Dynamic Bounds and Transition Merging for Local First Search

D. Lugiez,P. Niebert, S. Zennou

Laboratoire d'Informatique Fondamentale (LIF) de Marseille
Université de Provence – CMI
39, rue Joliot-Curie / F-13453 Marseille Cedex 13
[lugiez,niebert,zennou]@cmi.univ-mrs.fr

Abstract. Local First Search (LFS) was recently introduced as a new partial order reduction approach. It is based on the observation that local properties can be found searching on paths with a low degree (LFS-number) of concurrent progress. It has shown its potential to find sequences to local states fast. Moreover, the method is very simple and thus allows easy integration into tools like Spin.
In this paper, we explore several improvements of LFS: On the one hand, we propose a replacement of the static bound on LFS-numbers by a dynamic criterion to detect exhaustion of reachable local properties faster. On the other hand, we explore the possibilities of combining LFS with other partial order reduction methods. It turns out that a combination with *transition merging* is possible and rewarding, whereas a combination with the sleep set method is not possible.

1 Introduction

Local First Search (LFS) was recently introduced by the authors as a heuristic method to reduce the state explosion problem in the search (verification) of *local properties* of distributed systems. It is based on the (theoretically founded) observation that in systems with local (binary, ternary, ...) communication the number of components progressing in true parallelism (called *LFS-number*) can be bounded to a logarithmic fraction of the number of processes when searching for path leading to local goals (local states of one or two processes independent of the states of the other processes). This is not the case if an arbitrary number of processes may synchronize in a single transition. As a heuristic search strategy, *local first* means to explore paths with a low LFS-number first.

The potential interest of LFS to the Spin community results from the fact that the method is based on state exploration while providing reductions related to the Unfolding approach [McM92,ERV96]: LFS may be much easier to integrate to a verification tool like Spin than Unfoldings. On the other hand, the reductions achievable by LFS are rather complementary to those partial order reductions already present in Spin.

However, the basic method presented in [NHZL01] still needs refinement to become a practical tool. The present work establishes important steps in this direction.

In [NHZL01], we observed that LFS does indeed expose paths to local properties rapidly if they exist, and the LFS-numbers found for these cases stayed far below the theoretically predicted bound. However, exploration up to this bound was needed to show the absence of such a path, and this turned out to yield state spaces exceeding those without reduction. The reason is that the LFS procedure presented requires some overhead information and that this overhead need not always be compensated by the reductions. In [BMNZ02], we combined LFS with ideas from unfoldings (adequate orders) in order to eliminate the double exploration of states due to the overhead information. With this refinement, one is guaranteed to obtain a state spaces inferior to the state space without reduction. On the other hand, *unfolding LFS* does not (as yet) seem to allow an efficient incremental exploration with growing LFS number: Its use would require either to unfold up to the theoretical bound or to recalculate unfoldings for some or all of the lower bounds.

At least for the non-unfolding version of LFS, we propose a solution for dynamically detecting that the exploration of local properties *has already been* exhaustive at some LFS-level. For the case of binary communication, the *dynamic LFS-bound* can be stated as follows: If for some LFS-number $n+1$ no local states were detected that were not already detected at LFS-number n then no new local states will be found at any higher level. In practice, this means that we can stop exploration at *just one level above* the actually highest LFS-number of local properties of the system.

As a second contribution of this paper, we have studied the potential of combining LFS with other partial order reduction methods.

On the positive side, we show the strong compatibility of LFS with a technique known as *transition merging* (see e.g. [DR99]): Transition merging preserves the parameters of a system essential to LFS and in particular excludes the possibility of an increased LFS-number of a local property. Moreover, it seems that the reductions achieved by LFS and transition merging are complementary. The experimental results indicate that this combination has a great potential.

On the other hand, we answer an imminent question negatively: LFS is not compatible with sleep-set reductions [GW93]. While it could be argued that the counterexample we give only shows that a naïve combination does not work, we believe that the two approaches are strictly incompatible.

Finally, we did some experimentation with a prototype implementation written in Caml. The rapid prototyping approach that we currently follow has allowed us to explore a number of combinations of methods in a short time, but of course it does not scale up as would an efficient implementation in an imperative language. Indeed, LFS only starts to unfold its potential (reductions for systems with *many* processes) where our implementation already suffers resource problems. We believe that nevertheless the potential of the method is visible through these results.

For the sake of a self contained presentation, we restate the formal development as given in [BMNZ02]. The organization of the paper is as follows: In Section 2, we develop the formal Framework of transition systems with an inde-

pendence relation. In Section 3, we restate the formal particularities of LFS and the theorem on the static LFS-bound taken from [NHZL01]. In Section 4, we indicate the functioning of the two LFS-methods of [NHZL01] and [BMNZ02], but for a complete coverage we refer the reader to those articles. Section 5 contains the main technical contributions of the present article: The dynamic bound for LFS and the combination of LFS with transition merging as well as the counterexample against the combination with sleep sets. In Section 6, we explain our current set of academic experiments.

2 Formal Framework

In this section, we introduce the formal framework common to all "partial order methods" (for a thorough treatment, see [DR95]): The notion of an independence relation with associated commutation properties on transition systems, the equivalence relation on transition sequences induced by an independence relation, and the relation to partial order representations of executions.

2.1 Independence and asynchronous transition systems

In this paper, we fix a finite alphabet Σ together with an *independence relation* $\| \subseteq \Sigma \times \Sigma$ which is symmetric and irreflexive. Intuitively, this relation represents concurrency between actions occurring on distinct processes in a distributed system.

Example 1. As an abstract running example consider the following system of processes (in Promela):

```
/* A simple example with communication via shared variables */
int X; int Y; int Z;
proctype A() { /* process A */
  /*action a*/ Z=1;
  /*action f*/ atomic{Y==2 -> Z=2;}}
proctype B() { /* process B */
  /*action b*/ Y=1;
  /*action e*/ atomic{X==2 ->Y=2;}}
proctype C() { /* process C */
  /*action c*/ X=1;
  /*action d*/ X=2;}
init {X=0; Y=0; Z=0; run A(); run B(); run C();}
```

It is a system consisting of three bounded domain variables (`X`, `Y`, `Z`, all initialized with 0) and three parallel processes (`A`, `B`, `C`).

We will consider the transitions `a` and `b` independent, because they belong to different threads and mention disjoint sets of variables. In contrast, the transitions `e` and `f` are considered dependent because the one writes a variable appearing in the condition of the other.

On the whole, the independence relation of the example is $\{(a,b),(b,a),(a,e),(e,a),(a,c),(c,a),(a,d),(d,a),(b,c),(c,b),(b,d),(d,b)\}$.

