A Compositional Sweep-Line State Space Exploration Method[*]

Lars Michael Kristensen[1,2**] and Thomas Mailund[2]

[1] School of Electrical and Information Engineering, University of South Australia, Mawson Lakes Campus, Mawson Lakes, SA 5095, AUSTRALIA
lars.kristensen@unisa.edu.au
[2] Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, DENMARK,
{lmkristensen,mailund}@daimi.au.dk

Abstract. State space exploration is a main approach to verification of finite-state systems. The sweep-line method exploits a certain kind of progress present in many systems to reduce peak memory usage during state space exploration. We present a new sweep-line algorithm for a compositional setting where systems are composed of subsystems. The compositional setting makes it possible to divide subsystem progress measures into monotone and non-monotone progress measures to further reduce peak memory usage. We show that in a compositional setting, it is possible to automatically obtain a progress measure for the system by combining the progress measure for each subsystem to a progress measure for the full system.

1 Introduction

A main method for verification of distributed and concurrent systems is state space exploration. The basic idea is to generate all the reachable states and state changes of the system under consideration, and represent these as a directed graph called the *state space*. The nodes in the state space represent the reachable states of the system, and the edges represent transitions between the reachable states. Because of the state explosion problem, the limiting factor in verification based on state spaces is, in most cases, the available computer memory. This has lead to the development of a wide range of *state space reduction methods* capable of reducing the memory required to conduct state space exploration. Partial-order methods [16,14,19] explore a subset of the state space, whereas the symmetry method [2] and symbolic methods based on Binary Decision Diagrams (BDDs) [12] provide a compact representation of the state space.

Another main paradigm is that of compositional state space verification [17]. In this paradigm, systems are specified as a parallel composition of subcomponents, and the state space of the full system is computed from the state spaces

[*] Supported by an Australian Research Council (ARC) Discovery Grant (DP02105).
[**] Supported by the Danish Natural Science Research Council.

of the subcomponents. Moreover, the state spaces of subcomponents can be replaced by smaller and behaviourally equivalent state spaces before constructing the state space of the full system. Process algebras [5, 13] are examples of formal description techniques that naturally fit the compositional paradigm.

The basic idea of the sweep-line method [1, 9] is to exploit a certain kind of *progress* exhibited by many systems. Exploiting progress makes it possible to explore all the reachable states of a system, while only storing small fragments of the state space in memory at a time. The key concept of the sweep-line method is the notion of a *progress measure* which is used to capture the progress in the system. A progress measure associates a *progress value* to each state of the system, and it is the progress values of states that are used to determine which states can be deleted during the *sweep* through the state space. The sweep-line method is aimed at on-the-fly verification of safety properties, e.g., determining whether a reachable state exists satisfying a given state predicate. The sweep-line method was used in [4] for verification of transactions in the Wireless Application Protocol (WAP) with a reduction in peak memory usage to 20%.

The sweep-line method is similar in spirit to the *state space caching method* [7, 3] in that they are both based on the paradigm of deleting states encountered during the state space exploration. The criteria for deletion of states is, however, fundamentally different. In the state space caching method, states not on the depth-first stack can be deleted. With the sweep-line method, deletion is based on progress values. The sweep-line method and state space caching both guarantee complete coverage of the state space, i.e., exploration of all reachable states. For the monotone progress measures of [1], the sweep-line method explores each state exactly once, whereas state space caching may explore states multiple times. For the generalised sweep-line method [9] the main difference is in the search order. State space caching relies on depth-first exploration, whereas the sweep-line method visits the reachable states in a least-progress-first order.

A disadvantage of the sweep-line method is that it requires additional information from the user in the form of a progress measure. It is therefore desirable to automatically compute progress measures for systems. One contribution of this paper is to show that in a compositional setting where state spaces of the subcomponents are explicitly represented, a progress measures for each subcomponent can be obtained algorithmically and combined to a progress measure for the full system. This makes the sweep-line method fully automatic in such a setting. Furthermore, we present a new sweep-line state space exploration algorithm which exploits a division of subcomponent progress measures into monotone and non-monotone progress measures to further reduce peak memory usage.

The paper is organised as follows. Section 2 gives the necessary background on the sweep-line method. Sections 3 and 4 show how to obtain a global progress measure from progress measures for the subcomponents of the system, and presents the sweep-line algorithm for compositional progress measures. Section 5 presents two algorithms for computing progress measure for subcomponents. Section 6 presents some experimental results and compares the use of the compositional sweep-line method with the use of the state space caching method. Finally in Sect. 7, we sum up the conclusions. The reader is assumed to be familiar with the basic ideas of state space exploration.

2 The Sweep-Line Method

This section contains the necessary background information on the sweep-line method [1, 9]. To make our presentation independent of the concrete modelling language used for specification of the system, we present our results in the framework of *labelled transition systems*.

Definition 1. *A* ***labelled transition system*** *(LTS) is a tuple* $\mathcal{L} = (S, \Sigma, \Delta, \iota)$*, where* S *is a finite set of* states*,* Σ *is a finite set of* transition labels*,* $\Delta \subseteq S \times \Sigma \times S$ *is the* transition relation*, and* $\iota \in S$ *is the* initial state*.* $\square$

We will write $s \xrightarrow{a} s' \in \Delta$ if $(s, a, s') \in \Delta$. A state s_n is *reachable* from a state s_1 iff there exists states $s_2, s_3, \ldots, s_{n-1}$ and transition labels $a_1, a_2, \ldots a_{n-1}$ such that $s_i \xrightarrow{a_i} s_{i+1} \in \Delta$ for $1 \leq i \leq n-1$. If a state s' is reachable from a state s in the LTS $\mathcal{L}$, we write $s \rightarrow^*_{\mathcal{L}} s'$. We will refer to $\rightarrow^*_{\mathcal{L}}$ as the *reachability relation*. For a state s, $reach_{\mathcal{L}}(s) = \{ s' \in S \mid s \rightarrow^*_{\mathcal{L}} s' \}$ denotes the set of states reachable from s. The set of *reachable states* of an LTS $\mathcal{L}$ is then $reach_{\mathcal{L}}(\iota)$. The transition labels of the LTSs are not important for the sweep-line method, and hence we omit them in the figures. An LTS can be viewed as representing the state space of a system, and we therefore use the terms state space and LTS interchangeably.

A common way of constructing systems is through composition, and several parallel composition operators have been defined in the literature. In this paper we do not consider a particular parallel composition operator. Our only requirement to the parallel composition is captured in the definition below. This requirement is satisfied by most parallel composition operators used in practice.

