Is CADP an Applicable Formal Method?

Hubert Garavel Frédéric Lang Radu Mateescu Wendelin Serwe
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP[*], LIG, 38000 Grenoble, France
{Hubert.Garavel,Frederic.Lang,Radu.Mateescu,Wendelin.Serwe}@inria.fr

CADP is a comprehensive toolbox implementing results of concurrency theory. This paper addresses the question, whether CADP qualifies as an applicable formal method, based on the experience of the authors and feedback reported by users.

1 Introduction

Formal Methods [17, 16] are a wide spectrum of languages, techniques, and tools, strongly rooted in theory, for the design, analysis, and validation of systems. CADP (Construction and Analysis of Distributed Processes)[1] [21] is a comprehensive toolbox that implements the results of concurrency theory. Started in the mid 80's, CADP initially consisted of only two tools. Over the past 35 years, CADP has been continuously improved and extended, and contains now more than 50 tools and almost 20 software components. CADP offers functionalities covering the entire design cycle of asynchronous concurrent systems: specification, interactive simulation, rapid prototyping, verification, testing, and performance evaluation. For verification, CADP supports the three essential approaches existing in the field: model checking, equivalence checking, and visual checking. To deal with complex systems, CADP implements a wide range of verification techniques (reachability analysis, static analysis, on-the-fly, compositional, and distributed verification) and provides a scripting language for describing elaborate verification scenarios. In addition, CADP supports many different specification languages.

CADP benefits from a worldwide user community: The CADP web site lists more than 200 published case studies in numerous application domains and more than 100 published third-party tools. Most of them were not carried out by the authors of the toolbox. Some case studies were conducted with only a minimal amount of training and/or consulting, by almost novice users of formal methods [4]. Many applications of CADP were carried out by industrial companies, including Airbus, Bull, Crouzet/Innovista Sensors, Google, Nokia Bell Labs, Orange Labs, STMicroelectronics, and Tiempo Secure.

The rest of this paper is organized as follows: Sections 2 to 9 address the questions listed in the topics of interest[2] of the appFM 2021 workshop. The answers to each question are not only based on our own experience in teaching as well as academic and industrial collaborative projects, but also on feedback reported in publications on case studies and third-party tools connected to CADP. Section 10 concludes the discussion with additional criteria considered important by the authors.

2 Applicability

αFM: *How can the approach be applied in practice?*

CADP is used by mainly two audiences. The first one consists of students learning concurrency theory

[*]Institute of Engineering Univ. Grenoble Alpes
[1]http://cadp.inria.fr
[2]https://sites.google.com/view/appfm21/scope

Submitted to:
appFM 2021

concepts, for which CADP provides concrete representations (concurrent processes, synchronization and communication, temporal logic formulas, etc.). For example, the abstract notion of automaton is instantiated as files in the BCG (Binary Coded Graphs) format, which can be visualized, minimized modulo bisimulation relations, modified by hiding or renaming transition labels, etc.

The second audience consists of scientists or industry engineers, who build complex and critical systems, and strive to follow a rigorous design process and to obtain guarantees of correct functioning and performance. Here, CADP assists in all the main design phases: specification (modeling concurrent processes and their interactions), design-time analysis (verification and performance evaluation), validation of an existing implementation (co-simulation, conformance test generation). Design-time analysis of a formal model has the advantage of early error detection, which significantly reduces the cost of errors.

Among the case studies listed on the CADP web site, the tools are most frequently used for the formal specification and modeling of a system, which are then verified by model checking temporal logic properties and/or equivalence checking against a reference model (e.g., the expected service of a protocol). Some case studies additionally take advantage of more specific tools, e.g., for performance evaluation [18, 3, 11, 7, 9, 14, 45, 55], (conformance) test generation [33, 12, 28, 51, 54, 8, 29, 41, 35, 5], or generation of an executable prototype from the formal model [12, 28, 53, 27, 41, 52, 26].

3 Automation

αFM: *Which tool support is proposed? If abstraction is needed, how is it automated?*

Tool support has always been central in CADP, which is a toolbox rather than a monolithic tool supporting a single language and methodology (e.g., the B-method). CADP has completely automated tools for simulation, for which abstraction is not desirable, because an accurate model is needed.

