Stronger reduction criteria for Local First Search

Marcos E. Kurbán[1] Peter Niebert[2] Hongyang Qu[2] Walter Vogler[3]

[1] Formal Methods and Tools Group, University of Twente, EWI INF - PO Box 217, 7500 AE, Enschede, The Netherlands, mkurban@cs.utwente.nl

[2] Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence 39, rue Joliot-Curie / F-13453 Marseille Cedex 13, {niebert,hongyang}@cmi.univ-mrs.fr

[3] Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Walter.Vogler@Informatik.Uni-Augsburg.DE

Abstract. Local First Search (LFS) is a partial order technique for reducing the number of states to be explored when trying to decide reachability of a local (component) property in a parallel system; it is based on an analysis of the structure of the partial orders of executions in such systems. Intuitively, LFS is based on a criterion that allows to guide the search for such local properties by limiting the "concurrent progress" of components.

In this paper, we elaborate the analysis of the partial orders in question and obtain related but significantly stronger criteria for reductions, show their relation to the previously established criterion, and discuss the algorithmics of the proposed improvement. Our contribution is both fundamental in providing better insights into LFS and practical in providing an improvement of high potential.

1 Introduction

Partial order methods [15, 5, 8, 7, 12, 16, 9, 10, 13, 4, 6] exploit the structural property of independence that occurs naturally in asynchronous parallel systems. The basic observation exploited by partial order methods is the commutation of pairs of independent transitions which, by definition, lead to the same state independent of their order of execution. This structural information can be applied in order to remove redundant transitions or, if the property in question permits, even states, without changing the validity of the property. Independence is typically derived from distribution, i.e. transitions of distinct processes in a system *may commute* (unless they access shared variables or synchronize). This commutation of independent transitions gives rise to a notion of equivalent executions, and the equivalence classes are called *Mazurkiewicz traces.*

Among these methods, *Local First Search* (LFS) [11, 1] is specialized for the complete search for *local properties*, i.e. properties that can only be modified by dependent transitions. The definition and justification of LFS highly depend on the characterization of equivalent executions as labelled partial orders. In [11], it is shown that *prime traces*, i.e. partial orders with a single maximal element, suffice to search for local properties; in turn, to approximate all prime traces, it suffices to consider only traces (partial orders) with a logarithmic number of maximal elements (compared to the overall parallelism in the system); this number is called *LFS*-bound.

In [11], a first method for exploiting this criterion was given, which however did not guarantee that the number of states actually explored would be inferior to the global number of states. In [1] in contrast, the LFS-bound is combined with ideas from McMillan unfoldings [4] to obtain a breadth first search based algorithm that is complete and never explores the same state twice. For a number of benchmarks, it was observed that (asymptotically) LFS with the unfolding approach gives decent reductions where the stubborn set method [16], the ample set [12] and related methods fail.

In the current work, we revisit the LFS correctness theorem and derive a hierarchy of criteria, *peak rest compliance* (pr-compliance), *peak width sequence compliance* (pws-compliance), a *recursive LFS-bound* and finally the previously published *logarithmic LFS-bound*. These criteria characterize subsets of traces, ordered by inclusion: *pr-compliance* defines the smallest set of traces and the *logarithmic LFS-bound* the biggest. We prove that any prime trace can be reached through a sequence of prefixes such that each one is pr-compliant, and for that matter pws-compliant, and satisfies the LFS-bounds. On the whole, we thus obtain a modular proof of the original theorem and stronger reduction criteria. Efficient exploration algorithms have been implemented using the technique from [1].

The paper is structured as follows. Section 2 presents the necessary background on Marzurkiewicz trace theory. Section 3 explains the basic concepts of the LFS technique. Section 4 introduces *pr-compliance* based on a tree like recursive decomposition of traces, and a proof of the preservation of local properties is given. In Section 5, we derive a simplified version of pr-compliance, *pws-compliance*, which is computationally less expensive. In Section 6 in turn, we derive a *recursive LFS-bound* from pws-compliance and the previously published logarithmic bound from the recursive bound. In Section 7, we explain the complexity and steps needed to implement a pws-compliance procedure. In Section 8, we report experimental results obtained with our prototype implementation and conclude in Section 9.

2 Basic Concepts

The theory of Marzurkiewicz traces is built on the concept of a *concurrent alphabet*, which is a tuple (Σ, I) with Σ a finite set of *actions* and I an irreflexive symmetric binary relation on Σ. I is called the *independence relation* of the alphabet, and we will refer to $D = (\Sigma \times \Sigma) \backslash I$ as the *dependence relation* of such an alphabet. We will assume that (Σ, I) is fixed for this paper.

A transition system over Σ is a triple $T = (S, \rightarrow, s_0)$ with S a set of states, $s_0 \in S$ the initial state, and $\rightarrow \subseteq S \times \Sigma \times S$ a transition relation. For $(s, a, s') \in \rightarrow$

we also write $s \xrightarrow{a} s'$. We only consider deterministic transition systems, i.e. systems such that $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ implies $s_1 = s_2$. Moreover, we only consider systems that respect the independence relation in the following way: If $s \xrightarrow{a} s_1 \xrightarrow{b} s_2$ and $a\ I\ b$ then there exists s_1' with $s \xrightarrow{b} s_1' \xrightarrow{a} s_2$.

A word over Σ is a – possibly empty – finite sequence of symbols from Σ; the set of words is Σ^*, ranged over by u, v, w, etc.; the empty sequence will be denoted by ε. When used on words, $\preceq$ will denote the usual prefix ordering on words.