Definition 2. We say that $(\Sigma, \|)$ has *parallel degree* m if m is the maximal number of pairwise independent actions in Σ,
(i.e. $m = max\{|A| \mid A \subseteq \Sigma \text{ and } a,b \in A, a \neq b \implies a \parallel b\}$)

The parallel degree of the independence relation of Example 1 is 3.

Next, we look at the interleaving semantics of parallel systems, transition systems.

A transition system is a triple $T = (S, \rightarrow, s_0)$ with S a finite set of states, $s_0 \in S$ the initial state, and $\rightarrow \subseteq S \times \Sigma \times S$ a transition relation. In this paper, we require that transition systems are *deterministic*. Any non-deterministic system can be transformed into a deterministic one by renaming actions without modifying the properties we are interested in. By the *language* $L(T) \subseteq \Sigma^*$ we denote the set of all sequences corresponding to paths in T in the usual sense.

The transition system corresponding to the process system of Example 1 is depicted in Figure 1. The states are named after the values of the variables, vectors (X, Y, Z). The progress of the three processes is indicated by three dimensions.

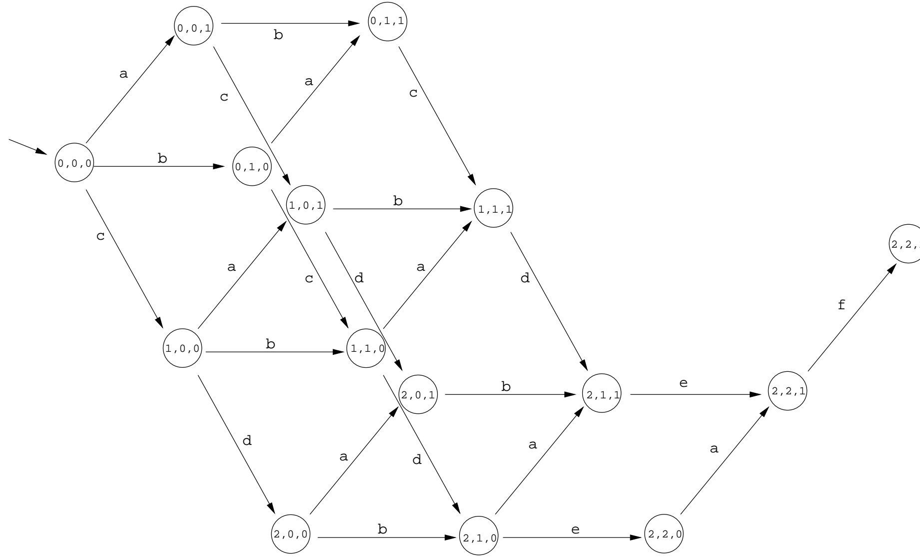


Fig. 1. The transition system of the process system of Example 1.

On the level of transition systems, the independence relation results in the following structural properties for independent actions a, b:

ID: $s \xrightarrow{a} s_1 \xrightarrow{b} s_2$ implies $s \xrightarrow{b} s'_1 \xrightarrow{a} s_2$ for some state s'_1 *[Independent Diamond]*
FD: $s \xrightarrow{a} s_1$ and $s \xrightarrow{b} s'_1$ implies $s_1 \xrightarrow{b} s_2$ for some state s_2 *[Forward Diamond]*

A transition system respecting the axioms ID and FD is called an *asynchronous transition system.*

The example in Figure 1 clearly respects the axioms ID and FD. Note that there is indeed dependence between the process due to the shared variables.

From now on, we consider only an asynchronous transition system $T = (S, \rightarrow, s_0)$ w.r.t. an independence alphabet $(\Sigma, \|)$.

2.2 Mazurkiewicz traces and partial orders

The axiom FD states that it is possible to exchange consecutive occurrences of independent transitions on paths in an asynchronous transition system. The axiom ID in exchange says that independent transitions cannot be in conflict (taking one will not disable the other). This gives rise to the idea of equivalence classes of executions, where the equivalence is *generated* by (iteratively) exchanging independent transitions.

Formally, the *Mazurkiewicz trace equivalence* associated to the independence alphabet $(\Sigma, \|)$ is the least congruence $\sim$ over $\Sigma^\star$ such that $ab \sim ba$ for any pair of independent actions $a \| b$. A *trace* $[u]$ is the equivalence class of a word $u \in \Sigma^\star$.

Since all paths corresponding to one trace lead to the same state, it is natural to consider *traces as abstractions of executions.* For Example 1, we have $abcd \sim cadb$ and the trace $[abcd]$ corresponds to all paths leading to the state $(2, 1, 1)$.

In particular for cyclic systems, it is useful to consider an *unfolding* representing the paths and the prefix relation. For a transition system without independence relation, this unfolding will typically be a tree with paths as vertices. A tree can also be considered as a transition system with a unique path from the initial state to each reachable state. Uniqueness of paths implies that every state has only one parent.

For acyclic transition systems however, we prefer not to make a distinction between paths of the same equivalence class. This gives rise to a natural generalization of tree unfoldings called *trace systems.* These are acyclic asynchronous transition system with all paths leading to the same state belonging to the same equivalence class. It is a useful notion for reasoning about paths in an asynchronous transition system.

Definition 3. Let $T = (S, \rightarrow, s_0)$ be an asynchronous transition system w.r.t. $(\Sigma, \|)$. Then the *trace system* of T is the transition system $\mathcal{TS}(T)$ whose states are the traces associated to an execution sequence, with the empty trace $[\varepsilon]$ has initial state and such that the transition relation is $\rightarrow = \{([w], a, [w.a]) \mid w.a \in L(T)\}$.

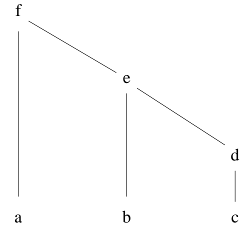
Given a sequence $u = a_1 a_2 \ldots a_n$, there is a unique $s = \sigma(u) \in S$ such that $s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s$. Similarly, we define $\sigma([u]) = \sigma(u)$ for any trace $[u]$ (for any trace $[v]$ equivalent to $[u]$ we have $\sigma([v]) = \sigma([u])$).

The asynchronous transition system in Figure 1 in fact has just one trace leading to any state, so the corresponding trace system has exactly the same structure.

