Expand, Enlarge and Check... Made Efficient

Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin

Université Libre de Bruxelles – Département d'Informatique – CPI 212
Boulevard du Triomphe, B–1050 Bruxelles, Belgium
{gigeerae,jraskin,lvbegin}@ulb.ac.be

Abstract. The coverability problem is decidable for the class of well-structured transition systems. Until recently, the only known algorithm to solve this problem was based on symbolic backward reachability. In a recent paper, we have introduced the theory underlying a new algorithmic solution, called 'Expand, Enlarge and Check', which can be implemented in a forward manner. In this paper, we provide additional concepts and algorithms to turn this theory into efficient forward algorithms for monotonic extensions of Petri nets and Lossy Channels Systems. We have implemented a prototype and applied it on a large set of examples. This prototype outperforms a previous fine tuned prototype based on backward symbolic exploration and shows the practical interest of our new algorithmic solution. The data used in the experiments and more relevant information about 'Expand, Enlarge and Check' can be found at: `http://www.ulb.ac.be/di/ssd/ggeeraer/eec/`

1 Introduction

Model-checking is nowadays widely accepted as a powerful technique for the automatic verification of reactive systems that have natural finite state abstractions. However, many reactive systems are only naturally modelled as infinite-state systems. Consequently, a large (and successful) research effort has recently focused on the application of model-checking techniques to infinite-state models such as FIFO channel systems [2], (extensions of) Petri nets and automata with counters [14], broadcast protocols [8], etc.

One of the positive results is the decidability of the *coverability problem* for *well-structured transition systems* (WSTS for short). WSTS enjoy an infinite set of states that is well-quasi ordered by $\leq$ and their transition relation is monotonic w.r.t $\leq$. Examples of such systems are Petri nets and their monotonic extensions [5, 14], broadcast protocols [7], lossy channel systems [2]. The *coverability problem* asks whether a given WSTS S can reach a state of a given $\leq$-upward closed set of states U.

A general algorithm (i.e. a procedure that always terminates) is known to solve the coverability problem for WSTS [1, 11]. It symbolically manipulates upward-closed sets of states, obtained by unrolling the transition relation in a *backward* fashion. Unfortunately, backward search is seldom efficient in practice [13], and the only complete forward approach known so far is the Karp-Miller algorithm that can only be applied to a small subclass of WSTS: Petri nets. All the previous attempts to generalize this procedure have led to incomplete forward approaches that are either not guaranteed to terminate (e.g.: [7], as shown in [8]) or that can be inconclusive due to over-approximation [4].

Nevertheless, we have recently proposed a new schema of algorithms, called 'Expand, Enlarge and Check' (EEC for short), to solve the coverability problem for a large class of WSTS (those that enjoy reasonable effectiveness requirements, see [12] for the details). EEC works basically as follows. It constructs a sequence of pairs of approximations of the set of reachable states: an under-approximation (built during the 'Expand phase') and an over-approximation (built during the 'Enlarge' phase). Some basic results from the theory of well-quasi ordering and recursively enumerable sets allow us to show that positive instances of the coverability problem are answered by the sequence of under-approximations while negative instances are answered by the over-approximations after a finite number of iterations. The theory and the proofs are very elegant and furthermore the schema is really promising from the practical point of view because it can be implemented in a forward manner.

In this paper, we show that, indeed, EEC can be turned into an efficient algorithm to solve the coverability problem in a forward manner. In particular, we show how to implement the EEC efficiently for the two most practically important classes of WSTS in the literature: monotonic extensions of Petri Nets (EPN for short) and for Lossy Channel Systems (LCS for short). Those two classes are useful for the analysis of parametric systems and communication protocols. To obtain efficient algorithms from the EEC schema, we have to get over two obstacles: first, during the 'Expand' phase, we have to analyze finite graphs that can be very large. Second, during the 'Enlarge' phase, we have to approximate sets of successors efficiently. To solve the first problem, we show that we can always turn a WSTS into a *lossy* WSTS that respects the same coverability properties and for which the graph during the 'Expand' phase is monotonic. The coverability problem can often be solved efficiently in monotonic graphs because (roughly) only $\leq$-maximal states of the graph must be explored. We provide an efficient algorithm for that exploration. This algorithm is also applicable during the 'Enlarge' phase in the case of EPN. The second problem is difficult for LCS only. We provide here a way to construct efficiently the most precise approximations of the set of the successors of a downward-closed set of LCS configurations.

On the basis of those two conceptual tools, we have implemented a prototype to analyze coverability properties of EPN and LCS. We have applied the prototype to a large set of examples taken in the literature and compared its performances with our fine-tuned implementation of the backward search in the case of EPN. For LCS, the only available tools are implementing either a potentially non-terminating analysis or an over-approximation algorithm that is not guaranteed to conclude due to over-approximation. The performance of our prototype are very encouraging and often much better than those of the backward search prototype.

The rest of the paper is organized as follows. In Section 2, we recall some basic notions about well-quasi orderings, WSTS and the coverability problem. In Section 3, we summarize the results of our previous paper about the EEC schema. In Section 4, we show how to efficiently explore monotonic graphs to establish coverability properties and show that the graphs that we have to analyze during the 'Expand' phase are monotonic when the lossy abstraction is applied. In Section 5, we show how EPN and LCS can be analyzed efficiently with the EEC schema. and provide practical evidence that it can compete advantageously with other techniques. Finally, we draw some conclusions in a last section.

2 Preliminaries

In this section, we recall some fundamental results about *well-quasi orderings* and *well-structured transition systems* (the systems we analyze here). We show how to *finitely* represent upward- and downward-closed sets of states (which will allow us to devise *symbolic* algorithms), and discuss And-Or graphs and monotonic graphs (useful to represent abstractions of systems).

Well quasi-orderings and adequate domains of limits A *well quasi ordering* $\leq$ on C (wqo for short) is a *reflexive* and *transitive* relation s. t. for any infinite sequence $c_0 c_1 \ldots c_n \ldots$ of elements in C, there exist i and j, with $i < j$ and $c_i \leq c_j$. We note $c_i < c_j$ if $c_i \leq c_j$ but $c_j \not\leq c_i$.

Let $\langle C, \leq \rangle$ be a well-quasi ordered set. A $\leq$-*upward closed set* $U \subseteq C$ is such that for any $c \in U$, for any $c' \in C$ such that $c \leq c'$, $c' \in U$. A $\leq$-*downward-closed set* $D \subseteq C$ is such that for any $c \in D$, for any $c' \in C$ such that $c' \leq c$, $c' \in D$. The set of $\leq$-*minimal* elements $\mathsf{Min}(U)$ of a set $U \subseteq C$ is a minimal set such that $\mathsf{Min}(U) \subseteq U$ and $\forall s' \in U : \exists s \in \mathsf{Min}(U) : s \leq s'$. The next proposition is a consequence of wqo:

Proposition 1. *Let $\langle C, \leq \rangle$ be a wqo set and $U \subseteq C$ be an $\leq$-upward closed set, then:* $\mathsf{Min}(U)$ *is finite and* $U = \{c \mid \exists c' \in \mathsf{Min}(U) : c' \leq c\}$.

Thus, any $\leq$-upward closed set can be *effectively represented* by its finite set of minimal elements. To obtain a finite representation of $\leq$-downward-closed sets, we must use well-chosen limit elements $\ell \notin C$ that represent $\leq$-downward closures of infinite increasing chains of elements.

Definition 1 ([12]). *Let $\langle C, \leq \rangle$ be a well-quasi ordered set and L be a set s.t. $L \cap C = \emptyset$. The tuple $\langle L, \sqsubseteq, \gamma \rangle$ is called an* adequate domain of limits *for $\langle C, \leq \rangle$ if the following conditions are satisfied: (L_1: representation mapping) $\gamma : L \cup C \to 2^C$ associates to each element in $L \cup C$ a $\leq$-downward-closed set $D \subseteq C$, and for any $c \in C$, $\gamma(c) = \{c' \in C \mid c' \leq c\}$. γ is extended to sets $\mathcal{S} \subseteq L \cup C$ as follows: $\gamma(\mathcal{S}) = \cup_{c \in \mathcal{S}} \gamma(c)$; ($\mathsf{L}_2$: top element) There exists a special element $\top \in L$ such that $\gamma(\top) = C$; (L_3: precision order) The set $L \cup C$ is partially ordered by $\sqsubseteq$, where: $d_1 \sqsubseteq d_2$ iff $\gamma(d_1) \subseteq \gamma(d_2)$; ($\mathsf{L}_4$: completeness) for any $\leq$-downward-closed set $D \subseteq C$, there exists a finite set $D' \subseteq L \cup C$: $\gamma(D') = D$.*

Well-structured transition systems and coverability problem A *transition system* is a tuple $S = \langle C, c_0, \rightarrow \rangle$ where C is a (possibly infinite) set of states, $c_0 \in C$ is the initial state, $\rightarrow \subseteq C \times C$ is a transition relation. We note $c \rightarrow c'$ for $\langle c, c' \rangle \in \rightarrow$. For any state c, $\mathsf{Post}(c)$ denotes the set of one-step successors of c, i.e. $\mathsf{Post}(c) = \{c' | c \rightarrow c'\}$, this function is extended to sets: for any $C' \subseteq C$, $\mathsf{Post}(C') = \bigcup_{c \in C'} \mathsf{Post}(c)$. Without lost of generality, we assume that $\mathsf{Post}(c) \neq \emptyset$ for any $c \in C$. A *path* of S is a sequence of states $c_1, c_2, \ldots, c_k$ such that $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k$. A state c' is reachable from a state c, noted $c \rightarrow^* c'$, if there exists a path $c_1, c_2, \ldots, c_k$ in S with $c_1 = c$ and $c_k = c'$. Given a transition system $S = \langle C, c_0, \rightarrow \rangle$, $\mathsf{Reach}(S)$ denotes the set $\{c \in C \mid c_0 \rightarrow^* c\}$.

Definition 2. *A transition system $S = \langle C, c_0, \rightarrow \rangle$ is a* well-structured transition system *(*WSTS*) [1, 11] for the quasi order $\leq \subseteq C \times C$ if the two following properties hold: (*W_1*: well-ordering) $\leq$ is a well-quasi ordering and (*W_2*: monotonicity) $\forall c_1, c_2, c_3 \in C : c_1 \leq c_2$ and $c_1 \rightarrow c_3$ implies $\exists c_4 \in C : c_3 \leq c_4$ and $c_2 \rightarrow c_4$. It is* lossy *if its transition relation satisfies the following additional property: (*W_3*) $\forall c_1, c_2, c_3 \in C$ such that $c_1 \rightarrow c_2$ and $c_3 \leq c_2$, we have $c_1 \rightarrow c_3$.*

Problem 1. The *coverability problem for well-structured transition systems* is defined as follows: 'Given a well-structured transition system S and the $\leq$-upward closed set $U \subseteq C$, determine whether $\mathsf{Reach}(S) \cap U \neq \emptyset$?'