Let $\equiv_I$ be the least congruence on the monoid generated by Σ^* and concatenation such that $\forall a, b \in \Sigma : (a, b) \in I \Rightarrow ab \equiv_I ba$. The equivalence classes of $\equiv_I$ will be called *traces*, the equivalence class of u will be denoted by $[u]$ and the set of all traces by $[\Sigma^*]$. Since $\equiv_I$ is a congruence, concatenation carries over to traces: $[u][v] = [uv]$ is well-defined. Similarly, the *prefix relation* $\preceq$ carries over, i.e. $[u] \preceq [v]$ iff there exists $[w]$ with $[u][w] = [v]$.

For a transition system T, let $L(T) \subseteq \Sigma^*$ denote the words $u = a_1 \ldots a_n$ such that there exists a path $s_0 \xrightarrow{a_1} s_1 \ldots s_{n-1} \xrightarrow{a_n} s_n$ and let $\sigma(u) = s_n$ denote the state reached by the word. Obviously, if $u \in L(T)$ and $u \equiv_I u'$ then $u' \in L(T)$ and $\sigma(u) = \sigma(u')$. We therefore also write $\sigma([u]) := \sigma(u)$.

A *property* of a transition system T is a subset $P \subseteq S$. An action a is *visible for* P iff there exist $s_1 \in P$ and $s_2 \in S \setminus P$ such that $s_1 \xrightarrow{a} s_2$ or $s_2 \xrightarrow{a} s_1$ (i.e. a may "change" the validity of P). A property P is a *local property* iff, for all pairs of actions a and b both visible for P, we have $a\ D\ b$. Typically, a local property is a property of a single variable or a single process in a parallel product.

Local properties have an interesting link with traces, as has been observed in [11]: if some state satisfies local property P, then such a state can be reached by a trace which seen as a partial order has exactly one maximal element; cf. Section 3.

3 Local First Search

The aim of "Local First Search" is to optimize the search for local properties in transition systems. It is based on the following parameters of a concurrent alphabet.

Definition 1. *We say that* (Σ, I) *has* parallel degree m *if* m *is the maximal number of pairwise independent actions in* Σ*, i.e.*

$$m = \max\{|A| \mid A \subseteq \Sigma \text{ and } a, b \in A, a \neq b \Rightarrow aIb\}.$$

We say that (Σ, I) *has* communication degree cd *if* cd *is the maximal number of pairwise independent actions such that all of them depend on a common action, i.e.*

$$cd = \max\{|B| \mid B \subseteq \Sigma, \exists c \in \Sigma : (\forall b \in B : \neg cIb) \text{ and } (\forall b, b' \in B : b \neq b' \Rightarrow bIb')\}.$$

Intuitively, the parallel degree might correspond to the number of processes of a concurrent system, whereas the communication degree is related to synchronisation, e.g. systems based on binary channels have a communication degree 2.

The main idea of *Local First Search* (*LFS*) is better understood by viewing traces as partial orders. This is based on the well known one-to-one correspondence [3, Chapter 2] between traces and the class of finite Σ-labeled partial orders $(E, \leq, \lambda)$ such that

(1) For any $e, f \in E$ with $\lambda(e)\ D\ \lambda(f)$ we have $e \leq f$ or $f \leq e$.
(2) $\leq$ is equal to the transitive closure of $\leq \cap \{(e, f) \mid \lambda(e)\ D\ \lambda(f)\}$.

We will refer to such partial orders as a (Σ, I)-*lpo* or *lpo* for short. Any of them can be seen as an abstract representation of an execution. In this representation, two elements are unordered if and only if the actions labelling them could have occurred in any relative order (or in parallel). Correspondingly, any two such elements are labeled with independent actions.

By a *linearisation* we mean a word over E which contains each element of E once and where an element e occurs before f whenever $e < f$. We obtain a *labeled linearisation* from such a word, if we replace each element by its label. The relation between traces and lpo's is simply that the set of all labelled linearisations of an lpo is a trace and each trace, as described above, induces such a lpo.

If we have an lpo $(E, \leq, \lambda)$, we call subset F of E an *interval* iff for all $e, f \in F$ and $g \in E$ with $e \leq g \leq f$ also $g \in F$. We identify an interval F with the labeled partial order it induces by restricting $\leq$ and λ appropriately. Note that F is a (Σ, I)-lpo again. For a linearisation v of F we define $set(v)$ by $set(v) = F$.

The *downward closure* of $F \subseteq E$ is $\downarrow F = \{e \in E \mid \exists f \in F : e \leq f\}$, and we write $\downarrow f$ if $F = \{f\}$.

Element e of an lpo is an *immediate predecessor* of f and f an *immediate successor* of e iff $e < f$ and $\forall g :\ e \leq g \leq f \implies g = e \textit{ or } g = f$. We now define notions for $(E, \leq, \lambda)$ some of which correspond to the parallel and the communication degree of a concurrent alphabet.

Definition 2. *Let $(E, \leq, \lambda)$ be an lpo. An element $e \in E$ is* maximal *if there is no $f \in E$ such that $e < f$. We define* $\max(E)$ *as the set of maximal elements of E and call E* prime, *if* $\max(E)$ *has just one element.*

The width *of E (denoted by $width(E)$) is the maximal number of pairwise incomparable elements, i.e. $max\{|A| \mid A \subseteq E \land \forall e, f \in A : e \leq f \Rightarrow e = f\}$.*

The communication degree *of E is the maximal number of immediate predecessors of an element of E.*

The following proposition first relates these notions to the concurrent alphabet; the proof of this relation can be found in [11]. The last claim is easy to see.

Proposition 3. *Let $(E, \leq, \lambda)$ be an lpo. Then $|max(E)| \leq width(E)$ and $width(E)$ is at most the parallel degree of (Σ, I) and the communication degree of E is at most cd.*

For an interval $F \subseteq E$ we have $width(F) \leq width(E)$.

