} \mathcal{N}_{\mathcal{S}}^* &= Narrow_{\mathcal{S}}^*(I_{\mathcal{S}}) \\ (\mathcal{N}_{\mathcal{S}} + \mathcal{R}_{\mathcal{S}})^* &= (Narrow_{\mathcal{S}} + Reduce_{\mathcal{S}})^*(I_{\mathcal{S}}). \end{aligned}$$

In the computation of $\mathcal{N}_{\mathcal{S}}^*$, right narrowing on the one hand, bottom and left narrowing on the other hand, are independent (since they involve sub-words separated by W) and can be done in parallel. Since right narrowing (resp. bottom and left narrowing) can be simulated by prefix reduction (resp. suffix reduction), $\mathcal{N}_{\mathcal{S}}^*$ can be computed by iterated prefix and suffix reductions done in parallel, starting from the right-hand sides of $Top_{\mathcal{S}}$. By Caucal's result [8] (cf. [5,6]), it follows directly that:

Proposition 3. *$\mathcal{N}_{\mathcal{S}}^*$ is regular and can be constructed in polynomial time in the size of $\mathcal{S}$.*

We decompose $\mathcal{N}_{\mathcal{S}}^*$ into $\mathcal{M}_0 \cup \mathcal{M}_1$, where $\mathcal{M}_0$ (resp. $\mathcal{M}_1$) is the subset generated with 0 (resp. 1) application of bottom narrowing. (It is easy to see that bottom narrowing can be applied at most once). More precisely, $\mathcal{N}_{\mathcal{S}}^* = \mathcal{M}_0 \cup \mathcal{M}_1$ where

- $\mathcal{M}_0$ is the result of applying $(\rightsquigarrow^{right})^*$ to $I_{\mathcal{S}}$, and
- $\mathcal{M}_1$ the result of applying $\rightsquigarrow^{bot} \circ (\rightsquigarrow^{left})^*$ in parallel with $(\rightsquigarrow^{right})^*$ to $I_{\mathcal{S}}$.

Example. For the Beauquier-Debas system $\mathcal{BD}$, the initial set $I_{\mathcal{BD}}$ is $1W2$. Application of $(\rightsquigarrow^{right})^*$ to $I_{\mathcal{BD}}$ via $M_4 : X02Y \to X20Y$ yields $\{1W2, 1W20, 1W200, 1W2000, \ldots\}$. Thus the set $\mathcal{M}_0$ is $1W20^*$. On the other hand, the application of $\rightsquigarrow^{bot}$ to $I_{\mathcal{BD}}$ via B_1 yields $21W2$. Then subsequent application of $(\rightsquigarrow^{left})^*$ via $M_1 : X10Y \to X01Y$ yields $\{21W2, 201W2, 2001W2, \ldots\}$, that is 20^*1W2. By (parallel) application of $(\rightsquigarrow^{right})^*$, we finally generate $\mathcal{M}_1 = 20^*1W20^*$. We have $\mathcal{N}_{BD}^* = \mathcal{M}_0 \cup \mathcal{M}_1 = 1W20^* \cup 20^*1W20^*$.

3.2 Grounding Narrowing

We now consider an operation which removes W from an open word by replacing W with ε followed by a reduction step, thus yielding a ground word.

Definition 4. Consider a middle rule $M : XabY \to Xa'b'Y$, a bottom rule $B : abY \to a'b'Y$, an open word t and a ground word t'.

- We say that *grounding narrowing* applies from t to t' via M if $t = uaWbv$ and $t' = ua'b'v$, for some $u \in \Sigma^+, v \in \Sigma^*$. This is written $t \rightsquigarrow_M^{gr} t'$.
- We say that *grounding narrowing* applies from t to t' via B if $t = aWbv$ and $t' = a'b'v$, for some $v \in \Sigma^*$. This is written $t \rightsquigarrow_B^{gr} t'$.

We write $t \rightsquigarrow^{gr}_{\mathcal{S}} t'$ iff $t \rightsquigarrow^{gr}_{M} t'$ or $t \rightsquigarrow^{gr}_{B} t'$ for some rules M, B of $\mathcal{S}$. Given a set J of open words, the *grounding narrowing of J via $\mathcal{S}$*, written $Gr_{\mathcal{S}}(J)$ or more simply $Gr(J)$, is $Gr(J) = \{t' \mid \exists t \in J \wedge t \rightsquigarrow^{gr}_{\mathcal{S}} t'\}$ and we set $\mathcal{G}_{\mathcal{S}} = Gr_{\mathcal{S}}(\mathcal{N}^*_{\mathcal{S}})$.