Definition 2. *Let* $(S, \Sigma, \Delta, \iota) = \mathcal{L}_1 \| \cdots \| \mathcal{L}_n$ *be the parallel composition of* n *LTSs, where* $\mathcal{L}_i = (S_i, \Sigma_i, \Delta_i, \iota_i)$ *for* $1 \leq i \leq n$*, and* $S = S_1 \times \cdots \times S_n$*. The operator* $\|$ ***respects local reachability*** *if and only if the following holds:*

- $\iota = (\iota_1, \ldots, \iota_n)$.
- $(s_1, \ldots, s_n) \rightarrow^*_{\mathcal{L}} (s'_1, \ldots, s'_n) \Rightarrow s_i \rightarrow^*_{\mathcal{L}_i} s'_i$ *for* $1 \leq i \leq n$*.* $\square$

The sweep-line method is based on the concept of a *progress measure*. A progress measure specifies a *partial order* $(O, \sqsubseteq)$ (i.e., a reflexsive, antisymmetric, and transitive relation) on progress values O of the system, and a *progress mapping* ψ assigning a *progress value* $\psi(s) \in O$ to each state s.

Definition 3. *A* ***progress measure*** *on an LTS* $\mathcal{L} = (S, \Sigma, \Delta, \iota)$ *is a tuple* $\mathcal{P} = (O, \sqsubseteq, \psi)$ *such that* $(O, \sqsubseteq)$ *is a partial order and* $\psi : S \rightarrow O$ *is a progress mapping from states into* O*. A* ***monotone progress measure*** *is a progress measure satisfying:* $\forall s, s' \in reach_{\mathcal{L}}(\iota) : s \rightarrow^*_{\mathcal{L}} s' \Rightarrow \psi(s) \sqsubseteq \psi(s')$*.* $\square$

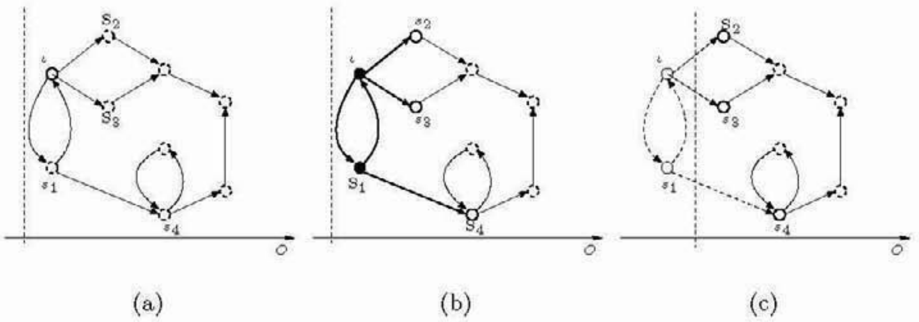


Fig. 1. Snapshots of the sweep-line method. The states are ordered left to right according to their progress value. Initially (a), only the initial state ι is stored and marked as unprocessed, and the sweep-line (the dashed vertical line), is to the left of the state space. Successors of ι are calculated and marked as unprocessed. After ι has been processed, s_1 has a minimal progress value among the unprocessed states (s_1, s_2, and s_3) and is selected for processing. After s_1 has been processed, all states with the initial progress value (i.e., ι and s_1) have been processed (b), and the sweep-line can move to the right. As it does this, the states ι and s_1 with progress value strictly less than the minimal progress value among the unprocessed states (i.e., s_2, s_3, and s_4) are deleted (c). Either s_2 or s_3 can now be selected for processing and the exploration continues.

Monotone progress measures preserve the reachability relation $\rightarrow_{\mathcal{L}}^{*}$ of the LTS. This means that a monotone progress measure $\mathcal{P}$ provides a conservative estimate of the reachability relation. In conventional state space exploration, the set of explored states is kept in memory to recognise already visited states. For a system with a monotone progress measure, the states with a progress value strictly less than the progress values of unprocessed states[1] cannot be reached from the set of unprocessed states. It is therefore safe to delete such states from memory as they are not required to determine whether a newly generated state has already been visited. Saving memory by deleting such states is the basic idea underlying the sweep-line method. Figure 1 [9] illustrates the sweep-line method for monotone progress measures. The progress measure orders the states of the LTS, and the states are processed in this order. New states to be processed are chosen with minimal progress value, and whenever the minimal progress value increases, all states with a strictly smaller progress value are deleted. Intuitively, a sweep-line is dragged through the LTS, while new states are calculated in front of the sweep-line and processed states are deleted behind the sweep-line.

For non-monotone progress measures it is not safe to delete states with a progress value less than the minimal progress values among the unprocessed states. The reason is that a state with a low progress value could be reachable from a state with a high progress value. State space exploration with non-monotone progress measure can be done by conducting multiple sweeps. Each sweep is done in a similar way to the monotone case, with one exception: to

[1] A state is said to be unprocessed if all its successors have not yet been calculated.

```
1: $\textsc{Roots} \leftarrow \{\iota\}$ ; $\textsc{Nodes} \leftarrow \{\iota\}$ ; $\textsc{Persistent} \leftarrow \emptyset$
2: while $\textsc{Roots} \neq \emptyset$ do
3:   $\textsc{Unprocessed} \leftarrow \textsc{Roots}$ // mark all root states for this sweep unprocessed.
4:   $\textsc{Roots} \leftarrow \emptyset$
5:   while $\textsc{Unprocessed} \neq \emptyset$ do
6:     // select s minimal wrt. $\psi$ and $\sqsubseteq$ in $\textsc{Unprocessed}$
7:     select $s \in \textsc{Unprocessed}$ such that $\forall s' \in \textsc{Unprocessed} : \psi(s') \not\sqsubset \psi(s)$
8:     $\textsc{Unprocessed} \leftarrow \textsc{Unprocessed} - \{s\}$
9:     for all $(a, s')$ such that $s \xrightarrow{a} s'$ do
10:      if $s' \notin \textsc{Nodes}$ then
11:        $\textsc{Nodes} \leftarrow \textsc{Nodes} \cup \{s'\}$
12:        if $\psi(s) \not\sqsubseteq \psi(s')$ then
13:          $\textsc{Persistent} \leftarrow \textsc{Persistent} \cup \{s'\}$
14:          $\textsc{Roots} \leftarrow \textsc{Roots} \cup \{s'\}$ // make $s'$ root for the next sweep.
15:        else
16:          $\textsc{Unprocessed} \leftarrow \textsc{Unprocessed} \cup \{s'\}$
17:        end if
18:      end if
19:    end for
20:    // delete unreachable non-persistent states from $\textsc{Nodes}$
21:    $\textsc{Nodes} \leftarrow \{s \in \textsc{Nodes} \mid \exists s' \in \textsc{Unprocessed} : \psi(s') \sqsubseteq \psi(s)\} \cup \textsc{Persistent}$
22:  end while
23: end while
```