From the definition of local properties, one gets immediately the following: If a system can reach a state satisfying such a property, then a satisfying state can be reached with a prime trace corresponding to a prime lpo (the proof can be seen in [1]). The following fundamental result of LFS shows that one can construct all prime lpo's by restricting attention to lpo's with a bounded number of maximal elements; this implies that checking for satisfaction of a local property can be performed on a restricted state space.

Theorem 4 (LFS theorem [11]). *Let $(E, \leq, \lambda)$ be an lpo of width m with at most cd maximal elements. Then there exists a linearisation w of E such that, for every prefix v of w, $set(v)$ has at most 1 maximal element if $cd = 1$, and at most $\lfloor (cd-1)log_{cd}(m) + 1 \rfloor$ maximal elements if $cd > 1$.*

This theorem provides a filter for excluding traces in the search of states satisfying local properties. The best known way of exploiting this filter in a search procedure is given in [1]. For guidance purposes, it is also outlined below. Let us first consider a kind of "unfolding" of transition system T that respects traces, the trace system of T:

Definition 5 (Trace system). *Let $T = (S, \rightarrow, s_0)$ be a transition system respecting (Σ, I). Then the* trace system *of T is the transition system $\mathcal{TS}(T)$ whose states are the traces associated to words in $L(T)$, with the empty trace $[\varepsilon]$ as initial state and such that the transition relation is $\rightarrow = \{([u], a, [ua]) \mid ua \in L(T)\}$.*

Based on $\sigma([u])$ we can lift properties of T to properties of $\mathcal{TS}(T)$, and we can restrict the test for a local property to the search for a suitable prime trace.

The next notion, originating from McMillan prefixes [10], is needed to avoid the exploration of an infinite number of traces.

Definition 6 (Adequate order). *A partial order $\sqsubseteq$ on the whole set of traces is called* adequate *if*

(Ad_1) *it is well-founded;*
(Ad_2) *it refines the prefix order, i.e. $[u] \preceq [v]$ implies $[u] \sqsubseteq [v]$;*
(Ad_3) *it is a right congruence, i.e. $[u] \sqsubseteq [v]$ implies $[u.z] \sqsubseteq [v.z]$ for any $z \in \Sigma^*$.*

In practice, only adequate orders that refine the length order, i.e. $|u| < |v|$ implies $[u] \sqsubseteq [v]$, are used. Together with the filter of Theorem 4, adequate orders are used to *cut* the search in the state space, as is shown in the following algorithm that refines breadth first search.

Algorithm 1 guarantees that each state of the system is explored at most once (i.e. for at most one Mazurkiewicz trace leading to it), while preserving reachability of local properties. In practice, it considerably reduces the set of states explored. The correctness proof of the algorithm and a detailed explanation was presented in [1], and this proof relies on Theorem 4 as a module. What is important here is the consequence that the LFS-criterion in the algorithm, which is "bounding the set of maximal elements of trace $[ua]$ by $\lfloor (cd-1)log_{cd}(m) + 1 \rfloor$", can be replaced by other criteria similar to Theorem 4 without changing the correct functioning. The aim of this paper is to provide more restrictive criteria or tighter filters for lpo's that suffice to guarantee reachability of all prime lpo's but exclude a lot of traces or lpo's violating the criterion.

Algorithm 1 Computation of a finite locally complete subsystem

```
Table ← $\{(s_0, [\varepsilon])\}$, Next_Level ← $\{(s_0, [\varepsilon])\}$
while Next_Level $\neq \emptyset$ do
  Current_Level ← Next_Level; Next_Level ← $\emptyset$
  for all $(s, [u]) \in$ Current_Level, $a \in \Sigma$, $s' \in S$ such that $s \xrightarrow{a} s'$ do
    if $s' \in P$ then
      Return($ua$)
    else
      if $[ua]$ respects LFS-criterion then
        if $(s', [v]) \in$ Table then
          if $|ua| = |v|$ and $(s', [v]) \in$ Next_Level and $[ua] \sqsubset [v]$ then
            Table ← (Table $\setminus \{(s', [v])\}) \cup \{(s', [ua])\}$
            Next_Level ← (Next_Level $\setminus \{(s', [v])\}) \cup \{(s', [ua])\}$
          end if
        else
          Table ← Table $\cup \{(s', [ua])\}$
          Next_Level ← Next_Level $\cup \{(s', [ua])\}$
        end if
      end if
    end if
  end for
end while
Return unreachable
```

4 A new approach for tighter constraints

We will show in this section that, for building up prime lpo's, it is sufficient to consider peak-rest-compliant lpo's, which we define below. We will discuss in the succeeding sections in some detail how one can check this condition, and how one can weaken it to make the check more efficient; in the course of this discussion we will also prove that each peak-rest-compliant lpo obeys the bound on the number of maximal elements given in Theorem 4.

Definition 7. *Let $(E, \leq, \lambda)$ be an lpo. Let $e_1, \ldots, e_k \in \max(E)$ be different, $k \geq 0$, and $F = \max(E) \setminus \{e_1, \ldots, e_k\}$ with $F \neq \emptyset$. Define $E_i = (\downarrow e_i) \setminus \bigcup\{\downarrow f \mid e_i \neq f \in \max(E)\}$ for $i = 1, \ldots, k$ and $E_{k+1} = (\downarrow F) \setminus \bigcup\{\downarrow e_i \mid i = 1, \ldots, k\}$.*

Then $(E_1, \ldots, E_{k+1})$ is a peak-rest-decomposition *of E, and in case that F is a singleton a* peak-decomposition*. We call $E_1, \ldots, E_k$, and also E_{k+1} in the latter case,* peaks *of E.*

A peak is defined by a maximal element e; it consists of all elements that are below e, but not below any other maximal element. From this, it is clear that there exists a peak-decomposition, which is unique up to the ordering of its components; further, peaks are disjoint, elements of different peaks are unordered and a label appearing in one peak is independent of any label appearing in another peak – and this even holds for $E_1, \ldots, E_{k+1}$ in the general case. From this, we see that the sum over the *width*(E_i) is at most *width*(E).