The mapping Gr can also be seen as a transducer, hence $Gr(J)$ is regular if J is regular. In particular, $\mathcal{G}_{\mathcal{S}}$ is regular. It can be obtained by applying grounding narrowing to $\mathcal{M}_0$ via a bottom rule B and grounding narrowing to $\mathcal{M}_1$ via a middle rule M.

Example. By application of grounding narrowing to $\mathcal{M}_0 = 1W20^*$ via the rule $B_1 : \ 12X \to 21X$, we get 210^*. There is no grounding narrowing applicable to $\mathcal{M}_1 = 20^*1W20^*$ via M_1 or M_4. Hence $\mathcal{G}_{\mathcal{BD}} = Gr(\mathcal{M}_0) = 210^*$.

4 Unavoidable Regular Languages and Convergence

We now focus on the problem of constructing a regular set unavoidable by $\mathcal{S}$. Let us first define the notion of "unavoidable set".

Definition 5. Given a rewrite system $\mathcal{S}$, a set $\mathcal{K}$ of ground words is said to be *unavoidable by $\mathcal{S}$* if, for all infinite ground derivation of the form $t_1 \to_{\mathcal{S}} t_2 \to_{\mathcal{S}} \cdots \to_{\mathcal{S}} t_n \to_{\mathcal{S}} \cdots$, there exists a word $u \in \mathcal{K}$ such that $u = t_i$ for some $i > 0$.

Using the notion of unavoidable set, we can now rephrase the main result (Theorem 1, Section 3.2) of [3] in a more elegant way, as follows:

Theorem 6. **[3]** *If the rewrite system $\mathcal{S} - Top_{\mathcal{S}}$ is Noetherian, then $Gr((\mathcal{N}_{\mathcal{S}} + \mathcal{R}_{\mathcal{S}})^*)$ is unavoidable by $\mathcal{S}$.*

The idea of the proof is as follows: Since $\mathcal{S} - Top_{\mathcal{S}}$ is Noetherian, all infinite ground derivations contain at least one application of $Top_{\mathcal{S}}$. After application of such a rule, say $bXa \to b'Xa'$, the ground word generated is of the form $b'ua'$ for some $u \in \Sigma^*$. Therefore the set of words derived from $b'ua'$ is unavoidable. However, many words are generated in a redundant manner. Exploiting some reordering properties of infinite ground derivations (first discovered by Dershowitz [10]), it is possible to prove that no unavoidable word is lost when focusing on the derivations that start from $b'Wa'$, and replace iteratively W via successive narrowings interleaved with reductions. See [3] for the full proof.

We claim that the Noetherian condition on $\mathcal{S} - Top_{\mathcal{S}}$ is generally met in practice. Such a condition implies that all infinite derivations affect the rightmost machine at least once (hence an infinite number of times). It can be considered as a weak form of fairness concerning the top machine. For instance, in "token passing" algorithms (as those considered in our examples), the tokens visit the top machine at least once during infinite derivations, so the Noetherian condition on $\mathcal{S} - Top_{\mathcal{S}}$ is always satisfied. Note however that, in order to prove formally that the condition actually holds, we must generally discover, by hand, a specific well-founded measure that decreases after each step of reduction, top excepted (as, e.g., ψ given in an example of Section 2, or norm F of [25], p. 464).

The construction of an unavoidable set thus reduces to the generation of $(\mathcal{N}_\mathcal{S} + \mathcal{R}_\mathcal{S})^*$. In [3], over-approximations of $(\mathcal{N}_\mathcal{S} + \mathcal{R}_\mathcal{S})^*$ were generated in the form of regular languages, but the process was done in an *ad hoc* manner for each treated example, using human insight. It also required the construction of a graph expressing reachability relations among the inferred sublanguages, in order to detect possible divergent loops of execution. In contrast, we now state a simple and sufficient condition that guarantees that $(\mathcal{N}_\mathcal{S} + \mathcal{R}_\mathcal{S})^*$ is regular and can be constructed in both an exact and automated way. Let us suppose that $\mathcal{N}_\mathcal{S}^*$ is closed under $\mathcal{S}$ (i.e., $Reduce_\mathcal{S}(\mathcal{N}_\mathcal{S}^*) \subseteq \mathcal{N}_\mathcal{S}^*$). Then $(\mathcal{N}_\mathcal{S} + \mathcal{R}_\mathcal{S})^*$ is simply $\mathcal{N}_\mathcal{S}^*$. Using proposition 3, Theorem 6 thus becomes:

Proposition 7. *If the rewrite system $\mathcal{S} - Top_\mathcal{S}$ is Noetherian and $\mathcal{N}_\mathcal{S}^*$ is closed under $\mathcal{S}$, then $\mathcal{G}_\mathcal{S}$ is a regular unavoidable set.*

Example. In the Beauquier-Debas example, $\mathcal{N}_{\mathcal{BD}}^* = 1W20^* \cup\ 20^*1W20^*$. It is closed under $\mathcal{BD}$, and $\mathcal{BD} - Top_{\mathcal{BD}}$ is Noetherian. Therefore $\mathcal{G}_{\mathcal{BD}} = 210^*$ is unavoidable.

