To Compose, Or Not to Compose, That is the Question: An Analysis of Compositional State Space Generation

Sander de Putter https://orcid.org/0000-0002-6819-4011[⋆] and Anton Wijs https://orcid.org/0000-0002-2071-9624

Eindhoven University of Technology
PO Box 513, 5600 MB, Eindhoven, The Netherlands
s.m.j.d.putter@tue.nl, a.j.wijs@tue.nl

Abstract. To combat state space explosion several compositional verification approaches have been proposed. One such approach is compositional aggregation, where a given system consisting of a number of parallel components is iteratively composed and minimised. Compositional aggregation has shown to perform better (in the size of the largest state space in memory at one time) than classical monolithic composition in a number of cases. However, there are also cases in which compositional aggregation performs much worse.
It is unclear when one should apply compositional aggregation in favor of other techniques and how it is affected by action hiding and the scale of the model.
This paper presents a descriptive analysis following the quantitiative experimental approach. The experiments were conducted in a controlled test bed setup in a computer laboratory environment. A total of eight scalable models with different network topologies considering a number of varying properties were investigated comprising 119 subjects. This makes it the most comprehensive study done so far on the topic. We investigate whether there is any systematic difference in the success of compositional aggregation based on the model, scaling, and action hiding. Our results indicate that both scaling up the model and hiding more behaviour has a positive influence on compositional aggregation.

1 Introduction

Although model checking [5] is one of the most successful approaches for the analysis and verification of the behaviour of concurrent systems, it is plagued with the so-called *state space explosion problem*: the state space of a concurrent system tends to increase exponentially as the number of parallel processes increases linearly.

[⋆] This work is supported by ARTEMIS Joint Undertaking project EMC2 (grant nr. 621429).

To combat state space explosion several compositional approaches have been proposed such as assume-guarantee reasoning [19,29] and partial model checking [2]. An evaluation of assume-guarantee reasoning was recently conducted [7]. The study raises doubt whether it is an effective alternative to classical, monolithic model checking.

A prominent alternative approach is compositional aggregation [9,10] (also known as compositional state space generation [36], incremental composition and reduction [33], incremental reachability analysis [34,35], and inductive compression [32]). Given a system consisting of a number of parallel components the *compositional aggregation* approach iteratively composes the components and minimises the result. *Action abstraction* or *hiding* [24] may be applied to abstract away all actions irrelevant for the property being verified such that minimisation is more effective. The idea of compositional aggregation is that incremental minimisation should warrant a lower maximum memory use than composing the system monolithicly. Compositional aggregation has shown to perform better (in the size of the largest state space in memory) than monolithic composition in a number of cases [8,9,11,30,34]. However, sometimes the former is not effective, even producing a (much) larger state space than the monolithic approach [11].

The aggregation order of a composition can be understood as a tree, where leaves are the parallel components and the nodes represent an operation that constructs a composite Labelled Transition System (LTS) from the children nodes and minimises the result. As such the number of possible aggregation orders is exponential in the number of parallel components. The selection of an efficient order, i.e., that results in compositional aggregation being as memory efficient as possible is still an unsolved issue [9].

To automate the selection of the aggregation order several heuristics have been proposed [9,8,34]. However, it is unpredictable whether aggregation orders selected by the heuristics are an improvement over the monolithic approach. Insights in the conditions in which compositional aggregation is expected to perform well are vital for successful application of the techniques, but these insights are currently limited. Evaluation of compositional aggregation and heuristics is, to the best of our knowledge, only limited to small benchmarks with no control on aggregation order, model scale, and action hiding. To gain understanding on how these variables influence the effectiveness of compositional aggregation, this paper presents a *characterisation* of the compositional aggregation method. The *objective* of this study is as follows:

> Analyse *compositional aggregation* for the purpose of *characterisation* of the *maximum memory use of the generated state space* in the context of *aggregation orderings of concurrent models with different scaling and action hiding.*

The goal is to find guidelines that help deciding whether to apply compositional aggregation. To this end we address the following main research question.

> **RQ main:** *When can compositional aggregation be expected to be more (memory) efficient than monolithic minimisation?*

To answer this question we first answer a number of smaller questions. First, we investigate the effect of three specific aspects of the application of compositional aggregation: the aggregation order, the amount of action hiding, and the number of parallel processes in the model that compositional aggregation is applied on.

> **RQ 1:** *How do action hiding, number of parallel processes, and aggregation order affect the memory consumption of compositional aggregation?*

As stated earlier, some aggregation orders are better than others. Heuristics are employed in an attempt to find the well performing aggregation orders. Therefore, to determine whether or not it is wise to apply compositional aggregation the performance of the heuristics must be kept in mind.

> **RQ 2:** *How effective are the aggregation orders chosen by current heuristics?*

Having established what minimisation approach is most efficient on which variants of the models, we finally investigate the relation between subjects within these two groups (compositional aggregation and monolithic minimisation). Answering this research question provides insights into which structural properties of models are indicative for the success or failure of compositional aggregation.

> **RQ 3:** *How can the success or failure of compositional aggregation be explained?*

In terms of scaling, due to the exponential growth of aggregation orders, we limit the number of analysed aggregation orders to 2,647, this is precisely the number of aggregation orders for a model consisting of six parallel components, hence, an optimum can be found for subjects up to a scale of six. The action hiding sets are derived from properties formulated for the corresponding models using the *maximal hiding* technique [24]. Finally, for minimisation we use branching bisimulation with explicit divergence [15] as it supports a broad range of safety and liveness properties.

Contributions. We present our findings after having conducted a thorough experiment to study the effectiveness of compositional aggregation when applied on models with varying network topologies. Having exhaustively analysed a significant number of possible aggregation orders, we are able to compare several heuristics proposed in the literature with the optimal composition results. In total, we have selected 119 subjects for the analysis, making this the most comprehensive study performed on the topic so far. Our main conclusion is that the amount of internal behaviour of individual processes in the model, and the amount of synchronisation between those processes, seem to be the two main factors influencing the success of compositional aggregation. Furthermore, our results suggest that there is real potential to construct better heuristics in the near future.

Note that the study was conducted on networks of LTSs and, therefore, the results are possibly limited to models represented as networks of LTSs.

Structure of the paper. In Section 2, we discuss related work. Preliminaries are given in Section 3. The methodology used in our experiment is discussed in Section 4. Section 5 presents our results, and finally, conclusions and future work are discussed in Section 6.

