Compositional Verification of Concurrent Systems by Combining Bisimulations

Frédéric Lang · Radu Mateescu · Franco Mazzanti

September 8, 2021

Abstract One approach to verify a property expressed as a modal μ-calculus formula on a system with several concurrent processes is to build the underlying state space compositionally (i.e., by minimizing and recomposing the state spaces of individual processes in a hierarchical way, keeping visible only the relevant actions occurring in the formula), and check the formula on the resulting state space. It was shown previously that, when checking the formulas of the L_μ^{dbr} fragment of the μ-calculus (consisting of weak modalities only), individual processes can be minimized modulo divergence-preserving branching (divbranching for short) bisimulation. In this paper, we refine this approach to handle formulas containing both strong and weak modalities, so as to enable a combined use of strong or divbranching bisimulation minimization on concurrent processes depending whether they contain or not the actions occurring in the strong modalities of the formula. We extend L_μ^{dbr} with strong modalities and show that the combined minimization approach preserves the truth value of formulas of the extended fragment. We implemented this approach on top of the CADP verification toolbox and demonstrated how it improves the capabilities of compositional verification on realistic examples of concurrent systems. In particular, we applied our approach to the verification problems of the RERS 2019 challenge and observed drastic reductions of the state space compared to the approach in which only strong bisimulation minimization is used, on formulas not preserved by divbranching bisimulation.

Keywords Concurrency theory, labelled transition system, modal mu-calculus, model checking, state space reduction, temporal logic

Frédéric Lang · Radu Mateescu
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP*, LIG, 38000 Grenoble, France
* Institute of Engineering Univ. Grenoble Alpes
E-mail: {Frederic.Lang,Radu.Mateescu}@inria.fr

Franco Mazzanti
ISTI-CNR, Pisa, Italy
E-mail: Franco.Mazzanti@isti.cnr.it

1 Introduction

We consider the problem of verifying a temporal logic property φ on a concurrent system $P_1 \,||\, ... \,||\, P_n$ consisting of n processes composed in parallel. We work in the action-based setting, the property φ being specified as a formula of the modal μ-calculus (L_μ) [29] and the processes P_i being described in a language with process algebraic flavour. A well-known problem is the state-space explosion that happens when the system state space exceeds the available computer memory.

Compositional verification is a set of techniques and tools that have proven efficient to palliate state-space explosion in many situations [15]. These techniques may be either independent of the property, i.e., focus only on the construction of the system state space, such as compositional state space construction [36, 49, 53, 52, 21, 55, 30]. Alternatively, they may depend on the property, e.g., verification of the property on the full system is decomposed in the verification of properties on (expectedly smaller) sub-systems, such as in compositional reachability analysis [58, 5], assume-guarantee reasoning [46], or partial model checking [1].

Nevertheless, the frontier between property-independent and property-dependent techniques is blurred. In compositional state space construction, to be able to reduce the system size, a set of actions is selected and a suitable equivalence relation (e.g., strong bisimulation, branching bisimulation, or divergence-preserving branching bisimulation[1] — divbranching for short) is chosen, restricting the set of properties preserved after hiding the selected actions and reducing the system w.r.t. the selected relation. Therefore, there is still a dependency between the state space construction and the set of properties that can be verified. Given a formula φ of L_μ to be verified on the system, Mateescu & Wijs [40] have pushed this idea and shown how to extract a maximal hiding set of actions and an equivalence relation (either strong or divbranching bisimulation) automatically from φ, thus inviting the compositional state space construction technique to the table of property-dependent reductions. To select the equivalence relation from the formula, they have identified an L_μ fragment named L_μ^{dbr}, which is adequate[2] with divbranching bisimulation [40]. This fragment consists of L_μ restricted to *weak* modalities, which match visible actions preceded by arbitrary sequences of hidden actions, as opposed to traditional strong modalities $\langle\alpha\rangle\,\varphi_0$ and $[\alpha]\,\varphi_0$, which match only a single action satisfying α. If φ belongs to L_μ^{dbr}, then the system can be reduced for divbranching bisimulation; otherwise, it can be reduced for strong bisimulation, the weakest equivalence relation preserving full L_μ.