Checking the closure of $\mathcal{N}_\mathcal{S}^*$ under $\mathcal{S}$ is decidable (since $\mathcal{N}_\mathcal{S}^*$ is regular). Note that, even when $\mathcal{N}_\mathcal{S}^*$ is closed, $\mathcal{G}_\mathcal{S}$ is generally not closed itself. Therefore, infinite derivations traverse $\mathcal{G}_\mathcal{S}$, without staying within it. But if, additionally, $\mathcal{G}_\mathcal{S}$ is contained in a *closed* set $\mathcal{L}$, all infinite ground derivations are eventually "captured" by $\mathcal{L}$; we have *convergence* to $\mathcal{L}$. Formally:

Definition 8. Let $\mathcal{S}$ be a rewrite system and $\mathcal{L}$ a set of configurations. We say that $\mathcal{S}$ *converges to* $\mathcal{L}$ and write $Conv(\mathcal{S}, \mathcal{L})$, if, for all infinite ground derivation of the form $t_1 \to_\mathcal{S} t_2 \to_\mathcal{S} \cdots \to_\mathcal{S} t_n \to_\mathcal{S} \cdots$, there exists $k > 0$ such that $t_i \in \mathcal{L}$ for all $i \geq k$.

Convergence properties are useful for proving correctness of self-stabilizing algorithms and liveness properties of termination detection algorithms, as illustrated in Section 6.

Proposition 9. *Let $\mathcal{S}$ be a rewrite system and $\mathcal{L}$ a set of configurations closed under $\mathcal{S}$. If there exists an unavoidable set $\mathcal{K}$ for $\mathcal{S}$ such that $\mathcal{K} \subseteq \mathcal{L}$, then we have $Conv(\mathcal{S}, \mathcal{L})$.*

From proposition 7 and proposition 9, it follows:

Theorem 10. *Consider a system $\mathcal{S}$ such that $\mathcal{S} - Top_\mathcal{S}$ is Noetherian, and a set $\mathcal{L}$ closed under $\mathcal{S}$. If $\mathcal{N}_\mathcal{S}^*$ is closed under $\mathcal{S}$ and if $\mathcal{G}_\mathcal{S}$ is included in $\mathcal{L}$, then we have $Conv(\mathcal{S}, \mathcal{L})$.*

Note that, if $\mathcal{L}$ is regular, the inclusion of $\mathcal{G}_\mathcal{S}$ in $\mathcal{L}$ is decidable (since $\mathcal{G}_\mathcal{S}$ is regular). Theorem 10 is a major enhancement of the results in [3], as it allows to mechanically prove convergence properties, using the construction of $\mathcal{N}_\mathcal{S}^*$ via prefix/suffix rewriting.

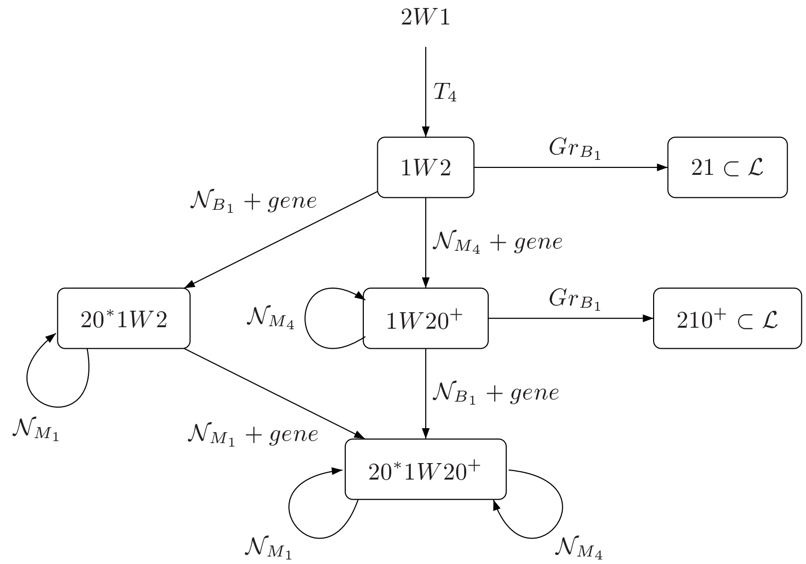

Fig. 1. Construction by hand of the graph in [3] for proving convergence of system $\mathcal{BD}$