2 Related Work

Compositional aggregation. In the past, compositional aggregation has been applied in a number of experiments [35,8,11]. In [35], the experiments do not involve the optimal aggregation order for each considered case, and they target a set of models mostly consisting of randomly generated models and variations of only one or two real use cases. Not involving the optimal order means that it is impossible to indicate the quality of the considered heuristics, i.e., how well they perform compared to how well they *could* potentially perform. The usefulness of insights gained by analysing randomly generated models heavily depends on how similar the models are to real models, in terms of their structural characteristics.

In [8], two of the three heuristics proposed in [35] are further developed and combined into what the authors call *smart reduction*. They consider a benchmark set of 28 models that are variants of 13 models. This is a relatively high number of subjects, but unfortunately, discussion of the results is very limited, and the differences between subjects based on the same model are not explained. Due to this, the effect of these differences between the subjects cannot be correlated to the presented performance.

In [11] the combined heuristic is subjected to another experiment to show the effect of action hiding, i.e., abstraction of behaviour irrelevant for the considered functional property. The experiment measures the largest number of states generated during aggregation with and without action hiding. The experiment considers 90 subjects; a single (industrial) use case consisting of 5 scenarios, each considering a subset of 25 properties. They report that action hiding improves the performance of the heuristic. It is not reported whether there is a correlation between the amount of reduction and the properties.

Other compositional approaches. A method for automatically generating context constraints for compositional aggregation methods is proposed in [6]. It consists of generating an interface LTS representing the communicating behaviour of a set of components, and then composing this interface with the remainder of the components. The resulting state space is observably equivalent to the monolithicly generated state space. To evaluate the approach the authors perform several experiments with client/server models that are scaled by adding clients to the model. In each experiment the aggregation order was fixed. In contrast, we both scale the models and vary the aggregation orders to see how they affect the effectiveness of the technique.

An evaluation of automated assume-guarantee reasoning was conducted in [7]. The authors study whether assume-guarantee reasoning provides an advantage over monolithic verification. They conclude by raising doubts whether assume-guarantee reasoning is an effective compositional verification approach. However, no attempts were made to investigate the effects of combining multiple components in one step, i.e., n-way decomposition, and action hiding. Assume-guarantee reasoning may be more effective when these approaches are involved.

Assume-guarantee reasoning by abstraction refinement [14] improves upon the approach. The technique is inspired by the experience that small interfaces between components positively affect compositional reasoning. The study considers four cases with a total of twelve subjects. The improved approach uses less memory than the original one in seven of the twelve subjects. However, it is not reported how the memory consumption is measured (i.e., of what the memory consumption is measured exactly), and furthermore, the results are not compared to monolithic verification.

An n-way decomposition with alphabet refinement is proposed in [1]. A benchmark consisting of three cases with a total of fifteen subjects is performed, but memory consumption is not reported. In eight of the fifteen subjects, the approach turned out to be faster than monolithic verification.

Other contributions to assume-guarantee reasoning [16,26] present similarly small benchmarks with the number of cases not exceeding four and the number of subjects not exceeding seventeen. In [16] memory consumption is reported as the number of states in an assumption LTS, however, no correlation with actual memory consumption is discussed. In [26] the memory consumption of the tools used is reported. Still, all these benchmarks suffer from the problem of repeated measures.

Concluding, compared to our study, none of the related studies consider (non-random) models of varying network topologies, and take those topologies explicitly into account. We also study in detail the effect of action hiding. Furthermore, in none of the studies the results are corrected for repeated measures, which occur when you obtain results from variations of test cases. Finally, it should be noted that most studies consider to few cases and subjects to extract general conclusions.

3 Background

Our experiments are performed using the Cadp toolbox [11]. In this section, we explain the computational model behind compositional aggregation as offered by Cadp.

Vectors. A vector $\bar{v}$ of size n contains n elements indexed from 1 to n. We write $1..n$ for the set of integers ranging from 1 to n. For all $i \in 1..n$, $\bar{v}_i$ represents the i^{th} element of $\bar{v}$. Given a vector of indices $I \subseteq 1..n$, the *projection* of a vector $\bar{v}$ on to I is defined as the vector $\bar{v}^I = \langle \bar{v}_{I_1}, \ldots, \bar{v}_{I_{|I|}} \rangle$ of length $|I|$.

Labelled Transition System (LTS). The semantics of a process, or a composition of several processes, can be formally expressed by an LTS as presented in Definition 1.

Definition 1 (Labelled Transition System). *An* LTS $\mathcal{G}$ *is a tuple* $(\mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G}, \mathcal{I}_\mathcal{G})$*, with*

– $\mathcal{S}_\mathcal{G}$ *a finite set of states;*
– $\mathcal{A}_\mathcal{G}$ *a set of action labels;*
– $\mathcal{T}_\mathcal{G} \subseteq \mathcal{S}_\mathcal{G} \times \mathcal{A}_\mathcal{G} \times \mathcal{S}_\mathcal{G}$ *a transition relation;*
– $\mathcal{I}_\mathcal{G} \subseteq \mathcal{S}_\mathcal{G}$ *a (non-empty) set of initial states.*

Internal, or hidden, system steps are represented by the special action label $\tau \in \mathcal{A}_\mathcal{G}$. A transition $(s, a, s') \in \mathcal{T}_\mathcal{G}$, or $s \xrightarrow{a}_\mathcal{G} s'$ for short, denotes that LTS $\mathcal{G}$ can move from state s to state s' by performing the a-action. A sequence consisting of at least one τ-transition is denoted by $\xrightarrow{\tau}^+_\mathcal{G}$.

An equivalence relation between two LTSs relates states that have equivalent behaviour. We use *divergence-preserving branching bisimulation*, also called *branching bisimulation with explicit divergence* [15]. It supports action hiding and preserves both safety and liveness properties, due to the fact that it is sensitive to cycles of τ-transitions, i.e., inifinite internal behaviour. The smallest infinite ordinal is denoted by ω.

