Compositional Verification in Action

Hubert Garavel[1], Frédéric Lang[1], and Laurent Mounier[2]

[1] Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, Verimag, F-38000 Grenoble, France
E-mail: {`hubert.garavel`,`frederic.lang`}`@inria.fr`,
`laurent.mounier@univ-grenoble-alpes.fr`

Abstract

Concurrent systems are intrinsically complex and their verification is hampered by the well-known “state-space explosion” issue. Compositional verification is a powerful approach, based on the divide-and-conquer paradigm, to address this issue. Despite impressive results, this approach is not used widely enough in practice, probably because it exists under multiple variants that make knowledge of the field hard to attain. In this article, we highlight the seminal results of Graf & Steffen and propose a survey of compositional verification techniques that exploit (or not) these results.

Keywords: Bisimulation, Compositional minimisation, Compositional reachability analysis, Compositional verification, Concurrency theory, Equivalence checking, Formal method, Labelled Transition System, Model checking, Process algebra, Process calculus, Validation, Verification

1 Introduction

The present article was written in honour of Susanne Graf and Bernhard Steffen at the occasion of their 60th birthdays.

Concurrent systems are commonly found in software programs, hardware circuits, and telecommunication networks, where many processes have to execute simultaneously, synchronise to properly access shared resources, and communicate together to achieve common tasks. Concurrent systems are notoriously hard to design correctly, as they are prone to subtle errors, such as deadlocks, livelocks, or synchronisation issues. To avoid or detect such errors, formal methods, supported by computer-aided verification tools, are established techniques for the design of concurrent systems [23].

Unfortunately, verification algorithms for concurrent systems are often hampered by the “state-space explosion” issue, which arises when the complexity of verification (which can be exponential in the number of concurrent processes) exceeds the capabilities of the computer on which verification is performed. This makes it difficult, if not unfeasible, to analyse large systems with many processes, such as most industrial case studies. Various verification approaches have been proposed to fight state-space explosion, but there is no silver bullet, as each approach works under specific assumptions, for particular classes of problems.

The present article focuses on one of these approaches, *compositional verification*, which relies on "divide-and-conquer" strategies that decompose a global system into local concurrent processes and seek to exploit locality properties of these processes. There are many different branches of formal methods and, consequently, very diverse forms of compositional verification. The present article is centred around a series of papers published between 1990 and 1996 by Susanne Graf and Bernhard Steffen [36] [37] [38] [39] [40], the three latter ones being co-authored with Gerald Lüttgen. More precisely, the scope of the present article is defined as follows:

- We consider the established framework of *asynchronous concurrency*, in which concurrent processes execute without assumption about their respective speeds. These processes can synchronise and communicate using Hoare's rendezvous[1] [44]. Communicating automata [1] and process calculi [7] naturally fit in this setting. Other communication schemes, such as shared memories or message queues, can be expressed, as particular cases, in terms of rendezvous.
- We do not consider compositional verification techniques designed for theorem proving or static analysis, but only those designed for *enumerative verification* (or *reachability analysis*) methods, which rely on state-space exploration and include both *model checking* (in which the properties to be verified are expressed in some temporal logic) and *equivalence checking* (in which the properties to be verified are expressed using bisimulations or behavioural preorders).
- We do not consider *state-based* models, such as Kripke structures (in which relevant information is attached to the states, usually in the form of *state variables*, so that the properties to be verified are expressed using predicates or invariants relating these variables); instead, we consider *action-based* models, such as labelled transition systems (in which relevant information is attached to the transitions, usually in the form of *transition labels*, so that the properties are expressed as sequences, trees, or graphs of actions).
- We consider both *explicit-state* methods (in which reachable states and transitions are analysed individually) and *symbolic* methods (in which sets of reachable states are analysed collectively). Actually, many papers discussed in this survey use explicit-state methods, but symbolic methods are also applicable. There is a common belief that symbolic methods systematically outperform explicit-state ones, which hardly exceed 10^{12} states on current

---

[1] Some authors consider rendezvous as synchronous and message queues as asynchronous.

machines; this is a misconception and the situation is more contrasted. In particular, explicit-state methods handle dynamic data structures (e.g., lists, trees, etc.) more easily, and, even in the case of pure control structures (e.g., Petri nets), recent results [48] show that explicit-state methods, combined with appropriate reductions, compete well with symbolic methods.

The compositional verification approaches we consider here are traditionally referred to as *compositional minimisation* or *compositional reachability analysis*; they are *action-based* and rely on *equivalence-checking* concepts, especially behavioural equivalence and preorder relations between labelled transition systems. In the sequel, *compositional verification* is often used as a synonym for *compositional minimisation*, although the latter is clearly more specific.

There exist indeed alternative approaches, referred to as *compositional reasoning*, *assume-guarantee*, or *rely-guarantee*, which are often *state-based* and rely on *model-checking* concepts, including assertions, logic formulas and satisfaction relations. See, e.g., [66] and [34] for detailed presentations of these approaches.

The present article is organised as follows. Section 2 introduces compositional minimisation in its simplest forms. Section 3 recalls the main concepts, namely *interfaces* and *semi-composition*, put forward in the seminal papers of Graf, Steffen & Lüttgen. Section 4 discusses enhanced compositional approaches that use interfaces without semi-composition. Section 5 presents the most advanced approach, in which interfaces and semi-composition are both used. Practical applications of compositional verification to realistic case studies are reported whenever possible. Finally, Section 6 gives a few concluding remarks.

2 Compositional Minimisation without Interfaces

2.1 Principles

To perform compositional minimisation in an action-based setting, one needs six ingredients carefully designed to fit well together:

1. A *low-level model* M, which is a state-transition formalism[2] in which the behaviour of the system S under verification can be encoded. This model is usually very simple, with a low abstraction level, so that the properties to be verified for S can be easily checked on M. As a counterpart, the encoding of S in M can get large and verbose. Two famous examples of such models are: *labelled transition systems* [63], which are the underlying semantic model of most process calculi and play a central role in major functional verification tools, and *interactive Markov chains* [42], which are performance evaluation

---

[2] For conciseness, we use the same term "model" and the same letter M to refer both to the "meta-model" (i.e., the low-level formalism) and the "models" (i.e., all particular instances expressed in this formalism).

models that combine ordinary transitions and stochastic ones, the firing time of the latter being governed by exponential distributions.

2. A *parallel composition operator* $||$ that takes n models M_1, ..., M_n and returns a new model $M' = M_1||...||M_n$. The notation $||$ is a crude simplification, as parallel composition operators usually carry extra information to determine which synchronisations have to be done (see, e.g., [30]). The resulting model M' is often referred to as a *composition*, while M_1, ..., M_n are referred to as *components*[3]. More often than not, the complexity of M' (measured in number of states and transitions) is the product (rather than the sum) of the complexities of M_1, ..., M_n: the state-space explosion problem precisely lies in such complexity growth.

3. An *equivalence relation* $\approx$ defined over models. This relation (which differs from graph isomorphism noted $=$) should be a congruence with respect to parallel composition, i.e., if $M_i \approx M'_i$ for all $i \in \{1, ..., n\}$, then $M_1||...||M_n \approx M'_1||...||M'_n$. Two examples of such equivalences are: strong bisimulation [60], which is a congruence for the parallel composition operators of most process calculi (see [88] for a discussion) and branching bisimulation [84]. Equivalence relations may incorporate *abstractions*: for instance, branching bisimulation can remove some τ-transitions ($\tau.M \approx M$), and Markov-chain lumpability can merge some stochastic transitions ($\lambda.M + \mu.M \approx (\lambda+\mu).M$).

4. A *minimisation function* $\mathsf{min} : M \to M$ that maps each model to a distinguished element of its equivalence class in the quotient set $M/\approx$; this distinguished element is usually chosen to minimise some complexity criterion. For bisimulation relations, for example, one chooses a labelled transition system that has the least number of states. Minimising a model applies to this model the abstractions inherent to relation $\approx$. Because of the congruence property, one has $M_1||...||M_n \approx \mathsf{min}(M_1)\,||...||\,\mathsf{min}(M_n)$ and $\mathsf{min}(M_1||...||M_n) = \mathsf{min}(\mathsf{min}(M_1)\,||...||\,\mathsf{min}(M_n))$.