Example. In the Beauquier-Debas example, $\mathcal{N}^*_{\mathcal{BD}} = 1W20^* \cup 20^*1W20^*$ is closed under $\mathcal{BD}$ and $\mathcal{G}_{\mathcal{BD}} = 210^*$ is included in $\mathcal{L} = 20^*10^* \cup 10^*20^*$. Since $\mathcal{BD} - Top_{\mathcal{BD}}$ is Noetherian and $\mathcal{L}$ is closed, we have $Conv(\mathcal{BD}, \mathcal{L})$. Such a proof has been automatically produced by an implemented program (see Section 7). This is a breakthrough with respect to [3], where the proof of convergence of $\mathcal{BD}$ to $\mathcal{L}$ required manual construction of a graph via an ad hoc process of generalization, as illustrated in Figure 1. (In the figure, an edge labeled with $\mathcal{N} + gene$ models the inference of a regular language through one step of narrowing followed by generalization. For example the edge, labeled with $\mathcal{N}_{M_4} + gene$, from $1W2$ to $1W20^+$, means that narrowing via M_4 leads from $1W2$ to $1W20$, which is generalized to $1W20^+$.)

5 A Sufficient Syntactic Condition for Closure of $\mathcal{N}^*_{\mathcal{S}}$

Although the closure condition of $\mathcal{N}^*_{\mathcal{S}}$ in Theorem 10 is decidable, it is difficult to verify, from simple inspection of the rules of $\mathcal{S}$, whether it holds or not. We next give a syntactic criterion on $\mathcal{S}$, which has a simple operational meaning, and guarantees the closure condition. This criterion is especially useful for self-stabilizing algorithms (see Section 6) involving mutual exclusion. In such a case, the reducible positions in configurations correspond to "tokens" or "privileges". The number φ of such tokens never increases (no creation of tokens). The problem is to show that φ always eventually decreases until just one token remains;

in which case the set $\mathcal{L}$ of "legitimate" configurations is reached. We explain now that, if we focus on derivations that preserve the number φ of tokens, then the system is likely to satisfy an operational condition of *unidirectionality* that guarantees the closure property. We need some preliminary definitions.

Definition 11. A *middle U-turn from left to left* (resp. *from right to right*) in a ground derivation is a sequence of reductions of the form $u\underline{ab}cv \to ua'\underline{b'c}v \to u\underline{a'b''}c'v \to ua''b'''c'v$ (resp. $ua\underline{bc}v \to u\underline{ab'}c'v \to ua'\underline{b''c'}v \to ua'b'''c''v$), where u and v are strings.

Definition 12. A *bottom U-turn* in a ground derivation is a sequence of reductions of the form $\underline{ab}cv \to a'\underline{b'c}v \to \underline{a'b''}c'v \to a''b'''c'v$. A *top right* (resp. *top left*) *U-turn* is a sequence of reductions $\underline{c}ua\underline{b} \to c'u\underline{ab'} \to c'ua'b''$ such that $\underline{d}v\underline{b''} \to d'vb'''$ for some $d \in \Sigma, v \in \Sigma^+$ (resp. $\underline{c}au\underline{b} \to \underline{c'a}ub' \to c''a'ub'$ such that $\underline{c''}v\underline{d} \to c'''vd'$ for some $d \in \Sigma, v \in \Sigma^+$).

Definition 13. A *repetition* in a derivation is a sequence of two consecutive reductions that rewrite the same positions in the term (e.g., $u\underline{ab}v \to u\underline{a'b'}v \to ua''b''v$).

A rewriting system $\mathcal{S}$ is said to be *unidirectional* if no U-turn (middle from left to left, middle from right to right, bottom, top left, top right) and no repetition are possible, along any derivation via $\mathcal{S}$. From an operational point of view, it roughly means that each token always travels in the same direction. It is easy to determine by inspection of the left- and right-hand sides of rules, whether a system $\mathcal{S}$ is unidirectional or not.

Proposition 14. *In a unidirectional system, we have* $(\mathcal{N}_{\mathcal{S}} + \mathcal{R}_{\mathcal{S}})^* = \mathcal{N}_{\mathcal{S}}^*$

More precisely, starting from a word of $I_{\mathcal{S}}$, once we have narrowed a word an arbitrary number of times at prefix or suffix position, there is no possibility of reducing the resulting word anywhere else. Unidirectionality is thus a syntactic sufficient condition that guarantees that $\mathcal{N}_{\mathcal{S}}^*$ is closed under $\mathcal{S}$.