Definition 2 (Divergence-Preserving Branching bisimulation). *A binary relation B between two LTSs $\mathcal{G}_1$ and $\mathcal{G}_2$ is a* divergence-preserving branching bisimulation *iff it is symmetric and for all $s \in \mathcal{S}_{\mathcal{G}_1}$ and $t \in \mathcal{S}_{\mathcal{G}_2}$, $s\ B\ t$ implies:*
1. *if $s \xrightarrow{a}_{\mathcal{G}_1} s'$ then*
   (a) either $a = \tau$ with $s'\ B\ t$;
   (b) or $t \xrightarrow{\tau}^*_{\mathcal{G}_2} \hat{t} \xrightarrow{a}_{\mathcal{G}_2} t'$ with $s\ B\ \hat{t}$ and $s'\ B\ t'$.
2. *if there is an infinite sequence of states $(s^k)_{k\in\omega}$ such that $s = s^0$, $s^k \xrightarrow{\tau}_{\mathcal{G}_1} s^{k+1}$ and $s^k\ B\ t$ for all $k \in \omega$, then there exists a state t' such that $t \xrightarrow{\tau}^+_{\mathcal{G}_2} t'$ and $s^k\ B\ t'$ for some $k \in \omega$.*

The *minimisation* of an LTS consists of the merging of all states that are related by a divergence-preserving branching bisimulation relation. To maximise the potential for minimisation, *maximal hiding* [24] can be applied, which identifies exactly which actions are essential to correctly determine whether an LTS satisfies a given functional property or not. This roughly corresponds to hiding all actions except those occurring in the formula. For this to work correctly, the property needs to be specified in a fragment of the modal μ-calculus, which is expressive enough to express most properties. When combined with compositional model checking, actions of one process that require synchronisation with those of another cannot be abstracted away prematurely.

LTS Network. An *LTS network* $\mathcal{M}$ is a tuple $(\Pi, \mathcal{V})$, with Π a vector of *process* LTSs and $\mathcal{V}$ a set of synchronisation laws that define by means of vectors of actions which actions of the corresponding LTSs can synchronise with each other. For example, a law $(\langle a, b\rangle, c)$ defines that in a network consisting of two LTSs, a transition labelled a of LTS Π_1 can synchronise with a b-transition of LTS Π_2, resulting in a c-transition in the resulting LTS, called the *system LTS*. This system LTS is the result of first combining the initial states of the individual process LTSs into state vectors, together defining the set of initial states, and then repeatedly combining process transitions according to the laws, and combining the target states of those transitions into vectors of process LTS states. This LTS can by obtained through monolithic state space construction.

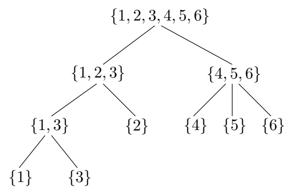


Fig. 1. Aggregation order; leaves are minimised first, then the tree is aggregated following an post-order walk of the tree

In line with the notion of projection for vectors, an LTS network $\mathcal{M}$ can be projected onto a vector of indices, by projecting both Π and $\mathcal{V}$ onto I. The result is an LTS network that can be considered as a subsystem or component of $\mathcal{M}$ consisting of the processes originally indexed in Π at the positions indicated by I.

Minimisation of processes in an LTS network (such as in compositional aggregation) is possible if the used equivalence relation is a congruence for LTS networks. Branching bisimulation, branching bisimulation with explicit divergence, observational equivalence, safety equivalence and weak trace equivalence, are congruences for *admissible* LTS networks [11,30]. An LTS network is called *admissible* if the synchronisation laws of the network do not synchronise, rename, or cut τ-transitions [22]. The intuition behind this is that internal behaviour, i.e., τ-transitions. should not be restricted by any operation.

Compositional order. The compositional aggregation of an LTS network $\mathcal{M} = (\Pi, \mathcal{V})$ is the incremental composition and minimisation of subsets of processes in Π. More specifically, the composition of a set of LTSs followed by a minimisation of the result is called an *aggregation.* The *compositional aggregation* modulo R of an LTS network $\mathcal{M}$ is the incremental aggregation of the processes in Π subject to $\mathcal{V}$ such that the result LTS is R-equivalent to the system LTS. Before we formally define compositional aggregation, we must first introduce aggregation orders.

The *aggregation order* organises the processes of an LTS network in a tree-structure as presented in Definition 3. The leaves represent the individual process LTSs in Π, and the nodes represent subsets of Π. The root represents all the processes in Π. For the sake of simplicity, the processes are represented by their index in the process vector Π. An example of an aggregation order is presented in Figure 1.

Definition 3 (Aggregation Order). *Given an LTS network $\mathcal{M} = (\Pi, \mathcal{V})$ of size n, an aggregation order of $\mathcal{M}$ is a tree $T_{\mathcal{M}} = (V, E)$ where $\emptyset \subset V \subset 2^{1..n}$ such that*

- *$1..n$ is the root of the tree,*
- *The singleton sets $\{i\} \in V$ with $i \in 1..n$ are the leaves of the tree, and*
- *For every non-leaf node $t \in V$, the children of t must form a partition of t.*

The *compositional aggregation* of a network $\mathcal{M}$ proceeds as follows. Let t be the root of aggregation order tree T. Compositional aggregation decomposes $\mathcal{M}$ by projecting $\mathcal{M}$ *on the sets* of the nodes in T and *by pre-order walk* of the aggregation order. That is, each component represented by a child of t is aggregated, i.e., the LTSs of its children are combined after which the result is minimised, before finally constructing and minimising the state space corresponding to t. Minimisation starts at the leaves. Aggregation is performed in a post-order walk of the aggregation tree (i.e., children are processed before their parents). At each non-leaf node t the state space of component t is constructed by concatenating the process vectors of the child networks and restoring synchronisations according to the sychronisation laws of the complete model $\mathcal{M}$.

The CADP toolbox offers several minimisation generation strategies.

- The monolithic approach, referred to as *root reduction*, directly computes the system LTS of an LTS network and then applies minimisation.
- *Root leaf reduction* applies minimisation on the process LTSs of a network and then applies root reduction on the resulting network.
- *Smart reduction* [8] is a heuristic that attempts to find an efficient aggregation order. First, all the process LTSs are minimised. Then, recursively, a set I of process LTSs is selected and the LTS in I are replaced by their aggregation.

4 Methodology

Setup. Our experiments were conducted in a controlled test bed comprising a set of homogeneous machines from the DAS-4 [4] cluster. Each machine has a dual quad-core INTEL XEON E5620 2.4 GHz CPU, 24 GB memory, and runs CENTOS LINUX 6. We used CADP version 2017-e "Sophia Antipolis" as implementation for the monolithic and compositional aggregation approaches.

The monolithic approach has been used as the control group. For compositional aggregation, all possible aggregation orders were computed using REFINER [37] in combination with the `decomposition.brute_force` plugin. The minimisation strategies were coded in the Script Verification Language (SVL) [13] of CADP. Given a property the hiding set was calculated using the maximal hiding technique [24]. This technique produces a set of property relevant actions that may not be hidden in the system. All other actions can be safely hidden without affecting the verification result.