Fig. 2. The sweep-line algorithm. The algorithm performs a number of sweeps (lines 2-23). In each sweep regress-edges are identified (line 12), and the destination states of regress-edges are marked as persistent and are used as root states (lines 13-14) for the next sweep. Once a node has been marked as persistent, it is not deleted in line 21.

ensure that the state space exploration terminates and that all states have been visited at least once, *regress-edges* are recognised during the state space exploration. A regress-edge is an edge $s \xrightarrow{a} s'$ of the LTS such that $\psi(s) \not\sqsubseteq \psi(s')$. When a regress-edge is discovered, the destination state of the regress-edge is marked as *persistent* preventing it from being deleted. Newly discovered persistent states are used as *roots* in the subsequent sweep. The algorithm, derived from the standard algorithm for explicit state enumeration, is listed in Fig. 2. Correctness of the algorithm was established in [9].

Theorem 1 (Thm. 1 in [9]). *The sweep-line algorithm terminates after having explored at most* $(|B| + 1) \cdot |reach_{\mathcal{L}}(\iota)|$ *states, where* B *denotes the set of destination states of regress-edges. Upon termination, all states reachable from the initial state have been explored at least once.* □

3 Compositional Sweep-Line Exploration

The construction of a progress measure for the parallel composition of LTSs is based on the observation that given n partial orders $(O_i, \sqsubseteq_i)$ for $1 \leq i \leq n$, we can construct the *product partial* order $(\prod_{i=1}^{n} O_i, \sqsubseteq)$ where $(o_1, \ldots, o_n) \sqsubseteq (o'_1, \ldots, o'_n)$ iff for all $1 \leq i \leq n : o_i \sqsubseteq_i o'_i$. Based on this we define the product of n progress measures.

Definition 4. *The* ***product*** *of progress measures $\mathcal{P}_i = (O_i, \sqsubseteq_i, \psi_i)$ for $1 \leq i \leq n$ is defined as $\mathcal{P} = (O, \sqsubseteq, \psi)$ where $(O, \sqsubseteq)$ is the product partial order of $(O_i, \sqsubseteq_i)$ for $1 \leq i \leq n$ as defined above, and $\psi(s_1, \ldots, s_n) = (\psi_1(s_1), \ldots, \psi_n(s_n))$.* □

The fact that the product of progress measures can be used as a progress measure in conjunction with parallel composition is established by the following proposition. The proof of the proposition is rather straightforward and has been omitted here. It can be found in [8].

Proposition 1. *Let $\mathcal{L}_i$ for $1 \leq i \leq n$ be LTSs with progress measures $\mathcal{P}_i = (O_i, \sqsubseteq_i, \psi_i)$, respectively. The product $\mathcal{P} = (O, \sqsubseteq, \psi)$ of the $\mathcal{P}_i$'s as defined in Def. 4 is a progress measure on $\mathcal{L} = \mathcal{L}_1 \| \cdots \| \mathcal{L}_n$. If $\mathcal{P}_i$ is a monotone progress measure on $\mathcal{L}_i$ for $1 \leq i \leq n$, then $\mathcal{P}$ is a monotone progress measure on $\mathcal{L}$.* □

Since by Prop. 1, the product progress measure is a progress measure on the parallel composition, we can immediately apply the algorithm from Fig. 2. However, we present a more elaborate algorithm which exploits that the progress measure on some components might be monotone. This makes it possible to delete persistent states when the monotone progress of a component ensures that they are no longer reachable from the unprocessed states. Persistent states cannot be deleted with the algorithm from Fig. 2.

To explain how the improved algorithm operates, we first revisit the algorithm from Fig. 2. The progress mapping for the parallel composition $\mathcal{L} = \mathcal{L}_1 \| \cdots \| \mathcal{L}_n$ is a vector: $\psi = (\psi_1, \ldots, \psi_n)$. This n-dimensional progress mapping makes a disjoint partitioning of the states of $\mathcal{L}$, and positions the states into an n-dimensional grid. Figure 3 illustrates this for $n = 2$ and progress measures based on total orders. The square with coordinates (x, y) contains all states with progress value (x, y). For example, the states s_1, s_2, and s_3 are the states with progress value $(1, 2)$.

The progress measure $\mathcal{P}_1$, corresponding to the x-axis in Fig. 3, is monotone, whereas $\mathcal{P}_2$, corresponding to the y-axis, is non-monotone. That $\mathcal{P}_1$ is monotone can be seen from the fact that all edges where the first coordinate changes points to the right. That $\mathcal{P}_2$ is non-monotone can be seen from, e.g., the regress-edge from s_5 to s_6 where the value of the y-coordinate decreases. As it can be seen, the non-monotone progress measures cause multiple sweeps. In this case, it caused all states reachable from s_6 to be re-explored in the next sweep. Revisiting all states reachable from s_6 can be avoided by exploring the states columnwise according to $\mathcal{P}_1$ (the monotone progress measure) and by conducting multiple sweeps only within each column. Figure 4 depicts a scenario in which the states are explored according to the columns determined by the monotone progress measure $\mathcal{P}_1$.

To formulate the sweep-line algorithm for n-dimensions we introduce two partial orderings derived from the partial order on the product progress values. The role of the two orderings is to distinguish between progress originating from monotone and non-monotone progress measures.