To solve the coverability problem, we use the notion of covering set:

Definition 3. *For any* WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$*, the* covering set *of S (*$\mathsf{Cover}(S)$*), is the $\leq$-downward closure of* $\mathsf{Reach}(S)$*:* $\mathsf{Cover}(S) = \{c | \exists c' \in \mathsf{Reach}(S) : c \leq c'\}$.

Proposition 2 ([14]). *For any* WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$, $\mathsf{Cover}(S)$ *is such that for any $\leq$-upward closed set $U \subseteq C$:* $\mathsf{Reach}(S) \cap U = \emptyset$ *iff* $\mathsf{Cover}(S) \cap U = \emptyset$.

Finite representation For any WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$ with an adequate domain of limits $\langle L, \sqsubseteq, \gamma \rangle$ for $\langle C, \leq \rangle$, by property L_4 of Definition 1, there exists a finite subset $\mathsf{CS}(S) \subseteq L \cup C$ s.t. $\gamma(\mathsf{CS}(S)) = \mathsf{Cover}(S)$. In the sequel, $\mathsf{CS}(S)$ is called a *coverability set* of the covering set $\mathsf{Cover}(S)$ and finitely represents that set.

Lossiness abstraction Any WSTS can be turned into a lossy WSTS that respects the same coverability property. Definition 4 and Proposition 3 formalize this:

Definition 4. *The lossy version of a* WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$ *is the lossy* WSTS $\mathsf{lossy}(S) = \langle C, c_0, \rightarrow_\ell, \leq \rangle$ *where* $\rightarrow_\ell = \{(c, c') \mid \exists c'' \in C : c \rightarrow c'' \wedge c' \leq c''\}$.

Proposition 3. *For any* WSTS S *with set of configurations C: (i)* $\mathsf{Cover}(S) = \mathsf{Cover}(\mathsf{lossy}(S))$*; and (ii) for any $\leq$-upward closed set $U \subseteq C$:* $\mathsf{Reach}(S) \cap U = \emptyset$ *iff* $\mathsf{Cover}(\mathsf{lossy}(S)) \cap U = \emptyset$.

Abstractions The EEC algorithm considers two kinds of abstractions. To represent them, we introduce two types of graphs. First, a $\overline{\preccurlyeq}$*-monotonic graph* is a finite graph $\langle V, \Rightarrow, v_i \rangle$ associated with an order $\overline{\preccurlyeq} \subseteq V \times V$ such that for any $v_1, v_2, v_3 \in V$ with $v_1 \Rightarrow v_2$ and $v_1 \overline{\preccurlyeq} v_3$, there exists $v_4 \in V$ with $v_2 \overline{\preccurlyeq} v_4$ and $v_3 \Rightarrow v_4$. Notice that $\overline{\preccurlyeq}$-monotonic graphs are finite WSTS. Second, an *And-Or graph* is a tuple $G = \langle V_A, V_O, v_i, \Rightarrow \rangle$ where $V = V_A \cup V_O$ is the (finite) set of nodes (V_A is the set of "And" nodes and V_O is the set of "Or" nodes), $V_A \cap V_O = \emptyset$, $v_i \in V_O$ is the initial node, and $\Rightarrow \subseteq (V_A \times V_O) \cup (V_O \times V_A)$ is the transition relation such that for any $v \in V_A \cup V_O$, there exists $v' \in V_A \cup V_O$ with $(v, v') \in \Rightarrow$.

Definition 5. *A* compatible unfolding *of an And-Or graph $G = \langle V_A, V_O, v_i, \Rightarrow \rangle$ is an infinite labelled tree $T_G = \langle N, root, B, \Lambda \rangle$ where: (i) N is the set of nodes of T_G, (ii) $root \in N$ is the root of T_G, (iii) $B \subseteq N \times N$ is the transition relation of T_G, (iv) $\Lambda : N \to V_A \cup V_0$ is the labelling function of the nodes of T_G by nodes of G that respects the three following compatibility conditions (Λ is extended to sets of nodes in the usual way):* (C_1)*$\Lambda(root) = v_i$; (C_2) for all $n \in N$ such that $\Lambda(n) \in V_A$, we have that (a) for all nodes $v' \in V_O$ such that $\Lambda(n) \Rightarrow v'$, there exists one and only one $n' \in N$ such that $B(n, n')$ and $\Lambda(n') = v'$, and conversely (b) for all nodes $n' \in N$ such that $B(n, n')$, we have $\Lambda(n) \Rightarrow \Lambda(n')$.* (C_3) *for all $n \in N$ such that $\Lambda(n) \in V_O$, we have that: there exists one and only one $n' \in N$ such that $B(n, n')$, and $\Lambda(n) \Rightarrow \Lambda(n')$.*

Problem 2. The *And-Or Graph Avoidability Problem* is defined as follows:'Given an And-Or graph $G = \langle V_A, V_O, v_i, \Rightarrow \rangle$ and a set $E \subseteq V_A \cup V_O$, does there exist $T = \langle N, root, \Lambda, B \rangle$, a compatible unfolding of G, such that $\Lambda(N) \cap E = \emptyset$?' When the answer is positive, we say that E is *avoidable* in G. It is well-known that this problem is complete for *PTIME*.

3 Expand, Enlarge and Check

This section recalls the fundamentals of the 'Expand, Enlarge and Check' algorithm (see [12] for more details). As stated in the introduction, the EEC algorithm builds a sequence of pairs of approximations. The first one, is an under-approximation of the set of reachable states, which allows one to decide positive instances of the coverability problem. The latter one over-approximates the set of reachable states and is suitable to decide negative instances.

More precisely, given a WSTS $S = \langle C, c_0, \to, \leq \rangle$, and a set of limits L, the algorithm considers in parallel a sequence of subsets of C: $C_0, C_1, \ldots$ and a sequence of limit elements: $L_0, L_1, \ldots$ s.t. (i) $\forall i \geq 0 : C_i \subseteq C_{i+1}$, (ii) $\forall c \in \mathsf{Reach}(S) : \exists i \geq 0 : c \in C_i$, and (iii) $c_0 \in C_0$; and (iv) $\forall i \geq 0 : L_i \subseteq L_{i+1}$, (v) $\forall \ell \in L : \exists i \geq 0 : \ell \in L_i$ and (vi) $\top \in L_0$. Given a set C_i, one can construct an *exact partial reachability graph* (EPRG for short) $\mathsf{EPRG}(S, C_i)$ which is an under-approximation of the system. Similarly, given a set L_i, one builds an over-approximation of the system under the form of an And-Or Graph:

Definition 6. *Given a* WSTS $S = \langle C, c_0, \to, \leq \rangle$ *and a set* $C' \subseteq C$*, the* EPRG *of S is the transition system* $\mathsf{EPRG}(S, C') = \big\langle C', c_0, \big(\to \cap (C' \times C') \big) \big\rangle$

Definition 7. *Given a* WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$*, an adequate domain of limits* $\langle L, \sqsubseteq, \gamma \rangle$ *for* $\langle C, \leq \rangle$*, a finite subset* $C' \subseteq C$ *with* $c_0 \in C'$*, and a finite subset* $L' \subseteq L$ *with* $\top \in L'$*, the And-Or graph* $G = \langle V_A, V_O, v_i, \Rightarrow \rangle$*, noted* $\mathsf{Abs}(S, C', L')$*, is defined as follows:* (A_1) $V_O = L' \cup C'$*;* (A_2) $V_A = \{S \in 2^{L' \cup C'} \setminus \{\emptyset\} \mid \nexists d_1 \neq d_2 \in S : d_1 \sqsubseteq d_2\}$*;* (A_3) $v_i = c_0$*;* $(\mathsf{A}_{4.1})$ *for any* $n_1 \in V_A, n_2 \in V_O$*:* $(n_1, n_2) \in \Rightarrow$ *if and only if* $n_2 \in n_1$*;* $(\mathsf{A}_{4.2})$ *for any* $n_1 \in V_O, n_2 \in V_A$ *:* $(n_1, n_2) \in \Rightarrow$ *if and only if (i)* successor covering: $\mathsf{Post}(\gamma(n_1)) \subseteq \gamma(n_2)$*, (ii)* preciseness: $\neg \exists n \in V_A : \mathsf{Post}(\gamma(n_1)) \subseteq \gamma(n) \subset \gamma(n_2)$*.*

We can now state (Theorem 1) the adequacy of these abstractions. Then, Theorem 2 tells us that we will eventually find the right abstractions in order to decide the coverability problem. The EEC algorithm directly follows from this last theorem: it enumerates the pairs of C_i and L_i, and, for each of them, (1 – 'Expand') builds $\mathsf{EPRG}(S, C_i)$, (2 – 'Enlarge') builds $\mathsf{Abs}(S, C_i, L_i)$ and (3 – 'Check') looks for an error trace in $\mathsf{EPRG}(S, C_i)$ and checks the avoidability of the bad states in $\mathsf{Abs}(S, C_i, L_i)$. Further details may be found in [12].

Theorem 1. *Given a* WSTS $S = \langle C, c_0, \rightarrow \rangle$ *with domain of limits* $\langle L, \sqsubseteq \rangle$*, and an* $\leq$*-upward-closed* $U \subseteq C$*:* $\forall i \geq 0$*:* **If** $\mathsf{Reach}(\mathsf{EPRG}(S, C_i)) \cap U \neq \emptyset$ **then** $\mathsf{Reach}(S) \cap U \neq \emptyset$*.* **If** U *is avoidable in* $\mathsf{Abs}(S, C_i, L_i)$*,* **then** $\mathsf{Reach}(S) \cap U = \emptyset$*.*

Theorem 2. *Given a* WSTS $S = \langle C, c_0, \rightarrow \rangle$ *with domain of limits* $\langle L, \sqsubseteq \rangle$*,* $C' \subseteq C$*,* $L' \subseteq L$*, and an* $\leq$*-upward-closed* $U \subseteq C$*:* **If** $\mathsf{Reach}(S) \cap U \neq \emptyset$*,* **then** $\exists i \geq 0 : \mathsf{Reach}(\mathsf{EPRG}(S, C_i)) \cap U \neq \emptyset$*.* **If** $\mathsf{Reach}(S) \cap U = \emptyset$*,* **then** $\exists i \geq 0$ *s.t.* U *is avoidable in* $\mathsf{Abs}(S, C_i, L_i)$*.*

The next propositions give properties of these abstractions. In particular, Proposition 5 says that, if C' is $\leq$-downward-closed and S is lossy, $\mathsf{EPRG}(S, C')$ is a finite $\leq$-monotonic graph. However, as stated in the introduction, these graphs are often too large. Thus, efficient procedures to decide the coverability problem on such graphs are highly desirable. This motivates Section 4.