As *cases* we consider LTS network models in CADP's EXP format. As *subjects* we consider case instances with a particular scale and hiding set. We use *minimisation strategy* to refer to both aggregation according to some order, and monolithic minimisation.

Research Questions. The variable of interest, i.e., the response variable, is the maximum memory cost of the state spaces produced by compositional aggregation. However, CADP reports the disk space cost of their LTS storage format, i.e., Binary Code Graphs (BCG), rather than the internal memory cost. As an alternative we use the *maximum number of transitions* generated as a measure for memory cost. The maximum number of transitions of an LTS has a strong and highly significant correlation with the disk space cost reported by CADP, i.e., they have a Kendall's τ_b coefficient [21] of 0.91 with a p-value of $2.2 \cdot 10^{-16}$. An additional advantage is that the metric is tool agnostic.

To answer RQ 1 (see Section 1) we measured the *maximum number of transitions* among the state spaces produced by compositional aggregation for *all possible* aggregation orders on a set of subjects. The effect of scaling and action hiding were investigating by controlling, respectively, the number of parallel processes and the property.

Next, *the performance of current heuristics are compared to that of other aggregation orders* in RQ 2. The *smart reduction* and *root leaf reduction* heuristics were applied on the subjects. Both heuristics are supported by CADP and have shown to be competitive w.r.t. other heuristics [8]. Again we measured the *maximum number of transitions* among the state spaces processed by compositional aggregation.

The intention of RQ 3 is to *explain the success or failure of composition aggregation.* Observed difference in performance between the subjects of the cases were investigated closer by inspecting the effect of action hiding, number of parallel processes, and aggregation order. Findings were verified with adjusted models fixing one or more aspects, therefore, obtaining more controlled measurements.

There are numerous variables that may affect the performance of compositional aggregation w.r.t. monolithic minimisation. Variables of interest are typically related to the size of a process LTS, or the reduction or interleaving that a process LTS or the composition of process LTSs may introduce.

Case and Subject Selection. The cases were sampled using *quota sampling* [25], i.e., cases with various characteristics were selected. To avoid source bias the cases were selected from four different sources, and where needed, converted to LTS networks.

Source 1 The BEnchmark for Explicit Model checkers (BEEM) database [27]. The benchmark includes 57 parameterised models with corresponding properties. [1]

Source 2 The demos of the CADP distribution. The CADP distribution contains a set of 42 demos. Many of the demos were extracted from the numerous real world verification case studies performed with CADP.

Source 3 The cases considered in an evaluation of automated assume-guarantee reasoning [7]. This set contains 6 scalable cases with corresponding properties. [2]

Source 4 The cases considered in our previous work [31]. In previous work we experimented with a set of 10 cases of which some are scalable.[3]

As mentioned by Cobleigh et al. [7] the generality of their work is threatened by the limited variety in network topology. To avoid this, we selected cases with

[1] paradise.fi.muni.cz/beem.
[2] http://laser.cs.umass.edu/breakingup-examples.
[3] http://www.win.tue.nl/mdse/property_preservation/FAC2017_experiments.zip.

a variety of network topologies. In addition, we took the following considerations into account:

1. The effect of action hiding was considered by selecting for each case various relevant safety and liveness properties.
2. To investigate the effect of the number of parallel LTSs on compositional aggregation we selected scalable cases. Each scalable case has one or more repeatable LTSs with which the model was scaled up; e.g., a model consisting of single server LTS and two client LTSs was scaled up by adding copies of the client LTSs.
3. The number of possible aggregation orders and the time required to construct state spaces grow exponentially with scale. Due to time considerations we limited each compositional aggregation to two hours. In addition, we prematurely terminated a compositional aggregation procedure as soon as it required more than the available (physical) memory, i.e., 24 GB. Any subjects violating the time or memory criteria were discarded from the experiment.
4. It is infeasible to calculate all 34,588 possible aggregation orders at seven parallel LTSs within reasonable time. For six parallel LTSs, this number is 2,647. To still find best and worst aggregation orders for up to six parallel LTS we limit the number of considered aggregation orders to 2,647.

Initially the sources above provided 115 models. We selected a number of scalable cases with a variety of network topologies. We discarded the cases for which it was infeasible to compute 2,647 aggregation orders for less than two scaled up version of the case. Finally, *eight* cases were selected covering *five* different network topologies. *Six* out of the *eight* cases were able scale to a size of six parallel LTSs while satisfying the time and memory criteria. The other two cases were scaled to *four* and *seven* parallel LTSs, respectively.

Next, we selected a range of properties relevant for the cases and modeled several scaled-up LTS networks. This finally resulted in a total of 129 subjects. The experiments were run on these 129 subjects. In total 117,879 decompositions were considered costing a total of 2.5 CPU-years. Finally, for 119 subjects all the run aggregation orders satisfied the time and memory criteria.

5 Results

5.1 Case and subject descriptions

Network topologies. The selected cases are characterised by the network topologies depicted in Figure 2. Dots indicate parallel processes and lines indicate synchronisation relations. Dashed lines show the synchronisation relations that are introduced by adding a repeatable process p.

Figure 2a shows a *client-server* topology. Such a network contains one or more servers and one or more clients.

In Figure 2b a *pipes and filter* topology is presented. The first process p_1 produces data and each process p_i $(i \in 1..n)$ in the sequence processes the data and filters before forwarding the filtered data to the next process p_{i+1}.

Table 1. Selected cases and their characteristics; with $p \geq 1$ the # of repeated LTSs

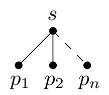
Case ID	Case description	Topology	Scaling	Source
1	The Gas Station problem [17]	a (3 servers)	$3 + p \geq 4$	1,3
2	Chiron user interface (single dispatcher) [20]	a (2 servers)	$3 + p \geq 4$	3
3	Eratosthenes' Sieve (distributed calculation of primes)	b	$1 + p \geq 3$	2
4	Le Lann leader election protocol [23]	c	$2 \cdot p \geq 4$	1
5	A simple token ring	c	$1 + p \geq 3$	4
6	Peterson's mutual exclusion protocol [28]	d	$2 \cdot p \geq 4$	1,2,3
7	Anderson's mutual exclusion protocol [3]	d	$1 + 2 \cdot p \geq 5$	1
8	Open Distributed Processing trader (ODP) [12]	e	$1 + p \geq 3$	2

A *ring* network topology is shown in Figure 2c. Communication between processes is organised as a ring structure. Often a token is passed along the edges that grants special privileges to the process holding the token.

