HAL
open science

Improved On-the-Fly Equivalence Checking using Boolean Equation Systems

Radu Mateescu, Emilie Oudot

To cite this version:

Radu Mateescu, Emilie Oudot. Improved On-the-Fly Equivalence Checking using Boolean Equation Systems. [Research Report] RR-6777, INRIA. 2008, pp.31. inria-00347627

HAL Id: inria-00347627

https://inria.hal.science/inria-00347627v1

Submitted on 16 Dec 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Improved On-the-Fly Equivalence Checking using Boolean Equation Systems

Radu Mateescu — Emilie Oudot

N° 6777

Décembre 2008

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6777--FR+ENG

Improved On-the-Fly Equivalence Checking using Boolean Equation Systems

Radu Mateescu* , Emilie Oudot**

Thème COM — Systèmes communicants

Projet Vasy

Rapport de recherche n° 6777 — Décembre 2008 — 31 pages

Abstract: Equivalence checking is a classical verification method for ensuring the compatibility of a finite-state concurrent system (*protocol*) with its desired external behaviour (*service*) by comparing their underlying labeled transition systems (LTSs) modulo an appropriate equivalence relation. The local (or *on-the-fly*) approach for equivalence checking combats state explosion by exploring the synchronous product of the LTSs incrementally, thus allowing an efficient detection of errors in complex systems. However, when the two LTSs being compared are equivalent, the on-the-fly approach is outperformed by the global one, which completely builds the LTSs and computes the equivalence classes between states using partition refinement. In this report, we consider the technique based on translating the on-the-fly equivalence checking problem in terms of the local resolution of a boolean equation system (BES). We propose two enhancements of this technique in the case of equivalent LTSs: a new, faster encoding of equivalence relations in terms of BESs, and a new local BES resolution algorithm with a better average complexity. These enhancements were incorporated into the BISIMULATOR 2.0 equivalence checker of the CADP toolbox, and they led to significant performance improvements w.r.t. existing on-the-fly equivalence checking algorithms.

Key-words: bisimulation, boolean equation system, equivalence checking, labeled transition system, on-the-fly verification

A short version of this report is also available as "Improved On-the-Fly Equivalence Checking using Boolean Equation Systems", *Proceedings of the 15th International SPIN Workshop on Model Checking of Software SPIN'2008 (Los Angeles, USA)*, August 2008.

* Radu.Mateescu@inria.fr

** Current e-mail address: Emilie.Oudot@lifc.univ-fcomte.fr

Centre de recherche INRIA Grenoble – Rhône-Alpes

Inovallée, 655, avenue de l'Europe, Montbonnot, 38334 Saint Ismier Cedex (France)

Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Vérification améliorée à la volée par équivalences au moyen de systèmes d'équations booléennes

Résumé : La vérification par équivalences (*equivalence checking*) est une méthode classique pour assurer la compatibilité entre un système concurrent à nombre fini d'états (*protocole*) et son comportement externe désiré (*service*) en comparant leurs systèmes de transitions étiquetées (STEs) sous-jacents selon une relation d'équivalence appropriée. L'approche locale (ou *à la volée*) pour la vérification par équivalences lutte contre l'explosion d'états en explorant le produit synchrone des STEs de manière incrémentale, permettant ainsi une détection efficace des erreurs dans des systèmes complexes. Cependant, quand les deux STEs à comparer sont équivalents, l'approche à la volée est outrepassée par l'approche globale, qui construit complètement les STEs et calcule les classes d'équivalence entre états par raffinement de partitions. Dans ce rapport, nous considérons la technique basée sur une traduction du problème de la vérification par équivalences vers la résolution locale d'un système d'équations booléennes (SEB). Nous proposons deux améliorations de cette technique dans le cas des STEs équivalents : un nouveau codage, plus rapide, des relations d'équivalence en termes de SEBs et un nouvel algorithme de résolution locale de SEBs avec une meilleure complexité moyenne. Ces améliorations one été incorporées dans le vérificateur par équivalences BISIMULATOR 2.0 de la boîte à outils CADP et ont conduit à des améliorations significatives des performances par rapport aux algorithmes existants de vérification à la volée par équivalences.

Mots-clés : bisimulation, système d'équations booléennes, système de transitions étiquetées, vérification par équivalences, vérification à la volée

1 Introduction

Equivalence checking is a classical verification method for finite-state concurrent systems that consists in comparing the behaviour of the system under design (typically, a *protocol* or a low-level hardware description) with its desired external behaviour (typically, a *service* or a high-level hardware description) modulo a suitable equivalence relation. Protocol and service behaviours are usually represented as labeled transition systems (LTSs), and the relations most used for comparing them are the *bisimulations* defined in the framework of process algebras, such as CCS [37], CSP [9], or ACP [6] and of the formal specification languages inspired from them, such as LOTOS [26] or CHP [31]. In practice, LTSs are often represented in two complementary ways, which also determine the nature of equivalence checking algorithms: either *explicitly*, by their list of states and transitions, or *implicitly*, by their "successor function" returning the set of transitions going out of a given state. The implicit and explicit LTS representations are suitable for protocols (which are usually large) and services (which are usually small), respectively.

There are basically two approaches for equivalence checking: the *global* one [15], which operates on explicit LTSs, computes the equivalence classes of states by using partition refinement and then checks whether the initial states of the two LTSs fall into the same equivalence class; and the *local* one [12], which operates on implicit LTSs, explores the synchronous product between the two LTSs and searches for mismatches indicating the non equivalence of their initial states. Global algorithms are more effective when the two LTSs are equivalent, but require their complete construction, which is limited for large systems by the amount of memory available. Local (or *on-the-fly*) algorithms are more effective when the LTSs are not equivalent, allowing the detection of errors in complex systems even when the global approach would fail. Therefore, on-the-fly algorithms are useful at the beginning of the verification process, when errors occur frequently and must be detected quickly, whereas global algorithms are more suitable at a later stage, once the formal descriptions of the protocol and the service are stable and their underlying LTSs become equivalent.

Our objective is to improve the performance of on-the-fly equivalence checking algorithms when the LTSs to be compared are equivalent, which is the worst case for this class of algorithms because it forces them to explore the synchronous product of the two LTSs entirely. This would combine the advantages of global and local verification, making the on-the-fly approach suitable throughout the verification process. We consider here the technique relying on the translation of on-the-fly equivalence checking to the local resolution of a boolean equation system (BES) [2, 35]. This technique involves two clearly separated aspects, namely the BES encodings of bisimulation relations and the local BES resolution algorithms, which can be developed and optimized independently. To improve performance, we seek to enhance both these aspects.

First, we devise new BES encodings of the branching [44] and weak [37] bisimulations, obtained by migrating a part of the computation of transitive reflexive closures over internal steps (τ-closures) into the boolean equations. This simplifies the structure of BES equations considerably and reveals to be faster than computing τ-closures separately by using spe-

cialized algorithms [34]. Second, we propose a new local BES resolution algorithm, which exhibits a smaller average complexity than previously published algorithms [1, 47, 17, 35]. Our algorithm is based on a suspend/resume depth first search (sr-DFS) of the dependencies between boolean variables, and stops as soon as the BES portion explored contains a single example or counterexample for the boolean variable to be solved, therefore being optimal from this point of view.

These two enhancements led to version 2.0 of the BISIMULATOR [35] equivalence checker of the CADP [22] verification toolbox. The tool was developed using the generic OPEN/CÆSAR [21] environment for on-the-fly manipulation of LTSs, and uses as verification engine the CÆSAR_SOLVE [35] library for on-the-fly resolution of BESs. The enhancements led to a significant performance increase w.r.t. BISIMULATOR 1.0, as we observed on LTSs generated from protocol and hardware descriptions or taken from the VLTS benchmark suite [46].

Related work. On-the-fly equivalence checking algorithms [12] received relatively little attention from the verification community, the research being mainly focused on optimizing global algorithms based on partition refinement [15, 20]. Among the first on-the-fly equivalence checking algorithms were those proposed in [18] and subsequently implemented in the ALDÉBARAN tool [19]. Two different algorithms were elaborated: the first one compares deterministic LTSs by searching their synchronous product for a pair of non equivalent states, and the second one handles nondeterministic LTSs by assuming that certain couples of states are equivalent and by backtracking in the synchronous product whenever such an assumption turns out to be wrong. The verification technique based on BES resolution allows one to reproduce the first algorithm by observing that the BESs corresponding to bisimulations between deterministic LTSs are conjunctive, and by devising a specialized local resolution algorithm for this case [35]. The algorithm for the nondeterministic case is outperformed in practice by local BES resolution algorithms, as it was observed experimentally [5].

Another approach of checking the equivalence of two LTSs is to rephrase the problem as the model checking on one LTS of a *characteristic formula* [25] in modal μ-calculus derived from the other LTS. This approach was elegantly implemented in the Concurrency Workbench [13, 10], but was hampered in practice for large LTSs by the prohibitive size of characteristic formulas, which is at least of the same order as the LTS size. The quest for performance was pursued by considering other intermediate formalisms suitable for representing equivalence checking, such as the BESs, which are lower-level than the modal μ-calculus and therefore are likely to require less computation effort.

Encodings of branching and weak bisimulation using BESs of alternation depth two were proposed in [2]. These BESs contain two mutually recursive equation blocks, a maximal fixed point one encoding the bisimulation relation, and a minimal fixed point one encoding the τ-closures to be computed in the input LTSs. The local resolution algorithms underlying this class of BESs have a quadratic complexity w.r.t. the BES size [47], which makes them impractical for large LTSs; no implementation of this approach was reported as far as we know. Although a subquadratic algorithm for solving BESs with disjunctive/conjunctive equation blocks of arbitrary alternation depth was proposed in [24], it does not seem to

capture the BESs for branching and weak bisimulations, which consist of a disjunctive block encoding τ-closures and a general block encoding the equivalence relation. Simpler encodings of weak equivalences using alternation-free BESs, obtained by leaving the computation of τ-closures (possibly enhanced with on-the-fly τ-confluence reduction [38]) outside the BES, proved to be practically effective [35]. The resulting BESs can be solved using the many local resolution algorithms available [28, 1, 47, 29, 17, 35].

An alternative approach consists in formulating equivalence checking by means of Horn clauses [40], which can be solved using classical HORNSAT resolution algorithms [16, 4]. We believe that BES encodings provide a more direct way of connecting on-the-fly equivalence checking to graph exploration algorithms. In fact, local BES resolution algorithms, such as the one presented in this report, can also be used for solving HORNSAT efficiently, by applying the translation from Horn clauses to BESs proposed in [29].

Report outline. Section 2 defines the class of BESs we use and illustrates the functioning of local resolution algorithms. Section 3 proposes new, faster BES encodings for branching and weak bisimulations. Section 4 describes our new local resolution algorithm, and Section 5 shows experimentally its performance when applied to equivalence checking. Section 6 gives some concluding remarks and directions for future work. Annex A contains the proofs of our BES encodings for strong and branching bisimulation.

2 Background

A boolean equation system (BES) is a set of possibly recursive equations $B = \{X_i \stackrel{\sigma}{=} X_{i1}\ op_i \cdots op_i\ X_{im_i}\}_{1 \leq i \leq n}$, where $X_i \in \mathcal{X}$ are boolean variables, $op_i \in \{\vee, \wedge\}$ are disjunctive or conjunctive connectors, and $\sigma \in \{\mu, \nu\}$ is a minimal or maximal fixed point sign. An empty disjunction (resp. conjunction) is equivalent to the false (resp. true) constant. Each boolean variable occurring in the right-hand side of an equation must be defined by some equation of the BES. A variable X_i is said to be disjunctive (resp. conjunctive) iff $op_i = \vee$ (resp. $\wedge$). BESs of this kind are called *simple*, because each of their equations contains a single type of boolean connector (either $\vee$, or $\wedge$) in its right-hand side. Any BES containing arbitrary combinations of boolean connectors in the right-hand sides of its equations can be brought to the simple form with at most a linear blow-up in size, by introducing new equations to factor subformulas [3]. We focus our attention on BESs with a single equation block (i.e., set of equations having the same fixed point sign), since they are suitable for encoding equivalence checking problems [35]; more general BESs with multiple blocks are used for encoding model checking problems [14, 36]. In-depth presentations of the theory and applications of BESs can be found in [1, 30].

For each equation i of a BES, the evaluation of the formula in its right-hand side yields a boolean value defined as follows:

$$[\![X_{i1}\ op_i \cdots op_i\ X_{im_i}]\!]\delta = \delta(X_{i1})\ op_i \cdots op_i\ \delta(X_{im_i}).$$

where the context $\delta : \mathcal{X} \rightarrow \mathsf{Bool}$ is a partial function assigning boolean values to all variables occurring in the formula. The solution of a BES is a vector $\langle v_1, ..., v_n \rangle \in \mathsf{Bool}^n$ equal to the fixed point $\sigma\Phi$ of the functional $\Phi : \mathsf{Bool}^n \rightarrow \mathsf{Bool}^n$ associated to the BES:

$$\Phi(b_1, ..., b_n) = \langle [\![X_{i1} \ op_i \cdots op_i \ X_{im_i}]\!][b_1/X_1, ..., b_n/X_n] \rangle_{1 \leq i \leq n}$$

where $[b_1/X_1, ..., b_n/X_n]$ is the context assigning the boolean value b_i to variable X_i for $1 \leq i \leq n$. Since the boolean formulas in a BES do not contain negation operators, the functional Φ is monotonic, which ensures the existence of its minimal and maximal fixed points on $\langle \mathsf{Bool}^n, \mathsf{false}^n, \mathsf{true}^n, \vee^n, \wedge^n \rangle$, the pointwise extension of the boolean lattice [27]. In the sequel, we consider only maximal fixed point BESs (i.e., with $\sigma = \nu$), which allow to encode equivalence checking.