Mazurkiewicz traces as generalizations of words also have an interpretation as certain finite labeled partial orders (lpo). The latter are triples $t = (E, \leq, \lambda)$ where $\leq$ is a partial order over the finite set of events E and λ maps any event $e \in E$ to the corresponding action $\lambda(e) \in \Sigma$. We require also that t has no auto-concurrency, i.e. $\lambda(e) = \lambda(e')$ implies $e \leq e'$ or $e' \leq e$ for all events $e, e' \in E$. We call *linear extension* of an lpo $t = (E, \leq, \lambda)$ any lpo $t' = (E, \leq', \lambda)$ such that $\leq'$ is a total order that refines $\leq$. Linear extensions of an lpo $t = (E, \leq, \lambda)$ can naturally be regarded as words over Σ.

Now the identification of any trace as an lpo stems from the following observation: For any trace $[u]$ there exists a *unique*[1] lpo t such that $[u]$ is precisely the set of linear extensions of t.

For Example 1, the partial order corresponding to the maximal trace $[abcdef]$ is depicted at right.



Traces themselves are partially ordered according to the *prefix relation* defined as follows: We put $[u] \preceq [v]$ whenever there exists a word $w \in \Sigma^\star$ such that $u.w \sim v$. This order has a natural interpretation through the representation of traces by labeled partial orders: Assume that a trace $[u]$ corresponds to the lpo $t = (E, \leq, \lambda)$ and let E' be a downward-closed subset of events, that is, $e \in E'$ and $f \leq e$ implies $f \in E'$. Then the restriction of t to E' is an lpo that corresponds to a trace $[v]$ and $[v] \preceq [u]$. Moreover, all prefixes of $[u]$ can be described in this way.

A crucial notion for our development concerns the number of maximal events in a trace seen as a labelled partial order.

Definition 4. For a trace $[w]$ let $last([w]) = \{a \mid [w] = [va]\}$ denote the set of labels of last (maximal) events in $[w]$, and let $\#_{last}([w]) = |last([w])|$ denote their number. A trace $[w]$ with $\#_{last}([w]) = 1$ is a *prime trace*.

In other words, a trace $[u]$ is prime if, and only if, it has a unique last event, i.e. for all words v_1, v_2 and all actions a and b, $v_1.a \sim u \sim v_2.b$ implies $a = b$.

Note, that there are two interpretations of $last([w])$ with respect to the structure of a trace system: It corresponds to the transitions with $[w]$ as target, and also it corresponds to the labels of the maximal elements in the lpo corresponding to $[w]$. Prime traces are traces with a unique predecessor in the trace system.

3 Paths leading to local properties

Our method aims at proving *local reachability properties* like *a component of a parallel composition reaches some particular state*, and it is not suited for *global*

[1] We identify here isomorphic labelled partial orders.

properties like deadlock. Local properties are also the basis of other partial order reduction methods such as the ample set method [Pel93].

In this section, we formalize local properties and explain their major structural properties concerning the trace system, which we exploit for the different variants of LFS.

3.1 Local properties

In our framework, we formalize the notion of a local property as follows:

Definition 5 (local properties). For a given set $P \subseteq S$ (called *property*), the set of *visible actions* $V_P \subseteq \Sigma$ is the set of all actions $a \in \Sigma$ such that there exist $s_1, s_2 \in S$ with $(s_1, a, s_2) \in \rightarrow$ and either $s_1 \in P$ and $s_2 \notin P$, or $s_2 \in P$ and $s_1 \notin P$. A property P has *parallel degree* n if the restriction of the independence alphabet to V_P has parallel degree n. A property is called *local* if it has parallel degree 1.

For Example 1, $X = Y$ is not a local property, because it is affected by the independent transitions a and b (for example). On the other hand, $Y = 2$ is a local property, as all transitions changing the value of Y belong to process $P2$ and are mutually dependent.

Proposition 6. *A property $P \subseteq S$ of parallel degree m is reachable (i.e. there exists a state $s \in P$ reachable from s_0) if, and only if, there exists an execution sequence $w = a_1 \dots a_k$ leading to a state $\sigma(w) \in P$ with $\#_{last}([w]) \leq m$ and all $a \in last([w])$ are visible.*

Instead of giving an (easy) proof, let us illustrate this fact on the property $Y = 1$ for Example 1. It is true for the reachable state $(2, 1, 1)$ corresponding to the trace $[abcd]$ with $\#_{last}([abcd]) = 3$. Note also that the state $(2, 1, 1)$ has three entering transitions labeled a, b and d, where a and d are invisible with respect to $Y = 1$. In following backwards along invisible transition (without loosing the property), we will end up in state $(0, 1, 0)$ corresponding to the prime trace $[b]$.

3.2 Communication degree and character

As stated in the introduction, the fundamental reason for the reductions achieved by LFS is that communication in parallel systems is typically rather local. More precisely, LFS will give good reductions for systems where the number of components/variables affected by transitions is very small compared to the parallelism in the system.

We formalize these important parameters for LFS as follows:

Definition 7. The *communication degree* of $(\Sigma, \|)$ is the maximal number n of pairwise independent actions which all depend on a common action, i.e. $n = max\{|B| \mid B \subseteq \Sigma$ *and*
$\forall b, b' \in B, b \neq b' \implies b \parallel b'$ *and*
$\exists c \in \Sigma$ *s.t.* $\forall b \in B,\ c \not\parallel b\}$.

The *character* of $(\Sigma, \|)$ is (m, n) where m is the parallel degree (Def. 2) and n the communication degree.

3.3 LFS-numbers of traces

The crucial ordering criterion for the "complexity" of a trace concerning LFS is called "LFS-number".

Definition 8. The *LFS-number* of a trace $[w]$, denoted by $LFS([w])$, is the least number l such that there exists a representative $v = a_1 \dots a_k \in [w]$ such that for each $1 \leq j \leq k$ we have $\#_{last}([a_1 \dots a_j]) \leq l$.

The LFS approach relies on the following theorem stated in [NHZL01]:

Theorem 9. *For all* prime *traces* $[w]$*, the LFS-number of* $[w]$ *is at most* $\lfloor (n-1) log_n(m) \rfloor + 1$*, where* (m, n) *is the character of* $(\Sigma, \|)$.

The bound $\lfloor (n-1) log_n(m) \rfloor + 1$ is referred to as the *static LFS-bound* of $(\Sigma, \|)$.

To illustrate the meaning of this theorem let us consider the case of a system of 1000 components with binary communication. While the trace system of this system may contain traces with up to 1000 last elements (otherwise stated: States with up to 1000 predecessors), its static LFS-bound states that all local properties are reachable on paths with intermediate traces with no more than 10 last elements (states with no more than 10 predecessors).