5. A *high-level language* L in which the system S can be specified. Theoretical papers on compositional verification often use M in place of L, but this is not realistic, as complex systems are never described using low-level models only. The language L should be equipped with a concept of components and a parallel composition, also noted $||$, for assembling these components. A composition $C_1||...||C_n$ is said to be *flat* if all components C_i are sequential, or *hierarchical* if some components C_i are themselves compositions.

6. A *translation function* $[\![\cdot]\!] : L \to M$ that maps each system S written in L to a corresponding low-level model $[\![S]\!]$. This function should be able to translate components taken individually, and should be a morphism for parallel composition, meaning that, given n components C_1, ..., C_n, $[\![C_1||...||C_n]\!] \approx [\![C_1]\!]\,||...||\,[\![C_n]\!]$. The translation of an entire system may very

---

[3] Also called *subsystems*, *agents*, or *processes* in the literature.

well fail due to state explosion[4], but the translation of individual components is expected to succeed, at least for a majority of them.

Given a system $S = C_1||...||C_n$ such that $[\![S]\!]$ is excessively large, compositional minimisation, in its simplest form, avoids to compute $[\![S]\!]$ directly and computes $\mathsf{min}\,[\![C_1]\!]\,||...||\,\mathsf{min}\,[\![C_n]\!]$ instead. This idea was advocated in many papers, both in the functional verification setting [18] [54] [68] [88] [76] [75] [79] [80] [83] and in the performance evaluation setting [42] [24].

2.2 Strategies

In practice, compositional minimisation is more complex than the simple form exposed above. For systems with many components, there are multiple ways (called *strategies*) to perform compositional minimisation, and all strategies do not necessarily have the same efficiency, i.e., provide the same amount of state-space reduction. The efficiency of a strategy is inversely proportional to the size (e.g., number of states) of the largest intermediate model that is generated; a good strategy strives to keep this size as small as possible, in order to avoid state-space explosion during the compositional verification process. There are several causes leading to the existence of multiple strategies.

First, if the system has a hierarchical structure, e.g., $(C_1||C_2)||(C_3||C_4)$, minimisation can be applied either to the leaf components only, i.e., $(\mathsf{min}\,[\![C_1]\!]\,||\,\mathsf{min}\,[\![C_2]\!])||(\mathsf{min}\,[\![C_3]\!]\,||\,\mathsf{min}\,[\![C_4]\!])$, or to every intermediate level in the hierarchy, i.e., $\mathsf{min}(\mathsf{min}(\mathsf{min}\,[\![C_1]\!]\,||\,\mathsf{min}\,[\![C_2]\!])||\,\mathsf{min}(\mathsf{min}\,[\![C_3]\!]\,||\,\mathsf{min}\,[\![C_4]\!]))$, or any intermediate combination between these two extremes. Such strategies are called *static* as they are uniformly applied to all components.

Second, compositional minimisation is sometimes counterproductive. Replacing, in a parallel composition $M_1||...||M_n$, some model M_i by its quotient $\mathsf{min}(M_i)$ never increases the complexity, but computing $(\mathsf{min}\,[\![C_1]\!]||...||\mathsf{min}\,[\![C_n]\!])$ rather than $[\![C_1||...||C_n]\!]$ may fail if the complexity of some $[\![C_i]\!]$ is larger than that of $[\![C_1||...||C_n]\!]$. This may very well occur when components are so tightly synchronised that the behaviour of a component C_i is strongly constrained by the other components; ignoring such components may lead to a huge, or even unbounded, state space for C_i. Shared memories, network links, and hardware buses are typical examples of components C_i whose models $[\![C_i]\!]$ cannot be generated in isolation because they allow a potentially infinite number of read/write or send/receive operations, whereas the components that use these memories, links, or buses actually employ a much smaller set of operations. Thus, when performing compositional minimisation on a system $S = C_1||...||C_n$, it is not necessarily optimal to minimise all components one by one; it might be more efficient to consider them two by two, three by three, etc., leading to a number of combinations that is an exponential of n.

---

[4] In theoretical papers that use M in place of L, there is a notational confusion between C_i and $[\![C_i]\!]$, which is particularly annoying when the latter cannot be computed.

Finding an optimal strategy is difficult, and computationally out of reach if the number of components is large. So, one can only rely on heuristics. Rather than using the aforementioned static strategies, which are probably suboptimal, it is more suitable to use *dynamic* strategies that decide, at each verification step, which subset of components is the best candidate for being generated and minimised.

Such a heuristic (called *smart reduction*) is proposed in [16], based on metrics that consider both the amount of synchronisations between components (trying to compose the most tightly synchronised components first, to avoid state-space explosion arising from the interleaving of loosely coupled components) and the proportion of transitions that can be hidden after composition (the more hidden transitions, the greater the gains during subsequent minimisation steps if a weak equivalence, e.g., branching bisimulation, is used).

2.3 Applications

Implementing compositional minimisation is a difficult challenge, because many software tools are required to implement M, $||$, $\approx$, min, L, and $[\![\cdot]\!]$. Moreover, if any of these tools is poorly implemented, the entire tool chain may become inefficient and useless for non-trivial applications.

A handful of tool prototypes have been developed in the 90s, but the best implementation of compositional minimisation available today is unquestionably the Cadp toolbox [27], the development of which started in the late 80s and has been steadily pursued until now. Compositional verification, at large, is a particular strength of Cadp [26]. Concerning compositional minimisation, Cadp provides the following software tools and libraries:

- M is implemented by Bcg[5] (*Binary-Coded Graphs*), a compact format, with its associated software tools and libraries, that enable large transition systems (with billions of states and transitions) to be stored as computer files.
- $||$ is implemented by Exp.Open[6], a tool that, among other features, computes the parallel composition of transition systems executing concurrently and synchronised using the parallel operators of various process calculi.
- $\approx$ and min are respectively implemented by Bcg_Cmp[7] and Bcg_Min[8], two state-of-the-art tools (see [9] for an assessment) that compare and minimise transition systems modulo various equivalence and preorder relations.
- L is implemented in multiple ways, as the Cadp toolbox supports several high-level languages for describing value-passing concurrent systems. For many years, Lotos (Iso/Iec international standard 8807) [46] has been the

---

[5] `http://cadp.inria.fr/man/bcg.html`
[6] `http://cadp.inria.fr/man/exp.open.html`
[7] `http://cadp.inria.fr/man/bcg_cmp.html`
[8] `http://cadp.inria.fr/man/bcg_min.html`

language of choice but, since 2010, it has been progressively replaced by LNT [28], a modern specification language combining features from process calculi, imperative languages, and functional languages.

- $[\![\cdot]\!]$ is implemented by the two LOTOS compilers CÆSAR[9] and CÆSAR.ADT[10], and by the LNT2LOTOS[11] translator, the combination of which delivers state-of-the-art user-friendliness and performance (see [57] [58] for an assessment).

Moreover, a unique feature of CADP is its scripting language SVL[12] [25], which can be seen as a process calculus extended with operations on labelled transition systems, e.g., comparison, minimisation, hiding and renaming of transition labels, detection of deadlocks and livelocks, etc. Designed with the goal of making compositional verification easily accessible to non-experts [51], SVL and its associated compiler[13] implement the aforementioned static and dynamic strategies, including smart reduction.

Compositional minimisation, as implemented in CADP, has been successfully used in many case studies. A dozen of small- or medium-size examples are available online, as part of the CADP demos[14]. In four of these examples (demos No. 05, 18, 25, and 35, which have between 5 and 20 components), compositional minimisation easily succeeds (generating intermediate models with 2.10^6 states at most) where direct generation fails. In seven other examples (demos No. 01, 02, 08, 17, 27, 28, and 36, which have between 4 and 11 components), the largest intermediate model generated by compositional minimisation is between 1.7 and 24 times smaller than the model obtained using direct generation.

Here is a chronological list (since 1991) of case studies in which compositional minimisation, as implemented in CADP, has been used to achieve functional verification. For conciseness, we use the symbol $\star$ to indicate those case studies in which the author's laboratories (INRIA Grenoble, LIG, and/or Verimag) have been involved:

- rel/REL reliable atomic multicast protocol[15] [3, 19], Hewlett-Packard (UK)*.
- Transit Node message router[16] [61]*.
- CoopScan framework for cooperative applications development[17] [47]*.
- Transmission Control Protocol (TCP)[18] [73], Berlin (DE).
- Distributed leader election for unidirectional ring networks[19] [29]*.