Definition 5. *Let $\mathcal{P}_i = (O_i, \sqsubseteq_i, \psi_i)$ for $1 \leq i \leq n$ be progress measures, and let $\mathcal{P} = (O, \sqsubseteq, \psi)$ be the product progress measure. Let $\mathsf{m} = \{i \mid \mathcal{P}_i$ is monotone $\}$,*

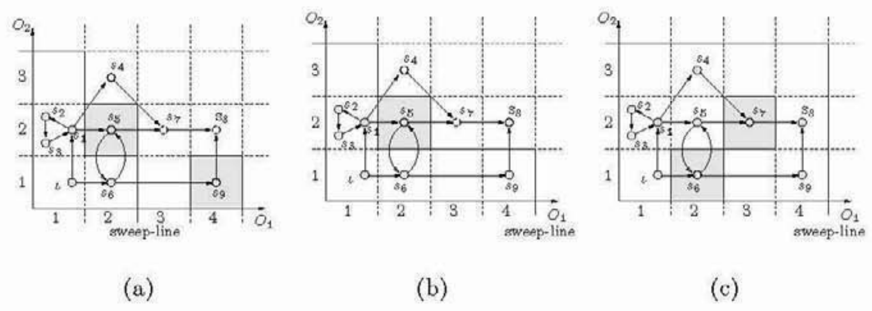


Fig. 3. Snapshots of state exploration. The sweep-line (indicated with thick lines) separates the squares (1, 1), (1, 2), (1, 3), and (2, 1) containing the explored (and now deleted) states (ι, s_1, s_2, s_3, s_6) from the squares containing some unprocessed states The squares (2, 2) and (4, 1) are the next squares that can be examined in the least-progress-first order. If (4, 1) is chosen to be examined, the fragment of the sweep-line corresponding to $y = 1$ can move to (4, 1) as shown in Fig. 3(b). If (2, 2) is chosen to be examined, the fragment of the sweep-line corresponding to $x = 2$ can move to (2, 2). Eventually, this sweep will terminate, giving the situation depicted in Fig. 3(c). During the sweep, the states s_6 and s_7 will have been marked as persistent states because of the regress-edges from s_4 to s_7 and s_5 to s_6, and they will be used as roots in the next sweep. The exploration of the state space will finish after this next sweep.

and $\overline{\mathsf{m}} = \{j \mid j \notin \mathsf{m}\}$. *The partial orders* $(O, \sqsubseteq_{\mathsf{m}})$ *and* $(O, \sqsubseteq_{\overline{\mathsf{m}}})$ *are defined by:*

$$(o_1, \ldots, o_n) \sqsubseteq_{\mathsf{m}} (o'_1, \ldots, o'_n) \Leftrightarrow \forall i \in \mathsf{m} : o_i \sqsubseteq_i o'_i$$
$$(o_1, \ldots, o_n) \sqsubseteq_{\overline{\mathsf{m}}} (o'_1, \ldots, o'_n) \Leftrightarrow \forall i \in \overline{\mathsf{m}} : o_i \sqsubseteq_i o'_i$$

□

In Fig. 3, the ordering on the monotone component determines the ordering on the columns, whereas the ordering on the non-monotone component determines the ordering within the columns. Figure 5 lists the compositional sweep-line algorithm. The algorithm consists of two procedures: COLUMNSWEEP and SWEEP. The procedure COLUMNSWEEP specifies the local sweep according to the non-monotone progress measures, i.e., the sweeps in the columns of Fig. 4. SWEEP specifies the global sweep based on the monotone progress measures.

4 Correctness

We now turn to the correctness of the compositional sweep-line algorithm, i.e., that the algorithm in Fig. 5 terminates for any LTS $\mathcal{L} = \mathcal{L}_1 \| \cdots \| \mathcal{L}_n$ and upon termination all reachable states of $\mathcal{L}$ have been visited at least once (full coverage). To do this, we first introduce some new notation: The monotone components of the product progress measure on $\mathcal{L}$ partitions the set of reachable states into equivalence classes (columns in terms of Fig. 4), where two states s and s'

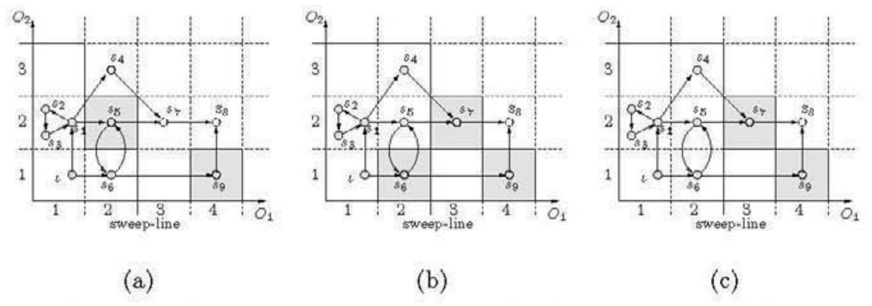


Fig. 4. Snapshot of improved state exploration. The situation in Fig. 4(a) is the same as the situation depicted in Fig. 3(a), but instead of exploring the square $(4, 1)$ we now explore the square $(2, 2)$ followed by exploring the square $(2, 3)$ in order to complete the exploration of the column $x = 2$ before moving on to the column $x = 3$. Figure 4(b) depicts the situation after the first sweep of the column $x = 2$. The state s_6 has been marked as persistent, and will be used as root in the subsequent sweep of the column. After the second sweep of column $x = 2$, we can move on to the next column as depicted in Fig. 4(c). The fact that the progress measure $\mathcal{P}_1$ determining the columns is monotone implies that no edges can point to the left, and hence it is safe to delete the states to the left of the sweep-line including the persistent state s_6. By exploring states columnwise, only the states s_5 and s_6 will be revisited as a consequence of the regress-edge from s_5 to s_6, and the monotonicity of ψ_1 makes it possible to delete s_6 before the next column is explored.

are equivalent iff $\psi(s) \sqsubseteq_{\mathsf{m}} \psi(s')$ and $\psi(s') \sqsubseteq_{\mathsf{m}} \psi(s)$, i.e., if the progress values on the monotone components are pairwise identical. We will use $\mathcal{C}_{\mathcal{L}}$ to denote the set of non-empty equivalence classes on $reach_{\mathcal{L}}(\iota)$. With this definition of equivalence classes, the ordering $\sqsubseteq_{\mathsf{m}}$ can be extended to equivalence classes by defining $C_1 \sqsubseteq_{\mathsf{m}} C_2$ for $C_1, C_2 \in \mathcal{C}_{\mathcal{L}}$ if and only if $\psi(s_1) \sqsubseteq_{\mathsf{m}} \psi(s_2)$ for $s_1 \in C_1$ and $s_2 \in C_2$.