Note that, in the general case, E_{k+1} could contain more elements than just the union of the peaks of the maximal elements in F, namely some elements that are below more than one maximal element of F.

Definition 8. *An lpo $(E, \leq, \lambda)$ is called* peak-rest-compliant *or* pr-compliant *for short if it has at most cd maximal elements or it has a peak-rest-decomposition $(E_1, \ldots, E_{k+1})$ with $1 \leq k < cd$ such that $width(E_{k+1}) \leq width(E_i)$ for $i = 1, \ldots, k$ and E_{k+1} is pr-compliant as well.*

Intuitively, a pr-compliant lpo has at most *cd* maximal elements or it is an initial part of an lpo with at most *cd* maximal elements and needed to build up the latter lpo; in the latter case, the idea is that k of the peaks of the latter lpo have already been built and that E_{k+1} will lead to the next peak. This idea will be formalized in the proof of our first main theorem, which we present now.

Theorem 9. *Let $(E, \leq, \lambda)$ be an lpo with at most cd maximal elements. Then there exists a linearisation w of E such that for each prefix v of w, $set(v)$ is pr-compliant.*

Proof. The proof will be by induction on the size of E, the case of $E = \emptyset$ being trivial. We assume that the claim has been shown for all smaller lpo's and make the following distinction of cases.

i) E has just one maximal element e. Then by Proposition 3, $E \setminus \{e\}$ has at most *cd* maximal elements, namely the immediate predecessors of e. Choose a suitable linearisation u of $E \setminus \{e\}$ by induction, and then we are done by setting $w = ue$.

ii) Let $\max(E) = \{e_1, \ldots, e_{k+1}\}$ with $1 \leq k < cd$. Let $(E_1, \ldots, E_{k+1})$ be the peak-decomposition of E ordered according to decreasing width. Choose linearisations u for $\downarrow (E_1 \cup \ldots \cup E_k)$ and u' for E_{k+1} by induction. Since these sets are a partition of E with no element of the latter below any element of the first, $w = uu'$ is a linearisation of E; we have to check all prefixes of w.

Let v be a prefix of w; if v is a prefix of u, we are done by induction, otherwise $v = uv'$ with v' a prefix of u'. Let $F = set(uv')$. Clearly, $e_1, \ldots, e_k \in F$ are still maximal; so let $\max(F) = \{e_1, \ldots, e_k, f_1, \ldots, f_l\}$, where $l \geq 1$ and $\max(set(v')) = \{f_1, \ldots, f_l\}$. Let $(F_1, \ldots, F_{k+1})$ be the peak-rest-decomposition of F induced by the maximal elements $e_1, \ldots, e_k$ and the set $\{f_1, \ldots, f_l\}$.

Since each $e \leq e_i$ for some $i \in \{1, \ldots, k\}$ occurs in u, we have $F_{k+1} \subseteq set(v') \subseteq E_{k+1}$, which implies $width(F_{k+1}) \leq width(E_{k+1})$ by Proposition 3. Vice versa, for each $f \in set(v') \subseteq E_{k+1}$, we cannot have $f \leq e_i$ for any $i \in \{1, \ldots, k\}$, hence we must have $F_{k+1} = set(v')$, which implies that F_{k+1} is pr-compliant by choice of u'.

For $i = 1, \ldots, k$, we have $E_i \subseteq F_i$: any $e \in E_i$ occurs in u, hence $e \in F$; we do not have $e \leq e_j$ for $j \neq i$ and we cannot have $e \leq f_j$ for some $j = 1, \ldots, l$ since $f_j \leq e_{k+1}$. Thus, we have $width(E_i) \leq width(F_i)$ by Proposition 3.

Hence, due to the chosen ordering of the E_i, we have $width(F_{k+1}) \leq width(E_{k+1}) \leq width(E_i) \leq width(F_i)$ for all $i = 1, \ldots, k$, and we are done. □

5 The peak-width-sequence criterion

In this section, we present a criterion that is slightly weaker than pr-conformance but avoids the recursive checks of the rest in a peak-rest-decomposition for pr-conformance. For this, we need the following notion for sequences of numbers, which is rather technical but easy to check.

Definition 10 (n-cumulative sequence). *For $n \geq 2$, a decreasing sequence of natural numbers $m_1 \geq m_2 \geq \ldots \geq m_l$ with $m_i \geq 1$ is called* n-cumulative*, if $l < n$ or there exists a j with $1 < j \leq n$ such that $m_{j-1} \geq \sum_{k=j}^{l} m_k$ and $m_j, \ldots, m_l$ is n-cumulative.*

Definition 11. *Let $m_1 \geq m_2 \geq \ldots \geq m_l$ be the widths of the peaks of an lpo $(E, \leq, \lambda)$; then the lpo is called* peak-width-sequence-compliant *or* pws-compliant *for short if this sequence is cd-cumulative.*

Together with Theorem 9, the following theorem demonstrates that we can restrict ourselves to pws-compliant lpo's if we want to build up all lpo's with at most cd maximal elements incrementally.