---

[9] `http://cadp.inria.fr/man/caesar.html`
[10] `http://cadp.inria.fr/man/caesar.adt.html`
[11] `http://cadp.inria.fr/man/lnt2lotos.html`
[12] `http://cadp.inria.fr/man/svl-lang.html`
[13] `http://cadp.inria.fr/man/svl.html`
[14] `http://cadp.inria.fr/demos`
[15] `http://cadp.inria.fr/case-studies/91-c-relrel.html`
[16] `http://cadp.inria.fr/case-studies/94-a-transitnode.html`
[17] `http://cadp.inria.fr/case-studies/95-c-groupware.html`
[18] `http://cadp.inria.fr/case-studies/96-d-tcp.html`
[19] `http://cadp.inria.fr/case-studies/96-f-leaderelection.html`

- Bus arbitration of the Powerscale architecture[20] [11], Bull, Les Clayes (FR)⋆.
- Eurocontrol's Departure Clearance Protocol[21] [17], Brussels (BE).
- OM/RR protocol for traffic control[22] [86, 85], Eindhoven (NL).
- INRES protocol[23] [53], Nokia Research Center (FI).
- Bull's CFS distributed file system for AIX[24] [64]⋆.
- Philips' HAVi leader election protocol[25] [67], Amsterdam (NL).
- Single pulser and bus arbitration hardware designs[26] [41], Stirling (UK).
- Sync-stop & Chandi-Lamport checkpoint algorithms[27] [35], Bucharest (RO)⋆.
- Chilean electronic invoices system[28] [2, 6], Sophia Antipolis (FR).
- FRACTAL software components[29] [4, 5], Sophia Antipolis (FR), London (UK).
- FAUST asynchronous network-on-chip[30] [71, 72], CEA/Leti, Grenoble (FR)⋆.
- Diagrams for choreographies[31] [70], Málaga (ES) and Santa Barbara (US).
- Trivial File Transfer Protocol (TFTP)[32] [31], Airbus, Toulouse (FR)⋆.
- CRESS diagrams for Web and grid services[33] [77], Stirling (UK).
- Logical regulatory modules[34] [59], Oeiras (PT), Evry-Paris-Marseille (FR)⋆.
- Fault-tolerant routing algorithm for a network-on-chip[35] [89], Utah (US)⋆.
- Graphical user interfaces[36] [62], Atos, Grenoble (FR)⋆.

Compositional minimisation has also been used for performance evaluation:

- Performance analysis of a Plain Old Telephone System[37] [43], Erlangen (DE).
- SCSI-2 bus arbitration protocol[38] [24], Twente (NL)⋆.
- European Train Control System[39] [8], Saarbrücken (DE), Freiburg (DE).

---

[20] `http://cadp.inria.fr/case-studies/96-h-powerscale.html`
[21] `http://cadp.inria.fr/case-studies/97-c-dcl.html`
[22] `http://cadp.inria.fr/case-studies/98-d-omrr.html`
[23] `http://cadp.inria.fr/case-studies/98-g-inres.html`
[24] `http://cadp.inria.fr/case-studies/98-i-cfs.html`
[25] `http://cadp.inria.fr/case-studies/99-a-havi.html`
[26] `http://cadp.inria.fr/case-studies/99-b-dill.html`
[27] `http://cadp.inria.fr/case-studies/01-d-checkpointing.html`
[28] `http://cadp.inria.fr/case-studies/04-a-electronic-invoices.html`
[29] `http://cadp.inria.fr/case-studies/05-c-components.html`
[30] `http://cadp.inria.fr/case-studies/07-a-faust.html`
[31] `http://cadp.inria.fr/case-studies/09-a-collab-diag.html`
[32] `http://cadp.inria.fr/case-studies/09-h-tftp.html`
[33] `http://cadp.inria.fr/case-studies/09-p-web-and-grid.html`
[34] `http://cadp.inria.fr/case-studies/13-c-regulatory-modules.html`
[35] `http://cadp.inria.fr/case-studies/13-f-utahnoc.html`
[36] `http://cadp.inria.fr/case-studies/14-d-hmi.html`
[37] `http://cadp.inria.fr/case-studies/98-b-markov-pots.html`
[38] `http://cadp.inria.fr/case-studies/02-f-scsi-2.html`
[39] `http://cadp.inria.fr/case-studies/06-e-etcs.html`

3 The Seminal Papers of Graf, Steffen, and Lüttgen

In spite of these achievements, compositional minimisation still faces practical limitations when some components (such as the aforementioned shared memories, network links, and hardware buses) cannot be analysed separately from their neighbour components. This problem was addressed, as early as 1990, by Graf & Steffen in a series of five scientific papers:

- [36]: the original paper, published at the first CAV workshop in 1990, which contains all the fundamental contributions;
- [37]: a technical report from RWTH Aachen, published in 1991, which gives the proofs for the theorems of [36];
- [38]: a technical report from Universität Passau, with Gerald Lüttgen as third author, published in 1995, which extends the theoretical developments of the former papers and includes a running example that illustrates the key steps of the approach;
- [39]: a 10-page extended abstract published in the paper version of the *Journal on Formal Aspects of Computing*; due to constraints on the number of pages, this paper does not contain more material than the initial paper [36];
- [40]: a 28-page journal article, which is based on [38] and can be considered as the most complete version; this article is available online from the electronic repository of the *Journal on Formal Aspects of Computing*[40].

These papers are a breakthrough in compositional verification, as they target the difficult case where some component C_i of a system $S = C_1||...||C_n$ cannot be minimised in isolation from its *environment*, i.e., from the other components $C_1||...||C_{i-1}||C_{i+1}||...||C_n$. Precisely, this is the case where the behaviour of C_i is potentially huge, so that state-space explosion occurs when computing $[\![C_i]\!]$, but only a fraction of this behaviour is actually permitted by the environment of C_i. To address such situations, Graf & Steffen propose the following approach, which we reformulate here in a didactic manner:

- The constraints[41] exerted on C_i by its neighbour components must be expressed as an *interface*[42] noted I, which is intended to be a set of traces containing all the sequences of actions allowed by the environment. Concretely, the interface is represented as a labelled transition system, and the traces are the words of the language recognised by this automaton. In practice, I is usually specified in the same high-level language L as the components $C_1, ..., C_n$ and later translated to the low-level formalism M. It is assumed that I is small enough that state-space explosion never occurs when computing $[\![I]\!]$.

---

[40] Online manuscript at `http://www-verimag.imag.fr/~graf/PAPERS/GLS96.pdf`.

[41] Also called *context constraints* or *environment constraints* in the literature.

[42] Also called *behavioural interface*, *interface specifications*, or *process interface*.

- Graf & Steffen assume that the interface I is provided by the user, based on his/her own intuition of how the environment behaves. Thus, the interface is not necessarily *exact* because of human errors or approximations:
    - If the interface is too *restrictive*, i.e., if it contains less traces than allowed by the environment, this is a severe problem, as the model computed for C_i will be truncated, so that subsequent verification steps will be done under false assumptions. In such case, the interface is said to be *incorrect*.
    - If the interface is too *permissive*, i.e., if it contains more traces than allowed by the environment, there is no correction problem, but there might be a performance problem, as the model computed for C_i will be larger than actually needed. The most permissive interface is the "chaos" automaton that accepts all actions of C_i in any order, which is equivalent to having no interface for C_i.

  So, a correct interface should be a superset of the traces allowed by the environment. Said differently, a correct interface should express some of, but not necessarily all, the constraints exerted by the neighbour components.

- Graf & Steffen define a *semi-composition*[43] operator $\Pi_I(C_i) = \pi_1([\![C_i || I]\!])$, where $||$ denotes the parallel composition operator of CSP [45] that forces C_i and I to synchronise on their common actions, while letting C_i (resp. I) interleave on its actions that are absent from I (resp. C_i), and where π_1 is a function that projects the product labelled transition system $[\![C_i || I]\!]$ onto the states of $[\![C_i]\!]$, meaning that each product state (x, y) is mapped to x and each product transition $(x, y) \xrightarrow{a} (x', y')$ is either mapped to $x \xrightarrow{a} x'$ if a is an action of C_i, or ignored otherwise.