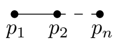
Figure 2d depicts communication via a number of *shared variables*. In the selected cases, for each repeatable process p_i there is a repeatable variable v_i.

In Figure 2e a peer-to-peer network topology is shown. Addresses and services of the peers p_i ($i \in 1..n$) are published via the tracker-server s after which the offered services can be employed on a peer-to-peer basis.

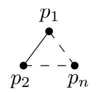
Case descriptions. We have selected *eight* scalable models as cases. An overview of these cases is given in Table 1. We identify the cases by their case number indicated in the *Case ID* column. The *Scaling* column shows the scaling of the cases in the number of repeated LTSs p and, on the right-hand side of the inequality, the minimum number of parallel LTSs; e.g., ODP's scaling $1 + p \geq 3$ states that there is one non-repeated LTSs (the trader) and one repeated LTS (the client), but the number of processes must be at least 3. Finally, in the *Source* column the sources of the cases are given, these correspond to the list of sources (Section 4). [4]



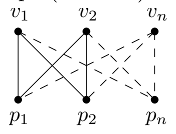
(a) Clients p_i ($i \in 1..n$) and server s



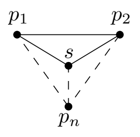
(b) Pipes and filters with processing nodes p_i ($i \in 1..n$)



(c) Ring with processing nodes p_i ($i \in 1..n$)



(d) Processes p_i sharing variables v_i ($i \in 1..n$)



(e) Peer-to-peer network with peers p_i ($i \in 1..n$) and tracker-server s

Fig. 2. Network topologies

Subject descriptions. Subjects correspond to instances of cases with a particular scale and hiding set, i.e., property. Subjects are identified by three alphanumeric characters: the first indicating the number of the case ID, the second indicating

[4] The models are available at `http://www.win.tue.nl/mdse/composition/test_cases.zip`.

the letter of a corresponding case property, and the third indicating the scale of the case model. With "_", we denote the absence of a property, i.e., no hiding is applied. For instance, `1e5` is the case `1` model where actions not relevant to property `e` (of case `1`) have been hidden and the subject has a total of `5` parallel LTSs. For each model, we identified between two and eight relevant properties.

The selected *scaling* is from the minimum scale of the case up to the possible scale nearest to six; e.g., for case `1` with property `a` the set of subjects is `1a4`, `1a5`, `1a6` and for case `6` with no property the set of subjects is `6_4`, `6_6`.

5.2 Analysis

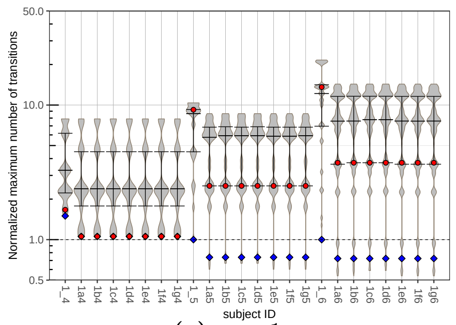
Figure 3 shows the distribution of the normalised maximum number of transitions of the generated state spaces for all possible aggregation orders of each subject, in the form of violin plots [18].[5] The black horizontal lines within each plot connected by a black vertical line indicate the first, second, and third quartiles. On the x-axis the subjects are displayed, grouped by case ID and scale. The y-axis displays the largest number of transitions the state space contained during compositioning on a $\log_{10}$-scale. Furthermore, the dashed horizontal line indicates the performance of monolithic construction. Finally, the normalised maximum number of transitions in memory during *smart reduction* and *root leaf reduction* are indicated by a red dot and blue diamond, respectively. It should be noted that the repeating of LTSs have a noticeable effect on the distribution of aggregation orders. Some peaks arise due to accumulation of sets of symmetric aggregation orders measuring the same normalized maximum number of transitions. However, as can be seen in the plots, in most cases this effect does not change significantly as more repeated LTSs are added.

RQ 1 How do action hiding, number of parallel processes, and aggregation order affect the memory consumption of compositional aggregation? We answer this research question using Figure 3. The chosen *aggregation order* has a major impact on the maximum number of transitions residing in memory. Two aggregation orders may differ up to several orders of magnitude depending on the subject.

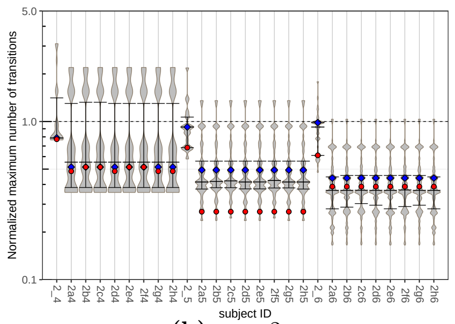
In general we observe that the range covered by the distribution of aggregation orders increases as the *number of parallel processes* increases. In all cases scaling up results in a better performance of the best aggregation orders w.r.t. monolithic verification, i.e., as the subjects increase in size, compositional aggregation becomes increasingly viable. In cases `1`, `3`, and `8` the range extends both upwards and downwards as the scale is increased; compared to the smaller subjects (in scale) the bad aggregation orders become worse and the good aggregation orders better. In the remaining cases the whole range shifts downwards as the number of parallel processes increases.

The shape of the distributions tends to change as the number of parallel processes increases. One of the factors contributing to this phenomenon is the

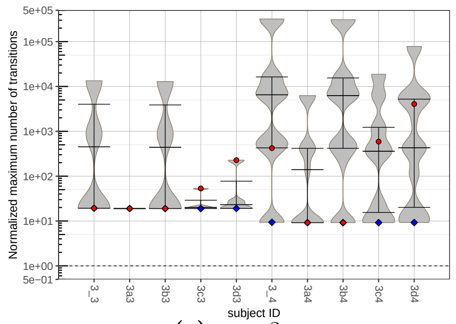
[5] All generated data is available at `http://www.win.tue.nl/mdse/composition/test_cases_data.zip`.