The local resolution of a BES B, which underlies on-the-fly verification (based on a forward exploration of LTSs), amounts to computing the solution v_i of a particular variable X_i by solving as few equations of B as possible. Local resolution algorithms are easier to devise and understand by representing BESs as *boolean graphs* [1]. Given a BES $B = \{X_i \stackrel{\sigma}{=} X_{i1} \ op_i \cdots op_i \ X_{im_i}\}_{1 \leq i \leq n}$, its associated boolean graph $G = (V, E, L)$ is defined as follows: $V = \{X_1, ..., X_n\}$ is the set of vertices (boolean variables), $E = \{(X_i, X_j) \mid 1 \leq i \leq n \wedge j \in \{i_1, ..., i_{m_i}\}\}$ is the set of edges (dependencies between variables), and $L : V \rightarrow \{\vee, \wedge\}$, $L(X_i) = op_i$ for $1 \leq i \leq n$ is the labeling of vertices as disjunctive or conjunctive. The constant false (resp. true) is represented as a sink $\vee$-vertex (resp. $\wedge$-vertex). The local resolution of a vertex X_i consists in two activities performed simultaneously [1, 47, 35]: a forward exploration of the boolean graph along its edges, starting at X_i; and a backward propagation of the *stable* variables found, i.e., whose boolean value has been computed. An example of local BES resolution is shown on Figure 1. The local resolution algorithm used is based on a depth-first search (DFS) of the boolean graph, starting at the variable of interest X_1. The light grey area delimits the boolean subgraph explored during resolution. Black (resp. white) vertices correspond to variables whose solution is true (resp. false).

$$\begin{cases} X_1 \stackrel{\nu}{=} X_2 \wedge X_7 \\ X_2 \stackrel{\nu}{=} X_3 \wedge X_5 \\ X_3 \stackrel{\nu}{=} X_1 \vee X_4 \\ X_4 \stackrel{\nu}{=} \mathsf{false} \\ X_5 \stackrel{\nu}{=} X_5 \vee X_6 \\ X_6 \stackrel{\nu}{=} \mathsf{true} \\ X_7 \stackrel{\nu}{=} X_2 \vee X_6 \vee X_8 \\ X_8 \stackrel{\nu}{=} X_7 \wedge X_9 \\ X_9 \stackrel{\nu}{=} \mathsf{false} \end{cases}$$

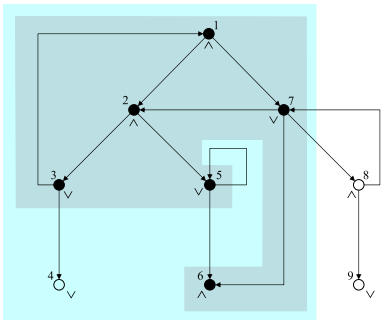

Figure 1: Boolean graph-based local resolution of variable X_1

The solution of a BES can also be characterized by interpreting on its boolean graph the following *example formula* [32] written in modal μ-calculus:

$$\phi_{ex} = \nu X.(P_\vee \wedge \langle - \rangle X) \vee (P_\wedge \wedge [-]X)$$

where the atomic propositions $P_\vee$ and $P_\wedge$ denote $\vee$-vertices and $\wedge$-vertices, respectively. Formula ϕ_{ex} expresses that every $\vee$-vertex (resp. $\wedge$-vertex) satisfying ϕ_{ex} must have one (resp. all) of its successors satisfying ϕ_{ex}. The solution v_i of a variable X_i is true iff the corresponding vertex X_i satisfies ϕ_{ex} in the boolean graph. A positive diagnostic (or *example*) for vertex X_i is a boolean subgraph that contains X_i and is a model for ϕ_{ex}. Dually, a negative diagnostic (or *counterexample*) for X_i is a boolean subgraph containing X_i and being a model for the counterexample formula $\phi_{cx} = \neg\phi_{ex}$. The dark grey area shown on Figure 1 delimits an example for X_1, generated by traversing again the boolean subgraph explored during resolution [32].

3 Encoding Bisimulation Relations as BESs

Labeled transition systems (LTSs) are the semantic model underlying process algebras [7] and the related languages, such as LOTOS [26] and CHP [31]. An LTS is a quadruple $M = \langle Q, A, T, q_0\rangle$, where Q is the set of states, A is the set of actions (including the internal action τ), $T \subseteq Q \times A \times Q$ is the transition relation, and $q_0 \in Q$ is the initial state. A transition $\langle p, a, q\rangle \in T$ (also written $p \xrightarrow{a} q$) indicates that the system can move from state p to state q by performing action a. The notation is extended to transition sequences: $p \xrightarrow{l} q$ denotes the existence of a sequence going from p to q and whose concatenated labels form a word of the language $l \subseteq A^*$.

To compare the LTSs modeling the behaviour of concurrent systems, various equivalence relations were proposed (see [45] for a survey), among which *bisimulations* are most useful in practice due to their congruence properties w.r.t. the parallel composition operators of process algebras. We consider here three widely-used bisimulations: strong [39], branching [44], and weak [37], the last two being originally proposed as native equivalence relations for ACP [6] and CCS [37], respectively. Given two LTSs $M_i = \langle Q_i, A_i, T_i, q_{0i}\rangle$ with $i \in \{1, 2\}$, a bisimulation $\approx \subseteq Q_1 \times Q_2$ is a relation such that $p \approx q$ if $\forall p \xrightarrow{a} p'.\exists q \xrightarrow{a} q'.p' \approx q'$ and $\forall q \xrightarrow{a} q'.\exists p \xrightarrow{a} p'.p' \approx q'$, where $p, p' \in Q_1$, $a \in A_1 \cup A_2$, and $q, q' \in Q_2$. Bisimulations are closed under union, and the strong bisimulation $\approx_s$ is defined as the greatest one, i.e., the union of all bisimulations. M_1 is strongly equivalent to M_2 (notation $M_1 \approx_s M_2$) iff $q_{01} \approx_s q_{02}$.

A basic encoding of this mathematical definition as a maximal fixed point BES is shown in Table 1 (upper part, first row). The fact that $p \approx_s q$ is encoded as a boolean variable X_{pq} defined by an equation whose right-hand side boolean formula is directly derived from the two bisimulation conditions. The correctness of this encoding scheme (see the proof in Annex A.1), which reformulates the definition of strong bisimulation in propositional logic instead of first-order logic, relies on a bijection between the set of bisimulations and the set of fixed point solutions of the functional associated to the BES. To obtain a simple BES compliant with the definition given in Section 2, we introduce the new variables $Y_{p'qa}$ and $Z_{pq'a}$ such that each right-hand side formula contains a single type of boolean connector (second row). The BES for the strong preorder relation $\preceq_s$ is obtained by keeping only the coloured parts of the equations. Checking the strong bisimilarity of M_1 and M_2 amounts to solving

the variable $X_{q_{01}q_{02}}$ of this BES, which can be carried out using a local resolution algorithm. The evaluation of the boolean formulas in the right-hand sides of the equations defining X_{pq}, $Y_{p'qa}$, and $Z_{pq'a}$ triggers a forward exploration of transitions in M_1 and M_2, which enables an incremental construction of both LTSs, and therefore an on-the-fly verification.

Table 1: Basic and full BES encodings of three widely-used bisimulations

Strong bisimulation		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} \bigvee_{q \xrightarrow{a} q'} X_{p'q'} \wedge \bigwedge_{q \xrightarrow{a} q'} \bigvee_{p \xrightarrow{a} p'} X_{p'q'}$		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{p'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pq'a}$	$Y_{p'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} X_{p'q'}$	$Z_{pq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} X_{p'q'}$
Branching bisimulation		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'}((a = \tau \wedge X_{p'q}) \vee \bigvee_{q \xrightarrow{\tau^*} q' \xrightarrow{a} q''}(X_{pq'} \wedge X_{p'q''})) \wedge$ $\bigwedge_{q \xrightarrow{a} q'}((a = \tau \wedge X_{pq'}) \vee \bigvee_{p \xrightarrow{\tau^*} p' \xrightarrow{a} p''}(X_{p'q} \wedge X_{p''q'}))$		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{pp'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pqq'a}$	$Y_{pp'qa} \stackrel{\nu}{=} (a = \tau \wedge X_{p'q}) \vee U_{pp'qa}$	
$Z_{pqq'a} \stackrel{\nu}{=} (a = \tau \wedge X_{pq'}) \vee V_{pqq'a}$	$U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} W_{pp'qq'} \vee \bigvee_{q \xrightarrow{\tau} q'} U_{pp'q'a}$	
$V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} W_{pp'qq'} \vee \bigvee_{p \xrightarrow{\tau} p'} V_{p'qq'a}$	$W_{pp'qq'} \stackrel{\nu}{=} X_{pq} \wedge X_{p'q'}$	
Weak bisimulation		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'}((a = \tau \wedge \bigvee_{q \xrightarrow{\tau^*} q'} X_{p'q'}) \vee \bigvee_{q \xrightarrow{\tau^*.a.\tau^*} q'} X_{p'q'}) \wedge$ $\bigwedge_{q \xrightarrow{a} q'}((a = \tau \wedge \bigvee_{p \xrightarrow{\tau^*} p'} X_{p'q'}) \vee \bigvee_{p \xrightarrow{\tau^*.a.\tau^*} p'} X_{p'q'})$		
$X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{p'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pq'a}$		
$Y_{p'qa} \stackrel{\nu}{=} (a = \tau \wedge U_{p'q}) \vee V_{p'qa}$	$U_{p'q} \stackrel{\nu}{=} X_{p'q} \vee \bigvee_{q \xrightarrow{\tau} q'} U_{p'q'}$	
$V_{p'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} U_{p'q'} \vee \bigvee_{q \xrightarrow{\tau} q'} V_{p'q'a}$	$Z_{pq'a} \stackrel{\nu}{=} (a = \tau \wedge W_{pq'}) \vee T_{pq'a}$	
$W_{pq'} \stackrel{\nu}{=} X_{pq'} \vee \bigvee_{p \xrightarrow{\tau} p'} W_{p'q'}$	$T_{pq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} W_{p'q'} \vee \bigvee_{p \xrightarrow{\tau} p'} T_{p'q'a}$	

Similar encoding schemes hold for the branching ($\approx_b$) and weak ($\approx_w$) bisimulations, as shown in Table 1 (middle and lower parts, first rows). The important difference w.r.t. strong bisimulation is the presence of transitive reflexive closures over τ-transitions, which correspond to the abstraction of internal activity done by these two bisimulations. The simple BESs derived from these encodings by introducing new variables (similarly to strong bisimulation as shown above) were successfully used as basis for on-the-fly equivalence checking in conjunction with linear-time local BES resolution algorithms [35]. For LTSs with a high percentage of τ-transitions, the encodings of weak bisimulations yield relatively small BESs, but shift the computation effort to the evaluation of right-hand side boolean formulas, which involve various forms of τ-closures ($p \xrightarrow{\tau^*} p' \xrightarrow{a} p''$, $p \xrightarrow{\tau^*.a.\tau^*} p'$, and $p \xrightarrow{\tau^*} p'$) having a quadratic worst-case complexity. Practical usage confirmed that computation of τ-closures, even using optimized algorithms [34], is the most time-consuming part of the verification process.