Definition 6. *Let $C \in \mathcal{C}_{\mathcal{L}}$. The states in C with an incoming edge from a state outside C are denoted* $\mathsf{in}(C)$*:* $\mathsf{in}(C) = \{s \in C \mid s = \iota \vee \exists s' \notin C : s' \to s\}$*. The states outside C with an incoming edge from C are denoted* $\mathsf{out}(C)$*:* $\mathsf{out}(C) = \{s \notin C \mid \exists s' \in C : s' \to s\}$*.*

The set of equivalence classes strictly smaller than C is denoted $\mathsf{under}(C)$*:* $\mathsf{under}(C) = \{C' \in \mathcal{C}_{\mathcal{L}} \mid C' \sqsubset_{\mathsf{m}} C\}$*. The equivalence classes strictly larger than C are denoted* $\mathsf{over}(C)$*:* $\mathsf{over}(C) = \{C' \in \mathcal{C}_{\mathcal{L}} \mid C \sqsubset_{\mathsf{m}} C'\}$*. A set $X \subseteq \mathcal{C}_{\mathcal{L}}$ is **downwards closed** if $\forall C \in X : \mathsf{under}(C) \subseteq X$.* □

We have included ι in $\mathsf{in}(C)$ if C contains ι to ensure that $\mathsf{in}(C) \neq \emptyset$ for all $C \in \mathcal{C}_{\mathcal{L}}$. For an equivalence class $C \in \mathcal{C}_{\mathcal{L}}$, $B(C)$ denotes the set of destination states of regress-edges with respect to $\sqsubseteq_{\overline{\mathsf{m}}}$ between nodes in C. Formally: $B(C) = \{s' \in C \mid \exists s \in C : s \to s' \wedge \psi(s) \not\sqsubseteq_{\overline{\mathsf{m}}} \psi(s')\}$.

The first step in establishing correctness is the following lemma which gives sufficient conditions for termination and full coverage in the sweep of an equiv-

```
Global sets:
 1: GlobalUnprocessed ← {$\iota$}; Nodes ← {$\iota$}; Persistent ← $\emptyset$

procedure ColumnSweep ()
 1: while Roots $\neq \emptyset$ do
 2:   Unprocessed ← Roots // mark all roots for this columnsweep unprocessed.
 3:   Roots ← $\emptyset$
 4:   while Unprocessed $\neq \emptyset$ do
 5:     // select s minimal wrt. $\psi$ and $\sqsubseteq_{\overline{\mathsf{m}}}$ in Unprocessed
 6:     select $s$ such that $\forall s' \in$ Unprocessed : $\psi(s') \not\sqsubseteq_{\overline{\mathsf{m}}} \psi(s)$
 7:     Unprocessed ← Unprocessed $-$ {$s$}
 8:     for all $(a, s')$ such that $s \xrightarrow{a} s'$ do
 9:       if $s' \notin$ Nodes then
10:         Nodes ← Nodes $\cup$ {$s'$}
11:         if $\psi(s) \sqsubset_{\mathsf{m}} \psi(s')$ then
12:           // $s'$ is in a subsequent column.
13:           GlobalUnprocessed ← GlobalUnprocessed $\cup$ {$s$}
14:         else if $\psi(s) \not\sqsubseteq_{\overline{\mathsf{m}}} \psi(s')$ then
15:           Persistent ← Persistent $\cup$ {$s'$}
16:           Roots ← Roots $\cup$ {$s'$}
17:         else
18:           Unprocessed ← Unprocessed $\cup$ {$s'$}
19:         end if
20:       end if
21:     end for
22:     // delete unreachable non-persistent states in this column from Nodes
23:     Nodes ← {$s \in$ Nodes | $\exists s' \in$ Unprocessed : $\psi(s') \sqsubseteq_{\overline{\mathsf{m}}} \psi(s)$}
24:     Nodes ← Nodes $\cup$ Persistent $\cup$ GlobalUnprocessed
25:   end while
26: end while

procedure Sweep ()
 1: while GlobalUnprocessed $\neq \emptyset$ do
 2:   Roots ← $R$ where $R \subseteq$ GlobalUnprocessed satisfy: $\forall s, s' \in R : \psi(s) = \psi(s')$
      and $\forall s \in R, s' \in$ GlobalUnprocessed $- R : \psi(s') \not\sqsubseteq_{\mathsf{m}} \psi(s)$
 3:   GlobalUnprocessed ← GlobalUnprocessed $-$ Roots
 4:   ColumnSweep() // sweep the next column.
 5:   Nodes ← Nodes $-$ Persistent
 6:   Persistent ← $\emptyset$
 7: end while
```

Fig. 5. The compositional sweep-line algorithm. Procedure ColumnSweep is identical to the algorithm in Fig. 2 except for lines 11-14 where it is checked whether s' is higher in the monotone order. If this is the case, its investigation should be postponed to a later invocation of the procedure as it belongs to a subsequent column. The state is therefore inserted into the set GlobalUnprocessed which is shared between the two procedures. The procedure Sweep conducts a sweep according to the monotone progress measures where for each fixed progress value of the monotone components, a non-monotone sweep is conducted by invoking the ColumnSweep procedure. After each invocation of ColumnSweep, the persistent states are deleted from the set Nodes in line 6 of Sweep.

alence class. We say that a state is being explored whenever its successor states are being calculated.

Lemma 1. *Let $C \in \mathcal{C}_\mathcal{L}$ and let $R \subseteq C$ be a set of states. If $\mathsf{in}(C) \subseteq R$, then invoking* ColumnSweep *on R explores all states in C, adds $\mathsf{out}(C)$ to* GlobalUnprocessed*, and explores at most $(|B(C)|+1) \cdot |C|$ states.*

Proof. The lemma follows by a straightforward generalisation of the proof of Thm. 1 given in [9], and the observation that all states in C are reachable from $\mathsf{in}(C)$ using only intermediate states in C. □

Definition 7. *Let $X \subseteq \mathcal{C}_\mathcal{L}$. The notation for* out *and* over *in Def. 6 is extended to sets of equivalence classes by:*

- $\mathsf{out}(X) = \{\iota\}$ *if* $X = \emptyset$, *and* $\mathsf{out}(X) = \bigcup_{C\in X} \mathsf{out}(C) - \bigcup_{C \in X} C$ *otherwise.*
- $\mathsf{over}(X) = \mathcal{C}_\mathcal{L}$ *if* $X = \emptyset$, *and* $\bigcup_{C \in X} \mathsf{over}(C) - X$ *otherwise.* □

For a non-empty set $X \subseteq \mathcal{C}_\mathcal{L}$ we will say that $C \in X$ is minimal in X if there exists no $C' \in X$ such that $C' \sqsubset_\mathsf{m} C$. Similarly, we will say that $s \in C$ for some $C \in \mathcal{C}_\mathcal{L}$ is minimal in C if there exists no $s' \in C$ such that $\psi(s') \sqsubset_{\overline{\mathsf{m}}} \psi(s)$. The correctness of the compositional sweep-line algorithm in Fig. 5 follows from Thm. 2 below which uses the following proposition. Its proof can be found in [8].