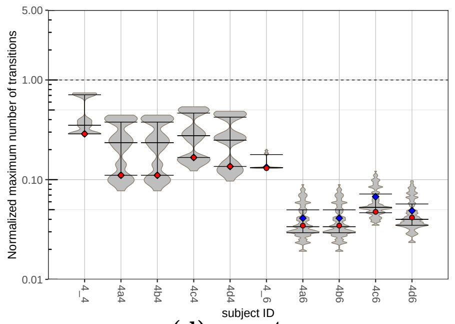
(a) case 1



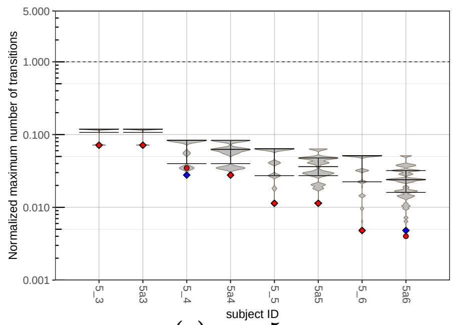
(b) case 2



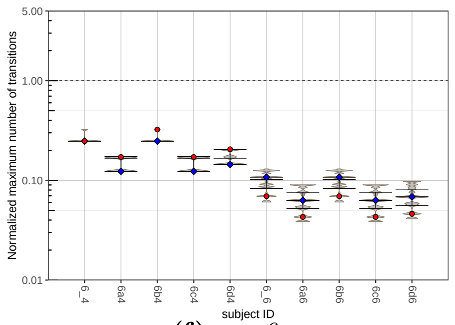
(c) case 3



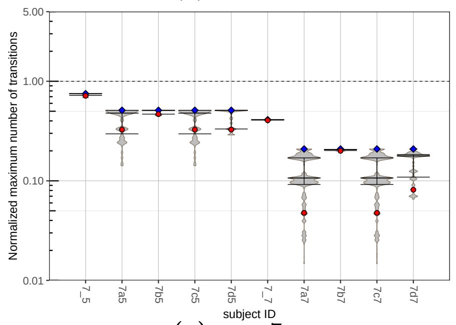
(d) case 4



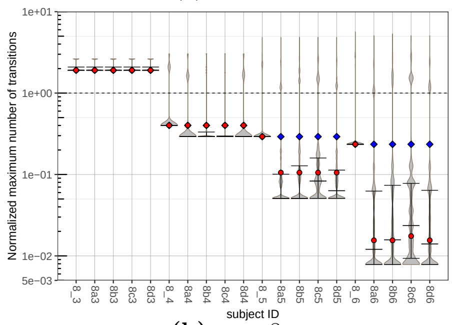
(e) case 5



(f) case 6



(g) case 7



(h) case 8

Fig. 3. Distribution of the *normalised maximum number of transitions* generated by the aggregation orders per subject (violin plots) and case (sub-figures).

increasing number of data points in the distributions as the scale increases; there are 4, 26, 236, and 2,647 distinct aggregation orders at `3`, `4`, `5`, and `6` parallel processes, respectively. At larger scales a sample of 2,647 orders was taken. This effect is particularly visible in case `3`, where the model at scales `3` and `4` are compared. However, most likely the changes are due to the number of repeated processes. Due to this the balance of constituents of the model changes causing the high density areas to change accordingly.

Applying action hiding practically always results in an improvement, the only exception being subjects `5a3` and `8a3` to `8d3`. In cases `1` and `2` practically no distinction in performance is observed between the applied hiding sets. Cases `3`, `4`, `6`, `7`, and `8` show moderate to significant variation in performance depending on the applied hiding set. For those subjects where the hiding sets have a noticeable impact, also the shape of the distribution is affected. For instance, subject `7c5` has a higher density around the optimal, forming a vase shape between the minimum and the first quartile, than `7d5`, which has a short tail in the same area.

RQ 2 How effective are the aggregation orders chosen by current heuristics? Figure 3 shows how smart reduction (indicated by a red dot) and root leaf reduction (indicated by a blue diamond) relate to the other orders. Both action hiding and the scaling can have a significant effect on their performance. However, there is no clear relation between these variables and the performance, which is particularly visible for case `3`.

Smart reduction requires fewer transitions in memory than the monolithic approach in 80 out of 119 subjects. Root leaf reduction performs better than monolithic minimisation in 94 out of 119 subjects. Furthermore, smart reduction and root leaf reduction find an optimal aggregation order for, respectively, 29 and 40 of the subjects.

Since our data is obtained from repeated measurements over eight cases, to make a fair and meaningful comparison we select cases under related conditions. We select the "*smallest*" and "*largest*" *subjects* in the number of parallel processes from the subjects in Fig. 3. From the properties we select the only two property IDs that all cases have in common; "`_`" (no property) and "`a`" (no deadlock). The intersection of these two pairs of selections yields four sets of subjects within which a comparison is made. First a comparison between the performance of the smart reduction and root leaf reduction is made, after which their performances are compared with the performance of optimal aggregation orders.

Table 2 compares the normalised maximum number of transitions of smart reduction and root leaf reduction. The first two columns indicate the selection criteria for the number of parallel processes. A comparison is made between smart reduction and root leaf reduction indicated by the *smart* and *root leaf* columns. The *Mean*, *Median* columns show the mean and median normalised maximum transitions. The final two columns, *# cases < monolithic* and *# cases < other heuristic*, indicate in how many cases the heuristics perform better than monolithic and the other heuristics, respectively.

Table 2. Normalised (w.r.t. monolithic) max. transitions descriptive statistics; with "Smallest" and "Largest" indicating, respectively, the smallest and largest number of parallel processes of subjects shown in Fig. 3

Size	Prop. ID	Mean		Median		# cases < monolithic		# cases < other heuristic	
		smart	root leaf	smart	root leaf	smart	root leaf	smart	root leaf
Smallest	_	3.12	3.10	0.74	0.77	5	5	3	1
Smallest	a	2.89	2.91	0.41	0.51	5	5	2	1
Largest	_	54.65	1.53	0.32	0.32	6	6	4	2
Largest	a	1.68	1.36	0.05	0.22	6	7	6	1

Table 3. Normalised (w.r.t opt. aggregation order) max. transitions descriptive statistics

Size	Prop. ID	Mean		Median		# opt. found	
		smart	root leaf	smart	root leaf	smart	root leaf
Smallest	_	1.02	1.02	1.00	1.00	5	5
Smallest	a	1.31	1.44	1.18	1.00	4	5
Largest	_	8.14	1.24	1.07	1.01	2	4
Largest	a	2.43	6.78	1.90	1.89	2	1

In the groups of "*smallest*" subjects there is little difference between the means of smart reduction and root leaf reduction. For both heuristics the mean performance is around 3 times that of the monolithic approach. On a positive note, the median is much lower than the mean for both heuristics. Smart reduction has a slightly better median performance. Both heuristics perform better than the monolithic approach in 5 out of 8 cases in both property ID groups. The remaining three cases being `1`, `3`, and `8` for both heuristics and property ID groups.

