Equivalence Checking 40 Years After: A Review of Bisimulation Tools
Hubert Garavel, Frédéric Lang

To cite this version:

HAL Id: hal-03920338
https://inria.hal.science/hal-03920338
Submitted on 3 Jan 2023
Equivalence Checking 40 Years After: 
A Review of Bisimulation Tools

Hubert Garavel  Frédéric Lang

Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, France
E-mail: \{hubert.garavel,frederic.lang\}@inria.fr

Abstract

Equivalence checking is a formal verification approach that consists in proving that two programs or models are related modulo some equivalence relation, or that one is included in the other modulo some preorder relation. In the case of concurrent systems, which are often represented using labelled transition systems, the relations used for equivalence checking are bisimulations and their simulation preorders. In the case of probabilistic or stochastic systems, which are usually represented using Markov chains, the relations used for equivalence checking are lumpability, probabilistic and stochastic equivalences, and their associated preorders. The present article provides a synthetic overview of 40 years of research in the design of algorithms and software tools for equivalence checking.

1 Introduction

The present article was written in honor of Frits Vaandrager and included in a collective Festschrift book offered to him at the occasion of his 60th birthday.

Frits Vaandrager has published an impressive list of papers addressing very diverse topics in formal methods and concurrency theory, among which: operational semantics and SOS rules, process algebra, Petri nets, input-output automata, timed automata and real-time models, probabilistic automata, hybrid input-output automata and hybrid systems, action-based and state-based temporal logics, testing theory, automata learning, as well as formal modelling and verification of many industrial protocols. Among such a wealth of contributions, we have chosen to focus on bisimulations and equivalence checking, a topic that Frits Vaandrager contributed to advance significantly.

In formal methods, one never proves that a system (or a program) is correct in itself, but only that it is correct with respect to its specifications. Thus, formal verification does not consist in checking one artefact, but in comparing two artefacts one against the other, i.e., a system against its specifications that
express desirable, expected properties. Depending on the formalism used for specifications, two cases need to be distinguished:

- If the system and its specifications are expressed in two different formalisms, one needs to prove that the system \textit{satisfies} its specifications. A major approach is \textit{model checking} [97], in which specifications are expressed in some temporal logic.

- If the system and its specifications are expressed in the same formalism, one needs to prove that the system \textit{is equivalent} to its specifications or, in a weaker form, that the system \textit{contains} or is \textit{included} in its specifications. Such verification approaches are usually referred to as \textit{equivalence checking} and, more often than not, the system is a much more complex artefact than its specifications, in which many low-level implementation details are abstracted away.

The present article presents a brief history of equivalence checking in the context of concurrent systems — leaving aside the widespread application of equivalence checking in the hardware-design industry to make sure that logic-synthesis tools preserve the intended behaviour of circuits.

The foundations of equivalence checking for concurrent systems have been laid in the 1970s. On the practical side, the protocol-engineering community developed verification techniques based on the systematic exploration of all reachable states of a concurrent system [371] [389]. On the theoretical side, it became clear that the semantics of concurrent systems could be adequately represented using state-transition models, such as LTSs (Labelled Transition Systems) [268] or Kripke structures [281] — although alternative models, such as Petri nets, would also coexist.

Finding the right equivalence relation to compare two state-transition models describing two concurrent systems was a non-trivial problem, as the two main equivalence relations known at that time were not appropriate: on the one hand, \textit{graph isomorphism} was too strong, requiring both models to be strictly identical modulo permutations, whereas both models often have different levels of abstraction; on the other hand, \textit{language equivalence} was too weak, only checking that the sets of traces of both models are the same, thus failing to make distinctions such as \(a.(b + c)\) vs \((a.b) + (a.c)\), which are essential as far as the semantics of concurrent systems is considered.

To address this problem, \textit{behavioural} equivalences have been introduced. These are equivalence relations situated between graph isomorphism and language equivalence: they are coarser than the former and finer than the latter. Important examples of behavioural equivalence are \textit{strong equivalence} and \textit{observational equivalence} [339] (the latter being also called \textit{weak equivalence}). A major breakthrough was made with the concept of \textit{bisimulation} [356], which provides a conceptual framework for these equivalences — see [20] for an insightful account of these foundational steps of concurrency theory. In a nutshell, bisimulation
identifies all states having the same future, i.e., all states from which one can execute the same (possibly infinite) trees of actions.

There exist many behavioural relations; an overview of them can be found in [406]. Each equivalence relation comes with its associated preorder. Therefore, equivalence checking consists in verifying that two systems are related modulo some equivalence relation, or that one system is included in the other modulo some preorder relation. Fortunately, not all of these relations are needed in practice: our experience shows that most real-life case studies can be addressed using only a handful of well-chosen relations.

In the case of probabilistic and stochastic systems, which are usually represented using Markov chains or models derived from Markov chains, bisimulation coincides with the concept of lumpability [269] and, thus, serves as a basis for the definition of probabilistic and stochastic equivalences on Markov chains.

The present article provides a retrospective account of 40 years of research in equivalence-checking techniques for concurrent systems. Compared to prior surveys on bisimulations [406] [20], we focus here on the design of algorithms and software tools for the implementation of bisimulations on finite-state systems, leaving aside all aspects related to theorem proving — see [364] for a survey on bisimulation as a proof method. The article is organized chronologically: Sections 2–5 present the main achievements done during the 1980s, 1990s, 2000s, and 2010s decades, respectively, and Section 6 gives a few concluding remarks.

2 Retrospective of the 1980s

2.1 Bisimulation Tools Based on Term Rewriting

The book of Robin Milner on CCS [339] and the seminal article of David Park providing a co-inductive definition of bisimulation [356], together with subsequent publications [340] [341], laid the theoretical foundations for a deep research area, in which bisimulations are closely associated to process calculi. The definition of process calculi using either structural operational semantics [361] [362] [363] or algebraic semantics [27] initially led to consider term rewrite engines and theorem provers as natural approaches for implementing process calculi and bisimulations.

Among such early attempts, CIRCAL [336] was a tool that used algebraic laws to perform proofs of equivalence between two process-calculus programs.

ECRINS [314] was a tool for manipulating process calculi, the operators of which were defined using conditional rewrite rules. A semi-decision algorithm for strong bisimulation was implemented in ECRINS, and the tool was able to prove strong bisimulation automatically between CCS-like programs, without process recursion but with free process variables. Another use of ECRINS was to prove the correctness of semantic translation from one process calculus to another [133] [134].
BIP (Bisimulation Prover) [84] was a verification tool for protocol specifications written in the ECCS language. This tool used an enhanced version of Sanderson’s algorithm for bisimulation [376]; this enhanced version removes one limitation of Sanderson’s by accepting the presence of nondeterminism and $\tau$-transitions, i.e., internal (or silent, or non-observable) actions of an LTS.

CRLAB [124] [125] was a tool that used term rewriting to compare processes written in CCS or basic LOTOS (i.e., the subset of LOTOS [254] without value passing) modulo observational equivalence.

2.2 Algorithms for Bisimulations on Finite-State Systems

However, alternative approaches to term rewriting quickly emerged. In these approaches, each program is first translated into a finite-state LTS, which is only possible if the program does not have an infinite number of states (e.g., does not handle unbounded data types nor infinitely many concurrent processes) and if the number of states is small enough to fit into the memory of available computers (i.e., does not face the well-known state explosion problem, which is a limiting factor for the verification of complex programs).