- The semi-composition operator enjoys nice properties: (i) $\Pi_I(C_i)$ is behaviourally included in $[\![C_i]\!]$, in the sense that both models have the same initial state and that any transition $x \xrightarrow{a} x'$ of $\Pi_I(C_i)$ is also a transition of $[\![C_i]\!]$; (ii) the number of states in $\Pi_I(C_i)$ is thus less or equal to the number of states in $[\![C_i]\!]$ (the more restrictive the interface, the smaller this number); (iii) if I is the chaos automaton allowing all the actions of C_i, then $\Pi_I(C_i) = [\![C_i]\!]$; (iv) interfaces can be safely minimised using language equivalence or any stronger equivalence.

- But the most important property is the following one: if interface I is correct, then $[\![C_1 || ... || C_n]\!] = [\![C_1]\!] || ... || [\![C_{i-1}]\!] || \Pi_I(C_i) || [\![C_{i+1}]\!] || ... || [\![C_n]\!]$, meaning that $[\![C_i]\!]$ can be safely replaced with $\Pi_I(C_i)$, which is presumably less complex[44], or even with $\mathsf{min}(\Pi_I(C_i))$, because $[\![C_1 || ... || C_n]\!] \approx [\![C_1]\!] || ... || [\![C_{i-1}]\!] || \mathsf{min}(\Pi_I(C_i)) || [\![C_{i+1}]\!] || ... || [\![C_n]\!]$ since $\approx$ is a congruence.

---

[43] This operator was actually named *reduction* in [36], but we prefer the term *semi-composition* later introduced by Krimm & Mounier [50], because the former term often denotes a minimisation operation that is incompletely done, yielding a smaller yet not necessarily minimal result: partial-order reduction, symmetry reduction, tau-confluence reduction, etc.

[44] In some cases [36, Sect. 6], interfaces reduce complexity from exponential to linear.

- Graf & Steffen also address the case of incorrect interfaces by extending $\Pi_I(C_i)$ with *undefinedness predicates* that indicate, for each state, which actions of C_i have been cut off by I. Later, when recombining $\Pi_I(C_i)$ with its environment, the parallel composition operator discharges those predicates corresponding to transitions of C_i that the environment is never ready to synchronise with, and would indeed never fire. If some predicates remain undischarged when the parallel composition is done, then I is incorrect; the user should analyse these predicates to understand why/where I is too restrictive, and restart compositional verification with a modified interface[45].

Figure 1 illustrates the semi-composition of a component C_i with an interface I, both having 0 as their initial state. All transitions of C_i labelled by actions a_0, a_2, b_0, b_2, c_0, and c_2 are cut off, because they never synchronise in the parallel composition with I. The action sets attached to the states 0, 2, and 5 of $\Pi_I(C_i)$ represent the undefinedness predicates; for instance, the set attached to state 0 indicates that transitions labelled by a_0 and a_2 have been cut off in this state.

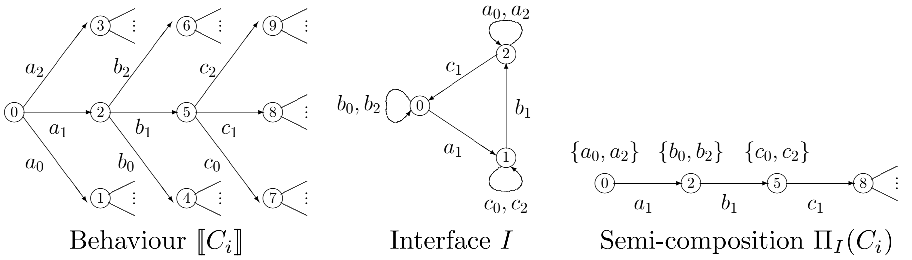


Figure 1: Reduction achieved using semi-composition with an interface

4 Compositional Minimisation with Interfaces but without Semi-Composition

We now examine two approaches that, given a set of asynchronous components $S = C_1||...||C_n$ synchronised by rendezvous, reuse the idea of interfaces to avoid state-space explosion when generating certain components. These approaches do not borrow the semi-composition operator concept, and thus technically differ from the work of Graf & Steffen.

The first approach was proposed by Cheung & Kramer [12] [13] [14] and implemented in the Tracta tool [33]:

- In this approach, a component C_i having an interface I is replaced by $[\![C_i||I]\!]$ instead of being replaced by the semi-composition $\Pi_I(C_i) = \pi_1([\![C_i||I]\!])$.

---

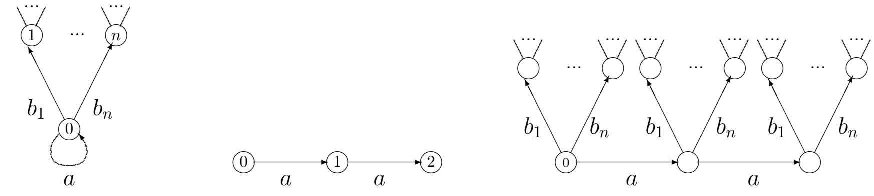
[45] Such an iterative approach based upon incremental refinement was very much the Cegar idea published ten years later [15].

- To ensure that $\llbracket C_1 \rrbracket \, || ... || \, \llbracket C_{i-1} \rrbracket \, || \, \llbracket C_i || I \rrbracket \, || \, \llbracket C_{i+1} \rrbracket \, || ... || \, \llbracket C_n \rrbracket$ is strongly bisimilar to $\llbracket C_1 || ... || C_n \rrbracket$, the interface I must not only be correct in the sense of Graf & Steffen, but also deterministic and free of internal actions[46].
- The initial paper [12] assumes that interfaces are correct without checking for correctness. In [13] [14], the approach is refined as follows to deal with incorrect interfaces. An *output-completion* operation is applied to transform the user-given interface I into an extended interface I': this is done by adding an *undefined* state π and by creating, for each state y of I and each action a not enabled in y, an additional transition $y \xrightarrow{a} \pi$. When computing $S' = \llbracket C_1 \rrbracket \, || ... || \, \llbracket C_{i-1} \rrbracket \, || \, \llbracket C_i || I' \rrbracket \, || \, \llbracket C_{i+1} \rrbracket \, || ... || \, \llbracket C_n \rrbracket$, each transition $(x, y) \xrightarrow{a} (x', \pi)$ of $\llbracket C_i || I' \rrbracket$ should normally disappear (i.e., be blocked) unless it can synchronise with another action a present in the environment of C_i, i.e., $\llbracket C_1 \rrbracket \, || ... || \, \llbracket C_{i-1} \rrbracket \, || \, \llbracket C_{i+1} \rrbracket \, || ... || \, \llbracket C_n \rrbracket$, thus signalling that I is too restrictive. Hence, interface I is correct iff S' contains no reachable state whose i^{th} element has the form (x', π).

The second approach was proposed by Valmari [81] and implemented in the Ara tool [82]. This approach is similar to the one of Cheung & Kramer, with two differences: interfaces are allowed to be nondeterministic, and the user must explicitly introduce the undefined state[47] π in the interface, i.e., provide an interface I' rather than I.

At first sight, these two approaches may look simpler and more elegant than the one of Graf & Steffen, because they do not require a semi-composition operator, but they are actually inferior (although they were published later than [36]), for at least three reasons:

1. Semi-composition is a reduction, meaning that $\Pi_I(C_i)$ is smaller than $\llbracket C_i \rrbracket$, but parallel composition is not. Indeed, $\llbracket C_i || I \rrbracket$ can be (much) larger than $\llbracket C_i \rrbracket$. Figure 2 shows a simple example in which $\llbracket C_i || I \rrbracket$ has three times more states than $\llbracket C_i \rrbracket$. Thus, using interfaces without semi-composition can be counter-productive, keeping in mind that $\llbracket C_i \rrbracket$ is expected to be huge.



Behaviour $\llbracket C_i \rrbracket$ Interface I Composition $\llbracket C_i || I \rrbracket$

Figure 2: Example where $\llbracket C_i || I \rrbracket$ is larger than $\llbracket C_i \rrbracket$

---

[46] Internal actions are usually noted τ in most process calculi.

[47] This state is called *cut state* in [81].

2. The approach of Cheung & Kramer requires nondeterministic interfaces to be determinised [12, Sect. 5.1]. In the worst case, this may cause an exponential blowup in the number of states of the interface (e.g., a small interface with 40 states may get larger than one trillion states), thus compromising the compositional verification approach.