Concerning enumerative verification, although often advertised as a push-button technique, practice shows that, like for theorem proving, a fair amount of human intervention may be needed (to fight state space explosion). For now, not everything is automatable, nor will it be in a foreseeable future. One still relies on experts, on their knowledge and competence to select an appropriate verification strategy. In general, on-the-fly techniques are used to quickly find bugs in initial versions of a model, whereas more involved strategies may be better to generate an abstract model (reduced for an appropriate bisimulation), on which temporal logic properties are to be verified formally. The most prominent technique implemented in CADP is compositional verification, which is a divide-and-conquer approach applying compositions and abstractions incrementally. CADP offers a collection of known abstractions, which are generic recipes that succeed in many cases. But, for some particularly involved problems, one must design specific abstractions after a careful analysis of the problem [20, 39, 40]. Compositional verification is supported by SVL (Script Verification Language) [19], providing, among others, also an automatic heuristic, called smart reduction [10], which selects an order for compositions. In the context of semi-composition [34], the EXP.OPEN tool [36] can automatically compute behavioral interfaces [37] and check the correctness of manually provided ones. Taking advantage of the integration of shell commands in SVL, CEGAR (Counter-Example Guided Abstraction Refinement)-style loops have been used to automatically reduce the model [35].

Turning informal specifications into formal models and/or properties is hardly automatable. However, the modeling languages of CADP ease this step by supporting concurrent programs with complex and/or dynamic data structures (records, unions, lists, trees, etc.). These languages are also convenient targets to compile domain-specific modeling languages [21, Figure 4]: dedicated compilers have been developed for instance for FSP [49], CHP [25], RT-UML [50], the (applied) π-calculus [44], dynamic fault trees [30], or AADL [48]. It is worth noting that the asynchronous subset of SystemVerilog can be translated almost line-by-line into LNT [4, Fig. 2].

In some cases, the rich data domains must be reduced by abstraction. Although not automatic, this is facilitated by the possibility to redefine basic data types [52] or adding constraints [4, Section V.C.1].

4 Integration

αFM: *If several approaches are to be applied in an integrated way; if the approach is to be used with a modeling technique or programming language; if the approach is to be integrated into an engineering process, what are the benefits?*

CADP is modular, providing many tools, libraries, formats, and languages. The formats and languages cover different abstraction levels (from process calculi to explicit automata descriptions for models, and from MCL formulas to XTL programs for properties). The tools and libraries provide well-defined functionalities, which can be combined and interconnected in various ways to improve the overall analysis approach by fighting state space explosion. The OPEN/CÆSAR architecture [15] separates language-dependent and language-independent aspects and has been identified as a key for smooth integration ("*the OPEN/CÆSAR interface has been underlying the success of CADP*" [2]).

CADP tools can be combined in several ways, and at various levels, e.g., through verification scripts, translations between languages, and tool interconnection using application programming interfaces, e.g., OPEN/CÆSAR, BCG, etc. Data-handling C code (e.g., existing optimized implementations of complex data structures, such as the directories of a cache-coherence protocol) can be called from the models. CADP provides comprehensive documentation for all tools and libraries, in the form of manual pages, totaling more than 800 pages. The benefit of this open, documented architecture is witnessed by the numerous third-party tools, which, taking advantage of functionalities offered by CADP, provide domain specific tools or particular verification features. This architecture also simplifies the implementation of new prototypes [38, 42].

5 Scalability

αFM: *How can the approach be applied at scale, for example, using composition and refinement?*

The main limiting factor for enumerative verification is the amount of available memory. Thus, the CADP tools and libraries are optimized to reduce memory usage before execution time. CADP is implemented mostly using shell scripts and the C programming language, which enables a fine control on the memory usage, enabling low-level optimization to reduce the memory footprint of each state at bit-level. The most prominent example is the BCG format, introduced in 1994, which enables compact storage of automata with up to 10^{13} states and transitions.

To benefit from the combined memory available in clusters and grids, CADP provides distributed tools for state space generation [23, 24] (possibly combined with on-the-fly reductions) and resolution of Boolean equation systems [32], into which a wide range of verification problems can be encoded.

However, compositional techniques [20], the most prominent among which is compositional state space construction, are the major asset of CADP for scalability. "*The advantage of using compositional construction in terms of space and time is apparent. Stepwise minimization keeps the size of state spaces low. This, in turns, reduces the duration of the minimization time in the next step, and so on, thus saving significant amount of time.*" [3, Section VI.C] Compositional techniques rely on the compositional properties of process calculi and the fact that interesting equivalences are congruences for the principal composition operators. CADP also relies on adequacy results between automata equivalences and temporal logics. Besides compositional construction, model checking can also be applied compositionally. This technique, called partial model checking, has been implemented as a companion tool to CADP [38]. Having access to many verification strategies allowed to win gold medals in the parallel tracks of the 2019 and 2020 editions of the RERS challenge (Rigorous Examination of Reactive Systems).[3]

In a comparison of formal verification tools for a railway problem [47], CADP ranked among both the fastest tools and among those with the lowest memory requirements.