Assuming that an LTS has been successfully generated, one then executes a different, bisimulation-specific algorithm to minimize this LTS according to a chosen bisimulation (e.g., strong, observational, etc.); given that bisimulations are equivalence relations, such a minimal LTS always exists and is unique modulo a renaming of states. The same minimization algorithm can also be reused for equivalence checking purpose, i.e., to compare whether two LTSs are bisimilar or not: this is done by applying a minimization algorithm on the disjoint union of both LTSs and checking whether initial states are in the same equivalence class.

In a nutshell, these alternative approaches give up the generality of term rewriting (which can handle infinite-state programs but, in practice, is quite limited in the size of programs that can be handled) to adopt a less general approach (which only handles finite-state programs, but of a greater complexity that exceeds the capabilities of human reasoning). Finite-state approaches also have the advantage of being fully automated, meaning that bisimulation can be decided without human intervention.

In the 1980s, two key algorithms for computing bisimulation on finite LTSs have been proposed. In two articles [262] [261] that recently received the 2021 Dijkstra Prize, Paris Kanellakis and Scott Smolka proposed an algorithm for checking the equivalence of CCS processes. Their algorithm performs relational coarsest partitioning (also known as coarsest partition refinement): initially, all states are in the same set, which is progressively partitioned to separate states having different futures. The time and space complexities of their algorithm are $O(mn)$ and $O(m + n)$, respectively, where $m$ is the number of transitions and $n$ the number of states.

1 We use the same definitions of $m$ and $n$ throughout this article.
In a subsequent article [354], Robert Paige and Robert Tarjan proposed a more efficient algorithm for the same problem. Their algorithm only addresses strong bisimulation. Its time and space complexities are $O(m \log n)$ and $O(m + n)$, respectively.

### 2.3 Early Bisimulation Tools

The Kanellakis-Smolka and Paige-Tarjan algorithms aroused great interest and triggered the development of numerous software tools.

SCAN [347] was probably the first implementation of bisimulations. This tool could reduce and compare, with respect to strong or observational equivalence, networks of finite automata composed together using parallel operators and filtering (a combination of hiding and relabeling to abstract away the internal behaviour of composed automata).

AUTO [410] [411] [412] [66] [315] [67] [129] was a tool for the analysis and manipulation of finite LTSs, which could be either specified using the MEIJE process calculus [17] or described graphically using the AUTOGRAPH editor and composed in parallel and connected together using synchronisation ports and communication wires. AUTO offered primitives to minimize an LTS modulo strong bisimulation, observational bisimulation, or trace language equivalence, to determinize an LTS, to compute the transitive closure of $\tau$-transitions, and to eliminate $\tau$-loops and single $\tau$-transitions. It could also check the equivalence of two LTSs for strong or observational bisimulation.

VENUS [344] [385] was a tool for minimizing and comparing CCS processes modulo strong or observational bisimulations. It also supported operations dedicated to $\tau$-transitions, such as the elimination of $\tau$-chains and $\tau$-circuits.

TAV [190] [53] [54] was a tool that could check strong or observational equivalence between two CCS processes, and also check whether a CCS process satisfied a temporal-logic formula expressed in the Hennessy-Milner logic extended with recursion. A distinctive feature of TAV was the possibility to provide an explanation given as a Hennessy-Milner logic formula computed using a backtracking algorithm [240]. Another algorithm for providing a temporal-logical formula that differentiates two LTSs that are not strongly bisimilar was proposed in [98].

SQUIGGLES [49] [50] was a tool that extended the Paige-Tarjan algorithm to compare LTSs or programs written in basic LOTOS with respect to strong equivalence, observational equivalence, or testing equivalence.

WINSTON [320] was a tool that could compare networks of finite-state processes using strong and observational equivalences implemented with the Kanellakis-Smolka algorithm.

ALDEBARAN [146] [147] [148] [149] was a tool for minimizing and comparing LTSs according to strong bisimulation, observational equivalence, acceptance model equivalence, or safety equivalence; it could also display the equivalence class of each state of the LTS. Contrary to most other tools written in function-
al/declarative languages such as Lisp, Prolog, etc., ALDEBARAN was written in C and implemented an adaptation of the Paige-Tarjan algorithm. The simple file format used by ALDEBARAN for storing LTSs became popular and has been known since as the AUT format².

The ACP Bisimulation Tool [436] was a tool to compare programs written in the ACP_τ process calculus [28] modulo a weak relation called bisimulation with τ-abstraction. It computed a transitive closure of τ-transitions, followed by a relational coarsest partition algorithm.

PIPN [19] was a tool that could minimize, according to observational equivalence or trace equivalence, the labelled reachability graphs generated from Petri nets. PIPN was part of ESTIM [378], a simulation and verification tool for communicating state machines specified in the ESTELLE language [253].

CWB (Concurrency Workbench) [105] [104] was an integrated tool for verifying networks of finite-state processes described in CCS, then converted to a state-transition model called transition graphs. Three kinds of analyses were supported by CWB: model checking (evaluation of temporal-logic formulas written in the propositional mu-calculus), equivalence checking (comparison of two transition graphs modulo C-bisimulation, a generic relation from which strong equivalence, observational equivalence, must equivalence, and testing failures equivalence can be derived), and preorder checking (comparison of two transition graphs modulo C-preorder, from which bisimulation divergence preorder, may preorder, must preorder, and testing preorders can be obtained). Equivalence checking was based on the Kanellakis-Smolka algorithm, while preorder checking was done using an ad hoc, less efficient algorithm. An early application of CWB to the analysis of mutual exclusion algorithms can be found in [418].

Most of the aforementioned tools performed at least two different tasks: (i) generate the LTSs corresponding to concurrent systems described either as networks of automata composed in parallel, or as process-calculi specifications; and (ii) minimize and/or compare these LTSs modulo various equivalence or preorder relations. Both tasks are subject to antagonistic implementation constraints: task (i) must store in memory the concrete contents of each state of the LTS being generated, while the transitions of the LTS can just be written to disk; conversely, task (ii) must store in memory all the transitions, and can ignore the concrete contents of states, which can just be treated like abstract numbers. It is thus difficult for the same tool to be optimally efficient in both tasks. One solution is to have separate tools, dedicated either to task (i) or to task (ii).

The first instance of a specialized tool for task (i) was CÆSAR [168] [182], a tool for translating value-passing LOTOS specifications into LTSs encoded in the various file formats accepted by ALDEBARAN, AUTO, CWB, PIPN, SCAN, SQUIGGLES, etc. A distinctive feature of ALDEBARAN, CÆSAR, and their companion tools later integrated in the CADP toolbox [151] was to be plain, ordinary Unix commands that could be directly invoked from the shell.

² https://cadp.inria.fr/man/aut.html
with appropriate command-line options, and that communicated via files containing LTSs. Such an architectural design was a major departure from other bisimulation tools, most of which were built around custom command-line interpreters (with ad hoc primitives for, e.g., loading, generating, minimizing, and saving LTSs). It has been progressively adopted by other tools during the next decades.

3 Retrospective of the 1990s

3.1 New Algorithms for Bisimulations

The 1990s have been a very active decade, in which new equivalences, preorders, and algorithms have been invented and implemented in tools.

Testing equivalences [123], introduced in the 1980s, define that two models (typically a high-level specification and a lower-level implementation) are equivalent iff an observer interacting with them cannot distinguish one from the other by means of testing. Although these equivalences are weaker (i.e., less discriminative) than bisimulations, Cleaveland and Hennessy gave an algorithm [101] based on a characterization of these relations in terms of bisimulation. This algorithm was implemented in CWB.

Safety equivalence [56], named so because it preserves safety properties, is the equivalence relation obtained by the logical conjunction of two $\tau^*$.a preorders [155] that abstract away $\tau$-transitions. Algorithms for minimizing and comparing LTSs modulo safety equivalence were implemented in ALDEBARAN.