In the groups of "*largest*" subjects there is signification difference between the means of smart reduction and root leaf reduction in group "_". In group "a" this difference is only 0.32 in favor of root leaf reduction. The high mean value for smart reduction is caused by its poor performance at cases 1 and 3. Again the median performance is much better than the mean performance for both heuristics. The median performance of root leaf reduction is over four times that of smart reduction. Both heuristics perform better than the monolithic approach in 6 out of 8 cases in property ID group "_", while root leaf reduction performs better in one additional case in group "a". The two remaining cases being `1` and `3`, excluding case `1` in group "a" for root leaf reduction.

Table 3 compares the maximum number of transitions of the smart reduction and root leaf reduction heuristics normalised w.r.t. the optimum performance of compositional aggregation. The final columns, *# opt. found*, indicate how many times an optimal aggregation order was found.

If we compare the groups "*smallest*" and "*largest*" both the means and medians increase, and the number of optimums found decrease. This may indicate that it becomes harder for the heuristics to find (near-)optimal aggregation orders as the number of parallel processes increases, however, this should be confirmed by further experiments.

RQ 3 How can the success or failure of compositional aggregation be explained? Although our experiment involves a large number of subjects, the number of different cases per topology is still rather limited. However, based on this data, we make the following observations, backed up by results obtained for additional models with the same topology that we constructed to focus on specific key aspects of the cases.

Two factors seem to be most influential regarding the effectiveness of compositional aggregation: the amount of internal behaviour within single process LTSs, and the amount of synchronisation among the process LTSs. In the latter case, the involvement of data has a noticeable effect, in particular the size of the data domain; for instance, when synchronisation on a Boolean value is specified, the receiver only needs to be able to synchronise on **true** and **false**, while the synchronisation on a Byte value already requires 256 transitions, many of which may be unnecessary in the complete model, since they handle values on which synchronisation actually never happens. However, if in an aggregation order, this receiver is selected before the corresponding sender, then in each step before selecting the sender, all 256 transitions of the receiver will remain, and interleave with the transitions of all LTSs that *are* added to the composition.

Among the subjects, case 3 demonstrates best that the involvement of a lot of (to be synchronised) data has a negative effect on compositional aggregation. Additional experiments with a simple pipes and filters model, one with a data domain ranging from 1 to 2 and the other from 1 to 100, underline this observation, the latter performing an order of magnitude worse than the former. Furthermore, the former performs very well compared to monolithic verification, demonstrating that the bad performance of compositional aggregation is not inherent to the pipes and filters topology.

The positive effect of involving a property to be checked, and therefore action hiding, demonstrates the importance of internal behaviour in the process LTSs, as action hiding adds internal behaviour. It seems of little importance which property is actually added, i.e., whether it allows abstraction from all actions in the case of deadlock detection, or only a subset. This is best demonstrated by the token ring cases, i.e., cases 4 and 5. We manipulated case 5 in two different ways: increasing the amount of synchronisation, and increasing the amount of process-local (but not hidden) behaviour. The results clearly show that the former has a negative impact on performance, while the latter results in much better performance (by two orders of magnitude) iff a property is involved that allows the additional behaviour to be abstracted away, such as deadlock freedom.

The mutual exclusion algorithms, i.e., cases 6 and 7, have exactly the same set of properties. Those results demonstrate that the effect of adding a property is not always the same for all models of the same topology; adding a property seems to have a bigger effect on case 7 than case 6, resulting in a bigger range between the worst and best performing aggregation orders.

In a follow-up experiment, we will extend the number of cases and/or subjects per topology, to achieve conclusive evidence that could generalise these observations.

5.3 Threats to validity

When interpreting the results of this study consider the following threats to validity:

- Only one tool has been involved to conduct the experiment, hence the results may be implementation specific. On the other hand, involving multiple tools introduces the problem that differences in implementations may affect the outcome.
- The scope of this study is limited to models that are represented as networks of LTSs. Therefore, the results of this study are possibly only applicable to models represented as networks of LTSs. As the compositional aggregation method is limited to these kind of models we have not considered alternative model representations.
- The study only considers the DPBB equivalence as aggregation relation. Results may vary depending on the chosen equivalence relation. The DPBB equivalence is the strongest aggregation order offered by CADP that still allows abstraction. Hence, other relations are expected to show better performance improvements.
- The scaled up models make use of a repeatable LTSs. It may be possible that the results are skewed due to lack of heterogeneous components. However, the used compositional aggregation methods do not take advantage of the symmetry in the model.
  The repeating of LTSs is noticeable in the violin plots (Fig. 3) as accumulation of sets of symmetric aggregation orders measuring the same normalized maximum number of transitions. Nevertheless, in most cases this effect does not change significantly as more repeated LTSs are added.
- A relatively small set of different cases has been studied, even though this experiment is the most comprehensive one performed thus far. In the future, we plan to extend this set considerably, but obtaining such a large set is very time-consuming. The lack of a (publicly available) set of nicely scalable models is a problem in general when analysing and designing formal verification techniques.
- Models with a relatively small number of parallel processes were considered. Beyond models with six parallel LTSs the experiments quickly become unfeasible. Extrapolation of the results presented in this work to models with more parallel LTSs should be done with caution. In the future, we plan to extend our analysis to subjects with more processes.

6 Conclusions

Our thorough analysis of compositional aggregation when applied on 119 subjects with varying topology, scale, and hiding set provides the following insights:

1. The amount of internal behaviour in process LTSs and the amount of synchronisation between process LTSs have the biggest impact on the performance, in terms of the largest number of generated transitions in memory.

2. The involvement of a functional property, and therefore a hiding set, is significant. The size of this hiding set is of less importance. For typical properties, maximal hiding already allows the hiding of a relatively large amount of behaviour.
3. Among the five network topologies we considered, none of them fundamentally rule out compositional aggregation as an effective technique.
4. As the number of processes in a model is increased, the effectiveness of compositional aggregation tends to increase as well.

It should be noted that we only considered a few cases per topology. To generalise our conclusions, we will have to work on extending our benchmark set. The first two conclusions underline observations made in earlier work [8]. Since they worked with a set of subjects of less variety, we can make these observations with more confidence.