6 Transfer

αFM: *How is teaching or training to be organized to transfer the approach?*

One of the most considered research priorities for formal methods research is the applicability and acceptability of tools [16]. Indeed, formal methods are reputed to have a steep learning curve hindering their easy acceptance outside academia. Since many years, the input languages of CADP are continuously being improved to flatten the learning curve, so as to enable our industrial partners to model their confidential systems in-house with as little guidance as possible.

Since 2010, the recommended modeling language of CADP is LNT [6], which is as close as possible to languages known by engineers and students [22], using a familiar syntax rather than algebraic notations, enabling a novice user to build upon known notions and to focus on important concepts.

For temporal logics, CADP provides MCL (Model Checking Language) [46], which takes advantage of regular expressions and offers the possibility of macro definitions. Together with a rich set of macro libraries, MCL hides most complex constructions (in particular fixed-point operators) from the user.

7 Usefulness

αFM: *How is usefulness achieved? Is the approach effective? What would have been different if a conventional or non-formal alternative was used (e.g., relative fault-avoidance or fault-detection effectiveness)?*

Usefulness of CADP is witnessed by the more than 200 case studies and 100 research tools published. Some case studies explicitly confirm that formal analyses are effective in the sense that they enable errors to be detected earlier: "*Architects detected a limitation in the IP [...]. This limitation manifests in a subset of the counterexamples for the data integrity property we verified 20 months before.*" [35, Section 6.2] "*It was agreed that the use of formal methods clearly increased the quality of the design, by detecting errors and by showing that, after correction, these errors would disappear from the modified versions of the cache coherency protocol.*" [28]

Effectiveness can even be improved by leveraging the modeling effort over several activities, e.g., formal verification, performance evaluation, and conformance testing. For the latter, it yields better coverage and uncovers bugs: "*We should notice that our 306 extracted tests trigger those checks 16 times, whereas the other tests of the [...] test library never trigger these checks*" and "*we observe that the coverage of the verification plan increased significantly and that the coverage of the svunit part of the verification plan is complete (100%), i.e., all the aspects corresponding to system-level behaviors are*

[3] `http://rers-challenge.org`

tested.” [35, Section 6.2] “*The time spent in specifying the Bull's CC_NUMA architecture, formalizing test purposes and generating the test cases with TGV is completely paid by the better correctness and the confidence to put in the implementation. This approach permitted to detect 5 bugs.*” [33]

The capability to generate counterexamples and witnesses, common to all enumerative techniques, is also judged an interesting asset. “*Another important advantage of using CADP is that, when a property does not hold, the model checking algorithm generates a counter-example, i.e., an execution trace leading to a state in which the property is violated. This ability to generate counterexamples can be exploited to pinpoint the cause of an error and possibly correct it.*” [43]

8 Ease of Use

αFM: *How is ease of use achieved? Is the approach efficient? How was its usability and maturity assessed (e.g., abstraction effort, proof complexity, productivity) and what are the results?*

The transfer-facilitating aspects mentioned in Section 6, in particular concerning the input languages, are also crucial for the ease of use. Process calculi and concurrency theory are useful, but notoriously difficult to grasp. One goal of CADP was to make a shift from abstract mathematics to concrete computer science, and a significant effort was invested in this direction:

For modeling languages, LOTOS [31] (based on algebra for both processes and data types) was progressively replaced by LNT (combining imperative and functional programming), which is closer to the practical needs and intuitive for users. 80% of LNT concepts are well-known to programmers, only 20% being related to concurrency (e.g., processes, parallel composition, and rendezvous). Moreover, functions and processes have a unified syntax, which makes a great step forward from previous languages based on process calculi [22]: “*Although modeling the DTD in a classical formal specification language, such as LOTOS [31], is theoretically possible, using LNT made the development of a formal model practically feasible. In particular, features such as predefined array data-types, loops, and modifiable variables helped to obtain a model easily understandable by hardware architects.*” [41, Introduction].