Proposition 4. *Let* S *be a* WSTS *and* $\langle L, \sqsubseteq \rangle$ *be an adequate domain of limits. For any And-Or graph* $\mathsf{Abs}(S, C', L') = \langle V_A, V_O, v_i, \Rightarrow \rangle$*, for any* $v_1, v_2, v_3 \in V_A \cup V_O$*:* $v_1 \Rightarrow v_2, \gamma(v_1) \subseteq \gamma(v_3)$ *implies* $\exists v_4 \in V_A \cup V_O : v_3 \Rightarrow v_4, \gamma(v_2) \subseteq \gamma(v_4)$*.*

Proposition 5. *Given a* lossy WSTS $S = \langle C, c_0, \rightarrow, \leq \rangle$*, and a* $\leq$-downward-closed set $C' \subseteq C$*:* $\mathsf{EPRG}(S, C')$ *is a* $\leq$*-monotonic graph.*

4 Efficient exploration of $\overline{\preccurlyeq}$-monotonic graphs

This section is devoted to the presentation of Algorithm 1 which efficiently decides the coverability problem on $\overline{\preccurlyeq}$-monotonic graphs. It is based on ideas borrowed from the algorithm to compute the minimal coverability set of Petri nets, presented in [9]. However, as recently pointed out in [10], this latter algorithm is flawed and may in certain cases compute an under-approximation of the actual

Algorithm 1: Coverability

Data : $\mathcal{G} = \langle V, \Rightarrow, v_i \rangle$: $\overline{\preccurlyeq}$-monotonic graph; $U \subseteq V$: $\overline{\preccurlyeq}$-upward-closed set of state.
Result : `true` when U is reachable in $\mathcal{G}$; `false` otherwise.
begin
  Let $\mathcal{T} = \langle N, n_0, B, \Lambda \rangle$ be the tree computed as follows:
  $\mathit{to\_treat} = \{n_0\}$ such that $\Lambda(n_0) = v_i$, $N = \{n_0\}$, $B = \emptyset$;
  while $\mathit{to\_treat} \neq \emptyset$ **do**
    while $\mathit{to\_treat} \neq \emptyset$ **do**
      chose and **remove** n in $\mathit{to\_treat}$;
      foreach *successor* v *of* $\Lambda(n)$ **do**
        Add n' with $\Lambda(n') = v$ as successor of n;
1   **if** $\neg \exists n' \in N : B^*(n', n) \wedge \Lambda(n) \overline{\preccurlyeq} \Lambda(n')$ **then** add n' into $\mathit{to\_treat}$;
        else $\mathsf{removed}(n') = \mathtt{true}$;
      Apply reduction rules (see Algorithm 2);
    /* reuse of nodes already computed */
2   **while** $\exists n, n' \in N : \neg\mathsf{removed}(n) \wedge \mathsf{removed}(n') \wedge \neg\mathsf{covered}(\Lambda(n'), N) \wedge B(n, n')$ **do**
      $\mathsf{removed}(n') = \mathtt{false}$;
    /*construction of new nodes */
3   **while** $\exists n \in (N \setminus \mathit{to\_treat}) : \exists v \in V : \Lambda(n) \Rightarrow' v \wedge \neg\mathsf{removed}(n) \wedge \neg\mathsf{covered}(v, N)$ **do**
      add n to $\mathit{to\_treat}$;
  if $\exists n \in N : \mathsf{removed}(n) = \mathtt{false}, \Lambda(n) \in U$ **then return** `true`;
  else return `false`;
end

Algorithm 2: Reduction rules

while *true* **do**
1 $\exists n, n' \in N : \Lambda(n) \overline{\prec} \Lambda(n') \wedge \neg B^*(n, n') \wedge \neg\mathsf{removed}(n) \wedge \neg\mathsf{removed}(n')$ **do**
    foreach $n'' \in \mathsf{nodes}(\mathsf{subtree}(n))$ **do** $\mathsf{removed}(n'') = \mathtt{true}$;
    foreach $n'' \in \mathsf{nodes}(\mathsf{subtree}(n)) \cap \mathit{to\_treat}$ **do**
      remove n'' from $\mathit{to\_treat}$;
  []
2 $\exists n, n' \in N : \Lambda(n) \overline{\prec} \Lambda(n') \wedge B^*(n, n') \wedge \neg\mathsf{removed}(n) \wedge \neg\mathsf{removed}(n')$ **do**
    Let $S = \{n'' \in \mathsf{leaves}(\mathsf{subtree}(n')) \mid n'' \notin \mathit{to\_treat}\}$;
    $\mathsf{Replace\_subtree}(n, n', \mathcal{T}, \mathit{to\_treat})$;
    foreach $n'' \in S$ **do**
      if $\neg \exists n' \in N : B^*(n', n) \wedge \Lambda(n) \overline{\preccurlyeq} \Lambda(n')$ **then** add n'' into $\mathit{to\_treat}$;
  []
  true **do**
    exit while loop ;

minimal coverability set. Algorithm 1 corrects this bug in the context of finite graphs. At the end of the section, we show how to exploit it in EEC.

Algorithm 1 receives a $\overline{\preccurlyeq}$-monotonic graph $\mathcal{G} = \langle V, \Rightarrow, v_i \rangle$ and constructs a finite tree $\mathcal{T} = \langle N, n_0, B, \Lambda \rangle$, with set of nodes N, root node $n_0 \in N$, set of arcs $B \subseteq N \times N$ (in the following we denote by B^* the transitive closure of B) and labelling function $\Lambda : N \mapsto V$. We denote by $\mathsf{leaves}(\mathcal{T})$ the set of leaves of $\mathcal{T}$; by $\mathsf{subtree}(n)$, the maximal sub-tree of $\mathcal{T}$ rooted in $n \in N$; and by $\mathsf{nodes}(\mathcal{T})$ the set of nodes of $\mathcal{T}$. Given a tree $\mathcal{T}$, and two nodes n and n', the function $\mathsf{Replace\_subtree}(n, n', \mathcal{T}, \mathit{to\_treat})$ replaces, in $\mathcal{T}$, the subtree $\mathsf{subtree}(n)$ by $\mathsf{subtree}(n')$ and removes from *to_treat* the nodes $n'' \in (\mathsf{leaves}(\mathsf{subtree}(n)) \setminus \mathsf{leaves}(\mathsf{subtree}(n'))) \cap \mathit{to\_treat}$. We also attach a predicate $\mathsf{removed}(n)$ to each node n of a tree, which is initially *false* for any node. When $\mathsf{removed}(n)$ is true, the node n is *virtually* removed from the tree. It is however kept in memory, so that it can be later put back in the tree (this makes sense since the bug in [9] occurs when nodes are deleted by mistake). The function $\mathsf{covered}(v, N)$ returns $\mathtt{true}$ iff there is a node $n \in N$ with $v \overline{\preccurlyeq} \Lambda(n)$ and $\mathsf{removed}(n) = \mathtt{false}$.

Sketch of the algorithm The tree built by Algorithm 1 is the reachability tree of the $\overline{\preccurlyeq}$-monotonic graph $\mathcal{G}$ on which some reduction rules are applied in order to keep maximal elements only, (so that the labels of the tree finally computed form a coverability set of $\mathcal{G}$). The sketch of the algorithm is as follows. The inner **while** loop constructs the tree by picking up a node n from *to_treat* and adding its successors. When a successor n' is smaller than or equal to one of its ancestors n'' ($\Lambda(n') \overline{\preccurlyeq} \Lambda(n'')$), we stop the development of n' (line 1) Then, reduction rules (Algorithm 2) are applied: *(i)* when the tree contains two nodes n and n' such that $\Lambda(n) \overline{\preccurlyeq} \Lambda(n')$ and n is not an ancestor of n', $\mathsf{subtree}(n)$ does not need to be developed anymore and is *removed* (that is, all the nodes n'' of $\mathsf{subtree}(n)$ are tagged: $\mathsf{removed}(n'') = \mathtt{true}$, and removed from *to_treat*); *(ii)* when the tree contains two nodes n and n' such that $\Lambda(n) \overline{\preccurlyeq} \Lambda(n')$ and n is an ancestor of n', we replace $\mathsf{subtree}(n)$ by $\mathsf{subtree}(n')$. As mentioned above, the inner **while** loop may fail to compute a coverability set of $\mathcal{G}$ and may only compute an under-approximation. To cope with this problem, we test, at the end of the inner **while** loop if a coverability set has been computed. More precisely (line 2), we look at all the nodes n' such that $\mathsf{removed}(n') = \mathtt{true}$ and that are direct successors of a node n actually in the tree (i.e.: $\mathsf{removed}(n) = \mathtt{false}$). When we find that such an n' is not covered by a node actually in the tree, we set $\mathsf{removed}(n')$ back to $\mathtt{false}$. This step is iterated up to stabilization. Then (line 3), we add into *to_treat* the nodes n with $\mathsf{removed}(n) = \mathtt{false}$ such that (i) the successor nodes of n have not been developed yet and (ii) there exists one successor v of $\Lambda(n)$ that is not covered by non-stopped nodes. If *to_treat* is not empty at the end of these steps, it means that the inner **while** loop has computed an under-approximation of the coverability set. In that case, the main loop is iterated again. Otherwise, when *to_treat* is empty, it is easy to see that for each node n in the tree such that $\mathsf{removed}(n) = \mathtt{false}$ all the successors of $\Lambda(n)$ are covered by nodes n' of the tree such that $\mathsf{removed}(n') = \mathtt{false}$. Since the root node of the tree covers

v_i, we conclude that $\{v \mid \exists$ a node n of the tree: $\Lambda(n) = v, \mathsf{removed}(n) = \mathtt{false}\}$ is a coverability set.
The next theorem states the correctness of Algorithm 1 (proof in appendix):

Theorem 3. *Algorithm 1, when applied to the $\overline{\preccurlyeq}$-monotonic graph $\mathcal{G}$ and the $\overline{\preccurlyeq}$-upward closed set U, always terminates and returns* `true` *if and only if there exists a node $v \in U$ such that v is reachable from v_i in $\mathcal{G}$.*

Remark 1. When $\Rightarrow$ is computable, Algorithm 1 can compute $\mathcal{T}$ without disposing of the whole graph $\mathcal{G}$ (v_i only is necessary). In that case, Algorithm 1 efficiently explores (large) $\overline{\preccurlyeq}$-monotonic graphs without building them entirely.