Theorem 12. *Each pr-compliant lpo is pws-compliant.*

Proof. Let $(E, \leq, \lambda)$ be a pr-compliant lpo and $(E_1, \ldots, E_{k+1})$ the respective peak-rest-decomposition with the peaks ordered by decreasing size. The proof is by induction on the size of E. Let $(F_{k+1}, \ldots, F_l)$ be the peak-decomposition of E_{k+1}, again with the peaks ordered by decreasing size. Since no element of E_{k+1} is below any element outside of E_{k+1}, $(E_1, \ldots, E_k, F_{k+1}, \ldots, F_l)$ is the peak-decomposition of E. Since $(F_{k+1}, \ldots, F_l)$ is cd-cumulative by induction and $k < cd$, it remains to check that $\mathit{width}(E_k) \geq \sum_{j=k+1}^{l} \mathit{width}(F_j)$. This follows from $\sum_{j=k+1}^{l} \mathit{width}(F_j) \leq \mathit{width}(E_{k+1})$, which is satisfied since the F_j are the peaks of E_{k+1}, and $\mathit{width}(E_{k+1}) \leq \mathit{width}(E_k)$ according to Definition 8. $\square$

The difference between pws-compliance and pr-compliance is that the width of the rest in a peak-rest-decomposition might be larger than the sum of the peak widths for the peaks in this rest; in such case, the lpo could be pws-compliant without being pr-compliant. Hence, pr-conformance may give a stronger restriction of the visited part of the state space. But algorithmically, checking for pr-conformance requires to identify the rest of a suitable peak-rest-decomposition, and this would presumably involve to determine all peaks and their widths first. Then one has additionally to compute the width of the rest, where the latter might be even larger than the union of the respective peaks. For the recursive checks for pr-conformance, the peaks of the rest and their widths are already given, but the problems with finding and checking a suitable peak-rest-decomposition of the rest occur repeatedly.

To test pws-compliance, we have to construct the peaks of the given lpo and to determine their widths $m_1 \geq m_2 \geq \ldots \geq m_l$; this sequence is cd-cumulative if – viewed from the end – there are never $cd - 1$ indices in a row where the cumulative sum is larger than the next (smaller) number. It is easy to check this in linear time by one scan from right to left, building the cumulative sums on the way.

6 Deriving the LFS bounds

In this section, we derive the classical LFS-bound as presented in [11], as well as some slight improvements, from the pws-criterion. Moreover, this is the first published proof of the LFS-bound for the general case (communication degree not limited to 2).

We begin by introducing a recursive formula for the bound that gives a relation of length and sum of n-cumulative sequences. Then, we derive the previously published logarithmic bound for the recursive formula.

Definition 13 (recursive bound). *For $n \geq 2$ and $m \geq 1$ let $L(n,m)$ be inductively defined by*

$$\begin{array}{lll} L(n,m) = & m & \text{for } m \leq n \\ L(n,m) = & n-1+L(n,\lfloor \frac{m}{n} \rfloor) & \text{for } m > n \end{array}$$

Lemma 14.
For $1 \leq k \leq n$ and $m \geq 1$ we have $k + L(n, \lfloor \frac{m}{k} \rfloor) \leq n + L(n, \lfloor \frac{m}{n} \rfloor)$.

Proof. By induction on m. We assume that the statement is already proven for all $m' < m$ with $m' \geq 1$. For an easier case analysis observe that $L(n,m) = n-1+L(n,\lfloor \frac{m}{n} \rfloor)$ also for $m = n$. This allows us to distinguish the following three cases: (a) $\lfloor \frac{m}{n} \rfloor \leq \lfloor \frac{m}{k} \rfloor < n$, (b) $\lfloor \frac{m}{n} \rfloor < n \leq \lfloor \frac{m}{k} \rfloor$ and (c) $n \leq \lfloor \frac{m}{n} \rfloor \leq \lfloor \frac{m}{k} \rfloor$.

For (a), we have to show that $k + \lfloor \frac{m}{k} \rfloor \leq n + \lfloor \frac{m}{n} \rfloor$. Due to properties of $\lfloor . \rfloor$ this follows from $k + \frac{m}{k} \leq n + \frac{m}{n}$ or equivalently $nk + n\frac{m}{k} \leq n^2 + k\frac{m}{k}$ or $\frac{m}{k}(n-k) \leq n(n-k)$. This follows, since $k \leq n$ and $\frac{m}{k} \leq n$ by $\lfloor \frac{m}{k} \rfloor < n$.

For (b), we have that $\lfloor \frac{\lfloor \frac{m}{k} \rfloor}{n} \rfloor = \lfloor \frac{m}{kn} \rfloor < n \leq \lfloor \frac{m}{k} \rfloor$. Therefore, we have to show that $k+n-1+\lfloor \frac{m}{kn} \rfloor \leq n + \lfloor \frac{m}{n} \rfloor$ which follows from $k - 1 + \frac{m}{kn} \leq \frac{m}{n}$. This is equivalent to $n(k-1) + \frac{m}{k} \leq k\frac{m}{k}$ or $n(k-1) \leq \frac{m}{k}(k-1)$. The latter follows since $k \geq 1$ and $n \leq \frac{m}{k}$ by the assumption $n \leq \lfloor \frac{m}{k} \rfloor$.

For (c), we have to show that $k+n-1+L(n, \lfloor \frac{\lfloor \frac{m}{k} \rfloor}{n} \rfloor) \leq n+n-1+L(n, \lfloor \frac{\lfloor \frac{m}{n} \rfloor}{n} \rfloor)$. Since $\lfloor \frac{\lfloor \frac{m}{k} \rfloor}{n} \rfloor = \lfloor \frac{m}{kn} \rfloor = \lfloor \frac{\lfloor \frac{m}{n} \rfloor}{k} \rfloor$, this follows immediately from induction for $m' = \lfloor \frac{m}{n} \rfloor < m$. $\square$

Proposition 15. *Let $m_1 \geq m_2 \geq \ldots \geq m_l$ be n-cumulative ($n \geq 2$) and $m = \sum_{i=1}^{l} m_i$. Then for the length l of the sequence we have $l \leq L(n,m)$.*

Proof. The proof is by induction on l.

For the case $m \leq n$ (in accordance with the first defining equation of L), we use $l \leq m$ (since $m_i \geq 1$ for each of the l summands m_i) and $m = L(m,n)$.