3. The approach of Valmari requires the introduction of the state π and its associated transitions into (possibly nondeterministic) interfaces. No algorithm is provided for such an operation, which might be trivial only for deterministic interfaces — unless determinisation (at the risk of exponential blowup) is first applied to nondeterministic interfaces. Figure 3 shows indeed that the aforementioned output-completion operation works for a deterministic interface I_1, but not for a nondeterministic interface I_2 language-equivalent to I_1: the output-complete interfaces I'_1 and I'_2 are not language-equivalent (e.g., I'_2 accepts a trace $a.b$ ending in π, whereas I'_1 does not).

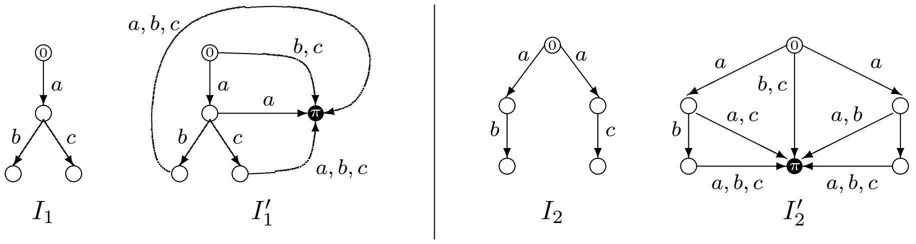


Figure 3: Example where language equivalence is not preserved by output completion

For the sake of completeness, one can mention a third approach [87] that, strictly speaking, does not use interfaces, but simulates their effect by introducing additional synchronisation actions *sleep* and *wake*. From a practical perspective, this approach is not suitable, as it requires to modify the code of components to insert these actions, and also changes the well-established semantic rules for the parallel composition operator, so as to perform look-ahead of *sleep* actions.

5 Compositional Minimisation with Interfaces and Semi-Composition

5.1 Principles

The first (and, to the best of our knowledge, the only) complete implementation of the ideas of Graf & Steffen has been done by Krimm & Mounier [49] [50], who adapted the approach to the case of LOTOS [46]. Such adaptation faces various changes in the base assumptions:

- Graf & Steffen (but also Cheung & Kramer and Valmari) consider the parallel composition operator $||$ of Csp [45], which forces synchronisation on all common actions. On the contrary, the parallel composition operator $|[g_1, ..., g_n]|$ forces synchronisation only on actions whose gate[48] belongs to the (possibly empty) list g_1, ..., g_n, whereas all other actions do not synchronise (i.e., interleave).
- The Lotos operator enables components to have common, yet non-synchronised actions, e.g., between two components C_1 and C_2 executing in full interleaving (i.e., $C_1 \,|[\,]|\, C_2$) and proposing the same actions.
- The Lotos operator also enables nondeterministic synchronisations, e.g., between three components C_1, C_2, and C_3 connected using $(C_1 \,|[\,]|\, C_2) \,|[g]|\, C_3$: any action having gate g proposed by C_3 may synchronise either with C_1 or C_2. This is a most useful pattern to describe pools of clients and servers.
- The parallel operator of Csp is associative whereas, in Lotos, $(C_1 \,|[g]| C_2) \,|[g']|\, C_3$ may be different from $C_1 \,|[g]|\, (C_2 \,|[g']|\, C_3)$ when $g \neq g'$.
- The Lotos operator for action hiding, which was not considered by Graf & Steffen, needs to be taken into account.

In a nutshell, the solution proposed by Krimm & Mounier works as follows:

- Interfaces are labelled transition systems, which can be nondeterministic and contain internal actions (same as in the approach of Graf & Steffen).
- The semi-composition operator $\Pi_I(C_i)$ is generalised to a new operator with four arguments: (i) a component C_i; (ii) an interface I; (iii) a list of gates $g_1, ..., g_n$ on which C_i and I must synchronise; (iv) a Boolean stating whether I is surely correct or possibly incorrect, the former case avoiding correctness checks. The useful properties of $\Pi_I(C_i)$ also hold for this new operator.
- The undefinedness predicates of [36], which are a state-based concept incompatible with labelled transition systems, are encoded by means of *fail-transitions*. In the labelled transition system computed by the semi-composition operator for a possibly incorrect interface I, state s has a self-loop transition $s \xrightarrow{\text{fail}(a)} s$ iff the interface has cut off action a in that state. The parallel composition operator of Lotos is also slightly modified to handle these fail-transitions.

The prototype tools developed by Krimm & Mounier have been rewritten and integrated in Cadp, which has become the reference framework for compositional minimisation techniques [26]. The Des2Aut tool has been subsumed by Svl[49].

---

[48] A Lotos action can be seen as a value tuple, the first element of which is the gate.

[49] `http://cadp.inria.fr/man/svl-lang.html` (see "abstraction")

The Projector tool[50] implements the semi-composition operator; it is built upon the Open/Cæsar application programming interface [22], which enables $\Pi_I(C_i)$ to be computed on the fly, without computing $[\![C_i]\!]$ first (this could cause state-space explosion), and also enables C_i to be expressed in any specification language connected to Open/Cæsar, including Lotos, Lnt, Exp, etc.

5.2 Interface Synthesis

Assume a system $S = C_1||...||C_n$, some components of which are too large to be generated separately from their neighbour components and thus require interfaces. Is it possible to generate automatically and correctly these interfaces, rather than asking the user to provide them, at the risk of human mistakes? This question has been studied in two papers.

The first paper [50] considers a process-algebraic setting in which components are combined inside expressions by means of the Lotos operators for action hiding and parallel composition. An algorithm is given [50, Sect. 3, operator Ψ] to automatically compute an interface for a given component, seen as a sub-expression contained in a larger expression describing the entire system or a part of it. This algorithm works recursively by structural induction on the syntax of Lotos expressions and calculates the set of actions on which the component and its environment have to synchronise.

The second paper [52] considers a more expressive setting, communicating-automata networks, the components of which are combined using synchronisation vectors [1] that can encode action hiding, action renaming, and the parallel composition operators of most process calculi (including Ccs, Csp, μCrl, Lnt, Lotos, etc.) as particular cases. An algorithm is given, which explores the synchronisation graph to compute a correct interface for a given component. This algorithm, which has been implemented in Svl[51], improves over the one of [50] in several respects:

- It is applicable to other process calculi than Lotos.
- It can compute an interface for a component, the environment of which can be arbitrarily chosen to be any subset of components, without requiring these components to be adjacent or closely connected in a process-algebraic expression (this is useful in presence of parallel composition operators that are not associative because they synchronise on different action sets).
- It handles the possible existence of common, yet non-synchronised actions between the component and its environment.
- It handles the possible existence of common, nondeterministically synchronised actions between the component and its environment (i.e., the environment can synchronise on a given action either with the component or

---

[50] `http://cadp.inria.fr/man/projector.html`

[51] `http://cadp.inria.fr/man/svl-lang.html` (see "refined abstraction")

with another component, and vice versa). Such actions are ignored in [50], leading to over-permissive interfaces.

- It can generate, in the case of LOTOS, less permissive interfaces than [50], possibly leading to better reductions [52, Examples 2–3 and Figure 1].

5.3 Applications

Four examples of compositional verification with interfaces and semi-composition are available online, as part of the CADP demos[52]. For these examples (demos No. 20, 33, 37, and 38, which have between 3 and 60 components), both direct generation and compositional minimisation without interfaces fail, but compositional minimisation with interfaces and semi-composition succeeds, the largest intermediate model generated having less than $700,000$ states.

Compositional verification with interfaces and semi-composition, as implemented in CADP, has been used with success in various (mostly industrial) case-studies:

- rel/REL reliable atomic multicast protocol[53] [50], Hewlett-Packard (UK)⋆.
- Distributed leader election for unidirectional ring networks[54] [50]⋆.
- ATC (Air Traffic Control) system[55] [69], Glasgow (UK).
- PolyKid CC-NUMA multiprocessor architecture[56] [32], Bull, Pregnana (IT)⋆.
- ScalAgent's deployment protocol for software components[57] [78]⋆.
- Mutual exclusion protocols for CC-NUMA architectures[58] [55, 56]⋆.
- Asynchronous circuit for the DES (Data Encryption Standard)[59] [74]⋆.
- Asynchronous Memory Protection Unit[60] [10], Tiempo, Grenoble (FR)⋆.