Branching bisimulation [407] was introduced by van Glabbeek and Weijland after observing that Milner’s observational equivalence does not fully respect the branching structure of LTSs. To compute branching bisimulation, which is slightly stronger than observational equivalence, Groote and Vaandrager proposed an algorithm [200] [201] for relational coarsest partitioning with stuttering. A key idea of the algorithm is the possibility to compress each strongly connected component of states connected by $\tau$-transitions into a single state beforehand, using existing linear-time algorithms. This algorithm has a worst-case time complexity $O(mn)$ and a space complexity $O(m + n)$. Thus, branching bisimulation can be implemented more efficiently than observational equivalence, which has progressively been superseded by branching equivalence, except in the CCS community where observational equivalence remained popular, probably by fidelity to the foundations set by Milner. The Groote-Vaandrager algorithm was implemented in an efficient prototype named Branching Tool, as well as in other tools, among which ALDEBARAN and AUTO.

Groote and Vaandrager also proposed a format of Plotkin-style structural operational semantics rules [202] that guarantees, among other properties, that strong bisimulation is a congruence on the states of any transition system described using this format.
It was shown [365] that, for all equivalences between strong bisimulation and trace equivalences (i.e., almost all equivalences of practical interest), checking equivalence of two networks of LTSs is PSPACE-hard because, in the general case, one cannot avoid computing the product state space of each network.

De Nicola and Vaandrager investigated the logical characterization of branching bisimulation, and exhibited three logics such that two LTSs are branching bisimilar iff they exactly satisfy the same formulas of these logics [127] [128].

There have been other attempts at computing bisimulations using model checkers, such as MEC [12] (later integrated in ALTARICA [13]) and SPIN [140] [141]. For instance, MEC did not implement dedicated algorithms for equivalence checking, but could define bisimulation as a concise formula in fix-point logic [11] [194], thus enabling bisimulation to be verified as a particular case of model checking.

### 3.2 Algorithms for On-the-Fly Verification

While the mainstream approach, so far, consisted in first generating LTSs before minimizing them or checking their equivalence, novel on-the-fly approaches emerged, where an LTS is reduced (i.e., partially minimized) while being generated, or where equivalence between two LTSs is checked while these LTSs are generated. Such approaches, which had already been experimented successfully for model checking, may avoid storing the entire set of states and/or transitions, especially when one does not need to fully explore both LTSs to decide that they are not equivalent.

An on-the-fly algorithm for equivalence checking was first proposed by Fernández and Mounier [153] [345]. Their algorithm was not based on partition refinement, which requires to compute the sets of states beforehand, but instead explored the states of the synchronous product of the LTSs under comparison, until a verdict can be given. Various adaptations of their algorithm to compute strong, branching, observational, and $\tau^*$-bisimulations, as well as safety equivalence and the corresponding preorders, have been implemented in ALDEBARAN [154]. A variant of their algorithm was later proposed, which does not store all states of the synchronous product, but only enough states so that the verification terminates [258].

A similar approach for on-the-fly equivalence checking modulo strong and observational bisimulations was implemented in the LOLA tool [348].

Lee and Yannakakis proposed another on-the-fly algorithm [300] for minimizing an LTS modulo strong bisimulation while this LTS is being generated.

A generic architecture, named OPEN/CÆSAR [170], was proposed for developing on-the-fly verification tools rationally. This architecture achieves a clear separation between, on the one hand, an LTS that is generated on-the-fly (e.g., from a process-calculus specification or a network of communicating automata) and, on the other hand, a verification algorithm that explores this LTS guided
by a specific goal or property to be proven. One of the first applications of OPEN/CÆSAR was REDUCTOR [170], a tool that reduced LTSs on the fly modulo $\tau^*.a$ equivalence.

Partial-order reductions are techniques for reducing the size of the state space, by exploiting the independence of transitions to avoid unnecessary interleavings. Such techniques, provided they preserve a behavioural relation of interest, may be particularly useful when doing minimization or equivalence checking on the fly. Partial-order reductions were first studied in the context of linear-time semantics, then in the context of branching-time semantics to check branching bisimulation [400] [184], strong bisimulation (between two LTSs having the same independent transitions) [244], and failures refinement [422].

3.3 Algorithms for Symbolic Verification

New algorithms were proposed, based on a symbolic representation of the system under verification in the form of a BDD (Binary Decision Diagram) [75].

An approach was proposed, in which CCS processes are translated to BDDs, thus allowing bisimulations (encoded as temporal logic formulas) to be computed on such a symbolic representation [139].

Bouali and De Simone proposed a symbolic algorithm [60] dedicated to the minimization of networks of LTSs according to strong bisimulation, as well as variants for observational and branching bisimulations.

Another symbolic algorithm [305] was proposed for comparing CCS processes modulo observational bisimulation, and compared, on a few benchmarks, to the equivalence-checking algorithm implemented in CWB.

Fisler and Vardi [158] [159] [160] implemented, using BDDs, three minimization algorithms in the setting of finite transition systems, the states of which are labelled with atomic propositions. Their experimental results indicate that BDDs are not a silver bullet for bisimulation minimization: the number of BDD nodes needed to compute the bisimulation relation grows quickly, outweighing the potential benefits of minimizing the global state space before performing symbolic model checking.

3.4 Algorithms for Compositional Verification

Given a concurrent system, e.g., a network of LTSs, compositional verification consists in generating a reduced (or even minimized) LTS for this system, modulo some behavioural relation of interest [146] [320] [372] [430] [391] [390] [399]. If this relation is a congruence for the operators of the network (typically, parallel composition and label hiding), which is the case of most bisimulations, then the generation can be done incrementally, by alternating operator applications and reductions of the intermediate LTSs.

However, while doing so, state explosion may happen in intermediate LTSs,
due to the existence of transitions that are fireable locally, but not globally, as they could not meet the synchronization constraints in the entire network. Graf, Steffen, and Lüttingen proposed an approach to solve this problem, based on interface specifications (represented as LTSs) that cut off globally unfireable transitions [192] [193]. This approach was extended by Krimm and Mounier, and implemented in PROJECTOR [280], an on-the-fly tool developed using the OPEN/CÆSAR architecture.

A related approach, called compositional reachability [92], performs compositional reduction modulo observational equivalence while verifying a property represented as an LTS. This approach was implemented in the TRACTA tool [92], which also supported a variant [91] of Graf-Steffen-Lüttingen interface specifications.

A comprehensive survey on compositional verification, from the 1990s to the present, can be found in [179].

### 3.5 Enhanced Bisimulation Tools

Most bisimulation tools developed in the 1980s quickly became unavailable, due to lack of software maintenance at a time where processors, operating systems, and programming languages were rapidly evolving. There was, however, the notable exception of three tools (namely, ALDEBARAN, AUTO, and CWB), the development of which steadily progressed during the 1990s.

ALDEBARAN [155], so far mainly based on the Paige-Tarjan algorithm, was enhanced in four directions: counterexample generation algorithms, implementation of the Groote-Vaandrager algorithm for branching bisimulation, novel on-the-fly equivalence-checking algorithms [155] [345], and symbolic verification algorithms based on BDDs [152] [271] [270]. ALDEBARAN was a key component of the CADP verification toolbox [151] [150] [169] [71] [172], which gathered a growing number of closely interconnected tools. Its synergy with CADP brought ALDEBARAN at least three benefits: the existence of an efficient LTS generator, the aforementioned compiler CÆSAR for LOTOS; the availability of BCG (Binary Coded Graphs), a compact file format for storing large LTSs; and the integration within EUCALYPTUS, a graphical user interface that simplified the invocation of ALDEBARAN with command-line options.