Proposition 2. *Let $C \in \mathcal{C}_\mathcal{L}$ and let $X \subseteq \mathcal{C}_\mathcal{L}$ be downwards closed. Then:*

1. *If C is minimal in $\mathsf{over}(X)$, then $X \cup \{C\}$ is downwards closed.*
2. *If $s \in C$ is minimal in $\mathsf{out}(X)$, then C is minimal in $\mathsf{over}(X)$.*
3. *If C is minimal in $\mathsf{over}(X)$, then $\mathsf{in}(C) \subseteq \mathsf{out}(X)$.* □

Theorem 2. *The sweep-line algorithm in Fig. 5 terminates after having explored at most $\sum_{C \in \mathcal{C}_\mathcal{L}} (|B(C)|+1) \cdot |C|$ states. Upon termination all states reachable from the initial state have been explored at least once.*

Proof. Let $E(i)$ denote the equivalence classes completely explored after i iterations of the loop in lines 1-7 of Sweep. The following loop invariant holds:

$E(i)$ is downwards closed, GlobalUnprocessed $= \mathsf{out}(E(i))$, *and* $|E(i)| = i$.

Before the first iterations ($i = 0$), $E(0) = \emptyset$ and hence is trivially downwards closed, $\mathsf{out}(E(0)) = \{\iota\}$ by the definition of out, and $|E(0)| = 0$. Assume that the invariant is valid before the $i+1$'th iteration, and let R be the set of states chosen by at line 2. By the definition, $R \subseteq C$ for some equivalence class C, and the states in R are minimal with respect to $\sqsubseteq_\mathsf{m}$ in GlobalUnprocessed. By Prop. 2(2-3) (with $X = E(i)$), $\mathsf{in}(C) \subseteq \mathsf{out}(E(i)) =$ GlobalUnprocessed. Since line 2 of Sweep ensures that R contains all states in GlobalUnprocessed which are in C, $\mathsf{in}(C) = R$. Hence by Lemma 1 the call to ColumnSweep in line 4 explores all of C. Since C is now explored, $E(i+1) = E(i) \cup \{C\}$. Since C is minimal in $\mathsf{over}(E(i))$ by Prop. 2(2), $E(i+1)$ is downwards closed by Prop. 2(1). Since $C \in \mathsf{over}(E(i))$, we have $C \notin E(i)$. Hence $|E(i+1)| = |E(i)| + 1 =$

$i{+}1$. Since R is removed from GlobalUnprocessed, and ColumnSweep adds $\mathsf{out}(C)$ to GlobalUnprocessed by Lemma 1, we have GlobalUnprocessed $= \mathsf{out}(E(i)) - R \ \cup\ \mathsf{out}(C)$. The monotonicity of $\sqsubseteq_{\mathsf{m}}$ and $C \notin E(i)$ ensures that $\bigcup_{C' \in E(i)} C' \ \cap\ \mathsf{out}(C) = \emptyset$, and hence that $\mathsf{out}(E(i)) - R \ \cup\ \mathsf{out}(C) = \mathsf{out}(E(i+1))$.

The number of equivalence classes are finite, and since the size of $E(i)$ grows by one in each iteration of the loop in lines 1-7 of Sweep we eventually have: $E(i) = \mathcal{C}_{\mathcal{L}}$. In this case GlobalUnprocessed $= \mathsf{out}(E(i)) = \emptyset$ by the definition of out, and the loop will eventually terminate with all equivalence classes having been explored. Each iteration of the loop explores a new equivalence C using ColumnSweep. The number of states explored when exploring the equivalence class C is bounded by $(|B(C)|+1) \cdot |C|$ by Lemma 1. The total number of states explored by Sweep is hence bounded by $\sum_{C \in \mathcal{C}_{\mathcal{L}}} (|B(C)|+1) \cdot |C|$. □

5 Computing Component Progress Measures

This section presents our initial approaches for automatically computing progress measures for the components LTSs. We present two algorithms: one for obtaining a monotone progress measure based on strongly connected components, and one for obtaining a non-monotone progress measure based on spanning trees. Both algorithms have a time and space complexity which is linear in the size of the component LTS for which the progress measure is being computed.

Strongly Connected Components. A monotone progress measure for a component LTS can be obtained by viewing the LTS as a directed graph and computing the *strongly connected component* (SCC) graph using e.g, Tarjan's algorithm [15]. The associated progress mapping ψ_{scc} maps each state s into the SCC $\mathsf{scc}(s)$ to which it belongs. The partial order $\sqsubseteq_{\mathsf{scc}}$ on progress values is determined by the reachability relation on the SCC graph, i.e., $\psi_{\mathsf{scc}}(s) \sqsubseteq_{\mathsf{scc}} \psi_{\mathsf{scc}}(s')$ if and only if $s' \in \mathit{reach}_{\mathcal{L}}(s)$. The relation $\sqsubseteq_{\mathsf{scc}}$ is a partial order since the SCC graph is a directed acyclic graph. The progress measure is monotone since ψ_{scc} preserves reachability.

The progress measure obtained using SCCs is optimal for monotone progress measures in the sense that all states in the same SCC must have the same progress value as a consequence of the definition of monotone progress measures. Hence, the progress measures based on SCC give the finest partitioning of the states of all monotone progress measures. However, the SCC based progress measure gives only the trivial progress measure if the component LTS itself is strongly connected, i.e., all its states belong to the same SCC.

Spanning Tree. The SCC based algorithm for monotone progress measure above is highly dependent on the component LTSs having many small SCCs. If too large SCCs exist, a non-monotone progress measure might give a better reduction of peak memory usage. A non-monotone progress measure can be obtained by computing a *spanning tree* for the component LTS using e.g., a depth-first search.

The reachability relation on the spanning tree determines the partial ordering in a similar way as for the SCC algorithm. A topological ordering of the nodes in the spanning tree gives the non-monotone progress measure consistent with the reachability relation on the spanning tree. Each edge not in the spanning tree will be treated as a regress-edge.