Future Work. In the near future, we will extend the current analysis to further explain the success and failure of compositional aggregation for the different subjects, and based on this, work on the construction of a new heuristic. For this to be successful, we will have to involve many more cases. As scalable models have now been thoroughly investigated, we can next focus on non-scalable models, of which many are publicly available.

Acknowledgements. The authors would like to thank Vrije Universiteit Amsterdam for their generosity in supplying the computing resources for the experiments.

References

1. Abd Elkader, K., Grumberg, O., Păsăreanu, C.S., Shoham, S.: Automated Circular Assume-Guarantee Reasoning with N-way Decomposition and Alphabet Refinement. In: CAV, part I. LNCS, vol. 9779, pp. 329–351. Springer (2016)
2. Andersen, H.: Partial Model Checking. In: LICS. pp. 398–407. IEEE Computer Society Press (1995)
3. Anderson, J.H., Kim, Y.J., Herman, T.: Shared-memory mutual exclusion: major research trends since 1986. Distrib. Comput. 16(2-3), 75–110 (2003)
4. ASCI: The Distributed ASCI Supercomputer DAS4. `http://www.cs.vu.nl/das4/`, accessed: 2017-08-09
5. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
6. Cheung, S.C., Kramer, J.: Context constraints for compositional reachability analysis. ACM Trans. Softw. Eng. Methodol. 5(4), 334–377 (Oct 1996)
7. Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Trans. Softw. Eng. Methodol. 17(2), 7:1–7:52 (May 2008)
8. Crouzen, P., Lang, F.: Smart Reduction. In: FASE. LNCS, vol. 6603, pp. 111–126. Springer, Berlin (2011)
9. Crouzen, P., Hermanns, H.: Aggregation ordering for massively compositional models. In: 10th International Conference on Application of Concurrency to System Design. pp. 171–180. IEEE (2010)

10. Fernandez, J.: ALDEBARAN : un système de vérification par réduction de processus communicants. (Aldebaran : a system of verification of communicating processes by using reduction). Ph.D. thesis, Joseph Fourier University, Grenoble, France (1988)
11. Garavel, H., Lang, F., Mateescu, R.: Compositional Verification of Asynchronous Concurrent Systems using CADP (extended version). Research Report RR-8708, INRIA Grenoble - Rhône-Alpes (Apr 2015), https://hal.inria.fr/hal-01138749
12. Garavel, H., Sighireanu, M.: A Graphical Parallel Composition Operator for Process Algebras. In: FORTE/PSTV'99. IFIP Conference Proceedings, vol. 156, pp. 185–202. Kluwer (1999)
13. Garavel, H., Lang, F.: Svl: A scripting language for compositional verification. In: 21st International Conference on Formal Techniques for Networked and Distributed Systems. pp. 377–392. Kluwer, Boston, MA (2002)
14. Gheorghiu Bobaru, M., Păsăreanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: CAV. pp. 135–148. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
15. Glabbeek, R.v., Luttik, S., Trčka, N.: Branching bisimilarity with explicit divergence. Fundam. Inf. 93(4), 371–392 (Dec 2009)
16. Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verification. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification. pp. 420–432. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
17. Heimbold, D., Luckham, D.: Debugging Ada Tasking Programs. IEEE Software 2, 47–57 (1985)
18. Hintze, J., Nelson, R.: Violin Plots: A Box Plot-Density Trace Synergism. The American Statistician 52(2), 181–184 (1998)
19. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP congress. vol. 83, pp. 321–332 (1983)
20. Keller, R.K., Cameron, M., Taylor, R.N., Troup, D.B.: User Interface Development and Software Environments: The Chiron-1 System. In: Proceedings of the 13th International Conference on Software Engineering. pp. 208–218. IEEE (1991)
21. Kendall, M., Gibbons, J.: Rank correlation methods, chap. 3. Oxford University Press, Oxford, 5 edn. (1990)
22. Lang, F.: Refined Interfaces for Compositional Verification. In: FORTE. LNCS, vol. 4229, pp. 159–174. Springer (2006)
23. Le Lann, G.: Distributed systems - towards a formal approach. In: IFIP Congress. pp. 155–160 (1977)
24. Mateescu, R., Wijs, A.: Property-Dependent Reductions Adequate with Divergence-Sensitive Branching Bisimilarity. Science of Computer Programming 96(3), 354–376 (2014)
25. O'Leary, Z.: The Essential Guide to Doing Research. SAGE Publications (2004)
26. Păsăreanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design 32(3), 175–205 (Jun 2008)
27. Pelánek, R.: BEEM: Benchmarks for Explicit Model Checkers. In: SPIN. LNCS, vol. 4595, pp. 263–267. Springer (2007)
28. Peterson, G.L.: Myths about the mutual exclusion problem. IPL 12, 115–116 (1981)
29. Pnueli, A.: In Transition From Global to Modular Temporal Reasoning about Programs. In: Logics and Models of Concurrent Systems. NATO ASI, vol. 13, pp. 123–144. Springer, Berlin (1985), https://doi.org/10.1007/978-3-642-82453-1_5

30. de Putter, S., Wijs, A.: Compositional Model Checking is Lively. In: FACS. LNCS, vol. 10487, pp. 117–136. Springer (2013)
31. de Putter, S., Wijs, A.: A formal verification technique for behavioural model-to-model transformations. Formal Aspects of Computing (Oct 2017), `https://doi.org/10.1007/s00165-017-0437-z`
32. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
33. Sabnani, K.K., Lapone, A.M., Uyar, M.U.: An algorithmic procedure for checking safety properties of protocols. IEEE Transactions on Communications 37(9), 940–948 (1989)
34. Tai, K.C., Koppol, P.V.: Hierarchy-based incremental analysis of communication protocols. In: 1993 International Conference on Network Protocols. pp. 318–325. IEEE (1993)
35. Tai, K.C., Koppol, P.V.: An incremental approach to reachability analysis of distributed programs. In: Proceedings of the 7th international workshop on Software specification and design. pp. 141–150. IEEE Computer Society Press (1993)
36. Valmari, A.: Compositional state space generation. In: Advances in Petri Nets. pp. 427–457. Springer, Berlin (1993)
37. Wijs, A.J., Engelen, L.J.P.: Refiner: Towards Formal Verification of Model Transformations. In: NFM. LNCS, vol. 8430, pp. 258–263. Springer (2014)