ALDEBARAN, together with companion tools of CADP, has been used in numerous case studies by scientists of many universities worldwide. Among the case studies involving equivalence checking, one can mention, in chronological order: a car overtaking protocol [142], dynamically changing communication structures [166], an ATM switch [157], a plain ordinary telephone service (starting from an existing specification [144]), a framework for groupware development [272], a trusted third-party protocol between video-on-demand service

---

3 https://cadp.inria.fr/man/bcg.html
4 http://cadp.inria.fr/case-studies
providers and customers [299] [183], a railway signalling system [164], a bounded retransmission protocol [323] [324] (starting from an existing specification [203]), the TCP Internet transport protocol [379], feature interactions in telephony systems [277], several variants of distributed leader election algorithms for unidirectional ring networks [181], a bus arbiter of a multiprocessor architecture [86], the link layer protocol of the IEEE-1394 serial bus [381], a departure clearance protocol for air traffic [121], testing of a distributed leader election algorithm [397] [396], a flow-control protocol for a high-speed network [222], patterns for software architecture styles [216], an invoicing system [380] [382], a protocol for road traffic control [424] [423], asynchronous circuits [431] [432] [433], a reliable data-transfer service [312], an abstraction-display-controller model for user interfaces [321], a distributed cluster file system [358] [359], an ISO high-speed transport protocol [25], highly reliable and reusable CORBA applications [279], a leader election protocol for home audio/video networks [367], synchronous hardware [215] [214] [213], a protocol for deploying intelligent telecommunication services [14] [15] [16], a handshake authentication protocol [298], a datalink system for air traffic control [373] [374] [375], a radio protocol for mobile telecommunications [309], a protection system against cloning of cellular phones [350] [349], and hardware/software codesign [429] [22]. Such an impressive list clearly indicates that formal verification should not be restricted to model checking only, and that equivalence checking and bisimulations also have a major role to play.

The evolution of AUTO and its associated graphical editor AUTOGRAPH also continued in the 1990s [369] [317] [318] [370]. Companion tools were developed, among which: MAUTO, which extended AUTO to a large family of process calculi; FCTOOL [58], which implemented efficient partitioning algorithms (derived from the Groote-Vaandrager algorithm) for strong, branching, and observational bisimulations; and the FC2TOOLS set [65] [64] [62] [63], which gathered a collection of tools offering both explicit and implicit (i.e., BDD-based) bisimulation algorithms, and designed around FC2, a dedicated file format for LTSs and networks of LTSs.

A few case studies were done using these tools, e.g., a bus instrumentation protocol specified in LOTOS [18], a sliding window protocol specified in LOTOS [316], a lift controller specified in ESTEREL [207], and a secure datagram protocol specified in LOTOS [188].

CWB also pursued its evolution during the 1990s. The performance of its LTS minimization algorithm was assessed in [143], the introduction of priorities was presented in [259], and an overall presentation of CWB can be found in [106]. Contrary to ALDEBARAN and AUTO, the development of which remained centralized in Grenoble and Sophia-Antipolis, respectively, CWB adopted a more decentralized approach: the original software was maintained in Edinburgh [342] [386]5 in collaboration with other universities (e.g., Sussex), while a new branch emerged in the United States under the successive names of Concurrency Factory [100] [102], NCSU Concurrency Workbench [108], Concurrency Workbench of

5 See also [387] and [388] for a reflection on the development of verification tools.
Besides the use of CWB by Swedish telecom companies to analyze parts of the GSM and ISDN protocols, other case studies used the equivalence-checking features of CWB, e.g., alternating bit protocol with lossy buffers [99], instruction pipelining and parallel memory models for the SPARC architecture [282], formalization in LOTOS of the GKS computer-graphics standard [366], and access monitoring for information flow security [161].

3.6 New Bisimulation Tools

Besides enhancements brought to already existing bisimulation tools, many new tools were developed during the 1990s.

PisaTool [249] [252] used term rewriting to compare CCS processes for strong, observational, or branching equivalences. Instead of relying on interleaving semantics, this tool took into account some aspects of true concurrency by introducing a parametric representation of finite-state systems and bisimulations.

SEVERO [83] was both an equivalence checker and a model checker for the ACTL temporal logic [126], applicable to finite-state processes that were converted to BDDs and reduced modulo observational equivalence using a symbolic minimization algorithm.

ARA (Advanced Reachability Analysis) [401] was a tool for minimization and equivalence checking of basic LOTOS processes modulo CFFD equivalence [398], a relation weaker than observational equivalence. ARA supported compositional LTS construction and partial-order reductions. It was used, e.g., to verify a small protocol for client-server communication [274] and, using compositional verification and in combination with CADP, a reliable data-transfer service [312].

CCSTOOL2 [409] was a modular tool performing various computations on finite-state CCS descriptions, including minimization for strong and observational bisimulations.

JACK (Just Another Concurrency Kit) [61] was an integrated environment, with a graphical user interface, gathering verification tools designed around the FC2 file format. JACK supported model checking of ACTL logic formulas, as well as equivalence checking using CRLAB (term rewriting) and AUTO (algorithms for finite-state models). The equivalence-checking features of JACK have been used to verify hardware components [122], a railway interlocking system [30] [31], and, in combination with CADP, an abstraction-display-controller model for user interfaces [321].

YAPV [36] was a research tool based on true concurrency, instead of interleaving semantics. It used a variant of the Kanellakis-Smolka algorithm to check bisimulation between processes.

BIDMIN [218] was a minimization tool written in Ada. It supported strong
bisimulation, as well as variants of observational and branching bisimulations (e.g., rooted, divergence-preserving, etc.), these variants differing only in the definition of the initial partition of the LTS state set.

FDR2 [368] was a verification tool for the CSP process calculus [74]. It implemented strong bisimulation, as well as failures and failures/divergences equivalences, which were preferred to observational or branching bisimulation for checking equivalence and refinement (i.e., preorder inclusion) of CSP processes. FDR2 was used to formally verify STATEMATE statecharts [167].

ARC [355] was another formal verification tool for CSP, which had similar equivalence checking features as FDR2, but in which LTSs were represented symbolically using ordered BDDs.

XEVE [59] was a verification environment for ESTEREL specifications, which used the Bouali-De Simone algorithm to perform symbolic minimization modulo strong bisimulation.

ObjectGeode, an industrial environment for simulating and verifying SDL programs, was equipped with on-the-fly equivalence checking capabilities by a connection to CADP [273].

Getting an accurate panorama of bisimulation tools by reading publications is uneasy, as related work was not always cited properly. Fortunately, a few benchmarks and surveys exist, e.g., a gentle introduction [145] to the principles of bisimulation tools, followed by an overview of ALDEBARAN, AUTO, CWB, PVE, TAV, etc.; a comparative evaluation [278] of bisimulation techniques in ALDEBARAN, AUTO, CWB, TAV, WINSTON, etc.; an overview [313] of tools available in the early 1990s, among which ALDEBARAN, AUTO, CWB, ECRINS, FCTOOL, MAUTO, MEC, TAV, etc.; a study [217] of the respective performances of ALDEBARAN, BIDMIN, CWB, and FCTOOL/HOGGAR; and a survey [250] [251] of tools for the analysis of distributed systems specified using process calculi.

3.7 Bisimulation Tools for Timed and Hybrid Systems

There has been a variety of tools for modelling and analyzing real-time systems. First attempts used a discrete-time model, in which a special “tick” action represented the elapsing of one time unit. Since this approach did not scale to realistic systems, requiring too many transitions to model long delays, dense-time models supported by methods from continuous mathematics were progressively adopted, either as timed extensions of process calculi [118] [195] [301] [302] or as timed automata [6], i.e., automata extended with real-valued clock variables. Tools for the analysis of timed systems often involved the generation of an untimed finite-state abstraction of the state space (e.g., a region graph [5]), that takes time constraints into account.