Application to EEC Thanks to Proposition 5, we can apply Algorithm 1 to any EPRG built at the 'Expand' phase, if the WSTS is lossy (but by Proposition 3, we can always take the lossy version of any WSTS), and the sets C_i are $\preccurlyeq_e$-downward-closed. We show in Section 5 that this is not restrictive in practice.

Algorithm 1 is also useful to improve the 'Enlarge' phase in the case where the And-Or graph is *degenerated*. An And-Or graph is degenerated whenever each Or-node has only one successor. Hence a degenerated And-Or graph $G = \langle V_A, V_O, \Rightarrow, v_i\rangle$ is equivalent to a plain graph $G' = \langle V_O, v_i, \Rightarrow'\rangle$ where we have $v \Rightarrow' v'$ if an only if $\exists v'' \in V_A : v \Rightarrow v'' \Rightarrow v'$. From Proposition 4, G' is a $\sqsubseteq$-monotonic graph, for any WSTS with adequate domain of limits $\langle L, \sqsubseteq, \gamma\rangle$.

5 Expand, Enlarge and Check in practice

In this section, we specialize the EEC schema to obtain efficient procedures to decide the coverability problem on two classes of WSTS of practical interest: the monotonic extensions of Petri nets (EPN) and the lossy channel systems (LCS).

Since And-Or graphs for EPN are always degenerated [12], we can exploit the efficient procedure described in Section 4 in both the 'Expand' and the 'Enlarge' phase. As far as LCS are concerned, the main difficulty relies in the construction of the And-Or graph: the 'Expand' phase requests an efficient procedure to compute the most precise successors of any Or-node. We show how to solve this problem.

5.1 Extended Petri nets

In this subsection we consider monotonic extensions of the well-known Petri net model (such as Petri nets with transfer arcs, a.s.o., see [5]). Due to the lack of space, we refer the reader to [12] for the syntax. An EPN P defines a WSTS $S = \langle \mathbb{N}^k, \mathbf{m}_0, \rightarrow\rangle$ where k is the number of places of P and $\rightarrow \subseteq \mathbb{N}^k \times \mathbb{N}^k$ is a transition relation induced by the transitions of the EPN (see [12] for details).

Domain of limits To apply the schema of algorithm to extensions of Petri nets, we proposed in [12] to consider the domain of limits $\langle \mathcal{L}, \preccurlyeq_e, \gamma(.)\rangle$ where $\mathcal{L} = (\mathbb{N} \cup \{+\infty\})^k \setminus \mathbb{N}^k$, $\preccurlyeq_e \subseteq (\mathbb{N} \cup \{+\infty\})^k \times (\mathbb{N} \cup \{+\infty\})^k$ is such that $\langle m_1, \ldots, m_k\rangle \preccurlyeq_e \langle m'_1, \ldots, m'_k\rangle$ if and only if $\forall 1 \le i \le k : m_i \le m'_i$ (where $\le$ is the natural order over $\mathbb{N} \cup \{+\infty\}$. In particular: $c < +\infty$ for all $c \in \mathbb{N}$). γ is defined as: $\gamma(\mathbf{m}) = \{\mathbf{m'} \in \mathbb{N}^k \mid \mathbf{m'} \preccurlyeq_e \mathbf{m}\}$.

The sequences of C_i's and L_i's are defined as follows: (D_1) $C_i = \{0, \ldots, i\}^k \cup \{\mathbf{m} | \mathbf{m} \preccurlyeq_e \mathbf{m}_0\}$, i.e. C_i is the set of markings where each place is bounded by i (plus the $\preccurlyeq_e$-downward closure of the initial marking); (D_2) $L_i = \{\mathbf{m} \in \{0, \ldots i, +\infty\}^k \mid \mathbf{m} \notin \mathbb{N}^k\}$.

Efficient algorithm To achieve an efficient implementation of EEC, and according to Proposition 3, we consider the lossy version of EPN (that are lossy WSTS) to decide the coverability problem on EPN. As the sets C_i are $\preccurlyeq_e$-downward-closed, we use the algorithm of Section 4 to efficiently compute the 'Expand' phase.

The 'Enlarge' phase is improved by using the method of [12] to compute the successors of any Or-node of the And-Or graph. Moreover, the And-Or graphs are always degenerated in the case of (lossy) EPN [12], hence we also apply Algorithm 1 during that phase.

Note that, although the set of successors of a configuration of a lossy EPN can be large, $\preccurlyeq_e$-monotonic graphs and And-Or graphs allow us to consider the maximal successors only.

Experiments We have implemented the techniques described so far in a prototype capable of analyzing EPN. We have run the prototype on several examples from the literature. Table 1 reports on selected results (See Appendix B for more results). The case studies retained here are mainly abstractions of multi-threaded Java programs (most of them taken from [14]).

When applied to these examples, the basic symbolic backward algorithm of [1] seldom produces a result within the time limit of 20 minutes we have fixed (column **Pre**). A heuristic presented in [6] uses place-invariants to guide the search and improves the performance of the prototype (which has been fined tuned during several years of research). Still, it might not terminate on some examples (column **Pre+Inv**). On the contrary, although the EEC prototype is still in its infancy, it *always* produces a result within the time limit, with *comparable performance*, on *all* the examples (column **EEC**). This demonstrates the practical superiority of the forward analysis at work in EEC.

5.2 Lossy channel systems

Lossy channel systems (LCS) are systems made up of a finite number of automata which communicate through lossy FIFO channels, by writing to or reading from the channels when a transition is fired. This model is well-studied, see e.g. [4, 3]. In particular, the Simple Regular Expressions (sre), a symbolic representation for downward-closed sets of configurations of LCS, have been defined. Algorithms to symbolically compute classical operations, such as the union, intersection or the Post, have been devised. In the sequel, we will rely on this background.

Example					EEC		Pre+Inv		Pre	
cat.	name	P	T	Safe	Time	Mem	Time	Mem	Time	Mem
PNT	Java	44	37	×	8.47	23,852	1.40	3,816	time out	
PNT	delegatebuffer	50	52	√	180.78	116,608	time out		time out	
PNT	leabasicapproach	16	12	×	0.01	1,608	0.02	2,248	0.03	2,376
PNT	queuedbusyflag	80	104	√	28.87	21,388	time out		time out	
PN	pncsacover	31	36	×	7.54	13,704	40.83	5,012	time out	

Table 1. results obtained on Intel Xeon 3Ghz with 4Gb of memory : **cat.**: category of example (PNT = Petri nets with transfer arcs, PN = (unbounded) Petri net); **P**: number of places; **T**: number of transitions; **EEC**: EEC algorithm; **Pre + Inv**: Backward approach, using invariant heuristics; **Pre**: same without invariants. All the memory consumptions in KB and times in second.

Preliminaries In order to keep the following discussion compact, we will consider, without loss of generality, a LCS $\mathcal{C}$ made up of a single automaton (with set of states Q) and a single FIFO channel (initially empty, with alphabet Σ). A configuration of $\mathcal{C}$ is a pair $\langle q, w\rangle$, where $q \in Q$ is the state of the automaton, and $w \in \Sigma^*$ is the content of the channel. Let $S_{\mathcal{C}}$ be the set of configurations of $\mathcal{C}$. A transition of $\mathcal{C}$ is of the form $\langle s_1, Op, s_2\rangle$ where $s_1, s_2 \in Q$ and Op is $!a$ (add a to the channel), or $?a$ (consume a on the channel), or *nop* (no modification of the channel), for any $a \in \Sigma$. The semantics is the classical one, see [4]. The w.q.o. $\preccurlyeq_w \subseteq \Sigma^* \times \Sigma^*$ is defined as follows: $w_1 \preccurlyeq_w w_2$ iff w_1 is a (non-necessarily contiguous) subword of w_2. A *downward-closed regular expression* (dc-re) is a regular expression that is either $(a+\varepsilon)$ for some $a \in \Sigma$, or $(a_1 + a_2 + \ldots + a_n)^*$ for $\{a_1, a_2, \ldots, a_n\} \subseteq \Sigma$. Given a dc-re d, $\alpha(d)$ (the *alphabet of* d) is the set of all the elements of Σ that occur in d. A *product* (of dc-re) is either ε or an expression of the form $d_1 \cdot d_2 \cdot \ldots \cdot d_n$, where d_1, d_2,..., d_n are dc-re. Given a product p, $[\![p]\!] \subseteq \Sigma^*$ denotes the ($\preccurlyeq_w$-downward-closed) language generated by p, and $|p|$, denotes its size, i.e., the number of dc-re that compose it (for $w \in \Sigma^*$, $|w|$ is defined the usual way). Let $P(\Sigma)$ denote the set of all products built from Σ.

Domain of limits Let $\mathcal{L}(\Sigma, Q)$ denote the set of limits $\{\langle q,p\rangle | q \in Q, p \in P(\Sigma)\} \cup \{\top\}$. For any $\langle q,p\rangle \in \mathcal{L}(\Sigma,Q)$, $[\![\langle q,p\rangle]\!]$ denotes the set of configurations $\langle q,w\rangle \in \mathcal{S}_{\mathcal{C}}$ such that $w \in [\![p]\!]$. We define the function $\gamma : \mathcal{L}(\Sigma, Q) \to 2^{\mathcal{S}_{\mathcal{C}}}$ such that (i) $\gamma(\top) = Q \times \Sigma^*$ and (ii) $\gamma(\langle q,p\rangle) = [\![\langle q,p\rangle]\!]$, for any $\langle q,p\rangle \in \mathcal{L}(\Sigma,Q) \setminus \{\top\}$.We define $\overline{\sqsubseteq} \subseteq \mathcal{L}(\Sigma,Q) \times \mathcal{L}(\Sigma,Q)$ as follows : $c_1 \overline{\sqsubseteq} c_2$ if and only if $\gamma(c_1) \subseteq \gamma(c_2)$. When $c_1 \overline{\sqsubseteq} c_2$ but $c_2 \overline{\not\sqsubseteq} c_1$, we write $c_1 \overline{\sqsubset} c_2$.

Let us now define the sets of concrete and limit elements we will consider at each step. We define $C_i = \{\langle q,w\rangle \mid \langle q,w\rangle \in \mathcal{S}_{\mathcal{C}}, |w| \leq i\}$, i.e. C_i is the set of states where the channel contains at most i characters. Similarly, we define L_i as follows: $L_i = \{\langle q,p\rangle \in \mathcal{L}(\Sigma,Q) \mid |p| \leq i\} \cup \{\top\}$, i.e. L_i contains the limits where a product of length at most i represents the channel (plus $\top$).