A priori, a distributed algorithm $\mathcal{S}$ does not satisfy the unidirectionality property. However we claim that $\mathcal{S}$ often can be transformed into a simplified system $\mathcal{T}$, which is unidirectional, and such that $\mathit{Conv}(\mathcal{T}, \mathcal{L}) \Rightarrow \mathit{Conv}(\mathcal{S}, \mathcal{L})$. This is done by restricting the possible rules of $\mathcal{S}$ with respect to the mapping φ. Since φ never increases, all the infinite derivations via $\mathcal{S}$ preserve φ after a finite number of initial steps. We construct $\mathcal{T}$ so that $\mathcal{T}$ preserves φ (i.e., $x \to_{\mathcal{T}} x'$ iff $x \to_{\mathcal{S}} x' \wedge \varphi(x) = \varphi(x')$). Then clearly $\mathit{Conv}(\mathcal{T}, \mathcal{L})$ holds iff $\mathit{Conv}(\mathcal{S}, \mathcal{L})$ does. The behavior of tokens in $\mathcal{T}$ is more restrained than in $\mathcal{S}$. (In particular, there is no collision of tokens since collision usually eliminates one or two tokens.) So $\mathcal{T}$ is more likely to satisfy the unidirectionality property. As an example, in Sect. 6.1, we derive a simplified system $\mathcal{BD}$ from the original Beauquier-Debas system $\mathcal{S}$ such that $\mathit{Conv}(\mathcal{BD}, \mathcal{L}) \Leftrightarrow \mathit{Conv}(\mathcal{S}, \mathcal{L})$. The original system is not unidirectional (for example, there is a U-turn: $u1\underline{10}v \to_{M_1} u\underline{10}1v \to_{M_1} u0\underline{11}v \to_{M_2} u002v$). In contrast, the simplified system $\mathcal{BD}$ is unidirectional and $\mathcal{N}_{\mathcal{BD}}^*$ is therefore closed.

Fig. 2. Execution-graph for self-stabilizing systems

Note that the unidirectionality criterion is a sufficient, but not necessary, condition for the closure of $\mathcal{N}_{\mathcal{S}}$. (There are examples, e.g., Hoepman's algorithm [18], which are not unidirectional, even in the simplified form, but are still closed.) However, unidirectionality allows us to understand better why our assumption of closure of $\mathcal{N}^*_{\mathcal{S}}$ is met in practice by several mutual exclusion self-stabilizing algorithms, like those of Beauquier-Debas and Ghosh [4,17].

6 Applications

We now show some applications of Theorem 10. Throughout this section, we make three assumptions about $\mathcal{S}$ and $\mathcal{L}$:

α. $\mathcal{L}$ is regular,
β. $\mathcal{L}$ is closed under $\mathcal{S}$,
γ. $\mathcal{S} - Top_{\mathcal{S}}$ is Noetherian.

We first show how to prove self-stabilization, and then a liveness property of termination detection algorithms.

6.1 Proof of Self-Stabilization

Self-stabilization is an important concept in distributed computing, and refers to a system's ability to recover automatically from unexpected faults [14]. Formally, a rewrite system $\mathcal{S}$ is *self-stabilizing* wrt a closed set $\mathcal{L}$ if all infinite derivations via $\mathcal{S}$ reach a configuration of $\mathcal{L}$ for all initial configurations [12]. The configurations of $\mathcal{L}$ are called *legitimate*. The self-stabilization of a system $\mathcal{S}$ wrt $\mathcal{L}$ is generally proven by showing two properties:(1) every ground configuration is reducible via $\mathcal{S}$, and (2) $\mathcal{S}$ converges to $\mathcal{L}$. The former is easy to check: it reduces to an equality test between two regular languages. The main problem is to prove property (2), depicted in Fig. 2.

In self-stabilizing algorithms for rings, the set $\mathcal{L}$ of legitimate configurations is always closed and can often be expressed naturally as a regular set. So (α) and (β) are met in practice. Condition (γ) is generally met by $\mathcal{S}$ as explained in Sect. 4. However the closure of $\mathcal{N}^*_{\mathcal{S}}$ under $\mathcal{S}$ is not always satisfied. In this case, as explained in Sect. 5, we try to find a simplified set $\mathcal{T}$ of rules such that $Conv(\mathcal{T}, \mathcal{L}) \Rightarrow Conv(\mathcal{S}, \mathcal{L})$.