The first minimization algorithms for timed automata [3] [4] generated a minimized region graph on the fly, by extending a prior algorithm [57] that directly
generates a minimal LTS for strong bisimulation (rather than first generating an LTS that is minimized later).

VERSA [96] [95] [93] was a tool for the analysis of discrete-time systems with prioritized resources and events, described in a timed process calculus called ACSR (Algebra of Communicating Shared Resources). Besides checking equivalences by means of term rewriting, VERSA automatically translated ACSR processes with bounded delays to LTSs, and compared these (finite-state) LTSs modulo strong bisimulation and \( \tau^*a \) equivalence using the Kanellakis-Smolka algorithm. XVERSA [94] was an extension of VERSA with a graphical user interface.

TPWB (Timing and Probability Workbench) [165] was a tool for analyzing finite-state, discrete-time systems described in TPCCS [208] [209], an extension of CCS with time and probabilities. It used an adaptation of the Kanellakis-Smolka algorithm with lumping of probabilities to perform minimization and equivalence checking on finite-state systems modulo strong bisimulation.

EPSILON [189] [191] [85] was an extension of TAV for analyzing dense-time systems described in TMS (Timed Modal Specifications), a formalism for timed networks inspired by TCCS [420] (a variant of CCS with a delay operator) extended with “may” and “must” modalities. EPSILON implemented equivalence and preorder checking for strong and observational bisimulations, as well as time-abstracted versions of these relations that gave finite representations of these networks. When the check was negative, EPSILON could generate, like TAV, a distinguishing formula in timed Hennessy-Milner logic. EPSILON was used to analyze a steam generator [284] [283].

KRONOS [119] [394] [434] [69] [70] was a tool for minimization, equivalence checking, and preorder checking of timed automata. It used a time-abstracting bisimulation and linear constraints on the clocks of the timed automaton to generate an untimed abstraction of the state space. The resulting LTS could be minimized and checked using ALDEBARAN. KRONOS was used in many case studies, and also as a back-end [238] for the verification of systems described in ET-LOTOS [301] [302], a timed extension of LOTOS. Later, Extended KRONOS (or OPEN/KRONOS) [393] enhanced KRONOS with a richer input language and on-the-fly verification capabilities based upon the OPEN/CÆSAR architecture.

TREAT (Timed Reachability Analysis Tool) [263] was a tool for timed automata. To fight the state explosion problem, TREAT generated untimed abstractions that preserved two behavioural relations named history equivalence and transition bisimulation.

RT-MEC [81], a component of PEP (Programming Environment based on Petri nets), was a tool for the analysis of systems modelled as Petri nets with dense time. It implemented equivalence checking modulo strong bisimulation and timed bisimulation, using partial-order reductions and on-the-fly techniques to generate a reduced region graph.

Hybrid automata are infinite-state models to describe digital programs interact-
ing with an analog environment. Although hybrid automata encompass timed automata, there have been few tools implementing bisimulations on hybrid automata. A notable exception was HYTECH \[221\] \[220\], which used bisimulations to reduce hybrid systems to finite LTSs. When such a reduction was possible, the hybrid automaton could be model checked.

A performance comparison between EPSILON, KRONOS, HYTECH, and the UPPAAL tool \[24\] (which does not use the concept of bisimulations) can be found in \[295\].

### 3.8 Bisimulation Tools for Probabilistic and Stochastic Systems

The advent of process calculi in the 1980s gave a new impulse to the study of Markovian models. In order to finely describe both the functional and performance aspects of concurrent systems, process calculi have been extended in various ways with non-functional concepts, such as probabilities and random durations, e.g., \[233\] \[242\]. At a lower abstraction level, extended models have been proposed, combining LTSs, to model functional aspects, and DTMCs (Discrete-Time Markov Chains) or CTMCs (Continuous-Time Markov Chains), to model performance aspects.

To analyze such models, in addition to traditional techniques (steady-state and transient analyses, simulation, etc.) and novel quantitative model-checking approaches, various bisimulations have also been defined, such as probabilistic bisimulations \[296\] \[297\] \[241\] \[245\] \[48\] \[310\] \[383\] and Markovian/stochastic bisimulations \[233\] \[226\] \[234\] \[76\]. These equivalences combine the concepts of bisimulation for the LTSs and of lumpability for the Markov chain aspects.

At first, such bisimulations were used for algebraic proofs of equivalence and performance calculations on simple models. But dedicated algorithms were progressively designed, e.g., \[21\], and implemented in already existing or novel software tools.

TIPPtool \[275\] \[232\] was a tool for creating and analyzing concurrent systems described in the TIPP language, a stochastic process calculus based on LOTOS. Initiated in 1992, TIPPtool has been progressively extended with many features for functional and performance analyses. Bisimulations on LTSs and Markovian models played a major role in TIPPtool, especially for applying compositional minimization techniques \[239\] \[227\] \[224\] \[225\] in order to contain state explosion. Strong and observational bisimulations were implemented using the Kanellakis-Smolka algorithm, but an export to CADP’s AUT format was also available; the Baier algorithm was used for Markovian bisimulations; symbolic algorithms based on BDDs were also developed for this purpose \[235\] \[236\]. Various systems have been analyzed using TIPPtool, e.g., an alternating bit protocol \[231\], a communication protocol \[162\], a plain-old telephone system (tackled using TIPPtool in combination with CADP) \[229\], and an hospital communication system \[225\].
TwoTowers [35] [34] was a tool for the functional and performance analysis of systems modelled in EMPA, a stochastic timed process calculus. TwoTowers combined two existing tools: the aforementioned Concurrency Workbench of North Carolina (for model checking, equivalence checking, and preorder checking) and MarCA (for steady-state and transient performance analysis); it also supported the strong extended Markovian reward bisimulation [33]. The equivalence-checking capabilities of TwoTowers have been used to analyze a randomized distributed algorithm for the dining philosophers problem [35] and a token ring protocol [32].

The APNN (Abstract Petri Net Notation) toolbox [23] [77] [78] was a set of tools for the functional and quantitative analysis of discrete-event dynamic systems. It offered many features, such as model checking, numerical analysis of Markov chains, and simulation, but also supported various kinds of bisimulations used to reduce, by means of compositional minimization techniques, the size of the generated state spaces.

### 3.9 Bisimulation Tools for Mobile Systems

Mobile process calculi, such as the π-calculus [337] [338], enable the description of concurrent systems with potentially infinite state spaces, due to the dynamic creation of agents and communication channels. For such systems, verification approaches based on finite-state systems are hardly applicable, especially if they require an exhaustive exploration of the state space before verification can take place. For such problems, formal proofs (which are outside the scope of this survey) are the approach of choice [364]. Nevertheless, there have been attempts at developing automated equivalence-checking tools based on bisimulation theory for analyzing such systems.

MWB (Mobility Workbench) [413] [416] [414] was a tool for checking whether two π-calculus programs are equivalent with respect to open bisimulation [377]. MWB was based upon an on-the-fly algorithm (inspired from the Fernandez-Mounier approach) in which LTSs where generated on demand. MWB was later extended with an algorithm for checking symbolic hyperequivalence [357] for the fusion calculus [415, Chapter 7].

The π-environment [156] was a tool for checking (strong or observational) early and late bisimulations on finite-state processes specified in the π-calculus. This tool did not work on the fly, but relied on the aforementioned AUTO and JACK tools.