Efficient algorithm In the case of LCS, the And-Or graph one obtains is, in general, not degenerated. Hence, the techniques presented in Section 4 can be used along the 'Expand' phase only (the C_is are $\preccurlyeq_w$-downward-closed and the

WSTS induced by LCS are lossy). In the sequel, we try nonetheless to improve the 'Enlarge' phase by showing how to directly compute (i.e. without enumeration of configurations) the set of (most precise) successors of any Or-node. Notice that following the semantics of LCS, the Post operation can add at most one character to the channel. Hence, we only need to be able to approximate precisely any $c \in L_{i+1} \cup C_{i+1}$ by elements in $L_i \cup C_i$.

Over-approximation of a product Given a product $p \neq \varepsilon$ and a natural number $i \geq 1$ such that $|p| \leq i+1$, let us show how to directly compute, the most complete and most precise set of products that over-approximate p, and whose size is at most i. For this purpose, we first define an auxiliary function $L(p)$. Let $p = d_1 \cdot d_2 \cdots d_n$ be a product. $L(p) = \bigcup_{1 \leq i \leq n-1} \{d_1 \cdots d_{i-1} \cdot (c_1 + \ldots + c_m)^* \cdot d_{i+2} \cdots d_n | \{c_1, \ldots, c_m\} = \alpha(d_i) \cup \alpha(d_{i+1})\}$. We can now define $\mathsf{Approx}(p,i)$ for $|p| \leq i+1$ and $i \geq 1$. $\mathsf{Approx}(p,i) = \{p\}$ when $|p| \leq i$, and $\mathsf{Approx}(p,i) = \{q \in L(p) \mid \nexists q' \in L(p) : q' \sqsubseteq q\}$ when $|p| = i+1$.

Proposition 6. *Given a natural number i and a product of* dc-re *p such that $|p| \leq i+1$, for all product of* dc-re *p' such that (i) $[\![p]\!] \subseteq [\![p']\!]$; (ii) $|p'| \leq i$ and (iii) $p' \notin \mathsf{Approx}(p,i)$: $\exists p'' \in \mathsf{Approx}(p,i) : [\![p'']\!] \subseteq [\![p']\!]$.*

Hence, Approx allows us to over-approximate any limit element of L_{i+1} by elements of L_i. In order to handle elements of C_{i+1}, we extend the definition of Approx as follows. Let i be a natural number and $w = a_1 \ldots a_n \in \Sigma^*$ (with $n \leq i+1$) be a word, then $\mathsf{Approx}(w,i) = w$ when $n \leq i$, and $\mathsf{Approx}(w,i) = \mathsf{Approx}(p_w,i)$ with $p_w = (a_1+\varepsilon) \cdots (a_n+\varepsilon)$ otherwise. Remark that w and p_w both define the same $\preccurlyeq_w$-downward-closed set, and Proposition 6 remains valid.

When the LCS has more than one channel, a state (limit) associates a word (or a product of dc-re) to each channel. It that case, the best approximation can be computed by taking the product of the best approximations for each channel.

Experiments We have built a prototype to decide the coverability problem for LCS. It implements the improvements of the 'Expand' and 'Enlarge' phases presented above. Another improvement in the construction of the And-Or graph consists in computing only the states that are reachable from the initial state. Table 2 reports on the performance of the prototype when applied to various examples of the literature: the Alternating Bit Protocol (ABP), and the Bounded Retransmission Protocol (BRP), on which we verify five different properties [4]. Table 2 shows very promising results with our simple prototype.

Case study	**S**	**E**	**C**	**EEC**	**Case study**	**S**	**E**	**C**	**EEC**
ABP	48	192	2	0.18	BRP_3	480	2,460	2	0.35
BRP_1	480	2,460	2	0.19	BRP_4	480	2,460	2	0.41
BRP_2	480	2,460	2	0.19	BRP_5	640	3,370	2	0.19

Table 2. Results obtained on Intel Xeon 3Ghz with 4Gb of memory : **S** and **E**: number of states and edges of the graph ; **C**: number of channels; **EEC**: execution time (in second) of an implantation of EEC.

6 Conclusion

In this paper we have pursued a line of research initiated in [12] with the introduction of the 'Expand, Enlarge and Check' algorithm. We have shown in the present work that, for a peculiar subclass of WSTS, one can derive efficient practical algorithms from this theoretical framework. We have presented an efficient method to decide the coverability problem on monotonic graphs. This solution fixes a bug, for the finite case, in the minimal coverability tree algorithm of [9]. It can always be applied to improve the 'Expand' phase. In the case of *extended Petri nets*, it can also be used to improve the 'Enlarge' phase. In the case of *lossy channel systems*, we have also shown how to improve the 'Expand' phase, by building the And-Or graph in an efficient way. We have implemented these techniques in two prototypes, working in a *forward* fashion. Their excellent behaviours clearly demonstrate the practical interest of EEC.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-state Systems. In *Proc. of LICS'96*, pages 313–321. IEEE, 1996.
2. P.A. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *Proc. of LICS'93*, pages 160–170. IEEE, 1993.
3. Parosh Abdulla, Aurore Annichini, and Ahmed Bouajjani. Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. In *Proc. of TACAS'99*, number 1579 in LNCS, pages 208–222. Springer-Verlag, 1999.
4. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Form. Methods Syst. Des.*, 25(1):39–65, 2004.
5. G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In *Proc. of ICATPN'94*, vol. 815 of *LNCS*, pages 179–198. Springer, 1994.
6. G. Delzanno, J.-F. Raskin, and L. Van Begin. Attacking Symbolic State Explosion. In *Proc. of CAV 2001*, vol. 2102 of *LNCS*, pages 298–310. Springer, 2001.
7. E. A. Emerson and K. S. Namjoshi. On Model Checking for Non-deterministic Infinite-state Systems. In *Proc. of LICS '98*, pages 70–80. IEEE, 1998.
8. J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. of LICS'99*, pages 352–359. IEEE, 1999.
9. A. Finkel. The minimal coverability graph for Petri nets. In *Proc. of APN'93*, vol. 674 of *LNCS*, pages 210–243. Springer, 1993.
10. A. Finkel, G. Geeraerts, J.-F. Raskin, and L. Van Begin. A counter-example to the minimal coverability tree algorithm Technical report ULB 535. Available at `http://www.ulb.ac.be/di/ssd/ggeeraer/eec/`.
11. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
12. G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, Enlarge and Check: new algorithms for the coverability problem of WSTS. In *Proc. of FSTTCS'04*, vol. 3328 of LNCS, pages 287–298. Springer-Verlag, 2004.
13. T. A. Henzinger, O. Kupferman, and S. Qadeer. From *pre*historic to *post*modern symbolic model checking. *Formal Methods in System Design*, 23(3):303–327, 2003.
14. L. Van Begin. *Efficient Verification of Counting Abstractions for Parametric systems.* PhD thesis, Université Libre de Bruxelles, Belgium, 2003.

A Proofs of Theorem 3

We split the proof in two propositions. The first one states that the algorithm terminates, the second states the partial correctness. The first one relies on the following (obvious) facts:

Fact 1 *At each step of Algorithm 1, the following holds. For all node n of the tree such that there exists a predecessor n' with $\Lambda(n)\overline{\preccurlyeq}\Lambda(n')$: n has no successors.*

Fact 2 *After loop of line 1 of Algorithm 1, if there exists $n \in N$: $\mathsf{removed}(n) = \mathtt{false}$ and a configuration v such that $\Lambda(n) \Rightarrow v$ and $\mathsf{covered}(v, \mathcal{T}) = \mathtt{false}$, then $n \in$ to_treat.*

Fact 3 *Algorithm 1 never loops forever into the loops inside the main while loop.*

To prove the first proposition, we also need the following lemma.

Lemma 1. *After loop of line 3 in Algorithm 1, if to_treat is not empty then at least one node is added into the tree during the next iteration of the main while loop.*

Proof. First, notice that *to_treat* is always empty when entering the loop of line 3.

Second, let us show that if the loop of line 3 adds the node n into *to_treat* then no successor of $\Lambda(n)$ has been developed, i.e. n has no successors in the tree built up to now. Suppose that it is not the case, i.e. n is such that $\mathsf{removed}(n) = \mathtt{false}$, there exists v such that $\Lambda(n) \Rightarrow v$, $\mathsf{covered}(v, N) = \mathtt{false}$, and there exists a successor n' of n with $\Lambda(n') = v$, $\mathsf{removed}(n') = \mathtt{true}$. However, following the condition of the loop of line 2, we have that when we exit that loop (to enter the loop of line 3), there is no $n, n' \in N$: $\mathsf{removed}(n) = \mathtt{false}$, $\mathsf{removed}(n') = \mathtt{true}$, $\mathsf{covered}(\Lambda(n'), N) = \mathtt{false}$ and n' is a successor of n. We obtain a contradiction and we conclude that when we exit the loop of line 3 all the nodes into *to_treat* are such that their successors are not developed into the tree, hence the successors of at least one node into *to_treat* will be added into the tree during the next iteration of the main while loop. □

Proposition 7. *Algorithm 1 terminates.*

Proof. First notice that a node cannot be replaced infinitely often. Indeed, when we replace a node n by another one n', we have that $\Lambda(n)\overline{\preccurlyeq}\Lambda(n')$. Since there is a finite number of nodes in the monotonic graph, after a finite amount of time i_1 the root node does not change anymore, after a finite amount of time i_2 the successors of the root node do not change anymore, etc.

Let us now prove that the algorithm does not compute an infinite tree. Suppose that it is not the case. Since that tree is finitely branching, by applying König's lemma we conclude that the infinite tree contains at least one infinite branch. Since the monotonic graph contains a finite number of nodes the infinite branch contains a node n with one predecessor node n' such that $\Lambda(n') = \Lambda(n)$.

By fact 1 the branch is not developed from n. We conclude that the branch is finite and we obtain a contradiction.