An alternative solution for computing τ-closures would be to encode them directly using boolean equations, yielding the BESs shown on Table 1 (middle and lower part, second rows); however, this works only for τ-convergent LTSs (i.e., without τ-cycles). To see this, consider the two LTSs $M_1 = \langle \{p_0, p_1\}, \{a, \tau\}, \{p_0 \xrightarrow{\tau} p_0, p_0 \xrightarrow{a} p_1\}, p_0 \rangle$ and $M_2 = \langle \{q_0, q_1\}, \{b, \tau\}, \{q_0 \xrightarrow{\tau} q_0, q_0 \xrightarrow{b} q_1\}, q_0 \rangle$, which are obviously not equivalent modulo any of the three bisimulations considered since they have different action sets. The comparison of M_1 and M_2 modulo weak bisimulation yields the BES below:

$$X_{p_0q_0} \stackrel{\nu}{=} Y_{p_0q_0\tau} \wedge Y_{p_1q_0a} \wedge Z_{p_0q_0\tau} \wedge Z_{p_0q_1b}$$

$$\begin{array}{lll} Y_{p_0q_0\tau} \stackrel{\nu}{=} U_{p_0q_0} \vee V_{p_0q_0\tau} & U_{p_0q_0} \stackrel{\nu}{=} X_{p_0q_0} \vee U_{p_0q_0} & V_{p_0q_0\tau} \stackrel{\nu}{=} V_{p_0q_0\tau} \\ Z_{p_0q_0\tau} \stackrel{\nu}{=} W_{p_0q_0} \vee T_{p_0q_0\tau} & W_{p_0q_0} \stackrel{\nu}{=} X_{p_0q_0} \vee W_{p_0q_0} & T_{p_0q_0\tau} \stackrel{\nu}{=} T_{p_0q_0\tau} \\ Y_{p_1q_0a} \stackrel{\nu}{=} V_{p_1q_0a} & V_{p_1q_0a} \stackrel{\nu}{=} V_{p_1q_0a} \quad Z_{p_0q_1b} \stackrel{\nu}{=} T_{p_0q_1b} & T_{p_0q_1b} \stackrel{\nu}{=} T_{p_0q_1b} \end{array}$$

We can easily compute the maximal fixed point solution of this BES by using Kleene's iterative characterization [27], which consists in initializing all variables to true and repeatedly evaluating the right-hand sides of equations until the values of all variables become stable; the process converges in one iteration and all variables remain true, erroneously indicating that $M_1 \approx_w M_2$. The problem here is that τ-closures express the existence of *finite* τ-sequences in the LTSs, and hence they correspond to *minimal* fixed point computations, which cannot be done accurately by solving the equations of a *maximal* fixed point BES. On the other hand, if we eliminate the two τ-loops in M_1 and M_2, the BES becomes:

$$X_{p_0q_0} \stackrel{\nu}{=} Y_{p_1q_0a} \wedge Z_{p_0q_1b} \qquad Y_{p_1q_0a} \stackrel{\nu}{=} \mathsf{false} \qquad Z_{p_0q_1b} \stackrel{\nu}{=} \mathsf{false}$$

and yields the correct solution false for the variable $X_{p_0q_0}$. This is a consequence of the fact that minimal and maximal fixed points have the same interpretation on acyclic models, as shown in [33] for modal μ-calculus formulas. Thus, if the LTSs being compared are τ-convergent, the encoding of τ-closures using maximal fixed point equations is correct. A proof of this fact for branching bisimulation is available in Annex A.2. The elimination of τ-cycles by collapsing their states (also called *τ-compression*), which preserves both branching and weak bisimulation, can be performed in linear-time during an on-the-fly LTS exploration [34], using an adaptation of Tarjan's algorithm [41] for detecting strongly connected components (SCCs). To make the BES encodings in Table 1 correct, it is therefore sufficient to reduce both LTSs on-the-fly by applying τ-compression simultaneously with the local BES resolution.

The new BESs obtained in this way for branching and weak bisimulation have a size comparable (see Figure 3 (a), (b)) with the BESs resulting from the previous encodings in which τ-closures were computed separately by specialized algorithms [34]; however, we observed experimentally that their resolution (using the same algorithms) is about one order of magnitude faster. This is due to the fact that intermediate results of τ-closure computations are stored as values of the boolean variables used to encode τ-closures (e.g., variables $U_{pp'qa}$ and $V_{pqq'a}$ of the BES for branching bisimulation), which are retrieved immediately if needed again during resolution; the only risk with this scheme was a too important quantity of such variables, which was not observed in practice. We also encoded as BESs, using similar schemes, the $\tau^*.a$ [18] and safety [8] equivalences, which are weaker than branching bisimulation and slightly less used in practice.

4 Local BES Resolution based on Suspend/Resume DFS

Several local BES resolution algorithms with a linear-time complexity are available [1, 47, 17, 35], typically based on DFS or breadth-first search (BFS) strategies for exploring the dependencies between boolean variables, i.e., the edges of the boolean graph. Here we aim to satisfy the following optimality criterion for local BES resolution algorithms, based on the notion of diagnostic [32]: the resolution must stop as soon as the boolean subgraph already explored contains exactly one diagnostic (example or counterexample) for the variable of interest. To our knowledge, all existing algorithms satisfy only a half of this criterion, i.e., they detect optimally either the presence of examples, or of counterexamples, but not of both of them. The LMC algorithm proposed in [17], based on a DFS traversal of the boolean graph with computation of SCCs, detects counterexamples optimally and speeds up the search of examples (in maximal fixed point BESs) without attempting their optimal detection.

In the BESs produced from equivalence checking problems, false constants (sink $\vee$-vertices) denote couples of non equivalent states; if their backward propagation along edges in the boolean graph is done as soon as these vertices are encountered, it leads to an optimal detection of counterexamples, as in the A0 algorithm proposed in [35]. However, when the LTSs being compared are equivalent, the variable of interest is true and the associated diagnostic is an *example*, which must be detected as soon as possible during resolution. Using the characterization of examples induced by the μ-calculus formula ϕ_{ex} given in Section 2, we can draw an alternative graph-based characterization: an example for vertex X is a subgraph containing X in which every $\vee$-vertex (resp. $\wedge$-vertex) must have exactly one successor (resp. all its successors) contained in the example. Each example can be split into maximal SCCs, which are connected acyclically; in the sequel, we denote them as *pseudo*-SCC*s*, since they are special cases of SCCs in the boolean graph (for instance, a trivial SCC containing a single sink $\vee$-vertex denotes a false constant, which is neither an example, nor a pseudo-SCC). These pseudo-SCCs are the smallest "building blocks" of the examples, and therefore their presence in the boolean subgraph already explored must be determined as soon as possible in order to achieve an optimal detection of examples.

To detect pseudo-SCCs, one can adapt Tarjan's algorithm [41], which relies on a DFS traversal. The problem is that a classical DFS of the boolean graph does not allow the detection of pseudo-SCCs as soon as they occur, because Tarjan's algorithm identifies SCCs only when their root vertex is popped from the DFS stack, meaning that the subgraph reachable from the root has been entirely explored; this subgraph may very well contain other pseudo-SCCs, which could make several examples to be contained in the boolean subgraph explored at the end of the resolution, i.e., when the variable of interest will be popped in turn from the DFS stack (if it evaluates to true, this variable is the root of the last pseudo-SCC identified). In order to detect the first pseudo-SCC encountered, it is necessary to *suspend* the DFS for each $\vee$-vertex when one of its successors was already visited; if this successor turns out to be false at some later stage (and thus not part of a pseudo-SCC), and the $\vee$-vertex is encountered again, it is necessary to *resume* the DFS by considering some other successor of the $\vee$-vertex that may belong to a pseudo-SCC. The exploration of $\wedge$-vertices is done as in the classical

DFS, since for each $\wedge$-vertex, all its successors must be visited before attempting to detect a pseudo-SCC containing it.

The local resolution algorithm sr-DFS that we propose, based on this suspend/resume DFS, is illustrated below. Taking as input a boolean graph $G = (V, E, L)$ represented implicitly (i.e., by its successor function) and a variable of interest x, the algorithm performs iteratively a forward search of G starting at vertex x. Visited vertices are stored in a set $A \subseteq V$. The DFS stack is stored in a variable *dfs* and the stack used for detecting pseudo-SCCs is stored in a variable *scc*. The variable *count* keeps a global counter allowing the assignement of unique DFS numbers to visited vertices. To each vertex v are associated the following fields:

- a counter $c(v)$ which counts the number of remaining successors of v to visit in order to stabilize v;
- a number $p(v)$ recording the index of the next successor of v to be visited (the successors in $E(v)$ are supposed to be indexed from 0 to $|E(v)| - 1$);
- a number $n(v)$ representing the DFS number of v;
- a number $l(v)$ representing the "lowlink" number [41] of v, used to detect if a vertex is the root of a pseudo-SCC;
- a set $d(v)$ containing the vertices that currently depend upon v;
- a boolean $on_scc(v)$ which is true if v is on the *scc* stack and false otherwise;
- a boolean $stop(v)$ which is true if the DFS must be suspended for v (i.e., v is a $\vee$-vertex and one of its successors has been visited);
- a boolean $stable(v)$, which is true if v is stable;
- a boolean $value(v)$, which represents the value of v (this field is of interest only if v is stable).

At each iteration of the main while-loop (lines 20–122), the vertex y at the top of the *dfs* stack is explored. If y is stable, or the DFS must be suspended for y (that is, y is a $\vee$-vertex and one of its successors has already been visited), the value of y is back-propagated along its predecessors d (lines 30–62). For each vertex w which is not stabilized by the back-propagation, the algorithm must keep on visiting its successors, if w will be visited again during the DFS (the variable $stop(w)$ becomes false). Due to the suspend/resume principle, this propagation phase may influence the contents of the pseudo-SCC currently stored on the *scc* stack. Indeed, each $\vee$-vertex which is visited during the propagation phase is stored on the *scc* stack. The definition of pseudo-SCCs requires that each $\vee$-vertex must have exactly one successor contained in its pseudo-SCC. But, as the vertex was not stabilized by the value of the successor which was propagated, it does not meet any more the definition of the pseudo-SCC. Since the *scc* does not contain a pseudo-SCC anymore, it must be cleared. A variable called *purge* is used for this purpose and becomes true when the *scc* stack must

Algorithm 1 Local BES resolution based on suspend/resume DFS

```
function sr-DFS (x, (V, E, L)) : Bool is
var A, B : 2^V; d : V → 2^V;
    u, w, y, z : V; dfs, scc : V*;
    c, p, n, l : V → Nat;
    stop, stable : V → Bool;
    value, on_scc : V → Bool;
    count, max, min : Nat;
    purge : Bool;
if L(x) = ∧ then
  c(x) := |E(x)|
else
  c(x) := 1
end if
p(x) := 0; stable(x) := false;
d(x) := ∅; value(x) := false;
A := {x}; count := 0;
dfs := push(x, nil);
scc := push(x, nil);
on_scc(x) := true; stop(x) := false;
while dfs ≠ nil do
  y := top(dfs);
  n(y) := count;
  l(y) := n(y);
  count := count + 1;
  max := 0;
  min := n(y);
  if stable(y) ∨ stop(y) then
    if d(y) ≠ ∅ then
      B := {y};
      while B ≠ ∅ do
        let u ∈ B;
        B := B \ {u};
        for all w ∈ d(u) do
          if ¬stable(w) then
            if ((L(w) = ∨) ∧ value(u)) ∨ ((L(w) =
            ∧) ∧ ¬value(u)) then
              c(w) := 0
            else
              c(w) := c(w) − 1
            end if
            if c(w) = 0 then
              stable(w) := true;
              value(w) := value(u);
              B := B ∪ {w};
              if n(w) < min then
                min := n(w)
              end if
            else
              stop(w) := false;
              if L(w) = ∧ then
                c(w) := 1;
                p(w) := 0
              else
                E(w) := E(w) \ {u};
                if n(w) > max then
                  max := n(w)
                end if
              end if
            end if
          end if
        end for
        d(u) := ∅
      end while;
      if max > min then
        purge := true
      end if
    else
      dfs := pop(dfs);
      if dfs ≠ nil then
        l(top(dfs)) :=
                min (l(top(dfs)), l(y))
      end if
    end if
  else
    if purge then
      while top(scc) ≠ top(dfs) do
        scc := pop(scc)
      end while;
      purge := false
    end if
    if p(y) < |E(y)| then
      if L(y) = ∨ then
        stop(y) := true
      end if
      z := (E(y))_p(y);
      p(y) := p(y) + 1;
      d(z) := d(z) ∪ {y};
      if z ∈ A then
        if on_scc(z) then
          if n(z) < n(y) then
            l(y) := min (n(z), n(y))
          end if
        else
          dfs := push(z, dfs);
          scc := push(z, scc);
          on_scc(z) := true
        end if
      else
        if L(z) = ∧ then
          c(z) := |E(z)|
        else
          c(z) := 1
        end if;
        p(z) := 0;
        A := A ∪ {z};
        dfs := push(z, dfs);
        scc := push(z, scc);
        on_scc(z) := true
      end if
    else
      if (l(y) = n(y)) ∧ (top(scc) ≠ y) then
        repeat
          z := top(scc);
          c(z) := 0;
          scc := pop(scc)
        until top(scc) = y
      end if
      dfs := pop(dfs);
      if dfs ≠ nil then
        l(top(dfs)) :=
                min (l(top(dfs)), l(y))
      end if
    end if
  end if
end while;
return value(x)
```

be cleared (two variables *min* and *max* are used to determine if *scc* must be cleared: *min* represents the least DFS number among all the DFS numbers of vertices stabilized during the propagation phase and *max* represents the greatest DFS number among all $\vee$-vertices towards which a false value has been propagated).