For tools, besides the command-line syntax documented in manual pages, CADP provides for more novice users a graphical user interface with contextual menus for easy invocation of the various tools: “*Through the Xeuca interface (see Figure 4) [the] CADP toolbox allows an easy access to the offered functionalities.*” [1, Section 5.3]. The SVL language enables a versatile description of complex verification scenarios, applies automatically various heuristics for combating state explosion, and provides support for batch-mode execution of verification experiments. “*CADP provides a scripting language, SVL [19], which is particularly convenient to experiment with different strategies to alternate construction and minimization steps.*” [3, Section V]

In [47], the authors highlight among the advantages of CADP the imperative modeling style of LNT fitting their state-machine oriented representation and the expressive power of the property definition language MCL (that subsumes both LTL, albeit at a non-linear cost, and CTL). “*CADP [tools] can be used in their default configuration without having to specify any particular evaluation choice.*” [47]

9 Evaluation

αFM: *Why will the approach be useful for a wide range of critical applications?*

Being based on the generic concepts of concurrency theory, and due to its flexible architecture and open interfaces, CADP has already been used in various application domains, including typical critical systems (for instance, avionics, autonomous vehicles, cryptography, embedded software, hardware design, railways, and security), but also more exotic applications, such as human cognitive processes [53].

To further promote the use of formal methods, we make results of some case studies available to the research community, by contributing models in various forms as challenges to competitions (such as the *Model Checking Contest*[4]), repositories (such as the *Models for Formal Analysis of Real Systems* repository[5]), or benchmarks.[6]

10 Conclusion

CADP is an efficient and usable verification toolbox developed on and for the long run by a small team of researchers, with limited resources. It was always meant to be a *true* software, intended for real users, rather than a mere research prototype to accompany publications. This has implications beyond the questions addressed before:

Software primacy. There is a true difference between a scientific publication and a software tool. Ideally, a publication is a fixed artefact, which is rarely updated after print. On the other hand, software maintenance represents the largest part of the software life-cycle: without maintenance, the software tool soon deprecates and becomes unusable, as the processors, operating systems, and software libraries continue their evolution.

Stability. Learning the CADP tools is an investment from our users, and we want to preserve such investment over the years. All the prototype tools we develop are not automatically integrated in CADP. But once a tool is integrated, this means that its functionality has been found to be useful, and we try to maintain it over the years. Sometimes, certain tools are eliminated (e.g., ALDEBARAN), but we replace them with equivalent tools and shell scripts that preserve backward compatibility.

Testing. We have accumulated hundreds of thousands of artefacts (programs, models, formulas, automata, etc.) that are routinely used to test the quality and stability of CADP after each modification.

Documentation. Each tool and library is documented by a detailed manual page. CADP comes with a set of demonstration examples covering various application domains, a collection of frequently asked questions, and a user forum[7] to create a user community and archive answers to specific questions in an easily accessible way. Besides easing transfer, these features also make CADP popular for teaching. There is no textbook, since it would give a frozen vision, as the toolbox is evolving (still keeping compatibility with previous versions). Instead, all resources are available on the CADP web site.

We conclude with words of CADP users: "*We exploit CADP since it is a popular toolbox maintained, regularly improved, and used in many industrial projects, as a verification framework.*" [43] The overall usability is so good that "*main barriers [to a more widespread inclusion] are the limited support for development functionalities, such as traceability, and other process-integration features.*" [13]

[4] `https://mcc.lip6.fr/`
[5] `http://www.mars-workshop.org/repository.html`
[6] `http://cadp.inria.fr/resources`
[7] `http://cadp.forumotion.com`

References

[1] Rabea Ameur-Boulifa, Ana Cavalli & Stephane Maag (2020): *From Formal Test Objectives to TTCN-3 for Verifying ETCS Complex Software Control Systems*. In Marten van Sinderen & Leszek A. Maciaszek, editors: *Software Technologies*, Springer International Publishing, Cham, pp. 156–178, doi:`10.1007/978-3-030-52991-8_8`.

[2] Stefan Blom, Jaco van de Pol & Michael Weber (2010): *LTSmin: Distributed and Symbolic Reachability*. In Tayssir Touili, Byron Cook & Paul Jackson, editors: *Proceedings of the 22nd International conference on Computer Aided Verification CAV 2010 (Edinburgh, UK)*, *Lecture Notes in Computer Science* 6174, Springer, pp. 354–359, doi:`10.1007/978-3-642-14295-6_31`.

[3] Eckard Böde, Marc Herbstritt, Holger Hermanns, Sven Johr, Thomas Peikenkamp, Reza Pulungan, Ralf Wimmer & Bernd Becker (2006): *Compositional Performability Evaluation for Statemate*. In: *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QUEST'06), Riverside, California, USA*, IEEE Computer Society Press, pp. 167–178, doi:`10.1109/QEST.2006.10`.