Suppose that Algorithm 1 does not terminate. Since it computes a finite tree and the nodes are replaced only a finite number of time, we conclude that at some point of the infinite execution, the tree is not modified anymore by the algorithm. Fact 3 implies that even if the tree is not modified anymore, Algorithm 1 passes through the loop of the line 3 and exits this loop. Since the algorithm does not terminate, *to_treat* is not empty when it exits the loop of line 3. By Lemma 1, we have that at least one node is added into the tree during the next execution of the main while loop, hence the tree is modified, and we obtain a contradiction. □

Proposition 8. *When Algorithm 1 terminates, the set of nodes* $N' = \{\Lambda(n) \mid n \in N, \mathsf{removed}(n) = \mathtt{false}\}$ *is a coverability set of of the* $\overline{\preccurlyeq}$*-monotonic graph* $\mathcal{G}$*.*

Proof. To prove the result, we have to prove that

1. $\{v \mid \exists v' \in N' : v \overline{\preccurlyeq} v'\} \subseteq \{v \mid \exists v' \in \mathsf{Reach}(\mathcal{G}) : v \overline{\preccurlyeq} v'\}$;
2. $\{v \mid \exists v' \in \mathsf{Reach}(\mathcal{G}) : v \overline{\preccurlyeq} v'\} \subseteq \{v \mid \exists v' \in N' : v \overline{\preccurlyeq} v'\}$.

Point 1 is immediate since it is easy to show that for any node n in the tree we have that $\Lambda(n)$ is reachable from the initial node of $\mathcal{G}$. Point 2 is equivalent to $\mathsf{Reach}(\mathcal{G}) \subseteq \{v \mid \exists v' \in N' : v \overline{\preccurlyeq} v'\}$ that can be easily proved by induction on the length of the paths (number of transitions) to reach any configuration in $\mathsf{Reach}(\mathcal{G})$.

Base case: The base case is trivial since it is easy to show that the root node *root* of the tree computed by Algorithm 1 is such that $v_i \overline{\preccurlyeq} \Lambda(root)$ (notice also that we have $\mathsf{removed}(root) = \mathtt{false}$).

Induction step: Suppose that v can be reached thanks to the sequence $v_i \Rightarrow v_1 \Rightarrow \cdots \Rightarrow v_{k-1} \Rightarrow v$ of length $k \geq 1$. By induction hypothesis, the node v_{k-1} (obtained after a sequence of length $k-1$) is such that there exists $n \in N : \mathsf{removed}(n) = \mathtt{false}, v_{k-1} \overline{\preccurlyeq} \Lambda(n)$. By Fact 2, we have that for all v'' such that $\Lambda(n) \Rightarrow v''$ there exists a $n' \in N : \mathsf{removed}(n') = \mathtt{false}, v'' \overline{\preccurlyeq} \Lambda(n')$. By monotonicity, that implies that there exists $n' \in N : \mathsf{removed}(n') = \mathtt{false}, v \overline{\preccurlyeq} \Lambda(n')$.
□

We can now state Theorem 3:

Theorem 3 *Algorithm 1, when applied to the* $\overline{\preccurlyeq}$*-monotonic graph* $\mathcal{G}$ *and the* $\overline{\preccurlyeq}$*-upward closed set* U*, always terminates and returns* `true` *if and only if there exists a node* $v \in U$ *such that* v *is reachable from* v_i *in* $\mathcal{G}$*.*

Proof. Immediate from Proposition 7 and Proposition 8. □

B Experiments on EPN

Table 3 contains the results of the whole set of experiments we made on EPN. A web page with a description of the examples may be found at
`http://www.ulb.ac.be/di/ssd/ggeeraer/eec/`

Example					EEC		Pre+Inv		Pre	
cat.	**name**	**P**	**T**	**Safe**	**Time**	**Mem**	**Time**	**Mem**	**Time**	**Mem**
PNT	CSMbroad	13	8	√	0.01	1,596	0.02	2,252	0.07	2,464
PNT	MOESI	9	11	√	0.01	1,536	0	2,160	0	2,156
PNT	german	12	8	√	0	1,472	0	2,128	0.07	2,476
PNT	Java	44	37	×	8.47	23,852	1.40	3,816	time out	
PNT	Javasanserreur	44	38	√	5.44	16,624	0.02	2,548	time out	
PNT	consprod	18	14	√	0.05	2,012	0	1,096	21.54	5,364
PNT	consprod2	18	14	√	0.05	2,012	0	2,196	0.64	2,556
PNT	delegatebuffer	50	52	√	180.78	116,608	time out		time out	
PNT	examplelea	48	42	√	6.23	10,448	4.92	9,184	time out	
PNT	leabasicapproach	16	12	×	0.01	1,608	0.02	2,248	0.03	2,376
PNT	queuedbusyflag	80	104	√	28.87	21,338	time out		time out	
PNT	simplejavaexample	32	28	×	0.19	2,656	0.56	2,784	2.70	3,720
PNT	transthesis	90	117	√	17.42	33,232	0.12	3,852	time out	
PNT	efm	6	5	√	0	1,316	0.01	2,096	0.01	2,096
PN	basicME	5	4	√	0	1,064	0	1,308	0	2,076
PN	csm	14	13	√	0.02	1,748	0.09	2,516	0.11	2,512
PN	fms	22	20	√	54.43	52,796	0	1,356	28.49	8,048
PN	kanban	16	16	√	4.18	10,688	0	1,068	1111.45	24,756
PN	mesh2x2	32	32	√	0.33	3,164	0.89	2,740	0.81	2,740
PN	mesh3x2	52	54	√	1.59	6,044	10.92	4,568	8.57	4,604
PN	multipool	18	21	√	0.76	4,052	0	1,124	1.46	2,980
PN	pncsacover	31	36	×	7.54	13,704	40.83	5,012	time out	
BPN	lamport	11	9	√	0	1,428	0.02	2,132	0.06	2,468
BPN	newdekker	16	14	√	0.01	1,684	0.05	2,260	0.54	2,536
BPN	newrtp	9	12	√	0	1,404	0	1,096	0.05	2,340
BPN	peterson	14	12	√	0	1,520	0.04	2,244	0.12	2,504
BPN	read-write	13	9	√	0.19	2,224	0.16	2,480	0.42	2,496

Table 3. results obtained on INTEL XEON 3Ghz with 4Gb of memory : **cat.** : category of example (PN = (unbounded) Petri net, PNT = Petri net with transfer arcs, BPN = bounded Petri net) **P** : number of places, **T**: number of transitions. **EEC** : results obtained with the "Expand, Enlarge and Check" algorithm. **Pre + Inv**: results obtained with a backward approach, using invariant heuristics, **Pre**: same without invariants. All the memory consumptions in KB and times in second.

C Proof of Proposition 6

This section is devoted to the proof of proposition 6, that states (roughly speaking) that $\mathsf{Approx}(p, i)$ is the set of most precise products of lengths at most i that over-approximate p. The proof of this proposition is a bit technical and we shall divide it into several auxiliary lemmata and propositions. But beforehand, we need to recall the definition of the *normal form* of a product of dc-re [4]. After this, we provide several obvious properties of dc-re. Then, we split the proof of Proposition 6 by considering separately the case where p is not in normal form (subsection C.2) and the case where it is in normal form (subsection C.3). These two results allow us to conclude (subsection C.4).

C.1 Definitions and properties

This section states several useful definitions and properties. In particular, we recall what is the normal form of a product of dc-re, and how to compute it.

Definition 8. *Given an alphabet Σ, an* ε-downward-closed regular expression *(ε-dc-re for short) is a dc-re of the form $(a+\varepsilon)$ (for some $a \in \Sigma$). A* $*$-downward-closed regular expression *($*$-dc-re for short) is a dc-re of the form $(a_1 + a_2 + \cdots + a_n)^*$ (with $\{a_1, a_2, \ldots, a_n\} \subseteq \Sigma$).*

Definition 9 ([4]). *A product of dc-re $p = d_1 \cdot d_2 \cdot \ldots \cdot d_n$ is in normal form if and only if for each $1 \leq i < n$: $[\![d_i \cdot d_{i+1}]\!] \not\subseteq [\![d_i]\!]$ and $[\![d_i \cdot d_{i+1}]\!] \not\subseteq [\![d_{i+1}]\!]$.*

Notice that to any product of dc-re p corresponds an unique product of dc-re in normal form p' such that $[\![p]\!] = [\![p']\!]$ (see [4]). To build that product of dc-re in normal form p' from p, we can apply Algorithm 3.

Algorithm 3: Normal_Form [4]

```
Data   : p : product of dc-re.
Result : P : product of dc-re in normal form.
begin
    Let P = p ;
    while P = d_1 · ... · d_k is not in normal form do
        Let i be s.t. 1 ≤ i < k and [[d_i · d_{i+1}]] ⊆ [[d_i]] or [[d_i · d_{i+1}]] ⊆ [[d_{i+1}]] ;
        if [[d_i · d_{i+1}]] ⊆ [[d_i]] then
            P = d_1 · ... · d_i · d_{i+2} · ... · d_k ;
        else
            P = d_1 · ... · d_{i-1} · d_{i+1} · ... · d_k ;
    return P;
end
```

From Definition 8, we immediately obtain the following obvious properties:

Property 1. The following holds:

1. If d and d' are two ε-dc-re, then $\alpha(d) \subseteq \alpha(d')$ if and only if $[\![d]\!] \subseteq [\![d']\!]$.
2. If d is a dc-re and d' is a $*$-dc-re, then $\alpha(d) \subseteq \alpha(d')$ if and only if $[\![d]\!] \subseteq [\![d']\!]$.
3. Let d be a $*$-dc-re. There is no ε-dc-re d' such that $[\![d]\!] \subseteq [\![d']\!]$.
4. As a consequence of the three previous points: if d and d' are two dc-re such that $[\![d]\!] \subseteq [\![d']\!]$, we have: $\alpha(d) \subseteq \alpha(d')$.

C.2 p is not in normal form

In this section we prove a first intermediate result, i.e. that Proposition 6 holds when the product p, on which Approx is applied, is *not* in normal form. This is done by first considering the following lemma:

Lemma 2. *Given a product of* dc-re p *such that* (i) $|p| \leq i+1$ *and* (ii) p *is not in normal form, we have* $\mathsf{Approx}(p,i) = \{p'\}$ *such that* $[\![p]\!] = [\![p']\!]$.

Proof. Suppose that $p = d_1 \cdot \ldots \cdot d_n$. We consider two cases : (i) $|p| \leq i$ and (ii) $|p| = i+1$. In the first case, we directly have $\mathsf{Approx}(p,i) = \{p\}$.