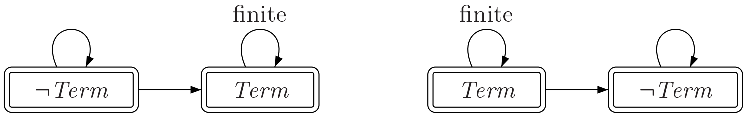

Fig. 3. Execution-graph for $\mathcal{S}$ satisfying $Liv(\mathcal{S}, Term)$ (left) and $\mathcal{S}^{-1}$ satisfying $Conv(\mathcal{S}^{-1}, \neg Term)$ (right)

Example. (*Beauquier-Debas*). The system $\mathcal{S}$ modeling the original Beauquier-Debas algorithm looks as follows:

$$
\begin{array}{ll}
B_1 : \ 12X \to 21X & T_1 : 0X0 \to 1X2 \\
M_1 : X10Y \to X01Y \ (X \neq \varepsilon) & T_2 : 0X1 \to 1X0 \\
M_2 : X11Y \to X02Y \ (X \neq \varepsilon) & T_3 : 0X2 \to 1X1 \\
M_3 : X12Y \to X00Y \ (X \neq \varepsilon) & T_4 : 2X1 \to 1X2 \\
M_4 : X02Y \to X20Y \ (X \neq \varepsilon) & T_5 : 2X2 \to 1X0 \\
M_5 : X22Y \to X10Y \ (X \neq \varepsilon) &
\end{array}
$$

$\mathcal{L}$ is defined as $20^*10^* \cup 10^*20^*$. One can show that $\mathcal{N}^*_{\mathcal{S}}$ is not closed under $\mathcal{S}$, but as remarked in [4], T_1, T_2, T_3 are applied at most once. As a consequence $\mathcal{S}$ converges to $\mathcal{L}$ iff $\mathcal{S}_0 \equiv \mathcal{S} - \{T_1, T_2, T_3\}$ converges to $\mathcal{L}$. Let φ be a measure mapping every ground word to its number of non-zero elements. Obviously, any application of rule M_2, M_3, M_5 or T_5 of $\mathcal{S}_0$ strictly decreases φ, while rules B_1, M_1, M_4, T_4 preserve it. Therefore, M_2, M_3, M_5 and T_5 can be applied only a finite number of times. It follows that $\mathcal{S}_0$ converges to $\mathcal{L}$ iff $\mathcal{BD} \equiv \{B_1, M_1, M_4, T_4\}$ converges to $\mathcal{L}$. Hence $Conv(\mathcal{BD}, \mathcal{L}) \Rightarrow Conv(\mathcal{S}, \mathcal{L})$. As explained above, we know that $Conv(\mathcal{BD}, \mathcal{L})$. Thus we have convergence of the original algorithm $\mathcal{S}$. Note that classical proof methods require human intervention involving quite subtle well-founded measures (see [4]).

6.2 Proving a Liveness Property of Termination Detection

A distributed algorithm terminates when it reaches a global configuration where no further step is applicable, but some individual machines may be unaware that the computation has terminated. A *termination detection* algorithm is an algorithm that observes the system computation and detects that the computation has reached a terminal configuration of the underlying algorithm (see [25], p.268). In our framework, given a set *Term* of terminal configurations (closed under $\mathcal{S}$), the *liveness* property $Liv(\mathcal{S}, Term)$ of a termination detection algorithm can be stated as follows: once a configuration of *Term* is reached, the algorithm terminates after a finite number of steps. This is depicted to the left in Fig. 3. Formally:

Definition 15. Let $\mathcal{S}$ be a rewrite system and *Term* a set of configurations closed under $\mathcal{S}$. We say that $\mathcal{S}$ satisfies the property $Liv(\mathcal{S}, Term)$, if there is no infinite ground derivation starting from an element of *Term*.

The method described before does not apply directly for proving $Liv(\mathcal{S}, Term)$, but can be easily adapted as follows. We consider the reverse system $\mathcal{S}^{-1}$, i.e., the system where the right- and left-hand sides of the rules have been exchanged. If we consider derivations via $\mathcal{S}^{-1}$ instead of $\mathcal{S}$, the property is now depicted to the right in Fig. 3. Proving that any execution in *Term* via $\mathcal{S}$ is finite is equivalent to proving that any execution in *Term* via $\mathcal{S}^{-1}$ is finite, i.e., $Conv(\mathcal{S}^{-1}, \neg Term)$. Assuming $(\alpha), (\beta), (\gamma)$ for $\mathcal{S}$ and *Term*, we can use our method to prove $Conv(\mathcal{S}^{-1}, \neg Term)$, hence $Liv(\mathcal{S}, Term)$, since the following conditions hold:

α'. $\neg Term$ is regular (since *Term* is regular),
β'. $\neg Term$ is closed under $\mathcal{S}^{-1}$ (since *Term* is closed under $\mathcal{S}$),
γ'. $\mathcal{S}^{-1} - Top_{\mathcal{S}^{-1}}$ is Noetherian (since $\mathcal{S} - Top_{\mathcal{S}}$ is Noetherian, and $\mathcal{S}$ is length-preserving).