One can also mention algorithms for checking symbolic (strong or observational) bisimulations between value-passing LTSs extended with variables, inputs, and assignments [306] [219] [307] [303] [248]. These algorithms have been transposed to compute bisimulations for the π-calculus [304] [308], but do not seem to have been implemented.
4 Retrospective of the 2000s

4.1 New Algorithm for Strong Bisimulation

A new fast bisimulation algorithm [136] [137] for strong bisimulation was proposed, which extends the Paige-Tarjan algorithm with a notion of state rank defined as the maximum distance to a sink state (if any), not counting transitions which are internal to strongly connected components. Fast bisimulation is more efficient than the Paige-Tarjan algorithm on LTSs containing sink states, especially acyclic LTSs. It was implemented in the COPS checker [360] for security properties. A symbolic version of this algorithm was implemented using BDDs [135].

A survey on the complexity of some of the behavioural equivalences presented in [406], considering their application to finite- and infinite-state models, can be found in [343].

4.2 New Bisimulation Tools

In the 2000s, a new generation of bisimulation tools appeared, which progressively superseded the tools developed during the previous decades.

BCG_MIN [174] used the Kanellakis-Smolka and the Groote-Vaandrager algorithms to minimize LTSs modulo strong and branching bisimilarities. BCG_MIN was released as a component of the CADP toolbox and used BCG as a native file format to represent LTSs compactly. Since BCG_MIN was globally more efficient than ALDEBARAN and could handle larger graphs, the original ALDEBARAN tool was replaced in 2005 by a backward-compatible shell script invoking BCG_MIN and other tools of CADP [176].

The \( \mu \text{CRL} \) toolset [38] [39] used partition-refinement algorithms to perform equivalence checking and minimization of LTSs modulo strong and branching bisimulations. It generated LTSs in the AUT and BCG formats of CADP, and implemented the OPEN/CÆSAR interface, so that most tools of CADP could be used on \( \mu \text{CRL} \) specifications. It was succeeded by the mCRL2 toolset [198], which also supports equivalence checking based on bisimulations.

TVT (Tampere Verification Tool) [417], which succeeded the ARA tool, implemented strong bisimulation and CFFD equivalence.

CHISIGMA [55] was a tool environment that could minimize, modulo strong and branching bisimulations, LTSs generated from specifications written in the process language \( \chi_\sigma \).

ABC [72] checked the equivalence of \( \pi \)-calculus processes modulo open bisimulation.

TAPAS [82], which was developed for teaching purpose, implemented comparison and minimization of LTSs modulo strong, observational, and branching
bisimulations.

LTSA (Labelled Transition System Analyser) [319] performed minimization modulo observational equivalence of LTSs generated from FSP (Finite-State Processes) specifications.

4.3 Bisimulation Tools Using On-the-Fly Verification

Significant advances in on-the-fly reduction and on-the-fly equivalence checking of LTSs have been made during the 2000s.

A key tool of CADP for on-the-fly verification is EXP.OPEN, which explores the state space of networks of LTSs composed together using synchronization vectors or parallel composition operators borrowed to various process calculi (CCS, CSP, LOTOS, LNT, or \(\mu\mathrm{CRL}\)), as well as hiding, renaming, cutting, or priority operators. EXP.OPEN was enhanced with on-the-fly partial-order reductions that preserve strong, branching, or stochastic branching bisimulations [289].

The ARCATS tool [90] [89] implemented a different approach to on-the-fly reduction: it incrementally generated an LTS reduced for branching bisimulation, by alternating steps of partial LTS generation and steps of branching minimization of the partial LTS already generated.

A generic library, named CÆSAR SOLVE [328], was developed, as part of CADP, for solving BESs (Boolean Equation Systems) [7] [326]. BESs are effective models in which both model-checking and equivalence-checking problems can be conveniently encoded. Given a BES, the CÆSAR SOLVE library explores an LTS on the fly, using the features provided by OPEN/CÆSAR, in order to compute the truth value of certain BES variables.

To reduce an LTS partially, during its generation, one can apply the notion of \(\tau\)-confluence [199], a form of partial-order reduction that analyzes \(\tau\)-transitions and preserves branching bisimulation. Based on this idea, an algorithm was proposed [45] and implemented in the \(\mu\mathrm{CRL}\) toolset, with the help of an automated theorem prover to identify \(\tau\)-confluent transitions.

This idea was also implemented in CADP by enhancing the aforementioned REDUCTOR tool that performed \(\tau\star\alpha\) reduction. The new version of REDUCTOR [327] supports many other reductions, among which \(\tau\)-confluence, \(\tau\)-closure (transitive reflexive closure over \(\tau\)-transitions), \(\tau\)-compression (collapsing of strongly connected components made of \(\tau\)-transitions), safety reduction, etc. It uses OPEN/CÆSAR and CÆSAR SOLVE to detect \(\tau\)-confluent transitions on the fly.

The definition of \(\tau\)-confluence was later generalized to visible actions [291], so as to enable better reductions in a compositional-verification setting.

CADP was also enriched with another tool, named BISIMULATOR [26] [330] [331], which checks whether two LTSs are equivalent modulo strong, branching, observational, or \(\tau\star\alpha\) bisimulations, as well as safety equivalence. The
comparison is expressed in terms of a BES, which is resolved on the fly using CÆSAR_SOLVE, one of the two LTSs being explored on demand using OPEN/CÆSAR. This general approach based on BESs subsumes the dedicated Fernandez-Mounier algorithms for checking bisimulations on the fly. BISIMULATOR, REDUCTOR, and BCG_MIN have been used in many case studies, e.g., the verification of a plant unit for drilling of metal products [329]. Conversely, BESs (which can be seen as a particular form of LTSs) can be minimized using strong bisimulation and an extension of strong bisimulation called idempotence-identifying bisimulation [267]. Such minimization preserves the truth values and, if applied before solving the original BESs, may speed up the resolution.

Equivalence checking of infinite-state models can be expressed in terms of PBESs (Parameterized Boolean Equation Systems) [326], an extension of BESs designed for model checking value-passing temporal-logic formulas [325]. This problem was addressed in [88] for branching and observational bisimulations. In this approach, the comparison of two models (represented using linear process equations) is encoded as a PBES, which is generated automatically, but whose resolution cannot be fully automated and may thus require human intervention.

4.4 Bisimulation Tools Based on Compositional Verification

Compositional verification makes intensive use of bisimulations and requires different tools for, e.g., generating, minimizing, and composing LTSs in parallel. To ease compositional verification, CADP was enriched with SVL [173] [288], which is both a scripting language and a compiler. SVL enables verification scenarios to be specified simply and executed efficiently, and thus offers an alternative to graphical user interfaces when dealing with complex, repetitive tasks. The SVL language has operators to generate, compose in parallel, minimize, and compare LTSs modulo various equivalence relations; LTSs can be either given in low-level formats (AUT, BCG, etc.) or specified using high-level languages (LOTOS, LNT [180], etc.); other operators support compositional-verification strategies and abstractions based on (handwritten or automatically generated [290]) interface specifications. The SVL compiler translates SVL scripts into shell scripts that invoke the appropriate CADP tools (BCG_MIN, EXP.OPEN, PROJECTOR, etc.), relieving users from taking care of command-line options and auxiliary files. SVL was used in several case studies [111] [395] [332].

4.5 Bisimulation Tools Based on Parallel/Distributed Computing

Blom and Orzan proposed both sequential and parallel/distributed algorithms for minimizing LTSs modulo strong bisimulation [40] [42] [43] [44] and branching bisimulation [41] (later improved in [46]). These algorithms, which are gathered in [352], are based on partition refinement and the novel concept of signatures (two states having different signatures cannot be bisimilar).
Although their sequential algorithms have worst-case time complexity \( O(mn^2) \), which is higher than the best algorithms for strong and branching bisimulations, they exhibit more opportunities for parallelization.