Let us consider the second case, i.e. when $|p| = i+1$. From Definition 9 we know that there exists $1 \leq j < n : [\![d_j \cdot d_{j+1}]\!] \subseteq [\![d_j]\!]$ or $[\![d_j \cdot d_{j+1}]\!] \subseteq [\![d_{j+1}]\!]$. By construction, there is in $L(p)$ a product $p' = d_1 \cdot d_2 \cdots d_{j-1} \cdot d' \cdot d_{j+2} \cdots d_n$ with $d' = (a_1 + a_2 + \cdots a_m)^*$ and $\{a_1, a_2, \ldots a_m\} = \alpha(d_j) \cup \alpha(d_{j+1})$. Thus $[\![d']\!] = [\![d_j \cdot d_{j+1}]\!]$ and $[\![p]\!] = [\![p']\!]$. Moreover, we have that $\forall q \in L(p) : [\![p]\!] \subseteq [\![q]\!]$, hence, $\forall q \in L(p) : [\![p']\!] \subseteq [\![q]\!]$. Since we keep in $\mathsf{Approx}(p,i)$ only the most precise elements from $L(p)$, we conclude that $\mathsf{Approx}(p,i) = \{p'\}$ with $[\![p]\!] = [\![p']\!]$. □

This allows us to prove the main result of this subsection:

Proposition 9. *Given a natural number* i *and a product of* dc-re p *that is not in normal form such that* $|p| \leq i+1$*: for all products of* dc-re p' *such that (i)* $[\![p]\!] \subseteq [\![p']\!]$*; (ii)* $|p'| \leq i$ *and (iii)* $p' \notin \mathsf{Approx}(p,i)$ *: there exists* $p'' \in \mathsf{Approx}(p,i)$ *such that* $[\![p'']\!] \subseteq [\![p']\!]$.

Proof. Let p' be a product of dc-re such that $[\![p]\!] \subseteq [\![p']\!]$, $|p'| \leq i$ and $p' \notin \mathsf{Approx}(p,i)$, and let us show that there exists a product in $\mathsf{Approx}(p,i)$ that is at least as precise as p'. From Lemma 2, we have that $\mathsf{Approx}(p,i) = \{p''\}$ with $[\![p]\!] = [\![p'']\!]$. Since $[\![p]\!] \subseteq [\![p']\!]$ and $[\![p]\!] = [\![p'']\!]$, we have $[\![p'']\!] \subseteq [\![p']\!]$. □

C.3 p is in normal form

We can now consider the case where p is in normal form. This case is a bit more difficult than the former one, and we need two auxiliary lemmata. The first one goes further than Property 1 and states that, whenever we want to over-approximate a given dc-re d by a *product* of dc-re, this product must contain at least another dc-re d' with $[\![d]\!] \subseteq [\![d']\!]$:

Lemma 3. *Let* d *be a* dc-re *and* $p = d_1 \cdot d_2 \cdots d_n$ *be a product of* dc-re*. If* $[\![d]\!] \subseteq [\![p]\!]$*, then there exists* $1 \leq i \leq n$ *such that* $[\![d]\!] \subseteq [\![d_i]\!]$.

Proof. We prove the contraposition, i.e. *if* for every $1 \leq i \leq n$ we have $[\![d]\!] \not\subseteq [\![d_i]\!]$, *then* $[\![d]\!] \not\subseteq [\![p]\!]$. We consider separately the cases where d is of the form $(a+\varepsilon)$, $(a)^*$, or $(a_1 + a_2 + \cdots + a_\ell)^*$ with $\ell \geq 2$.

1. d **is of the form** $(a+\varepsilon)$ for some a. Since, for any $1 \leq i \leq n$: $[\![(a+\varepsilon)]\!] \not\subseteq [\![d_i]\!]$, and since $(a+\varepsilon)$ is an ε-dc-re, this implies, by Property 1 (points 1 and 2) that, for any $1 \leq i \leq n : \alpha(d) = \{a\} \not\subseteq \alpha(d_i)$. Thus, the symbol '$a$' does not appear in p, and the word a does not belong to $[\![p]\!]$. However, the word a is in $[\![d]\!]$. Hence $[\![d]\!] \not\subseteq [\![p]\!]$.
2. d **is of the form** $(a)^*$ for some a. Since, for any $1 \leq i \leq n$: $[\![(a)^*]\!] \not\subseteq [\![d_i]\!]$, and since $(a)^*$ is a $*$-dc-re, this implies, by Property 1 (points 2 and 4) that for any $1 \leq i \leq n$, either d_i is an ε-dc-re, or d_i is a $*$-dc-re with $\alpha(d) \not\subseteq \alpha(d_i)$, i.e. $a \notin \alpha(d_i)$. In other words, there is no $*$-dc-re in p whose alphabet contains a. Hence the word $a^{|p|+1}$ is not accepted by p, since at most $|p|$ 'a' can be generated by the ε-dc-re possibly present in p. However, this word is clearly in $[\![d]\!]$. Hence $[\![d]\!] \not\subseteq [\![p]\!]$.
3. d **is of the form** $(a_1 + a_2 + \cdots + a_\ell)^*$ **with** $\ell \geq 2$**.** This part of the proof is by induction on the length of p.
   Base case: $|p| = 1$. In this case p is a single dc-re d_1. Since $[\![d]\!] \not\subseteq [\![d_1]\!]$, the property is trivially verified.
   Inductive case: $|p| = i + 1$. The induction hypothesis is as follows: *for any product of* dc-re $p = d_1 \cdot d_2 \cdots d_k$ *where* $k \leq i$*: if for every* $1 \leq j \leq i : [\![d]\!] \not\subseteq [\![d_j]\!]$*, then* $[\![d]\!] \not\subseteq [\![p]\!]$. Let us show that this holds for any k up to and including $i+1$. Let $p' = d_1 \cdot d_2 \cdots d_{i+1}$, such that for any $1 \leq j \leq i+1 : [\![d]\!] \not\subseteq [\![d_j]\!]$. By induction hypothesis, $[\![d]\!] \not\subseteq [\![d_1 \cdots d_i]\!]$, hence there exists a word w that is in $[\![d]\!]$ but not in $[\![d_1 \cdots d_i]\!]$. Since $[\![d]\!] \not\subseteq [\![d_{i+1}]\!]$, and since d is of the form $(a_1 + a_2 + \cdots + a_\ell)^*$ with $\ell \geq 2$, we deduce that, either d_{i+1} is an ε-dc-re, or it is a $*$-dc-re with $\alpha(d) \not\subseteq \alpha(d_{i+1})$. But in the case where d_{i+1} is a ε-dc-re we also have that $\alpha(d) \not\subseteq \alpha(d_{i+1})$, because $|\alpha(d_{i+1})| = 1 < |\alpha(d)|$. Hence, in both cases, $\alpha(d) \not\subseteq \alpha(d_{i+1})$. Let thus c be a character in $\alpha(d) \setminus \alpha(d_{i+1})$. Clearly, the word $w \cdot c$ is in $[\![d]\!]$. However, it is not in $[\![p']\!]$. Indeed, if $w \cdot c \in [\![p']\!]$, then $w \cdot c$ should be in $[\![d_1 \cdots d_i]\!]$ because c cannot be generated by d_{i+1}. But in this case, w would also be in $[\![d_1 \cdots d_i]\!]$, since this set is $\preccurlyeq_w$-downward-closed. We derive a contradiction and conclude that $[\![d]\!] \not\subseteq [\![p]\!]$. □

The second auxiliary lemma extends this result by comparing two *products* of dc-re p and p' such that $[\![p]\!] \subseteq [\![p']\!]$. This lemma states (intuitively speaking) that one can establish a correspondence between the dc-re of p and p'. This correspondence must respect the order of the dc-re in p and p': if p is of the form $p_1 \cdot d \cdot p_2 \cdot e \cdot p_3$ (where d and e are dc-re), and d' and e' are the corresponding dc-re in p' then, d' must appear in p' *before* e' (this must hold for any d and e).

Lemma 4. *Let* $p = d_1 \cdot d_2 \cdots d_n$ *and* $p' = d'_1 \cdot d'_2 \cdots d'_k$ *be two products of* dc-re *in normal form, such that* $[\![p]\!] \subseteq [\![p']\!]$*. Then there exists a function* $\rho : \{1, \ldots, n\} \mapsto \{1, \ldots, k\}$ *such that:*

- *for any* $1 \leq i \leq n$*,* $[\![d_i]\!] \subseteq [\![d'_{\rho(i)}]\!]$*;*

– *for any* $1 \leq i \leq n$, *for any* $i < j \leq n$: $\rho(i) \leq \rho(j)$ *and there is no* ℓ *with* $\rho(i-1) \leq \ell < \rho(i)$ *(we let* $\rho(0) = 1$*) such that* $[\![d_i]\!] \subseteq [\![d'_\ell]\!]$.

Proof. We first prove that there exists a function $\rho : \{1, \ldots, n\} \mapsto \{1, \ldots, k\}$ such that $\forall 1 \leq i \leq n : [\![d_i]\!] \subseteq [\![d_{\rho(i)}]\!]$. As a matter of fact, this property is a direct consequence of Lemma 3. Indeed, let us suppose this is not the case, and there is thus $1 \leq i \leq n$ such that there is no $1 \leq j \leq k$ with $[\![d_i]\!] \subseteq [\![d'_j]\!]$. By Lemma 3, this implies that $[\![d_i]\!] \not\subseteq [\![p']\!]$. But since $[\![d_i]\!] \subseteq [\![p]\!]$, we get $[\![p]\!] \not\subseteq [\![p']\!]$, which is a contradiction.

Let us now prove the lemma, by contradiction again. Suppose that the lemma does not hold. According to the first part of the proof, this implies that the second point of the lemma is false, i.e. there exists a function $\rho : \{1, \ldots, n\} \mapsto \{1, \ldots, k\}$ such that $\forall 1 \leq i \leq n : [\![d_i]\!] \subseteq [\![d_{\rho(i)}]\!]$ and there exists $\ell \geq 1$ such that (i) $\forall 1 \leq i \leq \ell : \nexists \rho(i-1) \leq k < \rho(i) - 1 : [\![d_i]\!] \subseteq [\![d'_k]\!]$ and (ii) $\forall 1 \leq i < j \leq \ell : \rho(i) \leq \rho(j)$. Moreover, $\forall \rho(\ell) \leq i \leq k : [\![d_{\ell+1}]\!] \not\subseteq [\![d'_i]\!]$. Intuitively, this means that we can construct the function ρ for any value up to ℓ, but cannot extend it further.

We first construct a word w which is in $[\![d_1 \cdots d_\ell]\!]$ but not in $[\![d'_1 \cdots d'_{\rho(\ell)-1}]\!]$. The following holds for any $1 \leq i \leq \ell$: since, by hypothesis, there is no $\rho(i-1) \leq k < \rho(i)$ (suppose that $\rho(0) = 1$) such that $[\![d_i]\!] \subseteq [\![d'_k]\!]$, one can apply Lemma 3 and find a word w_i that is in $[\![d_i]\!]$ and thus in $[\![d'_{\rho(i)}]\!]$, but not in $[\![d'_{\rho(i-1)} \cdots d'_{\rho(i)-1}]\!]$. Let $w = w_1 \cdot w_2 \cdots w_\ell$. Clearly, $w \in [\![d_1 \cdots d_\ell]\!]$. However, w is not in $[\![d'_1 \cdots d'_{\rho(\ell)-1}]\!]$ by construction (we need $d'_{\rho(\ell)}$ to accept the suffix w_ℓ).