If the vertex y at the top of the *dfs* stack is unstable or that the exploration must continue for this vertex, its next unexplored successor $z = E(y)_{p(y)}$ is visited. Before that, the *scc* stack is cleared if needed (i.e., if a back-propagation of a false value took place at some previous iteration of the main while-loop). If z is a new vertex (lines 96-107), it is pushed on the *dfs* stack. If z is an already explored vertex, two cases may appear. Either z is on the *scc* stack (lines 87-90) and therefore its lowlink number must be updated, or it is not on the stack (lines 91-95), and therefore it must be explored as if it was a new vertex (i.e., z was popped from the *scc* stack after a propagation phase which induced a clearing of this stack). Finally, if y is unstable and all its successors have been visited, the algorithm watches if y is the root of a pseudo-SCC (lines 108–120). If this is the case, the *scc* stack is cleared from its top until y and each vertex of the pseudo-SCC is stabilized. Then, y is popped from the *dfs* stack. Finally, after termination of the main while-loop, the value computed for x is returned.

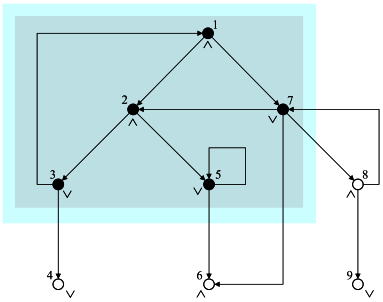

Figure 2: Local resolution of variable X_1 using the sr-DFS algorithm

The execution of algorithm sr-DFS on the boolean graph considered in Section 2 is illustrated on Figure 2. We observe on this example an optimal behaviour of sr-DFS: due to the suspension of the DFS for the $\vee$-vertices X_3, X_5, and X_7 when one of their successors was visited, the subgraph explored by the algorithm (light grey area) coincides with the example found for vertex X_1 (dark grey area), made of the pseudo-SCCs $\{X_1, X_2, X_3, X_7\}$ and $\{X_5\}$. The resolution previously shown on Figure 1 was done using the algorithm A0 [35], which is based on a classical DFS without computation of SCCs, and therefore explores a larger subgraph than sr-DFS in order to find another, larger example for X_1.

Complexity. For boolean graphs $G = (V, E, L)$ without sink $\vee$-vertices (i.e., for maximal fixed point BESs without false constants in the right-hand sides of their equations), the sr-DFS algorithm has a linear-time complexity $O(|V| + |E|)$. The presence of false constants could trigger the reexploration of some vertices (those present on the portions of the *scc* stack that were cleared after back-propagation of false constants), increasing the complexity

of the algorithm towards quadratic-time $O((|V|+|E|)^2)$, which is the theoretical worst-case. This is the price to pay for achieving an optimal detection of examples and counterexamples in the boolean graph. However, the behaviour of the sr-DFS algorithm that we observed in practice for equivalence checking (by measuring the number of variables explored and reexplored) shows that its complexity is close to linear-time.

5 Implementation and Experiments

The CÆSAR_SOLVE [35] library of CADP [22] provides a generic implementation of several local BES resolution algorithms. The library was developed using the OPEN/CÆSAR [21] generic environment for LTS manipulation, which offers many graph exploration primitives (stacks, hash tables, edge lists, etc.). BESs are handled by CÆSAR_SOLVE by means of their corresponding boolean graphs, represented implicitly in a way similar to LTSs in OPEN/CÆSAR. This representation is application-independent, allowing to employ the resolution algorithms as computing engines for several on-the-fly verification tools of CADP: the model checker EVALUATOR 3.x [36, 35], the equivalence checker BISIMULATOR 1.0 [5, 35], and the REDUCTOR 5.0 tool for LTS generation equipped with partial order reductions.

Table 2: Algorithms of CÆSAR_SOLVE and their application to equivalence checking

Alg.	BES type	Strategy	Time	Memory	Condition
A0	general	DFS	$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$	nondeterm. LTSs
A1		BFS			
A2	acyclic	DFS		$O(\|V\|)$	one LTS acyclic
A3	disjunctive				—
A4	conjunctive				one LTS determ., τ-free
A5	general			$O(\|V\|+\|E\|)$	nondeterm. LTSs
A6	disjunctive	BFS		$O(\|V\|)$	—
A7	conjunctive				one LTS determ., τ-free

Table 2 summarizes the local resolution algorithms currently available in the CÆSAR_SOLVE library and their application for equivalence checking within BISIMULATOR. All algorithms have a linear complexity w.r.t. the size of boolean graphs (number of vertices and edges). Algorithms A0, A1, and A5 can solve general BESs (without constraints on the structure of equations), A1 being BFS-based and thus able to produce small-depth diagnostics. When one LTS is deterministic (for strong equivalence) and τ-free (for weak equivalences), the resulting BES is conjunctive and can be solved using the memory-efficient algorithm A4 [35], which stores only the vertices of the boolean graph (and not its edges), i.e., only the states of the LTSs (and not their transitions). Also, when one LTS is acyclic, the resulting BES is also acyclic (i.e., it has an acyclic boolean graph) and can be solved using the memory-efficient algorithm A2. The BFS-based algorithm A7, recently added to the library, can be applied to conjunctive BESs and combines the advantages of algorithms A1 (small-depth diagnostics) and A4 (low memory consumption) when one LTS is deterministic and τ-free.

The version BISIMULATOR 2.0 includes the new BES encodings of weak equivalences defined in Section 3 and the new resolution algorithm sr-DFS given in Section 4, which was recently added to CÆSAR_SOLVE with the number A8. In the sequel, we present various performance measures showing the effect of these two enhancements. The LTSs considered were generated from the demo examples of CADP (specifications of communication protocols and asynchronous circuits) or taken from the VLTS benchmark suite [46].

New encodings of weak equivalences. The new BES encodings of weak equivalences that we proposed in Section 3 compute τ-closures by means of BES equations instead of relying on external, dedicated graph algorithms as the previous encodings used in BISIMULATOR 1.0. Figure 3(a)–(b) compares the performance of the two encodings for branching bisimulation as regards the size of the underlying BESs and their resolution time using algorithm A0. As expected, the BESs produced by the new encoding are larger (more variables but less operators) because intermediate results of τ-closure computations are stored as boolean variables, but they are solved faster due to the simpler structure of boolean equations. Of course, what matters from the end-user point of view is the overall performance of using sr-DFS in conjunction with the new BES encoding; this is illustrated below.

Resolution using the sr-DFS algorithm. The series of experiments shown in Figure 3(c)–(f) compare the behaviour of BISIMULATOR 1.0 (algorithm A0 and previous BES encoding) w.r.t. version 2.0 (algorithm sr-DFS and new BES encoding) for branching bisimulation. To improve readability, we separated the LTSs in two groups according to their number of transitions. When applying version 2.0, we observed reductions of both the number of vertices visited and edges explored, which determine the memory consumption and the execution time, respectively. These reductions become more important as the LTS size increases, as indicated by curves (e) and (f); in particular, the number of transitions traversed can decrease by a factor 8. It is worth noticing that some of the LTSs compared were not equivalent (e.g., certain erroneous variants of a leader election protocol examined in [23]), showing that version 2.0 of the tool exhibits a good behaviour also for counterexample detection. These experimental results indicate that the increase in BES size induced by the new encoding of weak equivalences is compensated by the reduction achieved using sr-DFS, leading to an overall improvement of the on-the-fly verification procedure. As regards strong bisimulation, sr-DFS reduces the number of variables explored by up to 25%, as shown in Figure 3(g).

Complexity w.r.t. theoretical worst-case. As pointed out in [18], the on-the-fly comparison of two nondeterministic LTSs $M_1 = \langle Q_1, A_1, T_1, q_{01} \rangle$ and $M_2 = \langle Q_2, A_2, T_2, q_{02} \rangle$ has a worst-case complexity $O((|Q_1| \cdot |T_2|) + (|Q_2| \cdot |T_1|))$. Considering the BES formulation of the problem, this complexity can be estimated in terms of BES size: the BESs given in Table 1 have a number of boolean variables proportional to the size of the synchronous product between the two LTSs. However in practice, the BESs produced from equivalence checking have a much smaller size (several orders of magnitude) than the theoretical worst-case, as it is illustrated in Figure 3(h) for strong bisimulation. This also holds for weak equivalences, in particular for branching bisimulation.

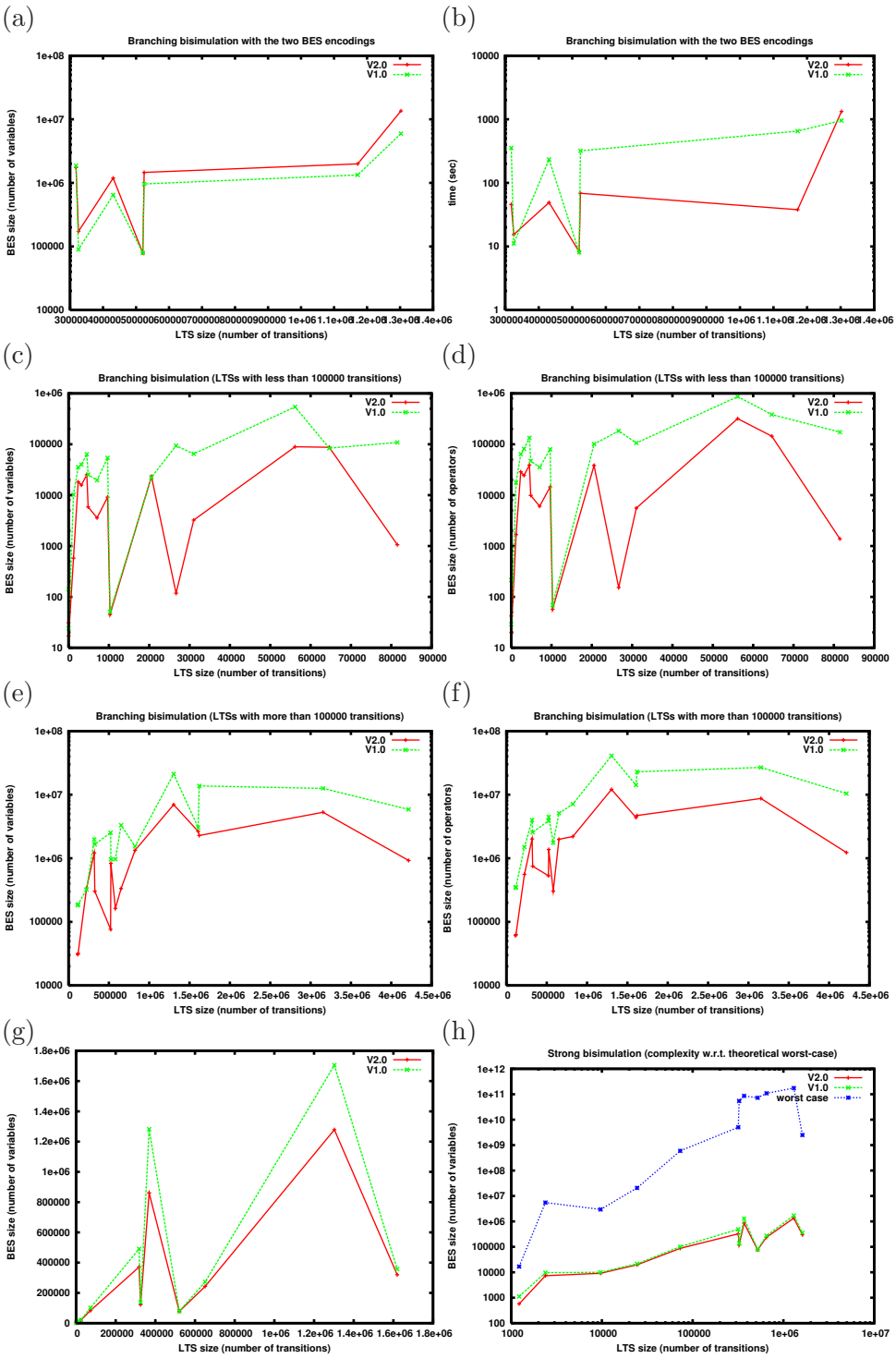

Figure 3: Performance of equivalence checking using BISIMULATOR 1.0 and 2.0

6 Conclusion and Future Work

Building efficient software tools for on-the-fly equivalence checking between LTSs is a difficult and time-consuming task. The usage of intermediate formalisms, such as BESs, allows one to separate the concerns of phrasing the verification problem and of solving it, leading to highly modular verification tools [36, 5]. The two optimizations we proposed, namely the new encodings of weak equivalences by applying τ-compression on the input LTSs and computing τ-closures using boolean equations (Section 3) and the new sr-DFS local BES resolution algorithm (Section 4) significantly increased the performance of on-the-fly equivalence checking w.r.t. existing approaches.

These optimizations underlie the new version 2.0 of the BISIMULATOR equivalence checker [35] of the CADP toolbox [22]. The sr-DFS algorithm was integrated to the generic CÆSAR_SOLVE library [35] for on-the-fly BES resolution, which is part of the generic OPEN/CÆSAR environment [21] for LTS manipulation. Local BES resolution proved to be a suitable alternative way for computing τ-closures on LTSs produced from protocols and distributed systems, competing favourably with general transitive closure algorithms. The sr-DFS algorithm is able to detect optimally the presence of both examples and counterexamples in the boolean graph, and appears to be quite effective for comparing LTSs modulo weak equivalences.