[4] Aymane Bouzafour, Marc Renaudin, Hubert Garavel, Radu Mateescu & Wendelin Serwe (2018): *Model-checking Synthesizable SystemVerilog Descriptions of Asynchronous Circuits*. In Milos Krstic & Ian W. Jones, editors: *Proceedings of the 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'18), Vienna, Austria*, IEEE, pp. 34–42, doi:`10.1109/ASYNC.2018.00021`.

[5] Josip Bozic, Lina Marsso, Radu Mateescu & Franz Wotawa (2018): *A Formal TLS Handshake Model in LNT*. In Rob van Glabbeek & Wendelin Serwe, editors: *Proceedings of the 3nd Workshop on Models for Formal Analysis of Real Systems (MARS'18), Thessaloniki, Greece*, *Electronic Proceedings in Theoretical Computer Science* 268, pp. 1–40, doi:`10.4204/EPTCS.268.1`.

[6] David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Christine McKinty, Vincent Powazny, Frédéric Lang, Wendelin Serwe & Gideon Smeding (2021): *Reference Manual of the LNT to LOTOS Translator (Version 7.0)*. Available at `http://cadp.inria.fr/publications/Champelovier-Clerc-Garavel-et-al-10.html`. INRIA, Grenoble, France.

[7] Ghassan Chehaibar, Meriem Zidouni & Radu Mateescu (2009): *Modeling Multiprocessor Cache Protocol Impact on MPI Performance*. In: *Proceedings of the 2009 IEEE International Workshop on Quantitative Evaluation of Large-Scale Systems and Technologies QuEST'09 (Bradford, UK)*, IEEE Computer Society Press, pp. 1073–1078, doi:`10.1109/WAINA.2009.117`.

[8] Valentin Chimisliu & Franz Wotawa (2013): *Improving Test Case Generation from UML Statecharts by Using Control, Data and Communication Dependencies*. In: *2013 13th International Conference on Quality Software*, pp. 125–134, doi:`10.1109/QSIC.2013.48`.

[9] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq & Wendelin Serwe (2009): *Towards Performance Prediction of Compositional Models in Industrial GALS Designs*. In Ahmed Bouajjani & Oded Maler, editors: *Proceedings of the 21th International Conference on Computer Aided Verification (CAV'09), Grenoble, France*, *Lecture Notes in Computer Science* 5643, Springer, pp. 204–218, doi:`10.1007/978-3-642-02658-4_18`.

[10] Pepijn Crouzen & Frédéric Lang (2011): *Smart Reduction*. In Dimitra Giannakopoulou & Fernando Orejas, editors: *Proceedings of Fundamental Approaches to Software Engineering (FASE'11), Saarbrücken, Germany*, *Lecture Notes in Computer Science* 6603, Springer, pp. 111–126, doi:`10.1007/978-3-642-19811-3_9`.

[11] Pepijn Crouzen, Jaco van de Pol & Arend Rensink (2008): *Applying Formal Methods to Gossiping Networks with mCRL and Groove*. *SIGMETRICS Performance Evaluation Review* 36(3), pp. 7–16, doi:`10.1145/1481506.1481510`.

[12] Lydie du Bousquet, Solofo Ramangalahy, Séverine Simon, César Viho, Axel Belinfante & René G. de Vries (2000): *Formal Test Automation: the Conference Protocol with TGV/TorX*. In Hasan Ural, Robert L. Probert & Gregor v. Bochmann, editors: *Proceedings of the 13th IFIP International Conference on Testing of Com-*

municating Systems (TestCom'00), Ottawa, Canada, University of Ottawa, Kluwer Academic Publishers, pp. 221–228, doi:10.1007/978-0-387-35516-0_14.

[13] Alessio Ferrari, Franco Mazzanti, Davide Basile & Maurice H. ter Beek (2021): *Systematic Evaluation and Usability Analysis of Formal Tools for Railway System Design*. Available at https://arxiv.org/abs/2101.11303. Submitted to IEEE Transactions on Software Engineering.

[14] Sahar Foroutan, Yvain Thonnart, Richard Hersemeule & Ahmed Jerraya (2010): *A Markov chain based method for NoC end-to-end latency evaluation*. In: *IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), (Atlanta, Georgia, USA)*, IEEE, pp. 1–8, doi:10.1109/IPDPSW.2010.5470788.

[15] Hubert Garavel (1998): *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*, *Lecture Notes in Computer Science* 1384, Springer, pp. 68–84, doi:10.1007/BFb0054165. Full version available as INRIA Research Report RR-3352.