For the small Example 1, the static LFS-bound is 2 and the reader will easily detect paths to any local reachable property avoiding the states $(1, 1, 1)$ and $(2, 1, 1)$, which correspond to the traces with 3 last elements.

In [NHZL01], we also show that the static LFS-bound given in Theorem 9 cannot essentially be improved, i.e. for any character it is possible to construct systems with local properties with LFS-number equal to the static LFS-bound (or in some cases equal to the static LFS-bound minus one).

We omit the proof of Theorem 9 here. Theorem 10 in Section 5.1 which gives a *dynamic LFS-bound* has a proof with a similar structure.

4 Search procedures

Theorem 9 assures the existence of a path with a bound on the number of last elements of the traces on the path, but there will still be an infinite number of such traces. We have found two ways to exploit this property of traces on finite structures.

Local First Search. The first method [NHZL01] is based on the following observation: $last([ua]) = \{a\} \cup \{b \in last([u]) \mid b \parallel a\}$. In particular, if $last([u]) \subseteq last([v])$ then $last([ua]) \subseteq last([va])$. This leads to the definition of the *last label tracking transition system*: Given the transition system $(S, \rightarrow, s_0)$, the last label tracking transition system $LLTTS(T, \Sigma)$ is defined by $LLTTS(T, \Sigma) = (S', \rightarrow, s'_0)$, where $s`_0 = (s_0, \emptyset), S' = \{(s, M) \mid s \in S, M \subseteq 2^{\Sigma}\}$, $(s, M) \xrightarrow{a} (s', M')$ iff $s \xrightarrow{a} s'$ and $M' = M \backslash \{b \in M \mid b \not\parallel a\}$. In [NHZL01], we proved that a local property is reachable in the original transition system T iff it is reachable in $LLTTS(T, \Sigma)$.

The basic algorithm for model-checking local properties is described by the following scheme (see [NHZL01] for details and correctness proof).

$B := \lfloor (n-1) \log_n m + 1 \rfloor$;
Explore$:=(s_0, \emptyset)$; *Visited*$:=\emptyset$;
repeat
  choose (s, M) from *Explore*;
  for each transition $s \xrightarrow{a} s'$
  do
    if $s' \models P$ **then** return(s')
    else $M' := \cup\{a\} \cup \{b \in M \mid a \parallel b\}$;
      if $|M'| \leq B$ and for all $(s'', M'') \in Explore \cup Visited$: $M'' \not\subseteq M$
      then add (s', M') to *Explore*
      fi
    fi
  od
  remove (s, M) from *Explore*, add (s, M) to *Visited*;
until *Explore*$=\emptyset$

The algorithm obviously terminates, but the necessity to keep the set of last labels may add a significant overhead to the construction, notably it can necessitate to explore a same state s with several incomparable sets M.

In [NHZL01], we defined the *Local First Search* as *choose* (s, M) *in Explore such that* $|M|$ *is minimal.* Our first experiments [NHZL01] showed that this heuristic could be quite effective to reduce the number of explored states when the general algorithm (explore all traces up to the theoretical bound) was not. A crucial remark is that usually the set of states corresponding to prime traces generated by the method quickly stabilizes which suggests that we could use a dynamic bound instead of the static theoretical bound. In this paper, we give such a dynamic bound (see Section 5.1), which will help make LFS usable also for showing the non reachability of local properties.

Unfolding. In order to avoid the overhead caused by exploring certain states more than once, in [BMNZ02] ideas were taken from McMillan prefixes (notably the notion of an *adequate order* [ERV96]) and adapted to LFS. As a result, it is possible to construct a *locally complete finite prefix* of the trace system with the following properties:

- It is not bigger than the initial system.

– It preserves the reachability of local properties.
– It doesn't contains trace with a LFS-number greater than the LFS-bound.

Since the development of adequate orders is quite involved, we do not explain this method here (see [BMNZ02] for the details). However, it has to be stated that the current "blackbox unfolding" setting does not allow an incremental computation of prime traces. Therefore, the dynamic bound elaborated in Theorem 10 is not immediately applicable to the unfolding approach.

5 LFS optimizations

This section contains the two main technical innovations of this paper: On the one hand, we prove a dynamic LFS-bound that allows to stop much earlier in an exhaustive LFS-search with the result "unreachable" than with the static worst case bound.

On the other hand, we discuss two options of combining LFS with other partial order reduction techniques. Positively, we show how to combine LFS with a wide spread technique called "transition merging". Negatively, we show that at least a straight forward combination of LFS with the "sleep set method" is not possible.

5.1 LFS with dynamic bound

The static LFS-bound of Theorem 9 is a bound for the worst case and our experiments have shown that apparently often the LFS-numbers of local properties are much lower. This has motivated us to look for tighter criteria for discovering bounds on the LFS-numbers of a concrete system.

Let $\mathcal{TS}(T)$ be the trace system associated to a transition system $T = (S, \rightarrow, s_0)$ w.r.t. $(\Sigma, \|)$ where $(\Sigma, \|)$ has character (m, n) with $n \geq 2$. Let L_p denote the set $\{[u] \mid [u] \textit{ is prime and } LFS([u]) \leq p\}$. Theorem 9 can be rephrased in saying that L_B, where B is the static LFS-bound, already contains all prime configurations, and thus $L_B = L_{B+1} = L_{B+2} = \ldots$. In short, the sequence $L_1, L_2, \ldots$ converges latest at L_B.

The following theorem gives a criterion to *detect* convergence:

Theorem 10. $L_p = L_{p+1} = \ldots = L_{p+n-1} \Rightarrow \forall k \geq 0 \; L_{p+k} = L_p$.

Proof. Let $[wa]$ be a prime trace in $L_{p+n} \setminus L_{p+n-1}$ and we assume that $[wa] = a_1 \ldots a_l$ is minimal, i.e. there is no prime trace $[w'] = b_1 \ldots b_{l'} \in L_{p+n} \setminus L_{p+n-1}$ with $l' < l$. Since $[wa]$ is prime and the communication degree is n, the l.p.o $(\leq, E, \lambda)$ representing $[w]$ has at most n last elements. Let $\downarrow e = \{f \in E \mid f \leq e\}$ and let $e_1, \ldots, e_r$ be the set of maximal elements of $(\leq, E, \lambda)$. Figure 2 shows the structure of the partial order $\leq$.

Since $[w]$ is minimal, the restriction of $\leq$ to $\downarrow e_i$ has LFS-number $n_i \leq p$: n_i is smaller than or equal to $p + n$ since $LFS([w]) \leq p + n$, but since $[wa]$ is minimal it must be that $n_i < p + n$ (otherwise $\leq_{e_i}$ gives a smaller prime trace

with LFS-number equal to $p+n$). Since $L_p = \ldots = L_{p+n-1}$ by hypothesis, we have $n_i \leq p$.