We plan to continue our work along two directions. First, the range of equivalences and preorders already available in BISIMULATOR 2.0 (strong, branching, weak, $\tau^*.a$, safety, trace, and weak trace) could be extended by devising BES encodings for other weak equivalences, such as CFFD [43] and testing equivalence [11], following the scheme in Section 3. Next, we will pursue experimenting the sr-DFS algorithm and study its applicability for solving BESs coming from other verification problems, such as the model checking of alternation-free modal μ-calculus and the on-the-fly LTS reduction modulo partial order relations (e.g., τ-confluence, τ-inertness, etc.) as formulated in [38].

Acknowledgements

The second author was supported by the FP6-NEST-PATH-COM European project no. 043235 "EC-MOAN".

References

[1] Henrik R. Andersen. Model Checking and Boolean Graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[2] Henrik R. Andersen and Bart Vergauwen. Efficient Checking of Behavioural Relations and Modal Assertions using Fixed-Point Inversion. In Pierre Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification CAV'95 (Liege,*

Belgium), volume 939 of *Lecture Notes in Computer Science*, pages 142–154. Springer Verlag, July 1995.

[3] André Arnold and Paul Crubillé. A Linear Algorithm to Solve Fixed-Point Equations on Transition Systems. *Information Processing Letters*, 29(1):57–66, September 1988.

[4] G. Ausiello and G. F. Italiano. On-Line Algorithms for Polynomially Solvable Satisfiability Problems. *Journal of Logic Programming*, 10(1/2/3&4):69–90, 1991.

[5] Damien Bergamini, Nicolas Descoubes, Christophe Joubert, and Radu Mateescu. BISIMULATOR: A Modular Tool for On-the-Fly Equivalence Checking. In Nicolas Halbwachs and Lenore Zuck, editors, *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2005 (Edinburgh, Scotland, UK)*, volume 3440 of *Lecture Notes in Computer Science*, pages 581–585. Springer Verlag, April 2005.

[6] J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Computation*, 60:109–137, 1984.

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

[8] Ahmed Bouajjani, Jean-Claude Fernandez, Susanne Graf, Carlos Rodríguez, and Joseph Sifakis. Safety for Branching Time Semantics. In *Proceedings of 18th ICALP*. Springer Verlag, July 1991.

[9] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, July 1984.

[10] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics-Based Verification Tool for Finite State Systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[11] Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, 5(1):1–20, 1993.

[12] Rance Cleaveland and Oleg Sokolsky. *Equivalence and Preorder Checking for Finite-State Systems*. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, chapter 6, pages 391–424. North-Holland, 2001.

[13] Rance Cleaveland and Bernhard Steffen. Computing Behavioural Relations, Logically. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Proceedings of the 18th Internationan Colloquium on Automata, Languages, and Programming ICALP'91 (Madrid, Spain)*, volume 510 of *Lecture Notes in Computer Science*, pages 127–138. Springer Verlag, July 1991.

[14] Rance Cleaveland and Bernhard Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. *Formal Methods in System Design*, 2(2):121–147, April 1993.

[15] Agostino Dovier, Carla Piazza, and Alberto Policriti. An Efficient Algorithm for Computing Bisimulation Equivalence. *Theoretical Computer Science*, 311(1–3):221–256, 2004.

[16] W.F. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[17] Xiaoqun Du, Scott A. Smolka, and Rance Cleaveland. Local Model Checking and Protocol Analysis. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(3):219–241, 1999.

[18] Jean-Claude Fernandez and Laurent Mounier. Verifying Bisimulations "On the Fly". In Juan Quemada, José Manas, and Enrique Vázquez, editors, *Proceedings of the 3rd International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*. North-Holland, November 1990.

[19] Jean-Claude Fernandez and Laurent Mounier. A Tool Set for Deciding Behavioral Equivalences. In *Proceedings of CONCUR'91 (Amsterdam, The Netherlands)*, August 1991.

[20] Kathi Fisler and Moshe Y. Vardi. Bisimulation Minimization and Symbolic Model Checking. 21(1):39–78, 2002.

[21] Hubert Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84, Berlin, March 1998. Springer Verlag. Full version available as INRIA Research Report RR-3352.

[22] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification CAV'2007 (Berlin, Germany)*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer Verlag, July 2007.

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

[24] Jan Friso Groote and Misa Keinänen. Solving Disjunctive/Conjunctive Boolean Equation Systems with Alternating Fixed Points. In Kurt Jensen and Andreas Podelski, editors, *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2004 (Barcelona, Spain)*, volume 2988 of *Lecture Notes in Computer Science*, pages 436–450. Springer Verlag, March 2004.

[25] A. Ingolfsdottir and B. Steffen. Characteristic Formulae for Processes with Divergence. *Information and Computation*, 110(1):149–163, June 1994.

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

[27] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[28] K. G. Larsen. Efficient Local Correctness Checking. In G. v. Bochmann and D. K. Probst, editors, *Proceedings of 4th International Workshop in Computer Aided Verification CAV'92 (Montréal, Canada)*, volume 663 of *Lecture Notes in Computer Science*, pages 30–43, Berlin, June-July 1992. Springer Verlag.

[29] X. Liu and S. A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming ICALP'98 (Aalborg, Denmark)*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer Verlag, July 1998.

[30] Angelika Mader. *Verification of Modal Properties Using Boolean Equation Systems*. VERSAL 8, Bertz Verlag, Berlin, 1997.

[31] Alain J. Martin. Compiling Communicating Processes into Delay-Insensitive VLSI Circuits. *Distributed Computing*, 1(4):226–234, 1986.

[32] Radu Mateescu. Efficient Diagnostic Generation for Boolean Equation Systems. In *Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000 (Berlin, Germany)*, LNCS. Springer Verlag, March 2000.

[33] Radu Mateescu. Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2002 (Grenoble, France)*, volume 2280 of *Lecture Notes in Computer Science*, pages 281–295. Springer Verlag, April 2002. Full version available as INRIA Research Report RR-4430.

[34] Radu Mateescu. On-the-fly State Space Reductions for Weak Equivalences. In Tiziana Margaria and Mieke Massink, editors, *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems FMICS'05 (Lisbon, Portugal)*, pages 80–89. ERCIM, ACM Computer Society Press, September 2005.

[35] Radu Mateescu. CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 8(1):37–56, February 2006. Full version available as INRIA Research Report RR-5948, July 2006.

[36] Radu Mateescu and Mihaela Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Science of Computer Programming*, 46(3):255–281, March 2003.

[37] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[38] Gordon Pace, Frédéric Lang, and Radu Mateescu. Calculating τ-Confluence Compositionally. In Jr Warren A. Hunt and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification CAV'2003 (Boulder, Colorado, USA)*, volume 2725 of *Lecture Notes in Computer Science*, pages 446–459. Springer Verlag, July 2003. Full version available as INRIA Research Report RR-4918.

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

[40] S. K. Shukla, H. B. Hunt III, and D. J. Rosenkrantz. Hornsat, Model Checking, Verification and Games. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification CAV'96 (New Brunswick, New Jersey, USA)*, volume 1102 of *Lecture Notes in Computer Science*, pages 99–110. Springer Verlag, August 2006.

[41] Robert E. Tarjan. Depth First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[42] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[43] Antti Valmari and Martti Tienari. Compositional Failure-Based Semantics Models for Basic LOTOS. *Formal Aspects of Computing*, 7(4):440–468, 1995.

[44] R. J. van Glabbeek and W. P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics (extended abstract). CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989. Also in proc. IFIP 11th World Computer Congress, San Francisco, 1989.

[45] Rob van Glabbeek. *The Linear Time — Branching Time Spectrum I. The Semantics of Concrete, Sequential Processes*. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–100. Elsevier, 2001.

[46] Vasy. The VLTS Benchmark Suite. `http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html`, June 2003.

[47] B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In S. Abiteboul and E. Shamir, editors, *Proceedings of the 21st ICALP (Vienna)*, volume 820 of *Lecture Notes in Computer Science*, pages 304–315, Berlin, July 1994. Springer Verlag.

A Proofs of the BES encodings

We prove in the sequel the correctness of our BES encodings for strong and branching bisimulation. The correctness proof for weak bisimulation is very similar to the one for branching bisimulation and is therefore left as exercise for the interested reader.

A few additional notions are needed in order to proceed. Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$ and $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two LTSs. Consider the two lattices $\langle \mathsf{Bool}^{|Q_1|\cdot|Q_2|}, \sqsubseteq, \mathsf{false}^{|Q_1|\cdot|Q_2|}, \mathsf{true}^{|Q_1|\cdot|Q_2|}, \sqcup, \sqcap\rangle$ and $\langle 2^{Q_1\times Q_2}, \subseteq, \emptyset, Q_1 \times Q_2, \cup, \cap\rangle$, where the relation $\sqsubseteq$ and the operations $\sqcup$, $\sqcap$ are defined as the pointwise extensions of the boolean connectors $\Rightarrow$, $\vee$, and $\wedge$, respectively. These lattices are isomorphic, being related by the function $\Gamma : \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \rightarrow 2^{Q_1\times Q_2}$ defined below:

$$\Gamma(\langle b_{pq}\rangle_{p\in Q_1, q\in Q_2}) = \{\langle p, q\rangle \mid b_{pq} = \mathsf{true}\}.$$

It is straightforward to show that Γ is an isomorphism, i.e., it is a bijection preserving the compatibility of operations ($\boldsymbol{b} \sqsubseteq \boldsymbol{b'} \Leftrightarrow \Gamma(\boldsymbol{b}) \subseteq \Gamma(\boldsymbol{b'})$, $\Gamma(\mathsf{false}^{|Q_1|\cdot|Q_2|}) = \emptyset$, $\Gamma(\mathsf{true}^{|Q_1|\cdot|Q_2|}) = Q_1 \times Q_2$, $\Gamma(\boldsymbol{b} \sqcup \boldsymbol{b'}) = \Gamma(\boldsymbol{b}) \cup \Gamma(\boldsymbol{b'})$, and $\Gamma(\boldsymbol{b} \sqcap \boldsymbol{b'}) = \Gamma(\boldsymbol{b}) \cap \Gamma(\boldsymbol{b'})$).

A.1 BES encoding of strong bisimulation

We recall below the definitions of strong bisimulation [39] and the corresponding BES encoding given in Section 3.

Definition 1 (Strong bisimulation) *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two* LTSs*. A relation $\approx \subseteq Q_1 \times Q_2$ is a* bisimulation *between M_1 and M_2 iff for every $p \in Q_1$ and $q \in Q_2$, $p \approx q$ if the two conditions below are satisfied:*

1. $\forall p \xrightarrow{a} p' \in T_1.\exists q \xrightarrow{a} q' \in T_2.p' \approx q'$;
2. $\forall q \xrightarrow{a} q' \in T_2.\exists p \xrightarrow{a} p' \in T_1.p' \approx q'$.

The strong bisimulation $\approx_s \subseteq Q_1 \times Q_2$ *is the union of all bisimulations between M_1 and M_2.*

Definition 2 (Strong bisimulation BES) *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two* LTSs*. The strong bisimulation $\approx_s \subseteq Q_1 \times Q_2$ is encoded by the maximal fixed point* BES *below:*

$$B_s = \left\{ X_{pq} \stackrel{\nu}{=} \bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} X_{p'q'} \wedge \bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} X_{p'q'} \right\}_{p\in Q_1, q\in Q_2}$$

The interpretation $[\![B_s]\!]$ is defined as the maximal fixed point $\nu\Phi_s$, where $\Phi_s : \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \rightarrow \mathsf{Bool}^{|Q_1|\cdot|Q_2|}$ is the (monotonic) functional associated to B_s:

$$\Phi_s(\langle b_{pq}\rangle_{p\in Q_1, q\in Q_2}) = \langle [\![\bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} X_{p'q'} \wedge \bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} X_{p'q'}]\!][b_{pq}/X_{pq}]\rangle_{p\in Q_1, q\in Q_2}.$$

According to Tarski's theorem [42], the maximal fixed point $\nu\Phi_s$ can be computed as follows:

$$\nu\Phi_s = \bigsqcup \{\boldsymbol{b} \in \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \mid \boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})\}.$$

The following lemma provides a link between bisimulations and the functional associated to the BES above.

Lemma 1 *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two* LTS*s, and let $\boldsymbol{b} \in \mathsf{Bool}^{|Q_1|\cdot|Q_2|}$. Then:*

$$\boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b}) \quad \textit{iff} \quad \Gamma(\boldsymbol{b}) \textit{ is a bisimulation.}$$

Proof. **If.** Let $\boldsymbol{b} = \langle b_{pq}\rangle_{p\in Q_1, q\in Q_2}$ such that $\Gamma(\boldsymbol{b})$ is a bisimulation. We must show that $\boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})$.