6 Conclusion

Although compositional verification is now thirty years old, and despite its true potential in overcoming state-space explosion (as demonstrated in many convincing case studies), it is not yet a widespread verification technique, and its use for practical problems remains rather an exception than the rule.

A major breakthrough was made in 1990 with a series of papers by Graf, Steffen & Lüttgen [36] [37] [38] [39] [40]. Unfortunately, the merits of these papers are

---

[52] `http://cadp.inria.fr/demos`

[53] `http://cadp.inria.fr/case-studies/91-c-relrel.html`

[54] `http://cadp.inria.fr/case-studies/96-f-leaderelection.html`

[55] `http://cadp.inria.fr/case-studies/99-e-atc.html`

[56] `http://cadp.inria.fr/case-studies/00-c-polykid.html`

[57] `http://cadp.inria.fr/case-studies/03-e-parfums.html`

[58] `http://cadp.inria.fr/case-studies/10-f-mutex.html`

[59] `http://cadp.inria.fr/case-studies/15-f-des.html`

[60] `http://cadp.inria.fr/case-studies/18-1-mpu.html`

not sufficiently understood. Either these papers are not mentioned in surveys [65] [20] [21] or they are merely cited without any further comment on the significance of their contributions [34]. These are injustices the present article tries to remedy.

The approach of [36] relies upon two key concepts: *interfaces* and *semi-composition*. Using these concepts is not mandatory, as it is possible to perform compositional minimisation without interfaces (Sect. 2) or with interfaces but without semi-composition (Sect. 4). We have shown, however, that the best results are obtained when both concepts are taken advantage of (Sect. 5).

The approach of [36] has been generalised to the case of LOTOS and its descendent languages, and fully implemented in the CADP verification toolbox [26] and successfully applied to numerous case studies, the most recent of which [10] shows impressive results, as an asynchronous hardware block containing not less than 660 concurrent processes was fully verified in a few hours by an industry engineer without prior training in formal methods. This is a clear indication that compositional minimisation techniques have reached a maturity level sufficient to enable their use in industry.

Concerning future research, we envision enhanced approaches for interface synthesis so as to generate interfaces automatically in complex cases that, today, must be dealt with manually, as well as applications of the ideas of Graf & Steffen to quantitative verification, including probabilistic, timed, and hybrid systems.

Acknowledgements

The present work has been partly funded by BPI France and FEDER (*Fonds Européen de Développement Economique Régional*) Rhône-Alpes Auvergne under national project SecurIoT-2 supported by the four competitiveness clusters Minalogic, SCS, Systematic Paris-Région, and Derbi.

References

[1] André Arnold. Synchronized Behaviours of Processes and Rational Relations. *Acta Informatica*, 17:21–29, 1982.

[2] Isabelle Attali, Tomás Barros, and Eric Madelaine. Parameterized Specification and Verification of the Chilean Electronic Invoices System. In *Proceedings of the 24th International Conference of the Chilean Computer Science Society (SCCC'04), Arica, Chili*, pages 14–25. Society for Computer Simulation International, IEEE, November 2004.

[3] Simon Bainbridge and Laurent Mounier. Specification and Verification of a Reliable Multicast Protocol. Technical Report HPL-91-163, Hewlett-Packard Laboratories, Bristol, U.K., October 1991.

[4] Tomás Barros, Ludovic Henrio, and Eric Madelaine. Behavioural Models for Hierarchical Components. In Patrice Godefroid, editor, *Proceedings of the 12th International Workshop on Model Checking of Software (SPIN'05), San Francisco, USA*, volume 3639 of *Lecture Notes in Computer Science*, pages 154–168. Springer, August 2005.

[5] Tomás Barros, Ludovic Henrio, and Eric Madelaine. Verification of Distributed Hierarchical Components. In *Proceedings of the International Workshop on Formal Aspects of Component Software (FACS'05), Macao*, Electronic Notes in Theoretical Computer Science, October 2005.

[6] Tomas Barros and Eric Madelaine. Formalization and Proofs of the Chilean Electronic Invoices System. INRIA Research Report 5527, INRIA, June 2004.

[7] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[8] Eckard Böde, Marc Herbstritt, Holger Hermanns, Sven Johr, Thomas Peikenkamp, Reza Pulungan, Ralf Wimmer, and Bernd Becker. Compositional Performability Evaluation for Statemate. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QUEST'06), Riverside, California, USA*, pages 167–178. IEEE Computer Society Press, September 2006.

[9] Alexandre Boulgakov, Thomas Gibson-Robinson, and A. W. Roscoe. Computing Maximal Weak and Other Bisimulations. *Formal Aspects of Computing*, 28(3):381–407, 2016.

[10] Aymane Bouzafour, Marc Renaudin, Hubert Garavel, Radu Mateescu, and Wendelin Serwe. Model-checking Synthesizable SystemVerilog Descriptions of Asynchronous Circuits. In Milos Krstic and Ian W. Jones, editors, *Proceedings of the 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'18), Vienna, Austria*. IEEE, May 2018.

[11] Ghassan Chehaibar, Hubert Garavel, Laurent Mounier, Nadia Tawbi, and Ferruccio Zulian. Specification and Verification of the PowerScale Bus Arbitration Protocol: An Industrial Experiment with LOTOS. In Reinhard Gotzhein and Jan Bredereke, editors, *Proceedings of the IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV'96), Kaiserslautern, Germany*, pages 435–450. Chapman & Hall, October 1996. Full version available as INRIA Research Report RR-2958.

[12] S. C. Cheung and J. Kramer. Enhancing Compositional Reachability Analysis with Context Constraints. In *Proceedings of the 1st ACM SIGSOFT International Symposium on the Foundations of Software Engineering (Los Angeles, CA, USA)*, pages 115–125. ACM Press, December 1993.

[13] S. C. Cheung and J. Kramer. Compositional Reachability Analysis of Finite-State Distributed Systems with User-Specified Constraints. In *Proceedings of the 3rd ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Washington, DC, USA*, pages 140–150. ACM Press, October 1995.

[14] S. C. Cheung and J. Kramer. Context Constraints for Compositional Reachability. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 5(4):334–377, October 1996.

[15] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00), Chicago, IL, USA*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, July 2000.

[16] Pepijn Crouzen and Frédéric Lang. Smart Reduction. In Dimitra Giannakopoulou and Fernando Orejas, editors, *Proceedings of Fundamental Approaches to Software Engineering (FASE'11), Saarbrücken, Germany*, volume 6603 of *Lecture Notes in Computer Science*, pages 111–126. Springer, March 2011.

[17] Anthony de Jacquier, Thierry Massart, and Christian Hernalsteen. Vérification et correction d'un protocole de contrôle aérien. Technical report 363, Université Libre de Bruxelles, May 1997.

[18] Jean-Claude Fernandez. *ALDEBARAN : un système de vérification par réduction de processus communicants.* Thèse de Doctorat, Université Joseph Fourier (Grenoble), May 1988.

[19] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A Toolbox for the Verification of LOTOS Programs. In Lori A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering (ICSE'14), Melbourne, Australia*, pages 246–259. ACM, May 1992.

[20] Jaroslav Fogel. A Survey of Verification Techniques for Solving the State Explosion Problem. In *Proceedings of the IFAC Conference on Control Systems Design (CSD'00), Bratislava, Slovak Republic, IFAC Proceedings Volumes, 33(13)*, pages 361–366, June 2000.

[21] Carlo Alberto Furia. A Compositional World: A Survey of Recent Works on Compositionality in Formal Methods. Technical Report 2005.22, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, March 2005.

[22] Hubert Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In Bernhard Steffen, editor, *Proceedings*

of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84. Springer, March 1998. Full version available as INRIA Research Report RR-3352.

[23] Hubert Garavel and Susanne Graf. Formal Methods for Safe and Secure Computers Systems. BSI Study 875, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany, December 2013.

[24] Hubert Garavel and Holger Hermanns. On Combining Functional Verification and Performance Evaluation using CADP. In Lars-Henrik Eriksson and Peter A. Lindsay, editors, *Proceedings of the 11th International Symposium of Formal Methods Europe (FME'02), Copenhagen, Denmark*, volume 2391 of *Lecture Notes in Computer Science*, pages 410–429. Springer, July 2002. Full version available as INRIA Research Report 4492.

[25] Hubert Garavel and Frédéric Lang. SVL: a Scripting Language for Compositional Verification. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'01), Cheju Island, Korea*, pages 377–392. Kluwer Academic Publishers, August 2001. Full version available as INRIA Research Report RR-4223.