Now, for $m > n$, let j be the first position in the sequence, such that $m_{j-1} \geq \sum_{k=j}^{l} m_k =: m'$. Then $j \leq n$. Since $m_j \geq \ldots \geq m_l$ (as a suffix of an n-cumulative sequence) is itself n-cumulative of length $l' = l-(j-1)$, it holds by induction that $l' \leq L(n, m')$ and consequently $l \leq j-1+L(n, m')$. Since $m' \leq m_{j-1} \leq \ldots \leq m_1$, we have that $m' \leq \lfloor \frac{m}{j} \rfloor$. By Lemma 14 and the monotonicity of L in the second argument, we finally obtain $l \leq n-1+L(n, \lfloor \frac{m}{n} \rfloor) = L(n, m)$ as desired. □

Corollary 16. *Let $(E, \leq, \lambda)$ be an lpo of width m with at most $cd \geq 2$ maximal elements. Then there exists an linearisation w of E such that, for every prefix v of w, $set(v)$ has at most $L(cd, m)$ maximal elements.*

Proof. We choose w as linearisation according to Theorem 9. Let v be a prefix of w; then $set(v)$ is by choice pr-compliant and according to Theorem 12 also pws-compliant, and $width(set(v)) \leq width(E) = m$ (Proposition 3). The peaks of $set(v)$ are mutually independent and hence the sum of their widths is bounded by m (antichains of the peaks freely combine to antichains of $set(v)$). Hence, the number of peaks of $set(v)$ is bounded by $L(cd, m)$ according to Proposition 15 and monotonicity of $L(n, m)$ in m. □

The recursive formula L is an improvement over the originally published bound of Theorem 4, as shown by the following statement.

Proposition 17.
For $2 \leq n$ and $1 \leq m$ we have $L(n, m) \leq \lfloor (n-1) \log_n m \rfloor + 1$.

Proof. By induction on m. For $m \leq n$ by definition $L(n, m) = m$. Observe that $\log_n m$ is concave in m and has for $m = 1$ and $m = n$ the same value as $\frac{m-1}{n-1}$, thus $\frac{m-1}{n-1} \leq \log_n m$ for $1 \leq m \leq n$. Hence, $m \leq (n-1)\log_n m + 1$ which implies $m \leq \lfloor (n-1) \log_n m \rfloor + 1$ and we are done.

Now for $m > n$ we get $L(n, m) = n - 1 + L(n, \lfloor \frac{m}{n} \rfloor)$. By induction, we obtain that $L(n, m) \leq (n-1) + \lfloor (n-1)(log_n \lfloor \frac{m}{n} \rfloor) \rfloor + 1 = \lfloor (n-1)(1 + log_n \lfloor \frac{m}{n} \rfloor) \rfloor + 1 = \lfloor (n-1)(log_n n \lfloor \frac{m}{n} \rfloor) \rfloor + 1 \leq \lfloor (n-1)(log_n m) \rfloor + 1$, as desired. □

Now, we can see how the original Theorem 4 is the end of a chain of reasoning in our present paper: We simply have to combine Corollary 16 with Proposition 17.

Concluding, we have seen how pr-compliance implies pws-compliance and pws-compliance induces a new recursive bound, which itself implies the original logarithmic bound.

7 Complexity and algorithmics of the new criteria

Pws-compliance may filter out significantly more traces than the LFS-bound, which means less states need to be stored. But its overhead, i.e. the cost of testing pws-compliance of each explored trace, has an impact on execution time.

The check for pws-compliance of a trace can be decomposed as follows:

- At first, the peaks need to be extracted as subtraces of the trace. Due to the characterization of dependency graphs at the beginning of Section 3, this task can be carried out in time at most $O(m^2)$, where m is the length of the trace. Depending on the representation of dependency, this can be improved to $O(m \cdot \log m)$.
- Then, for the partial order of each peak, its width has to be computed. Computing the width of a partial order is a problem that is known to be equivalent to that of finding a maximal matching in a bipartite graph [14]. The matching problem can then be solved with Hopcroft and Karp's algorithm [2] in $O(n^{\frac{5}{2}})$, where n is the size of the bipartite graph. This is in turn, twice the size of the peak.
- At last, the resulting widths have to be ordered and the test for an n-decreasing sequence has to be done. This is largely subsumed in the complexity of the previous task.

On the whole, the worst-case complexity of the pws-compliance test is thus subsumed by the $O(n^{\frac{5}{2}})$ of the matching algorithm. First experiments indicate however, that often the peaks are small compared to the length of the trace.

Exploring a transition system that has very long paths to some state is thus costly, the time complexity can only be limited by $O(n^{\frac{7}{2}})$ where n is the number of actually visited states (after reduction). Our filtering may lead to an exponential reduction of the size of the state space; in such cases, this at first sight high complexity pays well off with a significant speed up and even with dramatic space savings, as the experiments of the next section suggest.

It is conceivable that the pws-compliance test can be accelerated by a number of improvements:

- avoiding the execution of the test when it is not necessary (e.g. when the number of peaks is lower than the communication degree, ...);
- a caching technique that allows to avoid executing the test in certain cases;
- an incremental way of computing pws-compliance using some memory.

8 First experimental results

We have conducted first experiments with a new prototype using Algorithm 1. In one version, we use the bound of Proposition 15, which is slightly tighter than the previously published bound of Theorem 4. In another version, we use pws-compliance according to Theorem 12. In each parametric experiment, we compare the number of states, the memory consumption and the running time for exhaustive exploration on a machine with 2GB memory. We only give data for cases not running out of memory, which sometimes means that we can treat bigger problem instances with one or the other reduction method.