Let $p \in Q_1, q \in Q_2$ such that $b_{pq} = \mathsf{true}$. From the definition of Γ, this implies $\langle p, q\rangle \in \Gamma(\boldsymbol{b})$. Since $\Gamma(\boldsymbol{b})$ is a bisimulation, from Definition 1 this implies the two conditions below:

$$\forall p \xrightarrow{a} p' \in T_1.\exists q \xrightarrow{a} q' \in T_2.\langle p', q'\rangle \in \Gamma(\boldsymbol{b}) \quad \text{and} \quad \forall q \xrightarrow{a} q' \in T_2.\exists p \xrightarrow{a} p' \in T_1.\langle p', q'\rangle \in \Gamma(\boldsymbol{b})$$

Let $p \xrightarrow{a} p' \in T_1$. From the first condition above, there exists $q \xrightarrow{a} q' \in T_2$ such that $\langle p', q'\rangle \in \Gamma(\boldsymbol{b})$. From the definition of Γ, this implies $b_{p'q'} = \mathsf{true}$, which means that the boolean formula $\bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} b_{p'q'}$ is true. A symmetric reasoning applied to the second condition above shows that the formula $\bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} b_{p'q'}$ is also true. From the interpretation of boolean formulas given in Section 2 and Definition 2, this implies:

$$[\![\bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} X_{p'q'} \wedge \bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} X_{p'q'}]\!][b_{pq}/X_{pq}]_{p\in Q_1, q\in Q_2} = \mathsf{true}$$

meaning that $\Phi_s(\boldsymbol{b}) = \mathsf{true}$. Therefore, $\boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})$.

Only if. Let $\boldsymbol{b} = \langle b_{pq}\rangle_{p\in Q_1, q\in Q_2}$ such that $\boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})$. We must show that $\Gamma(\boldsymbol{b})$ is a bisimulation, i.e., it satisfies the two conditions stated in Definition 1.

Let $\langle p, q\rangle \in \Gamma(\boldsymbol{b})$. From the definition of Γ, this implies $b_{pq} = \mathsf{true}$. Since $\boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})$, from Definition 2 and the interpretation of boolean formulas defined in Section 2, this implies:

$$[\![\bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} X_{p'q'}]\!][b_{pq}/X_{pq}]_{p\in Q_1, q\in Q_2} \wedge [\![\bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} X_{p'q'}]\!][b_{pq}/X_{pq}]_{p\in Q_1, q\in Q_2} = \mathsf{true}.$$

In order this equality to hold, both conjuncts must be true, which yields, after further applying the interpretation of boolean formulas, the two conditions below:

$$\bigwedge_{p\xrightarrow{a}p'} \bigvee_{q\xrightarrow{a}q'} b_{p'q'} = \mathsf{true} \qquad \text{and} \qquad \bigwedge_{q\xrightarrow{a}q'} \bigvee_{p\xrightarrow{a}p'} b_{p'q'} = \mathsf{true}.$$

Let $p \xrightarrow{a} p' \in T_1$. From the first condition above, each conjunct associated to such a transition must be true, i.e., $\bigvee_{q\xrightarrow{a}q'} b_{p'q'} = \mathsf{true}$. This means that some disjunct corresponding to a transition $q \xrightarrow{a} q' \in T_2$ must be true, i.e., there exists such a transition such that $b_{p'q'} = \mathsf{true}$. From the definition of Γ, this implies $\langle p', q'\rangle \in \Gamma(\boldsymbol{b})$, which means that $\Gamma(\boldsymbol{b})$ satisfies condition 1 of Definition 1. A symmetric reasoning applied to the second condition above shows that $\Gamma(\boldsymbol{b})$ also satisfies condition 2 of Definition 1, and therefore $\Gamma(\boldsymbol{b})$ is a bisimulation. □

We are now ready to show the correctness of the BES encoding for strong bisimulation.

Proposition 1 (Correctness of strong bisimulation BES) *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two* LTS*s, and let B_s be the* BES *encoding the strong bisimulation between M_1 and M_2. Then:*

$$\Gamma([\![B_s]\!]) = \approx_s .$$

Proof.

$$\begin{aligned}
\Gamma([\![B_s]\!]) &= \Gamma(\nu\Phi_s) && \text{by Definition 2}\\
&= \Gamma(\textstyle\bigsqcup \{\boldsymbol{b} \mid \boldsymbol{b} \in \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \wedge \boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})\}) && \text{by Tarski's theorem}\\
&= \textstyle\bigcup \{\Gamma(\boldsymbol{b}) \mid \boldsymbol{b} \in \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \wedge \boldsymbol{b} \sqsubseteq \Phi_s(\boldsymbol{b})\} && \text{by } \Gamma \text{ isomorphism}\\
&= \textstyle\bigcup \{\Gamma(\boldsymbol{b}) \mid \boldsymbol{b} \in \mathsf{Bool}^{|Q_1|\cdot|Q_2|} \wedge \Gamma(\boldsymbol{b}) \text{ is a bisimulation}\} && \text{by Lemma 1}\\
&= \textstyle\bigcup \{\approx \subseteq Q_1 \times Q_2 \mid \approx \text{ is a bisimulation}\} && \text{by } \Gamma \text{ bijection}\\
&= \approx_s && \text{by Definition 1.}
\end{aligned}$$

□

A.2 BES encoding of branching bisimulation

We recall below the definitions of branching bisimulation [44] and the corresponding BES encoding given in Section 3.

Definition 3 (Branching bisimulation) *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two* LTS*s. A relation $\approx \subseteq Q_1 \times Q_2$ is a* branching-like bisimulation *between M_1 and M_2 iff for every $p \in Q_1$ and $q \in Q_2$, $p \approx q$ if the two conditions below are satisfied:*

1. $\forall p \xrightarrow{a} p' \in T_1.((a = \tau \wedge p' \approx q) \vee \exists q \xrightarrow{\tau^*} q' \xrightarrow{a} q'' \in T_2^*.(p \approx q' \wedge p' \approx q''))$;
2. $\forall q \xrightarrow{a} q' \in T_2.((a = \tau \wedge p \approx q') \vee \exists p \xrightarrow{\tau^*} p' \xrightarrow{a} p'' \in T_1^*.(p' \approx q \wedge p'' \approx q'))$.

The branching bisimulation *$\approx_b \subseteq Q_1 \times Q_2$ is the union of all branching-like bisimulations between M_1 and M_2.*

Definition 4 (Branching bisimulation BES) *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two τ-convergent* LTS*s. The branching bisimulation $\approx_b \subseteq Q_1 \times Q_2$ is encoded by the maximal fixed point* BES *below:*

$$B_b = \left\{ \begin{array}{r}
X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{pp'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pqq'a}\\
Y_{pp'qa} \stackrel{\nu}{=} (a = \tau \wedge X_{p'q}) \vee U_{pp'qa}\\
U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} W_{pp'qq'} \vee \bigvee_{q \xrightarrow{\tau} q'} U_{pp'q'a}\\
Z_{pqq'a} \stackrel{\nu}{=} (a = \tau \wedge X_{pq'}) \vee V_{pqq'a}\\
V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} W_{pp'qq'} \vee \bigvee_{p \xrightarrow{\tau} p'} V_{p'qq'a}\\
W_{pp'qq'} \stackrel{\nu}{=} X_{pq} \wedge X_{p'q'}
\end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2.}$$

We first prove several lemmas concerning BES manipulation, and then we prove the correctness of the BES encoding for branching bisimulation.

Definition 5 *Let $\mathcal{X}$ be a set of boolean variables including $X_1, ..., X_n, X_{n+1}$. Let $B = \{X_i \stackrel{\nu}{=} \varphi_i\}_{1 \leq i \leq n}$ and $B' = \{X_i \stackrel{\nu}{=} \varphi_i\}_{1 \leq i \leq n+1}$ be two* BES*s having their first n equations identical (φ_i are boolean formulas built from disjunctions and conjunctions) and $\delta : \mathcal{X} \to \mathsf{Bool}$ be a context. Let $\Phi_\delta : \mathsf{Bool}^n \to \mathsf{Bool}^n$ and $\Phi'_\delta : \mathsf{Bool}^{n+1} \to \mathsf{Bool}^{n+1}$ be the two functionals associated to B and B' in the context δ:*

$$\Phi_\delta(\langle b_i \rangle_{1 \leq i \leq n}) = \langle [\![\varphi_i]\!](\delta \oslash [b_j/X_j]_{1 \leq j \leq n}) \rangle_{1 \leq i \leq n}$$
$$\Phi'_\delta(\langle b_i \rangle_{1 \leq i \leq n+1}) = \langle [\![\varphi_i]\!](\delta \oslash [b_j/X_j]_{1 \leq j \leq n+1}) \rangle_{1 \leq i \leq n+1}$$

where $\delta \oslash [b_j/X_j]_{1 \leq j \leq n}$ denotes a context identical to δ except for variables $X_1, ..., X_n$, which are assigned values $b_1, ..., b_n$. According to Kleene's theorem [27], the maximal fixed points of the functionals Φ_δ and Φ'_δ can be computed as follows:

$$\nu\Phi_\delta = \sqcap_{k \geq 0} \Phi_\delta^k(\mathsf{true}^n) \qquad \nu\Phi'_\delta = \sqcap_{k \geq 0} {\Phi'_\delta}^k(\mathsf{true}^{n+1}).$$

The notation $\langle e_i, e \rangle_{1 \leq i \leq n}$, where e_i and e are boolean expressions, is a shorthand for $\langle e_1, ..., e_n, e \rangle$. We define the series $U_k \in \mathsf{Bool}^{n+1}$ associated to B, B', and δ as follows:

$$U_0 = \mathsf{true}^{n+1}, \qquad U_{k+1} = \langle (\nu\Phi_{\delta \oslash [(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1} \rangle_{1 \leq i \leq n}.$$

The following lemma relates the maximal fixed points of the two functionals Φ_δ and Φ'_δ.

Lemma 2 *Consider two* BES*s B, B' and a context δ as in Definition 5. Then:*

$$(\nu\Phi_{\delta \oslash [(\nu\Phi'_\delta)_{n+1}/X_{n+1}]})_i \sqsupseteq (\nu\Phi'_\delta)_i$$

for all $1 \leq i \leq n$.

Proof. According to Kleene's theorem [27], the maximal fixed point of Φ has the iterative characterization below:

$$\nu\Phi_{\delta \oslash [(\nu\Phi'_\delta)_{n+1}/X_{n+1}]} = \sqcap_{k \geq 0} \Phi^k_{\delta \oslash [(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\mathsf{true}^n).$$

Therefore, to show the desired inequality, it is sufficient to show the following statement for all $k \geq 0$ and $1 \leq i \leq n$:

$$(\Phi^k_{\delta \oslash [(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\mathsf{true}^n))_i \sqsupseteq (\nu\Phi'_\delta)_i.$$

We proceed by induction on k.

Base step. $(\Phi^0_{\delta \oslash [(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\mathsf{true}^n))_i = (\mathsf{true}^n)_i \sqsupseteq (\nu\Phi'_\delta)_i$.

Inductive step.

$$
\begin{aligned}
(\Phi^{k+1}_{\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\mathsf{true}^n))_i &= (\Phi_{\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\Phi^k_{\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\mathsf{true}^n)))_i \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition} \\
&\sqsupseteq (\Phi_{\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}]}(\langle(\nu\Phi'_\delta)_j\rangle_{1\le j\le n}))_i \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by induction hypothesis} \\
&= [\![\varphi_i]\!]((\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}])\oslash[(\nu\Phi'_\delta)_j/X_j]_{1\le j\le n}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition of } \Phi \\
&= [\![\varphi_i]\!](\delta\oslash[(\nu\Phi'_\delta)_j/X_j]_{1\le j\le n+1}) \\
&= (\Phi'_\delta(\nu\Phi'_\delta))_i \qquad\qquad \text{by definition of } \Phi' \\
&= (\nu\Phi'_\delta)_i \qquad\qquad \text{by definition of } \nu.
\end{aligned}
$$

□

The following lemma states two useful properties of the series U_k.

Lemma 3 *Consider two* BESs *B, B', a context δ, and the associated series U_k as in Definition 5. The following properties hold for all $k \ge 0$:*

1. $U_k \sqsupseteq \Phi'_\delta(U_k) \sqsupseteq U_{k+1}$;
2. ${\Phi'_\delta}^k(\mathsf{true}^{n+1}) \sqsupseteq U_k \sqsupseteq \nu\Phi'_\delta$.

Proof.
Property 1. We proceed by induction on k.

Base step. From the definition of U_k, we derive the first inequality: $U_0 = \mathsf{true}^{n+1} \sqsupseteq \Phi'_\delta(U_0)$.