Indeed, their parallel algorithms, which distribute an LTS across several machines, exhibit a linear speed-up, meaning that the time taken by these algorithms linearly decreases when the number of machines increases. A policy based on abstract interpretation for distributing the LTS across machines was proposed in [353].

4.6 Bisimulation Tools Based on Symbolic Verification

The SIGREF tool [428] implemented several equivalence relations (including strong, branching, observational, and orthogonal [29] bisimulations, as well as safety equivalence) using sequential algorithms combining Blom-Orzan signatures with a BDD representation of the LTS. Several papers were published, detailing an algorithm for branching bisimulation [426], optimization techniques for BDD-based bisimulation computation [427], and an efficient algorithm for Markov chains [425].

4.7 Bisimulation Tools for Timed Systems

Research on timed bisimulations, which was very active in the 1990s, has seemingly slowed down during the 2000s. This is most likely related to the decline of research on timed process calculi, progressively replaced by simpler models based on timed automata. As a consequence, timed bisimulations and timed modal \( \mu \)-calculus have been replaced by more elementary analyses, such as reachability and safety properties computed on networks of timed automata. In this respect, the good performance of UPPAAL may have favored this evolution (see the related discussion in [421]). However, one can mention two advances in timed bisimulations during the 2000s.

A new bisimulation-based faster-than preorder [311] was proposed to compare asynchronous processes with respect to their worst-case timing behaviour.

A new equivalence-checking algorithm for timed branching bisimulation of communicating timed automata was designed and implemented in the RED tool [419], improving over prior algorithms for timed branching bisimulation.

4.8 Bisimulation Tools for Probabilistic and Stochastic Systems

The aforementioned BCG_MIN tool was designed to handle, not only LTSs, but also extended models combining normal transitions (labelled with an action), probabilistic transitions (whose label is a probability), and/or stochastic transitions (whose label is the rate parameter of an exponential distribution governing a random delay). Such models encompass DTMCs, which contain only prob-
abilistic transitions, CTMCs, which contain only stochastic transitions, IMCs (Interactive Markov Chains) [73] [223] [230], which contain both normal and stochastic transitions, and IPCs (Interactive Probabilistic Chains) [114] [112], which contain both normal and probabilistic transitions. BCG_MIN can minimize such models for various equivalences that combine strong or branching bisimulation with lumpability and, in the stochastic case, maximal progress. BCG_MIN, together with other tools for steady-state and transient analysis [228], has made CADP the actual successor of the TIPPtool [171] and has been used in several performance studies.

Version 4.0 of TwoTowers [2] implemented equivalence checking, for strong and observational Markovian equivalences, of architectural specifications written in a language named Æmilia. This tool was used to assess the securing strategy implemented in a trusted device for security architectures [1].

The PEPA workbench [185] was extended to support PEPA nets [186], a formalism that describes mobile agent systems using Petri nets in which mobile program code (expressed using the stochastic process calculus PEPA) moves across the places of the net. This workbench implemented net bisimulation [187], an equivalence relation for minimizing marking graphs of PEPA nets, and was used in many case studies.

Bisimulation algorithms for minimizing DTMCs and CTMCs have been proposed [131] [132]. These algorithms, which are based on symbolic representations and Blom-Orzan signatures, have been implemented in the SIGREF tool [425]. Independently, it has been evidenced that minimizing a DTMC or CTMC before analysis improves performance [265].

Symmetry reductions that preserve strong probabilistic bisimulation have been proposed [286], which may generate DTMCs, CTMCs, and MDPs (Markov Decision Processes) by several orders of magnitude smaller. These reductions have been implemented, using BDDs, in the PRISM model checker [287].

Finally, a minimization algorithm for observational bisimulation of acyclic IMCs with inputs and outputs was proposed, with an application to the analysis of DFTs (Dynamic Fault Trees) [116].

5 Retrospective of the 2010s

5.1 New Bisimulation Tools

The BCG_MIN tool of CADP was entirely rewritten in 2010. The new version 2.0 [177] [178] relies on the sequential Blom-Orzan algorithms for strong and branching bisimulations. Despite the worst-case time complexity $O(mn^2)$ of the Blom-Orzan algorithms is higher than the worst-case time complexity $O(mn)$ of the Groote-Vaandrager and Kanellakis-Smolka algorithms implemented in BCG_MIN 1.0, the new version of BCG_MIN is statistically faster, which seems to indicate that the worst cases for signature-based algorithms rarely occur in
practice (crafted worst-case examples are given in [196, Section 8]).

BCG\_CMP\textsuperscript{6} uses the same algorithms as BCG\_MIN 2.0 to check the equivalence of two LTSs modulo strong, branching, divergence-preserving branching, and observational bisimulations. When both LTSs are not equivalent, it generates an LTS explaining where and why bisimulation does not hold.

LTS\_MIN [47] [264] is a comprehensive model-checking tool set, which also implements strong and branching bisimulations using the distributed Blom-Orzan algorithms. LTS\_MIN is used as a backend by, e.g., the mCRL2 toolset [115].

The aforementioned FDR\_2 toolset for analyzing CSP processes was extended with new features [10], among which minimization algorithms [68] for strong, observational, and delay bisimulations to reduce state spaces before verification.

RELTS [335] [334] and T-BEG [276] are tools that implement strong bisimulation in terms of game theory. Another tool [79] defines various simulation relations in terms of an antagonistic two-player game. Also, game-theoretic definitions of branching and divergence-preserving branching bisimulations have been given in [120].

Educational motivation was behind the development of tools such as CAAL (Concurrency Workbench, Aalborg Edition) [9], which implements various strong or observational, timed or untimed, equivalences and preorders, and PSEUCO [37], which supports strong bisimulation. These tools exhibit fancy Web interfaces that help teaching concurrency theory in university courses.

One can also mention SMART [346], which implements strong and observational bisimulations using multiway decision diagrams, GREASE [163], which checks strong and observational bisimulations on-the-fly using syntactic criteria to try finding a counter-example as soon as possible, and an implementation of branching bisimulation dedicated to the reduction of BIP (Behaviour-Interaction-Priority) models [351].

5.2 Bisimulation Tools for Probabilistic and Stochastic Systems

Foundations were laid for strong and observational bisimulations and preorders on Markov automata, a combination of probabilistic automata and IMCs [138].

The aforementioned minimization tool BCG\_MIN 2.0 was equipped with probabilistic and stochastic bisimulations [113]. For these relations (as well as for strong and branching bisimulations on LTSs), BCG\_MIN 2.0 was found to use less memory and to be faster than BCG\_MIN 1.0 [178]. Support for the same probabilistic and stochastic bisimulations was added in BCG\_CMP too.

MRMC (Markov Reward Model Checker) [435] [266] is a tool for verifying properties (expressed as CSL or PCTL temporal-logic formulas with their reward extensions) on probabilistic models. To alleviate state explosion, MRMC may minimize these models modulo strong bisimulation.

\textsuperscript{6} http://cadp.inria.fr/man/bcg\_cmp.html
Polynomial algorithms for probabilistic observational bisimulation on probabilistic automata \cite{237} \cite{211} and alternating probabilistic bisimulation on interval MDPs \cite{212} were proposed and implemented. The latter relation was shown to be compositional \cite{210}.

An algorithm \cite{130} for directly generating a DTMC minimized modulo probabilistic bisimulation from a probabilistic program described by guarded commands was proposed and implemented in the PRISM model checker using the SMT solver Z3. Another approach for generating, using PRISM, a DTMC from an RTL (Register Transfer Level) description and minimizing it, using SIGREF, modulo probabilistic bisimulation can be found in \cite{87}.