6 Experimental Results

A prototype [11] has been developed to conduct some initial experiments with the compositional sweep-line algorithm. This prototype consists of three programs. A program pslc for compiling process specifications into labelled transitions systems, a program progress implementing the two algorithms in Sect. 5 for computing progress measures for subcomponents, and a program sweepcheck implementing the algorithm in Fig. 5. The sweepcheck program implements the CSP [5] parallel composition operator. The deletion of states is handled by keeping track of which column each state belongs to. When the next unprocessed state selected has a progress value which is larger (wrt. to the non-monotone components) than the previous unprocessed state selected, all non-persistent states in the current column are deleted.

We present experimental results obtained with this prototype on three smaller examples, and we compare the use of the sweep-line method with the use of ordinary full state space exploration, and with the use of the state space caching method [7]. The three examples are available from [11]. The experiments were conducted on a 1 Ghz Linux PC with 1 Gb of RAM.

Master/Slave System. This example consists of a master process and a number of slave processes. The master process initially manages a set of jobs that it assigns to the slaves. Since the master process progresses with each job assigned, the SCC algorithm has been used to compute a monotone progress measure for the master. The slaves are all reactive, i.e., they return to their idle state after having completed a job. Hence they only have a single SCC, and the spanning tree algorithm was used to compute a non-monotone progress measure.

The performance of full state space exploration of the master/slave system is shown to the left in Table 1 for different number of jobs processed by 12 slaves. The reduction obtained with the sweep-line method is shown to the right in the table. The Total column shows the number of states processed by the sweep-line method relative to the total number of reachable states. The Peak column shows relative memory usage of the sweep-line method compared to the full state space, i.e., the peak number of states stored divided by the total number of states in the full exploration. As seen from the Time columns, the reduction comes at a minor runtime penalty. This is due to the larger number of states processed and the overhead in deletion of states.

Two-Phase Commit Protocol. The two-phase commit protocol consists of a single initiator process and a number of voter processes. The initiator process initiates an election on whether to commit or abort a transaction. The voters then decides

Table 1. The master/slave system with 12 slaves.

	Full State Space		Sweep-Line		
Config.	States	Time	Total	Peak	Time
20 jobs	126,976	00:00:14	200%	6.45%	00:00:24
24 jobs	159,744	00:00:18	200%	5.13%	00:00:30
28 jobs	192,512	00:00:23	200%	4.26%	00:00:36
34 jobs	241,664	00:00:29	200%	3.39%	00:00:46
40 jobs	290,816	00:00:36	200%	2.82%	00:00:55

locally whether to vote for commit or abort, and the initiator collects the votes. In this example, both the initiator and the voters have non-trivial SCCs. The SCC based algorithm has therefore been used on all subcomponents to compute the progress measures. Table 2 compares full state spaces exploration to the sweep-line method for different configurations. The independence of voters, together with the broadcast nature of the communication from initiator to voters, causes the state space to grow in width more than in depth as the number of voters increases. Consequently, the relative reduction obtained with the sweep-line method becomes smaller as the number of voters grows.

Table 2. The two-phase commit protocol.

	Full State Space		Sweep-Line		
Config.	States	Time	Total	Peak	Time
6 slaves	18,478	00:00:01	100%	21.16%	00:00:02
7 slaves	87,475	00:00:08	100%	22.33%	00:00:10
8 slaves	420,988	00:01:08	100%	23.19%	00:01:20
9 slaves	2,051,029	00:15:46	100%	23.81%	00:20:46

Stop-and-Wait Protocol. This example is a stop-and-wait data-link protocol, where a sender process sends packets attached with sequence numbers over an unreliable network to a receiver process. All processes in this system are cyclic, and the spanning tree algorithm was used to obtain non-monotone progress measures. Table 3 compares of the sweep-line method with full state space exploration for different values of the maximal sequence number.

State Space Caching. The state space caching method resembles the sweep-line method in that it deletes states on-the-fly in order to reduce peak memory usage. State space caching conducts a depth-first exploration of the state space, and allows states not on the depth-first stack to be deleted. To compare the two methods, we have implemented the state space caching method in our prototype tool, and tried it on the three examples above for different cache sizes. The time/space trade-off for different systems, configurations, and different choices

Table 3. The stop-and-wait protocol.

	Full State Space		Sweep-Line		
Config.	States	Time	Total	Peak	Time
20	3,640	00:00:01	103.84%	61.73%	00:00:01
30	5,460	00:00:01	103.84%	61.48%	00:00:01
40	7,280	00:00:02	103.84%	61.36%	00:00:02
50	9,100	00:00:04	103.84%	61.29%	00:00:04

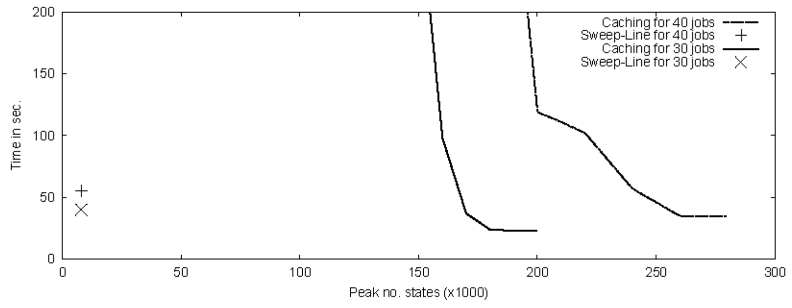
of cache size is plotted in Fig. 6. For all three examples, the state space caching methods runs in time comparable to ordinary full state space exploration for large cache sizes. Its performance, however, quickly degenerates, when the size of the state cache is reduced.

The performance degradation of state space caching has been observed before, and partial-order reduction has been used to alleviate this problem [3]. We have not added partial-order reduction to our prototype. Both the sweep-line method and state space caching will, however, benefit from the use of partial-order methods. We leave it for future work to investigate the relative impact of partial-order methods on the sweep-line method and the state space caching method. Compared to pure state space caching, the sweep-line method appears to provide a better time/space trade-off.