[26] Hubert Garavel, Frédéric Lang, and Radu Mateescu. Compositional Verification of Asynchronous Concurrent Systems Using CADP. *Acta Informatica*, 52(4):337–392, April 2015.

[27] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 15(2):89–107, April 2013.

[28] Hubert Garavel, Frédéric Lang, and Wendelin Serwe. From LOTOS to LNT. In Joost-Pieter Katoen, Rom Langerak, and Arend Rensink, editors, *ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, volume 10500 of *Lecture Notes in Computer Science*, pages 3–26. Springer, October 2017.

[29] Hubert Garavel and Laurent Mounier. Specification and Verification of Various Distributed Leader Election Algorithms for Unidirectional Ring Networks. *Science of Computer Programming*, 29(1–2):171–197, July 1997. Special issue on Industrially Relevant Applications of Formal Analysis Techniques. Full version available as INRIA Research Report RR-2986.

[30] Hubert Garavel and Mihaela Sighireanu. A Graphical Parallel Composition Operator for Process Algebras. In Jianping Wu, Qiang Gao, and Samuel T. Chanson, editors, *Proceedings of the IFIP Joint International*

Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV'99), Beijing, China, pages 185–202. Kluwer Academic Publishers, October 1999.

[31] Hubert Garavel and Damien Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In Corina Pasareanu, editor, *Proceedings of the 16th International SPIN Workshop on Model Checking of Software (SPIN'09), Grenoble, France*, volume 5578 of *Lecture Notes in Computer Science*, pages 241–260. Springer, June 2009.

[32] Hubert Garavel, César Viho, and Massimo Zendri. System Design of a CC-NUMA Multiprocessor Architecture using Formal Specification, Model-Checking, Co-Simulation, and Test Generation. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 3(3):314–331, July 2001. Also available as INRIA Research Report RR-4041.

[33] Dimitra Giannakopoulou. *Model Checking for Concurrent Software Architectures.* PhD thesis, Imperial College of Science, Technology and Medicine, University of London, Department of Computer Science, January 1999.

[34] Dimitra Giannakopoulou, Kedar S. Namjoshi, and Corina S. Pasareanu. Compositional Reasoning. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 345–383. Springer, 2018.

[35] Gavril Godza, Valentin Cristea, and Radu Mateescu. Formal Specification of Checkpointing Algorithms. In *Proceedings of 13th International Conference on Control Systems and Computer Science (CSCS'13), Bucharest, Romania*, pages 311–317. Polytechnic University of Bucharest, May 2001.

[36] Susanne Graf and Bernhard Steffen. Compositional Minimization of Finite State Systems. In Edmund M. Clarke and Robert P. Kurshan, editors, *Proceedings of the 2nd Workshop on Computer-Aided Verification (CAV'90), Rutgers, New Jersey, USA*, volume 531 of *Lecture Notes in Computer Science*, pages 186–196. Springer, June 1990.

[37] Susanne Graf and Bernhard Steffen. Compositional Minimization of Finite State Systems. Aachener Informatik-Berichte AIB 1991-23, RWTH Aachen University, Department of Computer Science, Germany, 1991.

[38] Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional Minimization of Finite State Systems Using Interface Specifications. Research Report MIP-9505, Universität Passau, Fakultät für Mathematik und Informatik, Germany, 1995.

[39] Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional Minimization of Finite State Systems Using Interface Specifications. *10-page article published in the paper version of the journal "Formal Aspects of Computing"*, 8(5):607–616, September 1996.

[40] Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional Minimization of Finite State Systems using Interface Specifications. *28-page article published in the electronic repository of the journal "Formal Aspects of Computing"*, 8E:286–313, September 1996. URL: `http://static-content.springer.com/esm/art%3A10.1007%2FBF01211911/MediaObjects/165_2005_BF01211911_MOESM1_ESM.pdf`.

[41] Ji He and Kenneth J. Turner. Specification and Verification of Synchronous Hardware using LOTOS. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Proceedings of the IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing, and Verification (FORTE/PSTV'99), Beijing, China*, pages 295–312. Kluwer Academic Publishers, October 1999.

[42] Holger Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.

[43] Holger Hermanns and Joost-Pieter Katoen. Automated Compositional Markov Chain Generation for a Plain-Old Telephone System. *Science of Computer Programming*, 36:97–127, 2000.

[44] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[45] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[46] ISO/IEC. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva, September 1989.

[47] Alain Kerbrat and Slim Ben Atallah. Formal Specification of a Framework for Groupware Development. In Gregor v. Bochmann, Rachida Dssouli, and Omar Rafiq, editors, *Proceedings of the 8th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'95), Montreal, Quebec, Canada*, IFIP Advances in Information and Communication Technology, pages 303–310. Springer, October 1995.

[48] Fabrice Kordon, Hubert Garavel, Lom Messan Hillah, Emmanuel Paviot-Adet, Loïg Jezequel, Francis Hulin-Hubard, Elvio Amparore, Marco Beccuti, Bernard Berthomieu, Hugues Evrard, Peter G. Jensen, Didier Le Botlan, Torsten Liebke, Jeroen Meijer, Jiří Srba, Yann Thierry-Mieg, Jaco van de Pol, and Karsten Wolf. MCC'2017 – The Seventh Model Checking Contest. *Transactions on Petri Nets and Other Models of Concurrency*, 2018.

[49] Jean-Pierre Krimm. Une approche compositionnelle pour la vérification de programmes LOTOS. Master's thesis, Université Joseph Fourier (Grenoble), June 1996.

[50] Jean-Pierre Krimm and Laurent Mounier. Compositional State Space Generation from LOTOS Programs. In Ed Brinksma, editor, *Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), University of Twente, Enschede, The Netherlands*, volume 1217 of *Lecture Notes in Computer Science*. Springer, April 1997. Extended version with proofs available as Research Report VERIMAG RR97-01.

[51] Frédéric Lang. Compositional Verification using SVL Scripts. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02), Grenoble, France*, volume 2280 of *Lecture Notes in Computer Science*, pages 465–469. Springer, April 2002.

[52] Frédéric Lang. Refined Interfaces for Compositional Verification. In Elie Najm, Jean-François Pradat-Peyre, and Véronique Viguié Donzeau-Gouge, editors, *Proceedings of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'06), Paris, France*, volume 4229 of *Lecture Notes in Computer Science*, pages 159–174. Springer, September 2006. Full version available as INRIA Research Report RR-5996.

[53] Matti Luukkainen and Ari Ahtiainen. Compositional Verification of Large SDL Systems. In *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM'98), Berlin, Germany*, June 1998.

[54] J. Malhotra, S. A. Smolka, A. Giacalone, and R. Shapiro. A Tool for Hierarchical Design and Simulation of Concurrent Systems. In *Proceedings of the BCS-FACS Workshop on Specification and Verification of Concurrent Systems, Stirling, Scotland, UK*, pages 140–152. British Computer Society, July 1988.

[55] Radu Mateescu and Wendelin Serwe. A Study of Shared-Memory Mutual Exclusion Protocols using CADP. In Stefan Kowalewski and Marco Roveri, editors, *Proceedings of the 15th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'10), Antwerp, Belgium*, volume 6371 of *Lecture Notes in Computer Science*, pages 180–197. Springer, September 2010.

[56] Radu Mateescu and Wendelin Serwe. Model Checking and Performance Evaluation with CADP Illustrated on Shared-Memory Mutual Exclusion Protocols. *Science of Computer Programming*, 78(7):843–861, July 2013.

[57] Franco Mazzanti and Alessio Ferrari. Ten Diverse Formal Models for a CBTC Automatic Train Supervision System. In John P. Gallagher, Rob van Glabbeek, and Wendelin Serwe, editors, *Proceedings of the 3rd Workshop on Models for Formal Analysis of Real Systems and the 6th International Workshop on Verification and Program Transformation (MARS/VPT'18),*

Thessaloniki, Greece, volume 268 of *Electronic Proceedings in Theoretical Computer Science*, pages 104–149, April 2018.

[58] Franco Mazzanti, Alessio Ferrari, and Giorgio Oronzo Spagnolo. Towards Formal Methods Diversity in Railways: An Experience Report with Seven Frameworks. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 20(3):263–288, 2018.