[16] Hubert Garavel, Maurice H. ter Beek & Jaco van de Pol (2020): *The 2020 Expert Survey on Formal Methods*. In Maurice H. ter Beek & Dejan Nickovic, editors: *Proceedings of the 25th International Conference Formal Methods for Industrial Critical Systems (FMICS'20), Vienna, Austria*, *Lecture Notes in Computer Science* 12327, Springer, pp. 3–69, doi:10.1007/978-3-030-58298-2_1.

[17] Hubert Garavel & Susanne Graf (2013): *Formal Methods for Safe and Secure Computers Systems*. BSI Study 875, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany. Available at https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.html.

[18] Hubert Garavel & Holger Hermanns (2002): *On Combining Functional Verification and Performance Evaluation using CADP*. In Lars-Henrik Eriksson & Peter A. Lindsay, editors: *Proceedings of the 11th International Symposium of Formal Methods Europe (FME'02), Copenhagen, Denmark*, *Lecture Notes in Computer Science* 2391, Springer, pp. 410–429, doi:10.1007/3-540-45614-7_23. Full version available as INRIA Research Report 4492.

[19] Hubert Garavel & Frédéric Lang (2001): *SVL: a Scripting Language for Compositional Verification*. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang & 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*, Kluwer Academic Publishers, pp. 377–392, doi:10.1007/0-306-47003-9_24. Full version available as INRIA Research Report RR-4223.

[20] Hubert Garavel, Frédéric Lang & Radu Mateescu (2015): *Compositional Verification of Asynchronous Concurrent Systems Using CADP*. *Acta Informatica* 52(4), pp. 337–392, doi:10.1007/s00236-015-0226-1.

[21] Hubert Garavel, Frédéric Lang, Radu Mateescu & Wendelin Serwe (2013): *CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes*. *Springer International Journal on Software Tools for Technology Transfer (STTT)* 15(2), pp. 89–107, doi:10.1007/s10009-012-0244-z.

[22] Hubert Garavel, Frédéric Lang & Wendelin Serwe (2017): *From LOTOS to LNT*. In Joost-Pieter Katoen, Rom Langerak & Arend Rensink, editors: *ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, *Lecture Notes in Computer Science* 10500, Springer, pp. 3–26, doi:10.1007/978-3-319-68270-9_1.

[23] Hubert Garavel, Radu Mateescu, Damien Bergamini, Adrian Curic, Nicolas Descoubes, Christophe Joubert, Irina Smarandache-Sturm & Gilles Stragier (2006): *DISTRIBUTOR and BCG_MERGE: Tools for Distributed Explicit State Space Generation*. In Holger Hermanns & Jens Palberg, editors: *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), Vienna, Austria*, *Lecture Notes in Computer Science* 3920, Springer, pp. 445–449, doi:10.1007/11691372_30.

[24] Hubert Garavel, Radu Mateescu & Wendelin Serwe (2013): *Large-scale Distributed Verification using CADP: Beyond Clusters to Grids*. Electronic Notes in Theoretical Computer Science 296, pp. 145–161, doi:10.1016/j.entcs.2013.07.010.

[25] Hubert Garavel, Gwen Salaün & Wendelin Serwe (2009): *On the Semantics of Communicating Hardware Processes and their Translation into LOTOS for the Verification of Asynchronous Circuits with CADP*. Science of Computer Programming 74(3), pp. 100–127, doi:10.1016/j.scico.2008.09.011.

[26] Hubert Garavel & Wendelin Serwe (2017): *The Unheralded Value of the Multiway Rendezvous: Illustration with the Production Cell Benchmark*. In Holger Hermanns & Peter Höfner, editors: Proceedings of the 2nd Workshop on Models for Formal Analysis of Real Systems (MARS'17), Uppsala, Sweden, Electronic Proceedings in Theoretical Computer Science 244, pp. 230–270, doi:10.4204/EPTCS.244.10.

[27] Hubert Garavel & Damien Thivolle (2009): *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, Lecture Notes in Computer Science 5578, Springer, pp. 241–260, doi:10.1007/978-3-642-02652-2_20.

[28] Hubert Garavel, César Viho & Massimo Zendri (2001): *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), pp. 314–331, doi:10.1007/s100090100044. Also available as INRIA Research Report RR-4041.

[29] Alexander Graf-Brill, Holger Hermanns & Hubert Garavel (2014): *A Model-based Certification Framework for the EnergyBus Standard*. In Erika Abraham & Catuscia Palamidessi, editors: Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'15), Berlin, Germany, Lecture Notes in Computer Science 8461, Springer, pp. 84–99, doi:10.1007/978-3-662-43613-4_6.