Let $E_j = \downarrow e_j \setminus \bigcup_{1 \leq i < j} \downarrow e_i$ be the set of elements $f \in E$ such that $f \leq e_j$, but $f \not\leq e_i$ for $i < j$. This set is not empty, because e_j is maximal and hence $e_j \in E_j$.

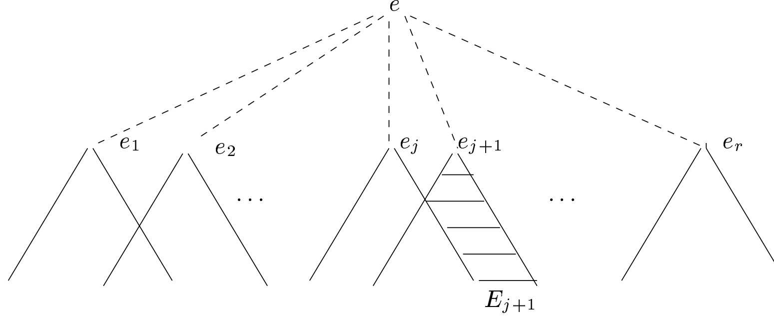


Fig. 2. The partial order corresponding to $[wa]$

We prove by induction on j that $\bigcup_{i=1}^{j} \downarrow e_i = \downarrow \{e_1, \ldots, e_j\}$ (with the corresponding restriction of the partial order) has LFS-number less than or equal to $p+j-1$ where $1 \leq j \leq l$.

- Base case $j = 1$. The corresponding l.p.o. is $\downarrow e_1$ for which the property holds as seen above.
- Inductive step. We assume that the property holds for $\downarrow \{e_1, \ldots, e_j\}$.
  Let $[u]$ be the trace corresponding to $\downarrow e_{j+1}$. Then $[u]$ can be factored into $[u_1 u2]$ where $[u_2]$ corresponds to E_{j+1}. Since by hypothesis the prime trace $[u_1 u2]$ has LFS-number $\leq p$, so has $[u_2]$: A representative of $[u_1 u_2]$ witnessing its LFS-number is an interleaving $u_1^1 u_2^1 u_1^2 u_2^2 \ldots u_1^s u_2^s$ such that $[u_1] = [u_1^1 \ldots u_1^s]$ and $[u_2] = [u_2^1 \ldots u_2^s]$ (where some factors may be empty). It is easy to see that for any prefix $\mathit{last}([u_2^1 \ldots u_2^t]) \subseteq \mathit{last}([u_1^1 u_2^1 u_1^2 \ldots u_1^t u_2^t])$, which shows that $[u_2]$ also has LFS-number $\leq p$.
  Let $[v]$ be the trace corresponding to $\downarrow \{e_1, \ldots, e_j\}$ with v a witness of its LFS-number $\leq p+j-1$. Then composing vu_2', where u_2' is a witness of LFS-number $\leq p$ of u_2, yields a witness of LFS-number $\leq j+p$ for $[vu_2]$. The latter trace corresponds to $\downarrow \{e_1, \ldots, e_j\} \cup E_{j+1} = \downarrow \{e_1, \ldots, e_{j+1}\}$.

The desired result holds by taking $j = r \leq n$, which proves that $[w]$ has LFS-number less than or equal to $p+n-1$. Then $[wa] \in L_{p+n-1} = L_p$. Therefore $L_{p+k} = L_p$ for all $k \geq 0$.

A immediate consequence of this theorem is the following improvement for the LFS basic algorithm: In the search algorithm, which iterates over the LFS-

number p, we check for *new* states (in $L_{p+1} \setminus L_p$) where $L_p = \{(s, M) \mid |M| = 1\}$ and we stop the search as soon as we have $L_{p+1} \setminus L_p = \ldots = L_{p+n-1} \setminus L_{p+n-2} = \emptyset$.

This improvement is especially interesting for systems with a low communication degree which happens in many applications ($n = 2$ in many cases). In that case, we can decide the non-reachability of local properties at one level above the actual LFS-bound of the prime traces. In the examples treated in Section 6, always the actual LFS-bound turns out to be 2, so exhaustive search stops at 3 rather than the logarithmic bound predicted by Theorem 9.

5.2 Transition merging

A simple and yet very effective reduction method in the partial order family is called *transition merging* (see e.g. [DR99]). The basic observation is that often system descriptions contain sequences of local transitions (of one process of a system) that do not interfere with the behavior of the rest of the system and the local property to verify. Merging such an *invisible transition* that does not take part in a choice to a predecessor thus preserves local properties.

It turns out that transition merging is compatible with LFS and that the reductions achieved by the two methods are nearly orthogonal.

In the following, we give a semantic formalization of transition merging for LFS. For the applicability from a practical point of view, criteria that can be checked by static analysis and are sufficient to guarantee these semantic conditions.

We assume a fixed local property and correspondingly a fixed set of visible transitions. Our formalization is a semantic one, defined on trace systems. In practice, sufficient syntactic criteria to be detected by the static analysis component of the verification system must be used.

We say that a transition $[u] \longrightarrow [ua]$ is *local* iff there exists a decomposition $[u] = [u_1 b u_2]$ such that $a \not\parallel b$ but $a \parallel u_2$ and $[u_1 b]$ is a prime trace. We say that a transition $[u] \longrightarrow [ua]$ is *in conflict* iff there exists a decomposition $[u] = [u_1 u_2]$ such that $b \parallel u_2$ for some b and furthermore there exists in the trace system a trace $[u_1 v b]$ such that $a \parallel v$ but $a \neq b$ and $a \not\parallel b$. Otherwise, we call $[u] \longrightarrow [ua]$ conflict free.

For a transition $[u] \to [ub]$, we say that the sequence $v = a_1 \ldots a_n$ is mergeable to b (yielding the merged transition $[u] \to [uba_1 \ldots a_n]$ iff for all i with $1 \leq i \leq n$ we have $a_i \not\parallel ba_1 \ldots a_{n-1}$, and furthermore $[uba_1 \ldots a_{i-1}] \to [uba_1 \ldots a_i]$ is a local invisible conflict free transition. For simplicity, we call the latter transitions "attachable".

Let a *merged trace system* be the trace system where each transition $[u] \to [ub]$ is replaced by an arbitrary set of mergeable transitions[2] $[u] \to [uba_1 \ldots a_n]$.

With these definitions, we observe the following properties for the merged trace system.