[59] Nuno Mendes, Frédéric Lang, Yves-Stan Le Cornec, Radu Mateescu, Grégory Batt, and Claudine Chaouiya. Composition and Abstraction of Logical Regulatory Modules: Application to Multicellular Systems. *Bioinformatics*, 29(6):749–757, March 2013.

[60] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980.

[61] Laurent Mounier. A LOTOS Specification of a Transit-Node. Rapport SPECTRE 94-8, VERIMAG, Grenoble, March 1994.

[62] Raquel Oliveira, Sophie Dupuy-Chessa, Gaelle Calvary, and Danielle Dadolle. Using Formal Models to Cross Check an Implementation. In Kris Luyten and Philippe Palanque, editors, *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'16), Brussels, Belgium*, pages 126–137. ACM, June 2016.

[63] David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, March 1981.

[64] Charles Pecheur. Advanced Modelling and Verification Techniques Applied to a Cluster File System. In Robert J. Hall and Ernst Tyugu, editors, *Proceedings of the 14th IEEE International Conference on Automated Software Engineering (ASE'99) Cocoa Beach, Florida, USA*. IEEE Computer Society, October 1999. Extended version available as INRIA Research Report RR-3416.

[65] Hong Peng and Sofiène Tahar. A Survey on Compositional Verification. Technical report, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, November 1998.

[66] Willem-Paul de Roever, Frank de Boer, Ulrich Hanneman, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification – Introduction to Compositional and Noncompositional Methods*, volume 54 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, November 2001.

[67] Judi Romijn. *Analysing Industrial Protocols with Formal Methods.* PhD thesis, University of Twente, The Netherlands, September 1999.

[68] Krishan K. Sabnani, Aleta M. Lapone, and M. Ümit Uyar. An Algorithmic Procedure for Checking Safety Properties of Protocols. *IEEE Transactions on Communications*, 37(9):940–948, September 1989.

[69] Meurig Sage and Chris Johnson. A Declarative Prototyping Environment for the Development of Multi-User Safety-Critical Systems. In *Proceedings of the 17th International System Safety Conference (ISSC'99) Orlando, Florida, USA*. System Safety Society, August 1999.

[70] Gwen Salaün and Tevfik Bultan. Realizability of Choreographies using Process Algebra Encodings. In Michael Leuschel and Heike Wehrheim, editors, *Proceedings of the 7th International Conference on Integrated Formal Methods (IFM'09), Düsseldorf, Germany*, volume 5423 of *Lecture Notes in Computer Science*, pages 167–182. Springer, February 2009.

[71] Gwen Salaün and Wendelin Serwe. Translating Hardware Process Algebras into Standard Process Algebras – Illustration with CHP and LOTOS. In Judi Romijn, Graeme Smith, and Jaco van de Pol, editors, *Proceedings of the 5th International Conference on Integrated Formal Methods (IFM'05), Eindhoven, The Netherlands*, volume 3771 of *Lecture Notes in Computer Science*, pages 287–306. Springer, November 2005. Full version available as INRIA Research Report RR-5666.

[72] Gwen Salaün, Wendelin Serwe, Yvain Thonnart, and Pascal Vivet. Formal Verification of CHP Specifications with CADP – Illustration on an Asynchronous Network-on-Chip. In Peter Beerel, Marly Roncken, Mark Greenstreet, and Montek Singh, editors, *Proceedings of the 13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'07), Berkeley, California, USA*, pages 73–82. IEEE Computer Society Press, March 2007.

[73] Ina Schieferdecker. Abruptly-Terminated Connections in TCP – A Verification Example. In Z. Brezočnik and T. Kapus, editors, *Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design, Maribor, Slovenia*, pages 136–145. University of Maribor, Slovenia, June 1996.

[74] Wendelin Serwe. Formal Specification and Verification of Fully Asynchronous Implementations of the Data Encryption Standard. In Rob van Glabbeek, Jan Friso Groote, and Peter Höfner, editors, *Proceedings of the International Workshop on Models for Formal Analysis of Real Systems (MARS'15), Suva, Fiji*, volume 196 of *Electronic Proceedings in Theoretical Computer Science*, 2015.

[75] Kuo-Chung Tai and Pramod V. Koppol. An Incremental Approach to Reachability Analysis of Distributed Programs. In *Proceedings of the 7th International Workshop on Software Specification and Design, Los Angeles, CA, USA*, pages 141–150, Piscataway, NJ, December 1993. IEEE Press.

[76] Kuo-Chung Tai and Pramod V. Koppol. Hierarchy-Based Incremental Reachability Analysis of Communication Protocols. In *Proceedings of the IEEE International Conference on Network Protocols, San Francisco, CA, USA*, pages 318–325, Piscataway, NJ, October 1993. IEEE Press.

[77] Larry Tan. Case Studies Using CRESS to Develop Web and Grid Services. Technical report, Department of Computing Science and Mathematics, University of Stirling, Scotland, UK, December 2009.

[78] Frédéric Tronel, Frédéric Lang, and Hubert Garavel. Compositional Verification Using CADP of the ScalAgent Deployment Protocol for Software Components. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *Proceedings of the 6th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'03), Paris, France*, volume 2884 of *Lecture Notes in Computer Science*, pages 244–260. Springer, November 2003. Full version available as INRIA Research Report RR-5012.

[79] Antti Valmari. Compositional State Space Generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993 – Papers from the 12th International Conference on Applications and Theory of Petri Nets (ICATPN'91), Gjern, Denmark*, volume 674 of *Lecture Notes in Computer Science*, pages 427–457. Springer, 1993.

[80] Antti Valmari. Compositionality in State Space Verification Methods. In Jonathan Billington and Wolfgang Reisig, editors, *Proceedings of the 17th International Conference on Application and Theory of Petri Nets (ICATPN'96), Osaka, Japan*, volume 1091 of *Lecture Notes in Computer Science*, pages 29–56. Springer, June 1996.

[81] Antti Valmari. Composition and Abstraction. In Franck Cassez, Claude Jard, Brigitte Rozoy, and Mark Dermot Ryan, editors, *Proceedings of the 4th Summer School on Modeling and Verification of Parallel Processes (MOVEP'00), Nantes, France*, volume 2067 of *Lecture Notes in Computer Science*, pages 58–98. Springer, June 2000.

[82] Antti Valmari, Jukka Kemppainen, Matthew Clegg, and Mikko Levanto. Putting Advanced Reachability Analysis Techniques Together: the ARA Tool. In Jim Woodcock and Peter Gorm Larsen, editors, *Proceedings of the 1st International Symposium of Formal Methods Europe (FME'93), Odense, Denmark*, volume 670 of *Lecture Notes in Computer Science*, pages 597–616. Springer, April 1993.

[83] Antti Valmari and Ilkka Kokkarinen. Unbounded Verification Results by Finite-State Compositional Techniques: 10^{any} States and Beyond. In *Proceedings of the 1st International Conference on Application of Concurrency to System Design (ACSD'98), Fukushima, Japan*, pages 75–85. IEEE Computer Society, March 1998.

[84] Rob J. van Glabbeek and W. Peter Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[85] Tim Willemse, Jan Tretmans, and Arjen Klomp. A Case Study in Formal Methods: Specification and Validation of the OM/RR Protocol. In Stefania Gnesi, Ina Schieferdecker, and Axel Rennoch, editors, *Proceedings of the 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'00), Berlin, Germany*, GMD Report 91, pages 331–344, Berlin, April 2000.

[86] Tim A.C. Willemse. The Specification and Validation of the OM/RR-Protocol. Master's thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, June 1998.

[87] Wei Jen Yeh. *Controlling State Explosion in Reachability Analysis.* PhD thesis, Software Engineering Research Center (SERC) Laboratory, Purdue University, December 1993. Technical Report SERC-TR-147-P.

[88] Wei Jen Yeh and Michal Young. Compositional Reachability Analysis Using Process Algebra. In *Proceedings of the ACM SIGSOFT Symposium on Testing, Analysis, and Verification (SIGSOFT'91), Victoria, British Columbia, Canada*, pages 49–59. ACM Press, October 1991.

[89] Zhen Zhang, Wendelin Serwe, Jian Wu, Tomohiro Yoneda Hao Zheng, and Chris Myers. An Improved Fault-Tolerant Routing Algorithm for a Network-on-Chip Derived with Formal Analysis. *Science of Computer Programming*, 118:24–39, 2016.