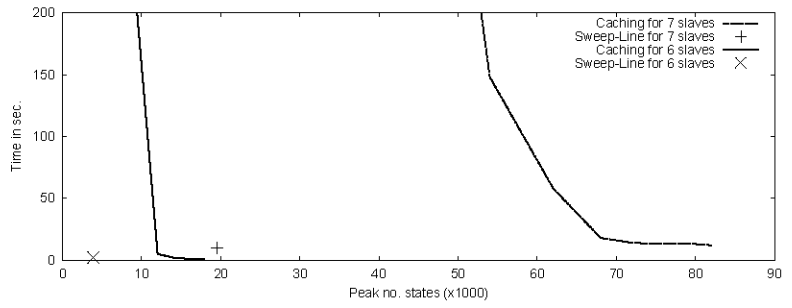
7 Conclusion and Future Work

We have presented a sweep-line algorithm applicable in a compositional framework where the components of the system are represented as labelled transition systems (LTSs). The key idea was to automatically compute progress measures for the component LTSs and compose these to obtain a progress measure for the full system. In addition, the developed sweep-line algorithm exploits that some components have monotone progress measures and some components have non-monotone progress measures. We have given algorithms based on strongly connected components and spanning trees for computing monotone and non-monotone progress measures for the component LTSs. Better progress measures can possibly be obtained by combining the two using spanning trees to split large strongly connected components.

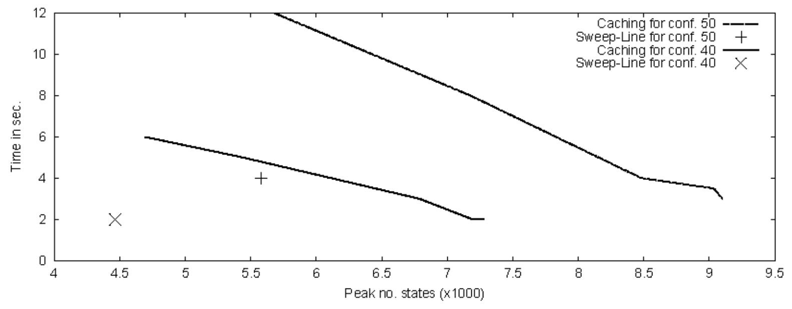
We have assumed that the system considered was a parallel composition of components represented as LTSs. It is, however, only the computation of progress measures for the components of the system which relies on the representation of the components in the form of LTSs. The compositional sweep-line algorithm itself can be applied independently of the origin of the progress measure for the components. This means that the algorithm is applicable for any system $\mathcal{S} = \mathcal{S}_1 \| \cdots \| \mathcal{S}_n$ which is a parallel composition of subsystems $\mathcal{S}_i$, and where the states of the full system $\mathcal{S}$ is a vector of states with an entry for each subsystem. This means that our results are applicable to many modelling and specification languages for concurrent systems, such as for instance Promela used in the



(a) Master/Slave system.



(b) Two-phase commit protocol.



(c) Stop-and-wait protocol.

Fig. 6. Time/space trade-off for state space caching and the sweep-line methods. The graphs shows, for different systems and configurations, the time and space usage of the sweep-line method and the state space caching method for different cache sizes. With smaller cache sizes the peak memory usage is smaller than for larger cache sizes, but at a significant runtime penalty. The sweep-line method reduces memory usage at a much lower runtime penalty.

SPIN tool [6]. Processes (subsystems) in PROMELA are specified in a C like language. In this case it seems possible to compute progress measures based on, e.g., the control flow in the individual processes. In other formalisms such as *timed automata networks* as used in, e.g., the UPPAAL tool [10], a progress measure for the individual timed automata could possibly be computed based on the locations of the automata. Details of computing progress measure for processes in various modelling formalisms are left for future work.

The sweep-line algorithm as presented is aimed at checking reachability properties as it explores all reachable states of the system at least once. It can however be observed that if all progress measures of the components are monotone progress measure, then all states of a cycle in the state space will have the same progress value, and hence all states in a cycle will reside in memory simultaneously at some point. This can be exploited for conducting Linear Temporal Logic (LTL) model checking, which can formulated as searching for cycles in the composition of two Büchi automata [18]. This means that the compositional approach presented in this paper can be used to develop an LTL model checking algorithm capable of exploiting the graph structure of the Büchi automata expressing the LTL property to be checked. Details of this is left for future work.

Future work also involves techniques for generation of error traces with the sweep-line method. A trace leading from, e.g., the initial state to a state satisfying a given predicate cannot immediately be obtained with the current version of the sweep-line method due to the deletion of states.

Acknowledgements. The authors would like to thank Michael Westergaard for assisting in the implementation of the prototype. Jens Bæk Jørgensen, Lisa Wells, and the anonymous reviewers for constructive comments on this paper.

References

1. S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proc. of TACAS'01*, volume 2031 of *LNCS*, pages 450–464. Springer-Verlag, 2001.
2. E. M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Logic Model Checking. In *Proc. of CAV'93*, pages 450–462. Springer-Verlag, 1993.
3. P. Godefroid, G.J. Holzman, and D. Pirottin. State Space Caching Revisited. In *Proc. of CAV'92*, volume 663 of *LNCS*, pages 178–191. Springer-Verlag, 1992.
4. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In *Proc. of ICATPN'02*, volume 2360 of *LNCS*, pages 182–202. Springer-Verlag, 2002.
5. C.A.R Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
6. G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.
7. C. Jard and T. Jeron. Bounded-memory Algorithms for Verification On-the-fly. In *Proc. of CAV'91*, volume 575 of *LNCS*, pages 192–202. Springer-Verlag, 1991.
8. L.M. Kristensen and T. Mailund. A Compositional Sweep-Line State Space Exploration Method. Technical report, Department of Computer Science, University of Aarhus, 2002. Available via: `www.daimi.au.dk/~mailund/ps/lmktm_compo.ps`.

9. L.M. Kristensen and T. Mailund. A Generalised Sweep-Line Method for Safety Properties. In *Proc. of FME'02*, volume 2391 of *LNCS*, pages 549–567. Springer-Verlag, 2002.
10. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1+2):134–152, October 1997.
11. T. Mailund. SweepChecker. `http://sweepchecker.sourceforge.net`.
12. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
13. R. Milner. *Communication and Concurrency*. Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989.
14. D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proc. of CAV'93*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.
15. R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.
16. A. Valmari. Stubborn Sets for Reduced State Space Generation. In *Advances in Petri Nets '90*, volume 483 of *LNCS*, pages 491–515. Springer-Verlag, 1990.
17. A. Valmari. Compositionality in State Space Verification Methods. In *Proc. of ICATPN'96*, volume 1091 of *LNCS*, pages 29–56. Springer-Verlag, 1996.
18. M. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. of IEEE Symposium on Logic in Computer Science*, pages 322–331, 1986.
19. P. Wolper and P. Godefroid. Partial Order Methods for Temporal Verification. In *Proc. of CONCUR'93*, volume 715 of *LNCS*, pages 233–246. Springer-Verlag, 1993.