The first experiment concerns a version of Dijkstra's dining philosophers with 5 states, which each choose nondeterministically, which fork they pick up first (Figure 1 at left). This is an example, where typical partial order techniques like stubborn sets [16] or ample sets [12] fail to achieve significant reduction; in

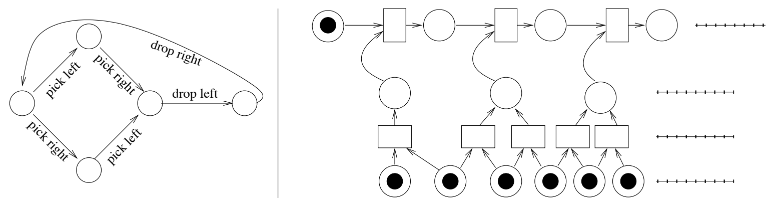

Fig. 1. An instance of the philosophers / the "comb" process as Petri net

fact, experimentally, Spin's partial order reduction does not remove any states. It turns out, that already the bound-based algorithm obtains a decent reduction and is a bit faster than the pws-compliant based reduction, which only has a slight advantage in states and memory on this series. Observe the sub-exponential growth with reduction, whereas growth is cleanly exponential without reduction.

N	No reduction			LFS bound			PWS compliant			SPIN PO red		
	states	time (s)	memory (m)	states	time (s)	memory (m)	states	time (s)	memory (m)	states	time (s)	memory (m)
2	13	0.01	4.1	13	0.01	4.6	13	0.01	4.8	13	0.00	2.6
3	51	0.01	4.1	49	0.01	4.7	49	0.01	4.8	51	0.00	2.6
4	193	0.01	4.1	191	0.01	4.7	147	0.01	4.9	193	0.01	2.6
5	723	0.01	4.1	651	0.01	4.7	441	0.01	4.9	723	0.02	2.6
6	2701	0.02	4.4	1937	0.02	4.8	1552	0.02	5.0	2701	0.02	2.7
7	10083	0.05	5.4	5041	0.05	5.4	4694	0.06	5.4	10083	0.09	3.1
8	37633	0.22	9.3	25939	0.25	8.8	11825	0.27	5.5	37633	0.35	7.9
9	140451	1.02	25.6	70225	0.76	17.3	26269	0.78	9.3	140451	1.59	43.8
10	524173	4.52	91.6	173031	2.13	38.1	63561	2.34	16.5	524173	7.03	74.1
11	1956243	21.06	357.5	392701	5.28	84.9	139788	5.92	32.6	1956243	31.03	325.1
12	7300801	106.49	1422.5	830415	12.33	183.5	340179	15.86	79.5	7300801	127.40	1030.1
13	—	—	—	1652587	26.99	378.3	808390	39.75	191.8	—	—	—
14	—	—	—	3121147	56.44	743.9	1817375	97.03	441.5	—	—	—
15	—	—	—	5633381	111.55	1399.0	3815044	213.60	948.4	—	—	—
16	—	—	—	—	—	—	7492734	966.79	1911.7	—	—	—

Fig. 2. Results of the philosophers example

The second example "comb" (indicated as Petri net in Figure 1 at right) is an artificial series of "best-case reductions" for the pws-criterion. While exponential without reduction, and while Spin's partial order reduction does fail to eliminate any states, it is clearly sub-exponential using the LFS-bound and it is not difficult to understand that it has cubic growth under pws-compliant reduction. Observe the jumps in state memory and time for the LFS-bound reduction: They occur when the LFS-bound increases.

« Comb »

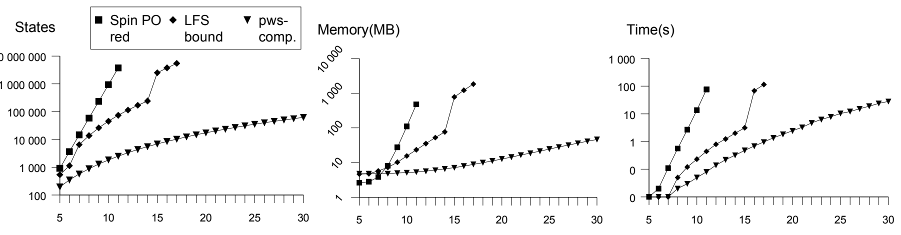

Fig. 3. Results of the comb example

The third example is a series of instances of an asynchronous version[1] of the Sieve of Erastosthenes, where N is the number of iterations of the sieve.

This is an example where the partial order methods like the ample set method are good at. Indeed, while LFS with bound is faster and less memory-consuming in our prototype than Spin without reduction, Spin with reduction is the fastest. However, pws-compliance does give a significantly better reduction with respect to memory, although it is the slowest. Here, the complexity of the pws-criterion is the bottle neck and almost all processor time is used for evaluating it.

N	No reduction			LFS bound			PWS compliant			SPIN PO red		
	states	time (s)	memory (m)	states	time (s)	memory (m)	states	time (s)	memory (m)	states	time (s)	memory (m)
1	340	0.01	4.1	198	0.01	4.7	198	0.01	4.8	188	0.00	2.3
2	1912	0.01	4.3	1456	0.01	4.8	910	0.02	4.9	630	0.00	2.3
3	8632	0.04	5.4	4560	0.05	5.5	2874	0.08	5.2	2134	0.00	2.5
4	63984	0.39	15.5	18252	0.19	8.5	14392	0.88	7.5	10599	0.04	3.6
5	178432	1.30	40.7	35072	0.46	12.9	26644	2.35	10.5	25048	0.13	6.2
6	1097296	10.31	259.2	361736	9.10	99.8	63212	16.95	20.3	109880	0.70	21.7
7	2978208	34.12	772.2	707120	19.75	206.7	112964	49.07	35.2	1076639	9.32	243.5
8	—	—	—	2072162	75.71	650.1	304386	229.57	95.0	4311167	86.52	1801.0
9	—	—	—	—	—	—	1158208	1380.12	395.1	—	—	—

Fig. 4. Results of the sieve example

But the running time of the pws-compliance test should be read with care: This is a first implementation without any optimization. In fact, we are positively surprised that the pws-compliance test *is practical*!