[30] Dennis Guck, Jip Spel & Marielle Stoelinga (2015): *DFTCalc: Reliability Centered Maintenance via Fault Tree Analysis*. In Michael Butler, Sylvain Conchon & Fatiha Zaïdi, editors: Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM'15), Paris, France, Lecture Notes in Computer Science 9407, Springer, pp. 304–311, doi:10.1007/978-3-319-25423-4_19.

[31] ISO/IEC (1989): *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. Available at https://www.iso.org/standard/16258.html.

[32] Christophe Joubert & Radu Mateescu (2004): *Distributed On-the-Fly Equivalence Checking*. In Lubos Brim & Martin Leucker, editors: Proceedings of the 3rd International Workshop on Parallel and Distributed Methods in Verification (PDMC'04), London, UK, Electronic Notes in Theoretical Computer Science 128, Elsevier, pp. 47–62, doi:10.1016/j.entcs.2004.10.018.

[33] Hakim Kahlouche, César Viho & Massimo Zendri (1999): *Hardware-Testing using a Communication Protocol Conformance Testing Tool*. In Rance Cleaveland, editor: Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Amsterdam, The Netherlands, Springer, pp. 315–329, doi:10.1007/3-540-49059-0_22.

[34] Jean-Pierre Krimm & Laurent Mounier (1997): *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, Lecture Notes in Computer Science 1217, Springer, pp. 239–258, doi:10.1007/BFb0035392. Extended version with proofs available as Research Report VERIMAG RR97-01.

[35] Abderahman Kriouile & Wendelin Serwe (2015): *Using a Formal Model to Improve Verification of a Cache-Coherent System-on-Chip*. In Christel Baier & Cesare Tinelli, editors: Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), London, United Kingdom, Lecture Notes in Computer Science 9035, Springer, pp. 708–722, doi:10.1007/978-3-662-46681-0_62.

[36] Frédéric Lang (2005): *EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods*. In Judi Romijn, Graeme Smith & Jaco van de Pol, editors: *Proceedings of the 5th International Conference on Integrated Formal Methods (IFM'05), Eindhoven, The Netherlands*, *Lecture Notes in Computer Science* 3771, Springer, pp. 70–88, doi:10.1007/11589976_6. Full version available as INRIA Research Report RR-5673.

[37] Frédéric Lang (2006): *Refined Interfaces for Compositional Verification*. In Elie Najm, Jean-François Pradat-Peyre & 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*, *Lecture Notes in Computer Science* 4229, Springer, pp. 159–174, doi:10.1007/11888116_13. Full version available as INRIA Research Report RR-5996.

[38] Frédéric Lang & Radu Mateescu (2013): *Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems*. *Logical Methods in Computer Science* 9(4), pp. 1–32, doi:10.1007/978-3-642-28756-5_11.

[39] Frédéric Lang, Radu Mateescu & Franco Mazzanti (2019): *Compositional Verification of Concurrent Systems by Combining Bisimulations*. In Annabelle McIver & Maurice ter Beek, editors: *Proceedings of the 23rd International Symposium on Formal Methods – 3rd World Congress on Formal Methods FM 2019 (Porto, Portugal)*, *Lecture Notes in Computer Science* 11800, Springer, pp. 196–213, doi:10.1007/s10703-021-00360-w.

[40] Frédéric Lang, Radu Mateescu & Franco Mazzanti (2020): *Sharp Congruences Adequate with Temporal Logics Combining Weak and Strong Modalities*. In Armin Biere & David Parker, editors: *Proceedings of the 26th International Confrence on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2020 (Dublin, Ireland), held online in 2021*, *Lecture Notes in Computer Science* 12079, Springer, pp. 57–76, doi:10.1007/978-3-030-45237-7_4.

[41] Etienne Lantreibecq & Wendelin Serwe (2014): *Formal Analysis of a Hardware Dynamic Task Dispatcher with CADP*. *Science of Computer Programming* 80(Part A), pp. 130–149, doi:10.1016/j.scico.2013.01.003.

[42] Lina Marsso, Radu Mateescu & Wendelin Serwe (2018): *TESTOR: A Modular Tool for On-the-Fly Conformance Test Case Generation*. In Dirk Beyer & Marieke Huisman, editors: *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'18), Thessaloniki, Greece*, *Lecture Notes in Computer Science* 10806, Springer, pp. 211–228, doi:10.1007/978-3-319-89963-3_13.

[43] Fabio Martinelli, Francesco Mercaldo, Vittoria Nardone, Albina Orlando, Antonella Santone & Gigliola Vaglini (2019): *Model Checking Based Approach for Compliance Checking*. *Inf. Technol. Control.* 48(2), pp. 278–298, doi:10.5755/j01.itc.48.2.21724.