[2] In practice, we will use a merging strategy in order to reduce the number of states visited. Intuitively, we want to join a maximal number of attachable transitions to b, thus obtaining a single replacement for b. Care has to be taken concerning cycles.

Proposition 11. *(1) The communication degree of transitions in the merged trace system is bounded by the communication degree of simple transitions.*

(2) The static and dynamic LFS-bounds determined for the simple trace system hold for the merged trace system.

(3) A local property reachable in the merged trace system is reachable in the simple trace system.

(4) A local property is reachable in the trace system with LFS-number l is reachable in the merged trace system with LFS-number $\leq l$.

Proof. For (1) it is sufficient to note that for two subsequent mergeable transitions $ba_1 \dots a_k$ and $b'a'_1 \dots a'_l$ we have $[u] \to [uba_1 \dots a_k] \to [uba_1 \dots a_k b'a'_1 \dots a'_l]$ we have $ba_1 \dots a_k \not\parallel b'a'_1 \dots a'_l$ iff $ba_1 \dots a_k \not\parallel b'$ and similarly for two mergeable transitions $[u] \to [uba_1 \dots a_k]$ and $[u] \to [ub'a'_1 \dots a'_l]$ we have $ba_1 \dots a_k \not\parallel b'a'_1 \dots a'_l$ iff $ba_1 \dots a_k \not\parallel b'$ or $b \not\parallel b'a'_1 \dots a'_k$. In either case, the number of predecessors a merged transition can depend upon is limited by the number of predecessors of the simple transitions b, b'.

(2) is an immediate consequence of (1). (3) is obvious.
(4) can be shown by induction: It suffices to take a path v witnessing the LFS-number of the local property and inductively replace the simple transitions by mergeable transitions. For a prefix u of v, this will result in a trace $[uu']$ such that u' consists of attachable transitions only. There are two cases for transitions $[u] \to [ub]$: Either $b \parallel u'$ in which case the merged transition $[uu'] \to [uu'bu'']$ leads to a trace $[ubu'u'']$; in the other case b itself must be local conflict free invisible and $[uu'] = [ubu'']$ for some u'', so that the transition is already taken and we skip it.

The resulting sequence of merged transitions leading to a state $[vv']$, where v' consists of invisible transitions, witnesses that the LFS-number of $[vv']$ in the merged trace system is limited by the LFS-number of $[v]$ in the original system, but this does not exclude the possibility of a decreased LFS-number.

In our prototype implementation, LFS on LLTTS as well as unfolding LFS integrating mergeable transitions uses non-empty sequences of transitions as labels, with a dependence between $[b, a_1, \dots, a_n]$ and $[b', a'_1, \dots, a'_m]$ iff there is a dependence between b and one of $b', a'_1, \dots, a'_m$ or between b' and one of $b, a_1, \dots, a_n$. Moreover, the attachable transitions are marked so by the transition generator (in a tool like Spin, this would have to be determined by static analysis). A generator for mergeable transitions is placed on top of the transition generator: In principle, it will try to attach a maximal number of attachable transitions to a first transition encountered and pass this sequence to the state space explorer (LFS or unfolding LFS).

Dealing with cycles. However, a sequence of attachable transitions may lead into a loop (we assume finite state systems).

Now, it easy to see that no future visible transition can depend directly or indirectly on a transition in a loop of attachable transitions. In order to avoid looping, the generator for mergeable transitions considers any attachable

transition closing a cycle as *dead*. It will continue to add attachable to a sequence until only dead attachable transitions remain.

The experimental results (where we have ourselves attachable transitions in the model generator, rather than implementing static analysis techniques) confirm that mergeable transitions alone give excellent reductions, but also that the combination with LFS gives reductions by far exceeding those achievable by either reduction in isolation.

Indeed, transition merging is rather orthogonal to LFS. This is a strong point for LFS, as it contrasts the situation of transition merging with respect to other partial order reduction methods. Note for example, that conditions similar to those on attachable transitions also been considered for partial order reductions in the style of ample sets for the case of branching time logics (see [GKPP95,WW96]). For this case, all potential reductions are already covered by transition merging.

5.3 Why sleep sets and LFS are incompatible

A well known partial order reduction technique is the sleep set method [GW93]. Its aim is to reduce the number of transitions (rather) than states to be explored when exploring a system with depth first search. It has the remarkable property of preserving exactly one representative of each trace.

The method is rather easy to describe: During depth first search, explored states are decorated with a set of "sleeping transitions". For the initial state, it is empty. For a state with sleep set M, transitions labeled with letters $a \in M$ are forbidden. For a state (s, M), let $(a_1, s_1), \dots (a_k, s_k)$ be the list of successor states, explored by depth first search in that order. Then the sleep sets for the successor states are calculated by the rule $M_j = (M \cup \{a_1, \dots, a_{j-1}\}) \setminus \{b \mid b \not\parallel a\}$. Otherwise speaking, sister transitions explored before by depth first search are put to sleep; transitions that depend on a transition taken are woken up. The idea is readily understood looking at a simple diamond *ab ba*: If DFS is begun exploring a, and after all successors of a have been explored, then the independent transition b in itself may lead to unexplored states. But after b there is no need to explore ba, which is a successor of a from the trace point of view and thus has already been explored.

There are several examples of combining the sleep set method with other reduction methods (e.g. with ample sets [Pel93]) and since LFS does not reduce interleavings on the small scale, it would be interesting to combine these methods.

An example reduction of the transition system of Example 1 is shown in Figure 3. We assumed that DFS explores transitions in the top down order.

In the same image, we indicated the transitions forbidden by LFS-bound 2. There exist paths to the last state (with the local property $Z = 2$ either avoiding transitions forbidden by the sleep sets ($abcdef$) or avoiding LFS-excessive traces (e.g. $cdbeaf$), but not both at a time.