9 Conclusions and future work

In this paper, we report on an improvement concerning both the theoretical basis and practice of Local First Search, a recent partial order reduction technique.

[1] Our prototype does not allow rendezvous yet, so to compare, we had to modify this example for Promela/Spin, see appendix.

The theory not only gives a better insight into previously published results, but also yields a stronger reduction method than previously known, using *peak-width-sequence* compliance. As regards the LFS-bound, pws-compliance can be used in a chain of reasoning that derives it.

We have also built an ambitious prototype implementation of the algorithm that starts to yield results. The first observations reported here are partially ambiguous and more experimental evidence would be desirable, but they clearly show the potential of the method. Problems with a high degree of independence are a good measure to see this. In treating them, established reduction methods fail to deliver the degree of space economy partial order semantics can yield. In this cases our method performs at its best, yielding robust space savings.

The sieve example shows that there is a space-time tradeoff between Spin's reductions and those of our prototype. Our method seems to be very strong in space savings, but the time complexity should be improved. This is a priority topic to be addressed: for instance, if the pws-sequence could be computed in a more incremental fashion by investing some memory, the algorithm could become much faster.

Apart from this topic and the need to stabilize and improve the prototype, we will study the performance of the algorithm on realistic case studies. We believe that in particular in the field of AI planning, the dependency analysis proposed by our algorithm could turn out to be very promising.

References

1. Bornot, S., Morin, R., Niebert, P., and Zennou, S. Black box unfolding with local first search. *LNCS*, 2280 (2002), 386–400.
2. Cormen, T., Leiserson, C., and Rivest, R. *Introduction to Algorithms*. MIT Press, 1990.
3. Diekert, V., and Rozemberg, G., Eds. *The Book of Traces*. World Scientific Publishing Co. Pte. Ltd., 1995.
4. Esparza, J., Romer, S., and Vogler, W. An improvement of McMillan's unfolding algorithm. In *Tools and Algorithms for Construction and Analysis of Systems* (1996), pp. 87–106.
5. Godefroid, P. Using partial orders to improve automatic verification methods. In *CAV '90: Proceedings of the 2nd International Workshop on Computer Aided Verification* (London, UK, 1991), Springer-Verlag, pp. 176–185.
6. Godefroid, P., Peled, D., and Staskauskas, M. Using partial-order methods in the formal validation of industrial concurrent programs. *IEEE Trans. Softw. Eng. 22*, 7 (1996), 496–507.
7. Godefroid, P., and Pirottin, D. Refining dependencies improves partial-order verification methods (extended abstract). In *Computer Aided Verification: Proc. of the 5th International Conference CAV'93*, C. Courcoubetis, Ed. Springer, Berlin, Heidelberg, 1993, pp. 438–449.
8. Godefroid, P., and Wolper, P. A partial approach to model checking. In *Logic in Computer Science* (1991), pp. 406–415.
9. Holzmann, G., and Peled, D. Partial order reduction of the state space. In *First SPIN Workshop* (Montrèal, Quebec, 1995).

10. McMillan, K. L. A technique of state space search based on unfolding. *Form. Methods Syst. Des. 6*, 1 (1995), 45–65.
11. Niebert, P., Huhn, M., Zennou, S., and Lugiez, D. Local first search: a new paradigm in partial order reductions. *LNCS*, 2154 (July 2001), 396–410.
12. Peled, D. All from one, one for all: on model checking using representatives. In *CAV* (1993), pp. 409–423.
13. Penczek, W., and Kuiper, R. Traces and logic. In Diekert and Rozemberg [3].
14. Reichmeider, P. F. *The Equivalence of Some Combinatorial Matching Theorems.* Polygonal Pub House, 1985.
15. Valmari, A. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets* (1989), pp. 491–515.
16. Valmari, A. On-the-fly verification with stubborn sets. In *CAV* (1993), pp. 397–408.

Appendix: Sieve of Erastosthenes in Promela for $N = 2$

```
#define len 2
#define lenplusone 3
#define MAX 6

int r[lenplusone] = 0;
int w[lenplusone] = 1;
int q[lenplusone] = 0;
int count;

active proctype left () {
    count=2;
    do
    :: count <= MAX ->
        atomic{ w[0]==1
                -> w[0]=0;}
        q[0]=count; r[0]=1;
        count++
    :: count>MAX -> break;
    od
}

active proctype right () {
    int next;
    do
    :: true ->
         atomic{r[len]==1
              -> r[len]=0;}
        next=q[len];
        w[len]=1;
    od
}
```

```
active proctype middle1 () {
    int myval, nextval;
    atomic{r[0]==1 -> r[0]=0;}
    myval=q[0]; w[0]=1;
    do
    ::  true -> atomic{r[0]==1
                           -> r[0]=0;}
        nextval=q[0]; w[0]=1;
        if
        :: (nextval % myval) != 0 ->
            atomic{w[1]==1 -> w[1]=0;}
            q[1]=nextval; r[1]=1;
        :: else -> nextval=nextval;
        fi
    od
}
active proctype middle2 () {
    int myval, nextval;
    atomic{r[1]==1 -> r[1]=0;}
    myval=q[1]; w[1]=1;
    do
    ::  true -> atomic{ r[1]==1
                            -> r[1] = 0;}
        nextval=q[1]; w[1]=1;
        if
        :: (nextval % myval) != 0 ->
            atomic{ w[2]==1 -> w[2] = 0;}
            q[2]=nextval; r[2]=1;
        :: else -> nextval=nextval;
        fi
    od
}
```