[44] Radu Mateescu & Gwen Salaün (2013): *PIC2LNT: Model Transformation for Model Checking and Applied Pi-Calculus*. In Nir Piterman & Scott A. Smolka, editors: *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13), Rome, Italy*, *Lecture Notes in Computer Science* 7795, Springer, pp. 192–198, doi:10.1007/978-3-642-36742-7_14.

[45] Radu Mateescu & Wendelin Serwe (2013): *Model Checking and Performance Evaluation with CADP Illustrated on Shared-Memory Mutual Exclusion Protocols*. *Science of Computer Programming* 78(7), pp. 843–861, doi:10.1016/j.scico.2012.01.003.

[46] Radu Mateescu & Damien Thivolle (2008): *A Model Checking Language for Concurrent Value-Passing Systems*. In Jorge Cuellar, Tom Maibaum & Kaisa Sere, editors: *Proceedings of the 15th International Symposium on Formal Methods (FM'08), Turku, Finland*, *Lecture Notes in Computer Science* 5014, Springer, pp. 148–164, doi:10.1007/978-3-540-68237-0_12.

[47] Franco Mazzanti & Alessio Ferrari (2018): *Ten Diverse Formal Models for a CBTC Automatic Train Supervision System*. In John P. Gallagher, Rob van Glabbeek & 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, *Electronic Proceedings in Theoretical Computer Science* 268, pp. 104–149, doi:10.4204/EPTCS.268.4.

[48] Hana Mkaouar, Bechir Zalila, Jérôme Hugues & Mohamed Jmaiel (2020): *A Formal Approach to AADL Model-Based Software Engineering*. *Springer International Journal on Software Tools for Technology Transfer (STTT)* 22(2), pp. 219–247, doi:10.1007/s10009-019-00513-7.

[49] Gwen Salaün, Jeff Kramer, Frédéric Lang & Jeff Magee (2007): *Translating FSP into LOTOS and Networks of Automata*. In Jim Davies, Wolfram Schulte & Jin Song Dong, editors: *Proceedings of the 6th International Conference on Integrated Formal Methods (IFM'07), Oxford, United Kingdom*, *Lecture Notes in Computer Science* 4591, Springer, pp. 558–578, doi:10.1007/s00165-009-0133-8.

[50] Pierre de Saqui-Sannes & Ludovic Apvrille (2009): *Making Formal Verification Amenable to Real-Time UML Practitioners*. In Hélène Waeselynck, editor: *Proceedings of the 12th European Workshop on Dependable Computing (EWDC'09), Toulouse, France*, IEEE Computer Society Press, pp. 1–2. Available at https://oatao.univ-toulouse.fr/2107/.

[51] Giuseppe Scollo & Silvia Zecchini (2005): *Architectural Unit Testing*. In: *Proceedings of the International Workshop on Model Based Testing (MBT'04), Barcelona, Spain*, *Electronic Notes in Theoretical Computer Science* 111, pp. 27–52, doi:10.1016/j.entcs.2004.12.006.

[52] Wendelin Serwe (2015): *Formal Specification and Verification of Fully Asynchronous Implementations of the Data Encryption Standard*. In Rob van Glabbeek, Jan Friso Groote & Peter Höfner, editors: *Proceedings of the International Workshop on Models for Formal Analysis of Real Systems (MARS'15), Suva, Fiji*, *Electronic Proceedings in Theoretical Computer Science* 196, pp. 61–147, doi:10.4204/EPTCS.196.6.

[53] Li Su, Howard Bowman, Philip Barnard & Brad Wyble (2009): *Process Algebraic Modelling of Attentional Capture and Human Electrophysiology in Interactive Systems*. *Formal Aspects of Computing* 21(6), pp. 513–539, doi:10.1007/s00165-008-0094-3.

[54] Kenneth J. Turner (2005): *Test generation for radiotherapy accelerators*. *Int. J. Softw. Tools Technol. Transf.* 7(4), pp. 361–375, doi:10.1007/s10009-004-0148-7.

[55] Hao Wu, Xiaoxiao Yang & Joost-Pieter Katoen (2016): *Performance Evaluation of Concurrent Data Structures*. In Martin Fränzle, Deepak Kapur & Naijun Zhan, editors: *Proceedings of the Symposium on Dependable Software Engineering (SETTA'16), Beijing, China*, *Lecture Notes in Computer Science* 9984, Springer, pp. 38–49, doi:10.1007/978-3-319-47677-3_3.