We derive now the second inequality:

$$
\begin{aligned}
\Phi'_\delta(U_0) &= \Phi'_\delta(\mathsf{true}^{n+1}) && \text{by definition of } U_k \\
&= \langle [\![\varphi_i]\!](\delta\oslash[\mathsf{true}/X_j]_{1\le j\le n+1})\rangle_{1\le i\le n+1} && \text{by definition of } \Phi'_\delta \\
&= \langle (\Phi_{\delta\oslash[\mathsf{true}/X_{n+1}]}(\mathsf{true}^n))_i, (\Phi'_\delta(\mathsf{true}^{n+1}))_{n+1}\rangle_{1\le i\le n} && \text{by definition of } \Phi_\delta,\ \Phi'_\delta \\
&\sqsupseteq \langle (\nu\Phi_{\delta\oslash[\mathsf{true}/X_{n+1}]})_i, (\Phi'_\delta(\mathsf{true}^{n+1}))_{n+1}\rangle_{1\le i\le n} && \text{by Kleene's theorem} \\
&\sqsupseteq \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(\mathsf{true}^{n+1}))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(\mathsf{true}^{n+1}))_{n+1}\rangle_{1\le i\le n} && \text{by monotonicity} \\
&= U_1 && \text{by definition of } U_k.
\end{aligned}
$$

Inductive step. We derive the first inequality:

$$
\begin{aligned}
U_{k+1} &= \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \qquad \text{by definition of } U_k \\
&= \langle (\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]}(\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]}))_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition of } \nu \\
&= \langle (\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]}(\langle(U_{k+1})_j\rangle_{1\le j\le n}))_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition of } U_k \\
&\sqsupseteq \langle (\Phi_{\delta\oslash[(U_{k+1})_{n+1}/X_{n+1}]}(\langle(U_{k+1})_j\rangle_{1\le j\le n}))_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{by induction hypothesis and monotonicity} \\
&= \langle [\![\varphi_i]\!](\delta\oslash[(U_{k+1})_j/X_j]_{1\le j\le n+1}), [\![\varphi_{n+1}]\!](\delta\oslash[(U_{k+1})_j/X_j]_{1\le j\le n+1})\rangle_{1\le i\le n} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{by definition of } \Phi_\delta \text{ and } \Phi'_\delta \\
&= \Phi'_\delta(U_{k+1}) \text{ by definition of } \Phi'_\delta.
\end{aligned}
$$

We derive now the second inequality:

$$
\begin{aligned}
\Phi'_\delta(U_{k+1}) &= \langle [\![\varphi_i]\!](\delta \oslash [(U_{k+1})_j/X_j]_{1\le j\le n+1}), [\![\varphi_{n+1}]\!](\delta \oslash [(U_{k+1})_j/X_j]_{1\le j\le n+1})\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } \Phi'_\delta \\
&= \langle (\Phi_{\delta\oslash[(U_{k+1})_{n+1}/X_{n+1}]}(\langle (U_{k+1})_j\rangle_{1\le j\le n}))_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } \Phi_\delta \text{ and } \Phi'_\delta \\
&\sqsupseteq \langle (\Phi_{\delta\oslash[(\Phi'_\delta(U_{k+1}))_{n+1}/X_{n+1}]}(\langle (U_{k+1})_j\rangle_{1\le j\le n}))_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by the property above and monotonicity} \\
&= \langle (\Phi_{\delta\oslash[(\Phi'_\delta(U_{k+1}))_{n+1}/X_{n+1}]}(\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]}))_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } U_k \\
&\sqsupseteq \langle (\Phi_{\delta\oslash[(\Phi'_\delta(U_{k+1}))_{n+1}/X_{n+1}]}(\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_{k+1}))_{n+1}/X_{n+1}]}))_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by induction hypothesis and monotonicity} \\
&= \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_{k+1}))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_{k+1}))_{n+1}\rangle_{1\le i\le n} \qquad \text{by definition of } \nu \\
&= U_{k+2} \qquad \text{by definition of } U_k.
\end{aligned}
$$

Property 2. We proceed by induction on k.

Base step. ${\Phi'_\delta}^0(\mathsf{true}^{n+1}) = \mathsf{true}^{n+1} = U_0 \sqsupseteq \nu\Phi'_\delta$.

Inductive step. We derive the first inequality:

$$
\begin{aligned}
{\Phi'_\delta}^{k+1}(\mathsf{true}^{n+1}) &= \Phi'_\delta({\Phi'_\delta}^k(\mathsf{true}^{n+1})) \qquad \text{by definition of } \Phi'_\delta \\
&\sqsupseteq \Phi'_\delta(U_k) \qquad \text{by induction hypothesis} \\
&= \langle [\![\varphi_i]\!](\delta \oslash [(U_k)_j/X_j]_{1\le j\le n+1}), [\![\varphi_{n+1}]\!](\delta \oslash [(U_k)_j/X_j]_{1\le j\le n+1})\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } \Phi'_\delta \\
&= \langle (\Phi_{\delta\oslash[(U_k)_{n+1}/X_{n+1}]}(\langle (U_k)_j\rangle_{1\le j\le n}))_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } \Phi_\delta \text{ and } \Phi'_\delta \\
&= \langle (\Phi_{\delta\oslash[(U_k)_{n+1}/X_{n+1}]}(\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_{k-1}))_{n+1}/X_{n+1}]}))_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by definition of } U_k \\
&\sqsupseteq \langle (\Phi_{\delta\oslash[(U_k)_{n+1}/X_{n+1}]}(\nu\Phi_{\delta\oslash[(U_k)_{n+1}/X_{n+1}]}))_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \\
&\qquad \text{by Property 1} \\
&= \langle (\nu\Phi_{\delta\oslash[(U_k)_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \qquad \text{by definition of } \nu \\
&\sqsupseteq \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \qquad \text{by Property 1} \\
&= U_{k+1} \qquad \text{by definition of } U_k.
\end{aligned}
$$

We derive now the second inequality:

$$
\begin{aligned}
U_{k+1} &= \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\le i\le n} \qquad \text{by definition of } U_k \\
&\sqsupseteq \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(\nu\Phi'_\delta))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(\nu\Phi'_\delta))_{n+1}\rangle_{1\le i\le n} \qquad \text{by induction hypothesis} \\
&= \langle (\nu\Phi_{\delta\oslash[(\nu\Phi'_\delta)_{n+1}/X_{n+1}]})_i, (\nu\Phi'_\delta)_{n+1}\rangle_{1\le i\le n} \qquad \text{by definition of } \nu \\
&\sqsupseteq \langle (\nu\Phi'_\delta)_i, (\nu\Phi'_\delta)_{n+1}\rangle_{1\le i\le n} \qquad \text{by Lemma 2} \\
&= \nu\Phi'_\delta \text{ by definition.}
\end{aligned}
$$

□

The following lemma states that appending an equation to two BESs that have the same interpretation yields new BESs still having the same interpretation.

Lemma 4 (Equation appending) *Consider two couples of* BESs $B = \{X_i \overset{\nu}{=} \varphi_i\}_{1\le i\le n}$, $B' = \{X_i \overset{\nu}{=} \varphi_i\}_{1\le i\le n+1}$ *and* $C = \{X_i \overset{\nu}{=} \psi_i\}_{1\le i\le n}$, $C' = \{X_i \overset{\nu}{=} \psi_i\}_{1\le i\le n+1}$ *like in Definition 5, such that* φ_{n+1} *and* ψ_{n+1} *are identical. Let* Φ_δ, Φ'_δ *and* Ψ_δ, Ψ'_δ *be the functionals*

associated to B, B' and C, C' in a context δ. Suppose that B and C have the same interpretation in any context δ, i.e., $\nu\Phi_\delta = \nu\Psi_\delta$. Then:

$$\nu\Phi'_\delta = \nu\Psi'_\delta$$

for any context δ.

Proof. Let δ be a context. We show the desired equality by double inclusion.

Inclusion $\nu\Phi'_\delta \sqsupseteq \nu\Psi'_\delta$. Consider the two series U_k and U'_k associated to the couples of BESs B, B' and C, C' in the context δ as in Definition 5. We show first that for all $k \geq 0$, $U_k = U'_k$. We proceed by induction on k.

Base step. $U_0 = \mathsf{true}^{n+1} = U'_0$.

Inductive step. We first show the statement below:

$$\begin{aligned}
(\Phi'_\delta(U_k))_{n+1} &= [\![\varphi_{n+1}]\!](\delta \oslash [(U_k)_j/X_j]_{1\leq j\leq n+1}) && \text{by definition of } \Phi'_\delta \\
&= [\![\psi_{n+1}]\!](\delta \oslash [(U_k)_j/X_j]_{1\leq j\leq n+1}) && \text{by hypothesis} \\
&= (\Psi'_\delta(U_k))_{n+1} && \text{by definition of } \Psi'_\delta \\
&= (\Psi'_\delta(U'_k))_{n+1} && \text{by induction hypothesis.}
\end{aligned}$$

We show the equality:

$$\begin{aligned}
U_{k+1} &= \langle (\nu\Phi_{\delta\oslash[(\Phi'_\delta(U_k))_{n+1}/X_{n+1}]})_i, (\Phi'_\delta(U_k))_{n+1}\rangle_{1\leq i\leq n} && \text{by definition of } U_k \\
&= \langle (\nu\Phi_{\delta\oslash[(\Psi'_\delta(U'_k))_{n+1}/X_{n+1}]})_i, (\Psi'_\delta(U'_k))_{n+1}\rangle_{1\leq i\leq n} && \text{by the property above} \\
&= \langle (\nu\Psi_{\delta\oslash[(\Psi'_\delta(U'_k))_{n+1}/X_{n+1}]})_i, (\Psi'_\delta(U'_k))_{n+1}\rangle_{1\leq i\leq n} && \text{by hypothesis} \\
&= U'_{k+1} && \text{by definition of } U'_k.
\end{aligned}$$

We now prove the desired inclusion by using the Kleene's characterization of Φ'_δ and proving that for all $k \geq 0$, ${\Phi'_\delta}^k(\mathsf{true}^{n+1}) \sqsupseteq \nu\Psi'_\delta$:

$$\begin{aligned}
{\Phi'_\delta}^k(\mathsf{true}^{n+1}) &\sqsupseteq U_k && \text{by Lemma 3, property 2} \\
&= U'_k && \text{by the property above} \\
&\sqsupseteq \nu\Psi'_\delta && \text{by Lemma 3, property 2.}
\end{aligned}$$

Inclusion $\nu\Phi'_\delta \sqsubseteq \nu\Psi'_\delta$. The proof is symmetric to the converse inclusion.

This concludes the proof of the desired equality. □

We are now ready to show the correctness of the BES encoding for branching bisimulation. We show first a lemma concerning the computation of τ-closures using boolean equations, and then we show the main proposition.

Lemma 5 *Let $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ be two τ-convergent* LTSs*, and consider the* BESs*:*

$$B = \left\{ \begin{array}{l} U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{\tau^*} q'} \bigvee_{q' \xrightarrow{a} q''} W_{pp'q'q''} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{\tau^*} p'} \bigvee_{p' \xrightarrow{a} p''} W_{p'p''qq'} \end{array} \right\}_{p,p'\in Q_1, q,q'\in Q_2, a\in A_1\cup A_2}$$

and

$$B' = \left\{ \begin{array}{l} U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \stackrel{a}{\to} q'} W_{pp'qq'} \vee \bigvee_{q \stackrel{\tau}{\to} q'} U_{pp'q'a} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \stackrel{a}{\to} p'} W_{pp'qq'} \vee \bigvee_{p \stackrel{\tau}{\to} p'} V_{p'qq'a} \end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2.}$$

Then:

$$[\![B]\!]\delta = [\![B']\!]\delta$$

for any context $\delta : \mathcal{X} \to \mathsf{Bool}$*, where the set* $\mathcal{X}$ *contains the variables* $U_{pp'qa}$ *and* $V_{pqq'a}$*.*

Proof. Let $\delta : \mathcal{X} \to \mathsf{Bool}$ be a context and $\Phi_\delta, \Phi'_\delta : \mathsf{Bool}^{|Q_1|^2 \cdot |Q_2| \cdot (|A_1|+|A_2|)} \times \mathsf{Bool}^{|Q_1| \cdot |Q_2|^2 \cdot (|A_1|+|A_2|)} \to \mathsf{Bool}^{|Q_1|^2 \cdot |Q_2| \cdot (|A_1|+|A_2|)} \times \mathsf{Bool}^{|Q_1| \cdot |Q_2|^2 \cdot (|A_1|+|A_2|)}$ be the functionals associated to B and B' in the context δ (for simplicity, we omit the subscript domains when their meaning is clear):

$$\begin{aligned} \Phi_\delta(\langle u_{pp'qa}, v_{pqq'a} \rangle) &= \langle [\![\bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} W_{pp'q'q''}]\!](\delta \oslash [u_{pp'qa}/U_{pp'qa}, v_{pqq'a}/V_{pqq'a}]), \\ &\quad [\![\bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} W_{p'p''qq'}]\!](\delta \oslash [u_{pp'qa}/U_{pp'qa}, v_{pqq'a}/V_{pqq'a}]) \rangle \\ &= \langle \bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} \delta(W_{pp'q'q''}), \bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} \delta(W_{p'p''qq'}) \rangle \end{aligned}$$

$$\begin{aligned} \Phi'_\delta(\langle u_{pp'qa}, v_{pqq'a} \rangle) &= \langle [\![\bigvee_{q \stackrel{a}{\to} q'} W_{pp'qq'} \vee \bigvee_{q \stackrel{\tau}{\to} q'} U_{pp'q'a}]\!](\delta \oslash [u_{pp'qa}/U_{pp'qa}, v_{pqq'a}/V_{pqq'a}]), \\ &\quad [\![\bigvee_{p \stackrel{a}{\to} p'} W_{pp'qq'} \vee \bigvee_{p \stackrel{\tau}{\to} p'} V_{p'qq'a}]\!](\delta \oslash [u_{pp'qa}/U_{pp'qa}, v_{pqq'a}/V_{pqq'a}]) \rangle \\ &= \langle \bigvee_{q \stackrel{a}{\to} q'} \delta(W_{pp'qq'}) \vee \bigvee_{q \stackrel{\tau}{\to} q'} u_{pp'q'a}, \bigvee_{p \stackrel{a}{\to} p'} \delta(W_{pp'qq'}) \vee \bigvee_{p \stackrel{\tau}{\to} p'} v_{p'qq'a} \rangle \end{aligned}$$