Formally we have:

Theorem 16. *Consider a system $\mathcal{S}$ such that $\mathcal{S} - Top_{\mathcal{S}}$ is Noetherian. Suppose also that Term is regular and closed under $\mathcal{S}$. If $\mathcal{N}^*_{\mathcal{S}^{-1}}$ is closed under $\mathcal{S}^{-1}$ and if $\mathcal{G}_{\mathcal{S}^{-1}} \subseteq \neg Term$, then we have $Liv(\mathcal{S}, Term)$.*

Checking the closure of $\mathcal{N}^*_{\mathcal{S}^{-1}}$ and inclusion of $\mathcal{G}_{\mathcal{S}^{-1}}$ are both decidable properties. In contrast to self-stabilizing algorithms where any configuration is reachable, termination detection algorithms often concern only a restricted set of "admissible" configurations, e.g., configurations with at most one token. Taking this into account, Theorem 16 can be refined by replacing $N^*_{\mathcal{S}^{-1}}$ with $N^*_{\mathcal{S}^{-1}} \cap Adm$ where *Adm* is a regular language of (open) words containing all admissible configurations. This is illustrated in [15] on the termination detection algorithm of Dijkstra-Feijen-van Gasteren (cf., [13,25] and [19]).

7 Implementation and Experimental Results

Our approach has been implemented in SICStus Prolog using the Finite State Automata Utilities of Gertjan van Noord [23]. The resulting program consists of approximately 400 lines of Prolog code (100 clauses) excluding the FSA utilities.

The core of the implementation uses Caucal's algorithm for computing the reachable configurations by prefix and suffix rewriting via systems $\mathcal{S}_{pre}$ (the prefix extensions of $\mathcal{S}$) and $\mathcal{S}_{suf}$ (the suffix extensions of $\mathcal{S}$) given an initial word t_0 of the form $u_0 \odot v_0$. The algorithm boils down to construction of finite automata $A_1(\mathcal{S}_{pre}, t_0)$ and $A_2(\mathcal{S}_{suf}, t_0)$ characterizing the reflexive-transitive closure of $\rightarrow_{\mathcal{S}_{pre}}$ and $\rightarrow_{\mathcal{S}_{suf}}$ applied to $\odot v_0$ and $u_0 \odot$ respectively.

We have run our program on several algorithms drawn from Beauquier-Debas [4], Ghosh [17], Hoepman [18] and Dijkstra-Feijen-van Gasteren [13]. (Ghosh's and Hoepman's systems involve 3-letters rewrite systems instead of 2, but our method extends to them in a natural way.) Running each example takes at most a few seconds on a 200MHz Pentium Pro with 128MB of memory (including various clean-up operations to facilitate human reading of the output). The main code of the implementation is given in [15] as well as typical automata output by the program on the examples.

8 Conclusions

We have given a new procedure, based on prefix and suffix rewriting, for constructing a regular set $\mathcal{G}_\mathcal{S}$ of "unavoidable" configurations of a rewrite system $\mathcal{S}$. This is useful e.g., for proving the convergence $Conv(\mathcal{S},\mathcal{L})$ of $\mathcal{S}$ to a set $\mathcal{L}$ of configurations. The procedure works under a novel and decidable sufficient condition: the closure of the narrowing language $\mathcal{N}^*_\mathcal{S}$. If $\mathcal{N}^*_\mathcal{S}$ is not closed, we suggest a way to find a simplified form $\mathcal{T}$ of $\mathcal{S}$ which is often closed in practice, and such that $Conv(\mathcal{T},\mathcal{L}) \Rightarrow Conv(\mathcal{S},\mathcal{L})$. An implementation exists, and all the examples treated manually in [3], have been processed here for the first time in a uniform automated manner. The method applies only to linear or circular arrays of machines. As an extension it would be interesting to investigate more complicated topologies such as trees, using concepts of tree automata and term rewriting [9] instead of string rewriting. Nevertheless

- We have simplified the framework of [3], giving a statement of the main theorem in a more elegant way.
- We have given a new and significant application of prefix rewriting (analysis of convergence of distributed algorithms) in addition to those given in [6,21].
- Using the idea to reverse the system $\mathcal{S}$, we have shown how to apply the method for proving the liveness of termination detection algorithms over parameterized rings (in addition to the correctness of self-stabilizing algorithms).

It is well-known that proving a property such as convergence of parametric linear array of finite machines is an undecidable problem [2]. Our work can be seen as a step towards isolating sufficient conditions of parametric systems, which are both met in practice and allow to verify some liveness properties mechanically.

Acknowledgement. We are grateful to Moshe Vardi for suggesting the use of prefix rewriting for automatic generation of regular languages.