In this paper, we revisit and refine this approach to accommodate L_μ formulas containing both strong and weak modalities. To do so, we define a logic named $L_\mu^{strong}(A_s)$, which extends L_μ^{dbr} with strong modalities matching only the actions belonging to a given set A_s of *strong* actions. The set A_s induces a partition of the processes $P_1 \,||\, ... \,||\, P_n$ into those containing at least one strong action, and those that do not. We show that a formula φ of $L_\mu^{strong}(A_s)$ is still preserved if

[1] In [40, 15], the name *divergence-sensitive* is used instead of *divergence-preserving* branching bisimulation (or branching bisimulation with explicit divergences) [56, 57]. This could lead to a confusion with the relation defined in [9], also called *divergence-sensitive* but slightly different from the former relation. To be consistent in notations, we replace by *dbr* the abbreviation *dsbr* used in earlier work.

[2] We say that a logic is adequate with an equivalence relation if equivalent processes are those satisfying exactly the same formulas expressed in the logic.

the processes containing strong actions are reduced modulo strong bisimulation and the other ones modulo divbranching bisimulation. We also provide guidelines for extracting the set A_s from particular L_μ formulas encoding the operators of widely-used temporal logics, such as CTL [6], ACTL [42], PDL [13], and PDL-Δ [50]. This combined use of bisimulations to reduce different parts of the same system makes possible to fine-tune the compositional state space construction by going smoothly from strong bisimulation (when all modalities are strong) to divbranching bisimulation (when A_s is empty, as in the previous approach based on L_μ^{dbr}). We implemented this approach on top of the CADP verification toolbox [16], and demonstrated how it improves the capabilities of compositional verification on two realistic case studies, namely the TFTP plane-ground communication protocol specified in [18] and the parallel CTL benchmarks of the RERS'2018 and RERS'2019 challenges.

This paper is an extended version of a previous paper [33]. Besides new examples and informal explanations, it incorporates the full proofs of all lemmas and theorems, a new section about the automatic extraction of a set of strong actions from a (test-free) PDL formula, a new theorem about the complexity of extracting a minimal set of strong actions from an arbitrary L_μ formula, more details on the TFTP case study, a new section on the RERS'2019 parallel challenge (which was not addressed in the short version), and a new related work section.

The paper is organized as follows. Section 2 recalls some definitions. Section 3 defines $L_\mu^{strong}(A_s)$ and proves the main result of its adequacy with the combined use of strong and divbranching bisimulations. Section 4 deals with the problem of extracting sets of strong actions in the general case of L_μ formulas and in particular fragments. Section 5 presents the experimental results obtained on the case studies. Section 6 discusses the relation between the compositional verification approach presented in this paper and two other approaches, namely partial model checking and partial order reductions. Finally, Section 7 contains concluding remarks and directions of future work. The case studies are available at `http://doi.org/10.5281/zenodo.2634148`.

2 Background

2.1 LTS compositions and reductions

We consider systems whose behavioural semantics can be represented using an LTS (*Labelled Transition System*).

Definition 1 (LTS) Let $\mathcal{A}$ denote an infinite set of actions, including the invisible action τ, which denotes internal behaviour. All actions in $\mathcal{A} \setminus \{\tau\}$ are called visible actions. An LTS is a tuple $(\Sigma, A, \longrightarrow, p_{init})$, where Σ is a set of states, $A \subseteq \mathcal{A}$ is a set of actions, $\longrightarrow \subseteq \Sigma \times A \times \Sigma$ is the (labelled) transition relation, and $p_{init} \in \Sigma$ is the initial state. Note that A may be empty only in the pathological case where $\longrightarrow$ is empty, whereas Σ has at least one element p_{init}. We write $p \xrightarrow{a} p'$ if $(p, a, p') \in \longrightarrow$ and $p \xrightarrow{\tau^*} p'$ if there is a (possibly empty) sequence of τ-transitions from p to p', i.e., states $p_0, \ldots, p_n$ $(n \geq 0)$ such that $p = p_0$, $p' = p_n$, and $p_i \xrightarrow{\tau} p_{i+1}$ for $i = 0, \ldots, n-1$.

LTS can be composed in parallel and their actions can be abstracted away using the parallel composition and hiding operators defined below. Prior to hiding, an action mapping operator is also introduced for the generality of the approach.