The intuitive reason behind the incompatibility of sleep sets and LFS as shown by this example is that sleep sets assume exhaustive search of a (reduced)

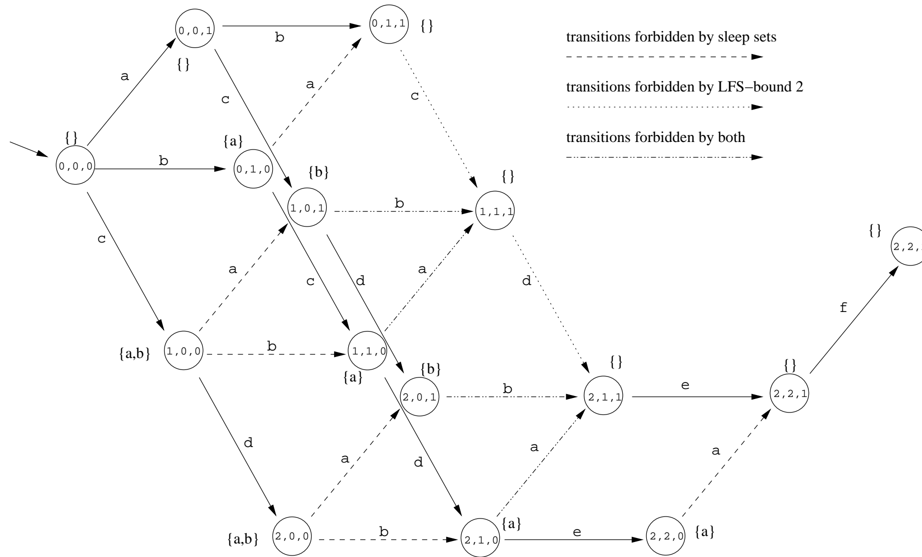


Fig. 3. Sleep set and LFS reduction for Example 1.

state space behind an arbitrary transition (a in the Example), but LFS does not guarantee this. In particular, LFS allows situations where a trace is forbidden but not all of its successors.

6 Experiments

Our current prototype implementation of all LFS methods relies on a modular structure as depicted in Figure 4. The LFS explorers (either via the LLTTS or via unfolding) are realized as polymorphic functions taking the model generator (likewise a collection of functions) as parameter. The model generator provides implicitly a data type for states, names of transitions, the independence relation and – required for transition merging – information about attachability of transitions. For each example, the model generator was written directly in Caml. In an integrated verification tool, this component must be replaced by a module for syntactic and static analysis of an input language such as Promela.

Note that from the point of view of the explorer functions, there is no difference in using them directly on a model or a model transformed by the transition merger.

The polymorphic function approach turned out to be very useful for trying out a number of combinations in a rapid prototyping fashion in a very short time. The price paid is a lack of run-time efficiency: The number of states we can actually explore in a reasonable amount of time is limited to largely less than 100000 states.

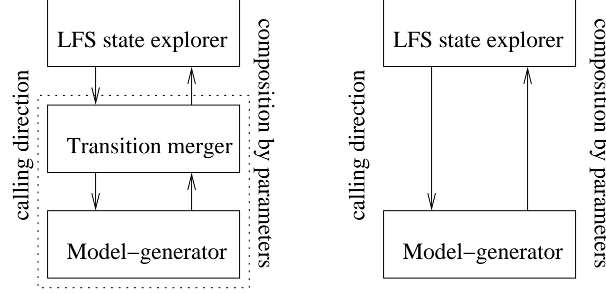
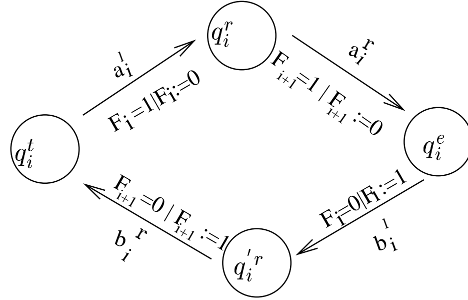


Fig. 4. Architecture of the prototype implementation

We experimented our prototype on three classes of concurrent systems. Theses examples are parameters with m the parallel degree (i.e. the number of components) and the communication is 2-bounded. The independence relation is uniformly chosen to represent concurrency in these models: Two transitions are independent if they are executed by different processes and do not reference the same shared variables.

6.1 The *dining philosophers* example

We have chosen the following instance of the (in)famous dining philosopher example. Philosophers are left-handed and the behavior of one philosopher is given in the picture at right. Each philosopher i can be in four possible states thinking q_i^t, aiming at taking the right fork q_i^r, eating q_i^e, aiming at giving back the right fork $q_i'^r$. The communication is achieved by the means of shared variables F_i's stating whether fork i is free or not.



6.2 The *n-buffer* example

The second concurrent system we have experimented is a classical example used to study the reduction potential of partial order methods.

Each process –called a cell– is allowed to use a resource only if it has a token. In our version, only one cell can create a token and sends it to its right neighbor; the other cells receive a token and from their left neighbor and send it to their right one (we don't care about an internal action of a cell to access the resource). As the last cell has no neighbor, we added to it an internal action before it can receives a token again.

The promela description of the n-buffer is as follows:

```
byte  X[size];
/*initial cell*/
proctype initial(){do  ::atomic{X[0]==0 -> X[0]=1;}  od;}
/*final cell*/
proctype final()  {do  ::atomic{X[size-1]==1 -> X[size-1]=0;}  od;}

/* cells in the middle */
proctype cell(int n){
  /* token passes from cell n-1 to cell n*/
  do ::atomic{X[n-1]==1&&X[n]==0 ->X[n-1]=0;X[n]=1;} od;}
```

6.3 The *echo protocol* example

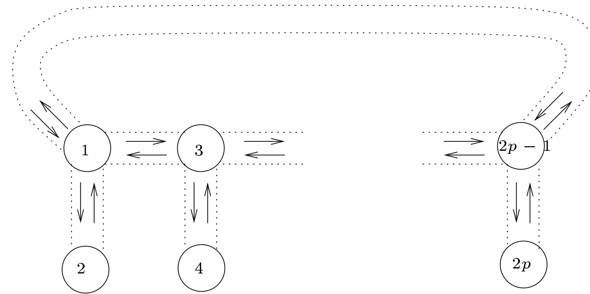


Fig. 5. The net structure for the echo protocol

Echo protocol is a simple asynchronous communication protocol realizing broadcast among a set of arbitrarily interconnected processes. Initially, all processes but one are sleeping. When a sleeping process receives a wakeup message from one of its neighbors, it sends a wakeup message to all its other neighbors. Then it waits for an acknowledgment from these one and sends to the process that woke it up an acknowledgment.

The kind of network chosen for the experiments is a ring-like net with neighbors attached to the processes on the ring (see Figure 5). Nodes are processes; communication channels are represented by dotted lines.