Let us now consider the fact that there is no m with $\rho(\ell) \leq m \leq k$ such that $[\![d_{\ell+1}]\!] \subseteq [\![d'_m]\!]$. By Lemma 3, this implies that $[\![d_{\ell+1}]\!] \not\subseteq [\![d'_{\rho(\ell)} \cdots d'_k]\!]$, and thus, there exists a word v that is in $[\![d_{\ell+1}]\!]$ but not in $[\![d'_{\rho(\ell)} \cdots d'_k]\!]$. But since $[\![d_{\ell+1}]\!] \subseteq [\![p]\!] \subseteq [\![p']\!]$, the word v must be accepted by p'. Hence, we can split v into v_1 and v_2 such that $v = v_1 \cdot v_2$, $v_2 \neq v$ is the longest suffix of v that can be accepted by $d'_{\rho(\ell)} \cdots d'_k$ and $v_1 \neq \varepsilon$ is accepted by $d'_1 \cdots d'_{\rho(\ell)-1}$. Since $v \in [\![d_{\ell+1}]\!]$, we have that $w \cdot v$ is in $[\![p]\!]$, and thus $w \cdot v \in [\![p']\!]$. But, by construction, v cannot be accepted by $d'_{\rho(\ell)} \cdots d'_k$ (only v_2 can), and thus $w \cdot v_1 \in [\![d'_1 \cdots d'_{\rho(\ell)-1}]\!]$. This implies that $w \in [\![d'_1 \cdots d'_{\rho(\ell)-1}]\!]$, since this set is $\preccurlyeq_w$-downward-closed. However, this contradicts the definition of w.

We conclude that there must exist $m \geq \rho(\ell)$ such that $[\![d_{\ell+1}]\!] \subseteq [\![d'_m]\!]$. Hence we have found m to be a suitable value for $\rho(\ell+1)$. Since this holds for any ℓ, we get the Lemma. □

We can now obtain the following result:

Proposition 10. *Given a natural number* i *and a product of* dc-re p *in normal form such that* $|p| \leq i + 1$*, for all product of* dc-re p' *such that (i)* $[\![p]\!] \subseteq [\![p']\!]$*; (ii)* $|p'| \leq i$ *and (iii)* $p' \notin \mathsf{Approx}(p, i)$*, we have that* $\exists p'' \in \mathsf{Approx}(p, i) : [\![p'']\!] \subseteq [\![p']\!]$.

Proof. We consider two cases. The first case is when $|p| \leq i$. In this case, the proposition is trivial since $\mathsf{Approx}(p, i) = \{p\}$. The latter case is when $|p| = i + 1$, which we prove by contradiction.

Let us suppose the proposition does not hold. Thus, there exists a product of dc-re p' that fits the three following conditions: (i) $[\![p]\!] \subseteq [\![p']\!]$, (ii) $|p'| \leq i$ and (ii) $p \notin \mathsf{Approx}(p,i)$, but $\neg\exists p'' \in \mathsf{Approx}(p,i) : [\![p'']\!] \subseteq [\![p']\!]$.

Let us suppose that $p = d_1 \cdot d_2 \cdots d_n$ and $p' = d'_1 \cdot d'_2 \cdots d'_k$. Since $[\![p]\!] \subseteq [\![p']\!]$, there exists, by Lemma 4, a function ρ that associates each d_i to a $d'_{\rho(i)}$. But since $k < n$, there must exist $1 \leq i < n$ such that $\rho(i) = \rho(i+1)$, that is, two successive dc-re d_i and d_{i+1} are associated to the same dc-re $d'_{\rho(i)}$. By construction, there exists in $L(p)$ a product $\mathcal{P}'$ of the form $d_1 \cdot \ldots \cdot d_{i-1} \cdot d \cdot d_{i+2} \cdot \ldots \cdot d_n$ with $[\![d_i]\!] \subseteq [\![d]\!]$ and $[\![d_{i+1}]\!] \subseteq [\![d]\!]$. Let $\mathcal{P}$ be the dc-re in normal form such that $[\![\mathcal{P}]\!] = [\![\mathcal{P}']\!]$ and obtained by applying Algorithm 3 to $\mathcal{P}'$. Let us assume that $\mathcal{P} = D_1 \cdot D_2 \cdots D_\ell$. Following Algorithm 3 and since the two products $d_1 \cdot \ldots \cdot d_{i-1}$ and $d_{i+2} \cdot \ldots \cdot d_n$ are both in normal form, there exists $1 \leq a \leq i+1$, such that the following holds for $i_1 = a-1$ and $i_2 = a+n-\ell+1$ (Fig. 1 illustrates the construction):

- $\forall 1 \leq j \leq i_1 : D_j = d_j$. The prefix of $\mathcal{P}$ of length i_1 is a prefix of p.
- $D_a = (c_1 + c_2 + \cdots + c_m)^*$ with $\{c_1, \ldots, c_m\} = \alpha(d_i) \cup \alpha(d_{i+1})$. The dc-re D_a over-approximates at least d_i and d_{i+1}.
- $\forall i_2 \leq j \leq n : D_{j-n+\ell} = d_j$. The suffix of $\mathcal{P}$ of length $n - i_2$ is a suffix of p.

Remark that it is possible that D_a over-approximates more than just d_i and d_{i+1}, because $\mathcal{P}$ is in normal form. Thus we have, by definition of the normal form, that $\{c_1, \ldots, c_m\} = \alpha(d_i) \cup \alpha(d_{i+1}) = \cup_{i_1 < j < i_2} \alpha(d_j)$.

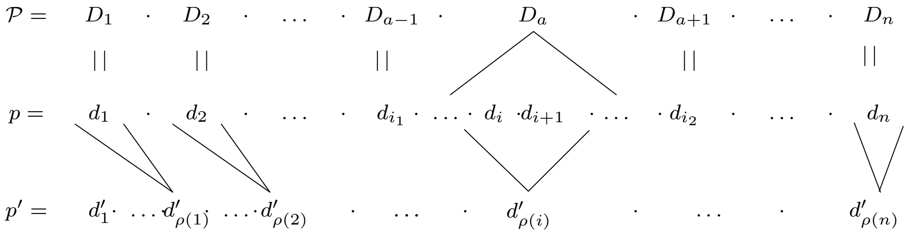

Fig. 1. The construction used in the proof of Proposition 10.

Let us apply Lemma 4 to $\mathcal{P}$ and construct a function π that associates the dc-re's of p to the dc-re of $\mathcal{P}$. Since both p and $\mathcal{P}$ are in normal form, it is clear that for any $1 \leq j \leq i_1 : \pi(j) = j$. Then, for any j such that $i_1 < j < i_2$, we have $\pi(j) = a$, because $\{c_1, \ldots, c_m\} = \cup_{i_1 < j < i_2} \alpha(d_j)$. Finally, for any $i_2 \leq j \leq n$, we have $\pi(j) = j - n + \ell$.

Now, let us consider a word w in $[\![\mathcal{P}]\!]$. We can decompose it into several subwords: $w = w_1 \cdot w_2 \cdots w_\ell$, such that $\forall 1 \leq j \leq \ell : w_j \in [\![D_j]\!]$. From this, we can deduce the following:

- for any $1 \leq j < a$, we know that $D_j = d_j$. Hence, $w_j \in [\![d_j]\!]$. But since $[\![d_j]\!] \subseteq [\![d'_{\rho(j)}]\!]$, this implies that for any $1 \leq j < a$, $w_j \in [\![d'_{\rho(j)}]\!]$.
- for any $a < j \leq \ell$, we know that $D_{j-n+\ell} = d_j$. By the same reasoning we deduce that for any $a < j \leq \ell$, $w_j \in [\![d'_{\rho(j+n-\ell)}]\!]$.

– we know that w_a is a word on $\alpha(d_i) \cup \alpha(d_{i+1})$. But since $\rho(i) = \rho(i+1)$, we know that $[\![d_i]\!] \subseteq [\![d'_{\rho(i)}]\!]$ and $[\![d_{i+1}]\!] \subseteq [\![d'_{\rho(i)}]\!]$, by definition of ρ. Thus $\alpha(d_i) \cup \alpha(d_{i+1}) \subseteq \alpha(d'_{\rho(i)})$ (by point 4 of Property 1), and we deduce that $w_a \in [\![d'_{\rho(i)}]\!]$.

Thus, we have shown that any word $w \in [\![\mathcal{P}]\!]$ is also in

$$[\![d'_{\rho(1)} \cdot d'_{\rho(2)} \cdots d'_{\rho(i_1)} \cdot d'_{\rho(i)} \cdot d'_{\rho(i_2)} \cdot d'_{\rho(i_2+1)} \cdots d'_{\rho(n)}]\!]$$

And since $[\![p']\!]$ is $\preccurlyeq_w$-downward-closed, $w \in [\![p']\!]$. Since this holds for any $w \in [\![\mathcal{P}]\!]$, we have $[\![\mathcal{P}]\!] \subseteq [\![p']\!]$, hence $[\![\mathcal{P}']\!] \subseteq [\![p']\!]$ since $[\![\mathcal{P}']\!] = [\![\mathcal{P}]\!]$. Thus, we have found a product of $\mathsf{dc\text{-}re}$ $\mathcal{P}'$ that is more precise than p' ($[\![\mathcal{P}']\!] \subseteq [\![p']\!]$) and that belongs to $L(p)$. By construction, there exists in $\mathsf{Approx}(p, i)$ a product $\mathcal{P}''$ such that $[\![\mathcal{P}'']\!] \subseteq [\![\mathcal{P}']\!]$, hence $[\![\mathcal{P}'']\!] \subseteq [\![p']\!]$. This is a contradiction with our hypothesis. Hence the lemma. □

C.4 Conclusion: Proposition 6

We can now obtain immediately Proposition 6:

Proposition 6. *Given a natural number i and a product of* dc-re *p such that $|p| \leq i + 1$, for all product of* dc-re *p' such that (i) $[\![p]\!] \subseteq [\![p']\!]$; (ii) $|p'| \leq i$ and (iii) $p' \notin \mathsf{Approx}(p, i)$, we have that: $\exists p'' \in \mathsf{Approx}(p, i) : [\![p'']\!] \subseteq [\![p']\!]$.*

Proof. The proof directly stems from Proposition 9 and Proposition 10. □