To prove that $\nu\Phi_\delta = \nu\Phi'_\delta$, we show first that $\nu\Phi_\delta \sqsubseteq \nu\Phi'_\delta$ and then we show that the strict inclusion $\nu\Phi_\delta \sqsubset \nu\Phi'_\delta$ does not hold. Since the right-hand sides of the equations of B contain neither occurrences of $U_{pp'qa}$, nor of $V_{pqq'a}$, the associated functional Φ_δ is constant and its maximal fixed point is obtained simply by evaluating the functional on some arbitrary arguments:

$$\nu\Phi_\delta = \langle (\bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} \delta(W_{pp'q'q''}))_{pp'qa}, (\bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} \delta(W_{p'p''qq'}))_{pqq'a} \rangle$$

By applying Φ'_δ on this fixed point, we obtain:

$$\begin{aligned} \Phi'_\delta(\nu\Phi_\delta) &= \Phi'_\delta(\langle \bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} \delta(W_{pp'q'q''}), \bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} \delta(W_{p'p''qq'}) \rangle) \\ &\qquad \text{by definition of } \nu\Phi_\delta \\ &= \langle \bigvee_{q \stackrel{a}{\to} q'} \delta(W_{pp'qq'}) \vee \bigvee_{q \stackrel{\tau}{\to} q'} (\bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} \delta(W_{pp'q'q''}))_{pp'q'a}, \\ &\quad \bigvee_{p \stackrel{a}{\to} p'} \delta(W_{pp'qq'}) \vee \bigvee_{p \stackrel{\tau}{\to} p'} (\bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} \delta(W_{p'p''qq'}))_{p'qq'a} \rangle \\ &\qquad \text{by definition of } \Phi'_\delta \\ &= \langle \bigvee_{q \stackrel{a}{\to} q'} \delta(W_{pp'qq'}) \vee \bigvee_{q \stackrel{\tau}{\to} q'} \bigvee_{q' \stackrel{\tau^*}{\to} q''} \bigvee_{q'' \stackrel{a}{\to} q'''} \delta(W_{pp'q''q'''}) \\ &\quad \bigvee_{p \stackrel{a}{\to} p'} \delta(W_{pp'qq'}) \vee \bigvee_{p \stackrel{\tau}{\to} p'} \bigvee_{p' \stackrel{\tau^*}{\to} p''} \bigvee_{p'' \stackrel{a}{\to} p'''} \delta(W_{p''p'''qq'}) \rangle \\ &\qquad \text{by subscript substitution} \\ &= \langle (\bigvee_{q \stackrel{\tau^*}{\to} q'} \bigvee_{q' \stackrel{a}{\to} q''} \delta(W_{pp'q'q''}))_{pp'qa}, (\bigvee_{p \stackrel{\tau^*}{\to} p'} \bigvee_{p' \stackrel{a}{\to} p''} \delta(W_{p'p''qq'}))_{pqq'a} \rangle \\ &\qquad \text{by definition of } \tau\text{-closure} \\ &= \nu\Phi_\delta \qquad \text{by definition of } \nu\Phi_\delta. \end{aligned}$$

From Tarski's theorem [42], this implies $\nu\Phi_\delta \sqsubseteq \nu\Phi'_\delta$. It remains to show that the strict inclusion $\nu\Phi_\delta \sqsubset \nu\Phi'_\delta$ does not hold. Suppose that $\nu\Phi_\delta \sqsubset \nu\Phi'_\delta$, meaning that:

$$\langle (\nu\Phi_\delta)_{pp'qa}, (\nu\Phi_\delta)_{pqq'a}\rangle \sqsubset \langle (\nu\Phi'_\delta)_{pp'qa}, (\nu\Phi'_\delta)_{pqq'a}\rangle$$

Two cases are possible, depending on whether the first or the second component of $\nu\Phi_\delta$ is smaller than the corresponding component of $\nu\Phi'_\delta$. We consider only the first case here, the other one being completely symmetric. Let $p, p' \in Q_1$, $q \in Q_2$, and $a \in A_1 \cup A_2$ such that $(\nu\Phi_\delta)_{pp'qa} = \mathsf{false}$ and $(\nu\Phi'_\delta)_{pp'qa} = \mathsf{true}$.

From the definition of Φ_δ, we infer that $\bigvee_{q \xrightarrow{\tau^*} q'} \bigvee_{q' \xrightarrow{a} q''} \delta(W_{pp'q'q''}) = \mathsf{false}$, meaning that there is no τ-sequence going out of q and leading to a state q' that has an outgoing transition $q' \xrightarrow{a} q''$ such that $\delta(W_{pp'q'q''}) = \mathsf{true}$.

From the definition of Φ'_δ and the fact that $\nu\Phi'_\delta$ is a fixed point, we infer that $\bigvee_{q \xrightarrow{a} q'} \delta(W_{pp'qq'}) \vee \bigvee_{q \xrightarrow{\tau} q'} (\nu\Phi'_\delta)_{pp'q'a} = \mathsf{true}$. But the disjunct $\bigvee_{q \xrightarrow{a} q'} \delta(W_{pp'qq'})$ cannot be true because this would imply the existence of a zero-step τ-sequence going out of q and leading, after an a-transition, to a state q' such that $W_{pp'qq'} = \mathsf{true}$, which is forbidden by the condition above. So the other disjunct must be true, meaning that there exists a transition $q \xrightarrow{\tau} q'$ such that $(\nu\Phi'_\delta)_{pp'q'a} = \mathsf{true}$.

By repeating the above reasoning, we can construct an infinite sequence $(q =)q_0 \xrightarrow{\tau} q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \cdots$ such that $(\nu\Phi'_\delta)_{pp'q_ia} = \mathsf{true}$ for all $i \geq 0$. This contradicts the hypothesis of M_2 being τ-convergent, and therefore concludes the proof. □

Proposition 2 (Correctness of branching bisimulation BES) *Let* $M_1 = \langle Q_1, A_1, T_1, q_{01}\rangle$, $M_2 = \langle Q_2, A_2, T_2, q_{02}\rangle$ *be two* τ*-convergent* LTSs*, and let* B_b *be the* BES *encoding the branching bisimulation between* M_1 *and* M_2*. Then:*

$$\Gamma([\![B_b]\!]) = \approx_b .$$

Proof. We start with the direct BES encoding of branching bisimulation (previously shown in Table 1, middle part, first row) and we progressively refine it until obtaining the full BES encoding given by Definition 4 (previously shown in Table 1, middle part, second row).

The direct encoding of branching bisimulation is given by the BES B_{b1} defined below:

$$B_{b1} = \left\{ \begin{array}{l} X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'}((a = \tau \wedge X_{p'q}) \vee \bigvee_{q \xrightarrow{\tau^*} q' \xrightarrow{a} q''}(X_{pq'} \wedge X_{p'q''})) \wedge \\ \qquad \bigwedge_{q \xrightarrow{a} q'}((a = \tau \wedge X_{pq'}) \vee \bigvee_{p \xrightarrow{\tau^*} p' \xrightarrow{a} p''}(X_{p'q} \wedge X_{p''q'})) \end{array} \right\}_{p \in Q_1, q \in Q_2}$$

This encoding is correct, i.e., $[\![B_{b1}]\!] = \approx_b$. The proof of this fact is similar to that of Proposition 1 and is omitted here for the sake of conciseness. Note that this proof is valid for arbitrary LTSs, and therefore also for τ-convergent LTSs.

We now refine the BES B_{b1} into a BES B_{b2} by replacing certain subformulas with new variables defined by additional equations, such that the right-hand side of each equation of

B_{b2} contains a single type of boolean operator:

$$B_{b2} = \left\{ \begin{array}{r} X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{pp'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pqq'a} \\ Y_{pp'qa} \stackrel{\nu}{=} (a = \tau \wedge X_{p'q}) \vee U_{pp'qa} \\ U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{\tau^*} q'} \bigvee_{q' \xrightarrow{a} q''} W_{pp'q'q''} \\ Z_{pqq'a} \stackrel{\nu}{=} (a = \tau \wedge X_{pq'}) \vee V_{pqq'a} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{\tau^*} p'} \bigvee_{p' \xrightarrow{a} p''} W_{p'p''qq'} \\ W_{pp'qq'} \stackrel{\nu}{=} X_{pq} \wedge X_{p'q'} \end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2}$$

This transformation into a simple BES B_{b2} as we consider in our framework (see Section 2) does not change the interpretation of the variables $X_{p,q}$ of the original BES, i.e., $(\llbracket B_{b1} \rrbracket)_{X_{p,q}} = (\llbracket B_{b2} \rrbracket)_{X_{p,q}}$ for all $p \in Q_1$ and $q \in Q_2$.

The final step towards the BES B_b given in Definition 4 is to get rid of the τ-closures present in the right-hand sides of the equations defining $U_{pp'qa}$ and $V_{pqq'a}$. To achieve this, we consider the following BES:

$$B = \left\{ \begin{array}{l} U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{\tau^*} q'} \bigvee_{q' \xrightarrow{a} q''} W_{pp'q'q''} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{\tau^*} p'} \bigvee_{p' \xrightarrow{a} p''} W_{p'p''qq'} \end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2}$$

Since both LTSs M_1 and M_2 are τ-convergent, Lemma 5 ensures that $\llbracket B \rrbracket \delta = \llbracket B' \rrbracket \delta$, where B' is defined as follows:

$$B' = \left\{ \begin{array}{l} U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} W_{pp'qq'} \vee \bigvee_{q \xrightarrow{\tau} q'} U_{pp'q'a} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} W_{pp'qq'} \vee \bigvee_{p \xrightarrow{\tau} p'} V_{p'qq'a} \end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2}$$

Starting with the BESs B and B', which have the same interpretation in any context δ, we can apply Lemma 4 repeatedly in order to add all the equations of B_{b2} defining variables X_{pq}, $Y_{pp'qa}$, $Z_{pqq'a}$, and $W_{pp'qq'}$ for all $p, p' \in Q_1$, $q, q' \in Q_2$, and $a \in A_1 \cup A_2$, still ensuring that the resulting BESs have the same interpretation. Upon completion of this process, the BES derived from B is B_{b2} and the BES derived from B' is B_b (note that we can permute freely the equations of a BES without changing the interpretation of its variables):

$$B_b = \left\{ \begin{array}{r} X_{pq} \stackrel{\nu}{=} \bigwedge_{p \xrightarrow{a} p'} Y_{pp'qa} \wedge \bigwedge_{q \xrightarrow{a} q'} Z_{pqq'a} \\ Y_{pp'qa} \stackrel{\nu}{=} (a = \tau \wedge X_{p'q}) \vee U_{pp'qa} \\ U_{pp'qa} \stackrel{\nu}{=} \bigvee_{q \xrightarrow{a} q'} W_{pp'qq'} \vee \bigvee_{q \xrightarrow{\tau} q'} U_{pp'q'a} \\ Z_{pqq'a} \stackrel{\nu}{=} (a = \tau \wedge X_{pq'}) \vee V_{pqq'a} \\ V_{pqq'a} \stackrel{\nu}{=} \bigvee_{p \xrightarrow{a} p'} W_{pp'qq'} \vee \bigvee_{p \xrightarrow{\tau} p'} V_{p'qq'a} \\ W_{pp'qq'} \stackrel{\nu}{=} X_{pq} \wedge X_{p'q'} \end{array} \right\}_{p,p' \in Q_1, q,q' \in Q_2, a \in A_1 \cup A_2}$$

By vertue of Lemma 4 these BESs have the same interpretation (the context δ becomes useless since both BESs are closed), meaning that B_b has in turn the same interpretation as B_{b1}, and thus it reflects the branching bisimulation of the τ-convergent LTSs M_1 and M_2 correctly. □

Centre de recherche INRIA Grenoble – Rhône-Alpes
Inovallée, 655, avenue de l'Europe, Montbonnot - 38334 Saint Ismier Cedex (France)

Centre de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes, 4, rue Jacques Monod - Bât. G - 91893 Orsay Cedex (France)
Centre de recherche INRIA Nancy – Grand Est : 615, rue du Jardin Botanique - 54600 Villers-lès-Nancy (France)
Centre de recherche INRIA Rennes – Bretagne Atlantique : Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399