All messages are undistinguished and a process cannot distinguish a wakeup message from an acknowledgment message. In other words, for a woken up process, any received message is an acknowledgment. External nodes are numbered by even numbers. Since we don't choose the initiator of the protocol among these processes, they have a simple behavior which is specified by the following Promela description (in this program the communication channel is separated into two channels, one for incoming messages the other one for sending messages):

```
proctype external(int n; chan in,out) {
  /* wait for a msg, when received send a message back */
  in?msg ->out!msg;
}
```

A internal node has a more complex behavior. It waits for a message from one of its neighbors (not fixed in advance), then send a message to the two other neighbors. Then it waits an acknowledgement from the latter neighbors and finally sends an acknowledgement to the neighbor which sent it the first message. This behavior can be described as follows:

```
proctype Inode(int n; chan in1,in2,in3,out1,out2,out3){
  int i;
  /*wait for a message on any input channel*/
if
 ::in1?msg ->i=1;
 ::in2?msg ->i=2;
 ::in3?msg ->i=3;
fi;
         /* wake up all neighbours in clockwise  order */
if
  :: i==1 -> /* wake up 2 and 3*/
             out2!msg; out3!msg;
             /* wait for acknowledgement from 2 and 3 */
             in2?msg; in3?msg;
             /*send acknowledgment to 1*/
             out1!msg;
  :: i==2 -> /*similarly up to permutation*/
             out1!msg; out3!msg; in2?msg; in3?msg; out2!msg;
  :: i==3 -> /*similarly up to permutation*/
             out1!msg; out2!msg; in1?msg; in2?msg; out3!msg;
fi;}
```

The initiator is an internal node which starts by sending a message to all its neighbors and waits for acknowledgements from these ones.

6.4 Experimental results

For each class of examples we have conducted a number of experiments with varying numbers of processes m and different combinations of reduction methods:

- LLTTS exploration with the dynamic bound and transition merging.
- LLTTS exploration with the dynamic bound.
- Transition merging alone (no LFS reduction) for reference.
- No reduction at all.
- Unfolding LFS with the static bound.

- Unfolding LFS with the dynamic bound found via the other experiments (oracle!). The interest in these experiments is to show the additional reduction potential of the dynamic bound for unfolding LFS, while we cannot execute unfolding LFS in the incremental way required.
- Unfolding LFS with the static bound and transition merging.

philosophers (m)	1	2	3	4	5	6	7	8	9	10	20
dynamic bound, merging	2	4	9	16	25	36	49	64	81	100	400
dynamic bound (no merging)	2	8	37	202	1006	4195	13981	38759	–	–	–
merging (no LFS)	2	5	16	45	116	283	666	1529	3448	7611	–
no reduction	2	8	26	80	242	728	2186	6560	19682	59048	–
unfolding LFS	2	8	25	79	226	598	1450	5347	13372	31286	–
unfolding LFS, oracle bound	2	8	25	79	226	598	1450	3229	6655	12806	–
unfolding LFS, merging	2	4	8	15	26	42	64	163	256	386	–

asynchronous buffers (m)	1	2	3	4	5	6	7	8	12	15	32
dynamic bound, merging	1	4	4	4	4	4	4	4	4	4	4
dynamic bound (no merging)	1	3	8	18	43	100	222	464	4850	17406	–
merging (no LFS)	1	3	4	4	4	4	4	4	4	4	4
no reduction	2	4	8	16	32	64	128	256	4096	32768	2^{32}
unfolding LFS	1	3	6	12	24	48	95	192	2829	–	–
unfolding LFS, oracle bound	1	3	6	12	24	48	95	184	1865	7359	–
unfolding LFS, merging	1	3	3	3	3	3	3	3	3	3	3

echo processes (m)	2	4	6	8	10	12
dynamic bound, merging	8	47	208	885	3631	14395
dynamic bound (no merging)	16	190	1763	10844	–	–
merging (no LFS)	8	51	216	843	3120	11139
no reduction	18	827	–	–	–	–
unfolding LFS	13	278	5646	–	–	–
unfolding LFS, oracle bound	13	278	5646	–	–	–
unfolding LFS, merging	6	18	54	162	482	1452

Fig. 6. Experimental results (number of states in hash table) with different methods

For each experiment, we measured the number of states explored on the LFS level (states in the hash table). It is important to note that this number is inferior to the number of actually explored states in the case of transition merging. This shows up in extreme for the case of transition merging for the n-buffer: For this example, almost all transitions are attachable so that the transition merger jumps across them.

For all examples, the stabilization according to Theorem 10 occurs at LFS-number 2, which is detected after exploring the LLTTS up to LFS-number 3. Unfortunately, the first occurrence of a difference between the dynamic bound and the static bound occurs at $m = 8$ and lacking an efficient implementation we cannot go much further than $m = 10$.

The state copying problem of the LLTTS approach causes for some examples or combinations an overhead beyond reduction. For other combinations, e.g. the combination with transition merging for the philosophers, it gives reductions that could not be improved.

As is well known, transition merging alone is a powerful reduction method, and for the n-buffer, it already does all the work. In other examples, it shows up that a combination in particular with the unfolding version of LFS gives excellent additional reductions.

It must be noted that LFS is a reduction method tailored on systems with a lot of parallelism – beyond our current experimental possibilities. On two process systems, it will give no reduction at all. While the results are encouraging already on our small examples, the potential on large examples is likely to be much stronger.

Acknowledgements

Thanks go to Rémi Morin for his comments on this work.

References

[BMNZ02] S. Bornot, R. Morin, P. Niebert, and S. Zennou, *Blackbox unfolding with local first search*, TACAS, Springer, 2002, accepted for publication.

[DR95] V. Diekert and G. Rozenberg (eds.), *The book of traces*, World Scientific, 1995.

[DR99] Y. Dong and C. Ramakrishnan, *An optimizing compiler for efficient model checking*, IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV), 1999.

[ERV96] J. Esparza, S. Römer, and W. Vogler, *An improvement of McMillan's unfolding algorithm*, TACAS (T. Margaria and B. Steffen, eds.), LNCS, vol. 1055, 1996, pp. 87–106.

[GKPP95] Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek, *A partial order approach to branching time model checking*, Israeli Symp. on Theoretical Computer Science, 1995.

[GW93] P. Godefroid and P. Wolper, *Using partial orders for the efficient verification of deadlock freedom and safety properties*, Formal Methods in System Design **2** (1993), 149–164.

[McM92] K.L. McMillan, *Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits*, Computer Aided Verification (CAV), 1992, pp. 164–174.

[NHZL01] P. Niebert, M. Huhn, S. Zennou, and D. Lugiez, *Local first search – a new paradigm in partial order reductions*, International Conference on Concurrency Theory (CONCUR), LNCS, no. 2154, 2001, pp. 396–410.

[Pel93] D. Peled, *All from one, one for all: On model checking using representatives*, International Conference on Computer Aided Verification (CAV), LNCS, vol. 697, 1993, pp. 409–423.

[WW96] Bernard Willems and Pierre Wolper, *Partial-order methods for model checking: From linear time to branching time*, IEEE Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1996, pp. 294–303.