An approach was proposed \cite{384} to accelerate the model checking of PCTL formulas on probabilistic automata, by iteratively refining an abstraction of a probabilistic automaton, using incrementally computed bisimulations and without resorting to any kind of counterexample analysis.

A fast algorithm for strong probabilistic bisimulation \cite{204} was proposed and implemented in the mCRL2 toolset \cite{80}.

Another fast algorithm \cite{257} was given to minimize, modulo branching bisimulation, DTMCs with labelled states.

Probabilistic bisimulation was also applied to infinite-state parameterized systems, i.e., systems with an arbitrary number of processes \cite{243}. The approach was experimented in a prototype tool that was not made public.

### 5.3 Bisimulation Tools for Mobile Systems

Two new tools for analyzing extensions of the π-calculus were released during the 2010s.

PWB (Psi-Calculus Workbench) \cite{51} \cite{52} was a generic tool for analyzing mobile processes by means of symbolic simulation and equivalence checking modulo symbolic (strong or observational) bisimulations \cite{260}.

SPEC \cite{392} was an equivalence-checking tool for open bisimulation on security protocols specified in the spi-calculus.

### 5.4 Bisimulation Tools Based on Parallel/Distributed Computing

There have been commendable efforts to parallelize mainstream partition-refinement algorithms for minimizing LTSs modulo strong bisimulation. A parallel version of the Paige-Tarjan algorithm, in combination with Blom-Orzan signatures, was proposed in \cite{285}, and a parallel version of the Kanellakis-Smolka algorithm is given in \cite{322}.

Combinations of symbolic techniques and parallel algorithms have also been explored in the second half of the 2010s. SIGREFMC \cite{402} \cite{405} was a bisimulation tool providing the same functionalities as SIGREF, but based on SYLVAN \cite{403}.
[404], a parallel implementation of BDDs on multi-core architectures.

While SIGREFMC encoded bisimulations as partitions of the state space, in the lineage of partition-refinement algorithms, a different approach was investigated in [246] [247], where strong bisimulation was encoded directly as a relation, like in the Bouali-De Simone algorithm.

### 5.5 Bisimulation Tools Based on Compositional Verification

A new approach, called smart reduction [117], was proposed for the compositional minimization of networks of LTSs. Smart reduction analyzes the synchronizations between concurrent processes to infer a suitable order in which processes are composed and minimized. Such a heuristic, which tries to avoid state explosion by keeping the size of intermediate LTSs as small as possible, was implemented in the SVL scripting language of CADP [175].

A new family of equivalence relations, named sharp bisimulations [294], which combine strong bisimulation and divergence-preserving branching bisimulation [333], was defined. Sharp bisimulations provide effective reductions while preserving given temporal-logic formulas [293]. They have been implemented in the BCG_MIN and BCG_CMP tools of CADP, with user-friendly support in SVL.

Smart reduction and sharp bisimulations play a major role in modern approaches to compositional verification. Together with recent developments [292] around the idea of partial model checking [8], they enabled scientists from Grenoble and Pisa to solve nearly all the parallel problems of the RERS\(^7\) verification challenge in 2019\(^8\) and 2020\(^9\).

### 5.6 Recent Results for Strong and Branching Bisimulations

An asymptotic lower bound \(\Omega((m+n) \log n)\) on the time complexity of partition refinement algorithms was established [197].

A new (successively revised, improved, and simplified) minimization algorithm for branching bisimulation was proposed [205] [206] [196] [255] [256]. Its worst-case time complexity \(O(m \log n)\) is lower than that of the Groote-Vaandrager algorithm, which has been the best-known algorithm since the early 1990s, and equal to that of the Paige-Tarjan algorithm, which is still the reference algorithm for strong bisimulation. This new algorithm has been implemented in the mCRL2 toolset.

\(^7\) http://rers-challenge.org
\(^8\) http://cadp.inria.fr/news12.html
\(^9\) http://cadp.inria.fr/news13.html#section-3
6 Conclusion

Although model checking and equivalence checking have been discovered nearly at the same time in the early 1980s, model checking is now widespread in academia and industry, whereas equivalence checking plays a more discrete role. It nevertheless found numerous applications in the verification of communication protocols, hardware circuits, distributed systems, security systems, web services, etc. Actually, compared to model checking, equivalence checking presents several advantages:

- It is conceptually simpler, as it does not require learning another language (i.e., temporal logics) to express the properties under verification.
- It enables visual checking, an easy form of verification done by abstracting away certain observable actions of the system (i.e., by renaming them to $\tau$-transitions), minimizing the resulting state space modulo some weak bisimulation, and visually inspecting the minimized state space if it is small enough.
- It may increase the effectiveness of model checking, as compositional state-space verification techniques based upon, e.g., congruence properties, smart reductions, and sharp bisimulations, are often capable of generating large state spaces that could not be explored otherwise.

For these reasons, we believe that equivalence checking should play a growing role in the future, in close combination with model checking. This could resolve the longstanding dilemma between state-based models (in which information is attached to states, as in Kripke structures) and action-based models (in which information is attached to transitions, as in LTSs and Markov chains) by giving an advantage to the latter models. Model checking is equally applicable to both action- and state-based models (although with slightly different temporal logics [126]), but most bisimulation tools have been designed to operate on action-based models, which suggests that the latter models are more suitable where model checking and equivalence checking are to be used together.

During the last forty years, the development of algorithms and tools for checking bisimulations on finite- or infinite-state systems has steadily progressed. These essential achievements have been spanning over several decades, which is no surprise, keeping in mind how theoretically involved are these algorithms and how technically involved are these tools subject to severe performance requirements. A remarkable example of such long-lasting research and commitment is the Groote-Vaandrager algorithm for branching bisimulation [201], which has been gradually refined to lower its complexity [256].

It is worth noticing that most of the bisimulation tools developed for equivalence checking are no longer available today. Quite often, publications are the only remaining indication that such tools have existed; in some cases (e.g., for the
promising BIDMIN tool), formal publications are even lacking. A counterexample is the CADP toolbox, the bisimulation tools of which have been, over several decades, constantly enhanced or replaced by better, backward-compatible tools.

Software tools get obsolete due to incompatible evolutions of programming languages and operating systems, but they also get abandoned when their authors leave academia or move from one university to another; this suggests that overemphasis on professional mobility may hamper long-term development of perennial software tools.

Finally, the development of bisimulation tools has probably suffered from additional factors, among which: (i) the lack of standard file formats agreed upon by the community, beyond the rather inefficient AUT format; (ii) the lack of benchmark examples, with the notable exception of VLTS\textsuperscript{10}, which plays the role of a de-facto test suite; and (iii) the lack of yearly software competitions dedicated to equivalence checking. We hope that the present survey will draw the attention to the past achievements and future promises of this research field.

Acknowledgements

We are grateful to Rance Cleaveland, Rocco De Nicola, Jan Friso Groote, Laurent Mounier, Elie Najm, and the anonymous referees for their valuable comments about this article. We also would like to thank the DBLP and Google Scholar teams, whose long-term undertaking made it possible to tackle such a comprehensive study.

\textsuperscript{10} \url{https://cadp.inria.fr/resources/vlts}
References


48


[305] Reiner Lichtenecker, Klaus Gotthardt, and Janusz Zalewski. Automated Verification of Communication Protocols Using CCS and BDDs. In José D. P. Rolim,


H. Garavel and F. Lang

Equivalence Checking 40 Years After...


56


H. Garavel and F. Lang  

Equivalence Checking 40 Years After...


on Digital System Design: Architectures, Methods and Tools (Milano, Italy). IEEE, September 1999.