References

1. P.A. Abdulla, A. Bouajjani, B. Jonsson and M. Nilsson. "Handling global conditions in parameterized system verification". *Proc. CAV'99*, LNCS 1633, Springer-Verlag, 1999, pp. 134-145.
2. K.R. Apt and D.C. Kozen. "Limits for automatic verification of finite-state concurrent systems". *Information Processing Letters 22*, 1986, pp. 307–309.
3. J. Beauquier, B. Bérard, L. Fribourg and F. Magniette. "Proving convergence of self-stabilizing systems using first-order rewriting and regular languages". *Distributed Computing 14:2*, 2001, pp. 83-95.

4. J. Beauquier and O. Debas. "An optimal self-stabilizing algorithm for mutual exclusion on uniform bidirectional rings". *Proc. 2nd Workshop on Self-Stabilizing Systems*, Las Vegas, 1995, pp. 226-239.
5. R.V. Book and F. Otto. *String-Rewriting Systems.* Springer-Verlag, 1993.
6. A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems and P. Wolper. "An Efficient Automata Approach to Some Problems on Context-Free Grammars". *Information Processing Letters* 74 (5-6), 2000, pp. 221-227.
7. A. Bouajjani, B. Jonsson, M. Nilsson and T. Touili. "Regular Model-Checking". *Proc. CAV'00*, LNCS 1855, Springer-Verlag, 2000, pp. 403-418.
8. D. Caucal. "On the regular structures of prefix rewriting". *Proc CAAP'90*, Copenhagen, 1990, LNCS 431, Springer, pp. 87-102.
9. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison and M. Tommasi. *Tree Automata Techniques and Applications.* Available on `http://www.grappa.univ-lille3.fr/tata/`.
10. N. Dershowitz. "Termination of Linear Rewriting Systems". *Proc. ICALP*, LNCS 115, Springer-Verlag, 1981, pp. 448-458.
11. N. Dershowitz and J.-P. Jouannaud. "Rewrite Systems". *Handbook of Theoretical Computer Science*, vol. B, Elsevier - MIT Press, 1990, pp. 243-320.
12. E.W. Dijkstra. "Self-stabilizing systems in spite of distributed control". *Comm. ACM* 17:11, 1974, pp. 643-644.
13. E.W. Dijkstra, W.H.J Feijen and A.J.M. van Gasteren. "Derivation of a Termination Detection Algorithm for Distributed Computations". *Information Processing Letters*, vol. 16, 1983, pp. 217-219.
14. S. Dolev. *Self-Stabilization.* MIT-Press, 2000.
15. M. Duflot, L. Fribourg and U. Nilsson. *Unavoidable Configurations of Parameterized Rings of Processes.* Research Report LSV-00-10, ENS de Cachan, Nov. 2000. (Available on `http://www.lsv.ens-cachan.fr/~fribourg/publis.html`)
16. L. Fribourg and H. Olsén. "Reachability sets of parametrized rings as regular languages". *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97)*, volume 9 of Electronical Notes in Theoretical Computer Science. Elsevier Science, 1997.
17. S. Ghosh. "An Alternative Solution to a Problem on Self-Stabilization". *ACM TOPLAS* 15:4, 1993, pp. 735-742.
18. J.-H. Hoepman. "Uniform Deterministic Self-Stabilizing Ring-Orientation on Odd-Length Rings". *Proc. 8th Workshop on Distributed Algorithms*, LNCS 857, Springer-Verlag. 1994, pp.265-279.
19. B. Jonsson and M. Nilsson. "Transitive closures of regular relations for verifying infinite-state systems". *Proc. TACAS'00*, LNCS 1785, Springer-Verlag, 2000.
20. Y. Kesten, O. Maler, M. Marcuse, A. Pnueli and E. Shahar. "Symbolic Model-Checking with Rich Assertional Languages". *Proc. CAV'97*, LNCS 1254, Springer-Verlag, 1997, pp. 424-435.
21. O. Kupferman and M.Y. Vardi. "An Automata-Theoretic Approach to Reasoning about Infinite-State Systems". *Proc. CAV'00*, LNCS 1855, Springer-Verlag, 2000.
22. D. Lesens, N. Halbwachs and P. Raymond. "Automatic Verification of Parametrized Linear Network of Processes". Proc. *POPL'97*, Paris, 1997.
23. G. van Noord. *Fsa Utilities User Manual Version 5*, 1998.
24. A. Pnueli and E. Shahar. "Liveness and Acceleration in Parameterized Verification". *Proc. CAV'00*, LNCS 1855, Springer-Verlag, 2000, pp. 328-343.
25. G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.
26. P. Wolper and B. Boigelot. "Verifying systems with infinite but regular state spaces". *Proc. CAV'98*, LNCS 1427, Springer-Verlag, 1998, pp. 88-97.