Definition 2 (Parallel composition of LTS) Let $P = (\Sigma_P, A_P, \longrightarrow_P, p_{init})$, $Q = (\Sigma_Q, A_Q, \longrightarrow_Q, q_{init})$, and $A_{sync} \subseteq \mathcal{A} \setminus \{\tau\}$. The parallel composition of P and Q with synchronization on A_{sync}, written "$P \,|[A_{sync}]|\, Q$", is defined as the LTS $(\Sigma_P \times \Sigma_Q, A_P \cup A_Q, \longrightarrow, (p_{init}, q_{init}))$, where $(p, q) \xrightarrow{a} (p', q')$ if and only if either:

- $p \xrightarrow{a}_P p'$, $q' = q$, and $a \notin A_{sync}$, or
- $p' = p$, $q \xrightarrow{a}_Q q'$, and $a \notin A_{sync}$, or
- $p \xrightarrow{a}_P p'$, $q \xrightarrow{a}_Q q'$, and $a \in A_{sync}$.

To alleviate notations, we generally omit the curly braces delimiting elements of the set A_{sync}, e.g., writing respectively $P \,|[a, b]|\, Q$ and $P \,|[]|\, Q$ instead of $P \,|[\{a, b\}]|\, Q$ and $P \,|[\{\}]|\, Q$.

Definition 3 (Action mapping) Consider an LTS $P = (\Sigma_P, A_P, \longrightarrow_P, p_{init})$ and a total function $\rho : A_P \to [\mathcal{A}]^{<\omega}$ (where $[\mathcal{A}]^{<\omega}$ denotes the set of finite subsets of $\mathcal{A}$) called action mapping. By abuse of notation, we write $\rho(A_P)$ for the image set of ρ, defined as $\bigcup_{a \in A_P} \rho(a)$, and $\rho(P)$ for the action mapping ρ applied to P, defined as $(\Sigma_P, \rho(A_P), \longrightarrow, p_{init})$, where $\longrightarrow = \{(p_1, a', p_2) \mid (\exists a \in A_P)\ p_1 \xrightarrow{a}_P p_2 \wedge a' \in \rho(a)\}$. An action mapping ρ is admissible if $\tau \in A_P \Rightarrow \rho(\tau) = \{\tau\}$.

Action mapping enables a single action a to be mapped onto the empty set of actions, onto a single action a', or onto more than one (but finitely many) actions $a'_0, \ldots, a'_{n+1}$ ($n \geq 0$). In the first case, every transition labelled by a is removed. In the second case, a is renamed into a'. In the third case, every transition labelled by a is replaced by $n + 2$ transitions with same source and target states, labelled by $a'_0, \ldots, a'_{n+1}$. Action hiding is a special case of admissible action mapping.

Definition 4 (Action hiding) Let $P = (\Sigma_P, A_P, \longrightarrow_P, p_{init})$ and $A \subseteq \mathcal{A} \setminus \{\tau\}$. We write "**hide** A **in** P" for the LTS $\rho(P)$, where ρ is the admissible action mapping defined by $(\forall a \in A_P \cap A)\ \rho(a) = \{\tau\}$ and $(\forall a \in A_P \setminus A)\ \rho(a) = \{a\}$.

Parallel composition and admissible action mapping subsume all abstraction and composition operators encodable as *networks of LTS* [31,15,10], such as the parallel composition, hiding, renaming, and cut (or restriction) operators of the process languages CCS [41], CSP [48], μCRL [23], mCRL2 [22], LOTOS [27], E-LOTOS [28], and LNT [4], as well as synchronization vectors. The ability to map a single action onto a set of actions is essential to attain such a generality. For instance, the composition of P and Q using synchronization vectors $a||b \to a_1$ and $a||c \to a_2$ (where action a of P synchronizes with either action b or c of Q), can be written e.g., as $\rho_2(\rho_1(P) \,|[b, c]|\, Q)$, where ρ_1 maps a onto $\{b, c\}$ and ρ_2 maps b onto $\{a_1\}$ and c onto $\{a_2\}$.

In the sequel, we write $P_1 \,||\, \ldots \,||\, P_n$ for any expression composing $P_1, \ldots, P_n$ using any combination of parallel composition and action mapping. Given any partition of $P_1, \ldots, P_n$ into arbitrary subsets $\mathcal{P}_1$ and $\mathcal{P}_2$, it is always possible to rewrite $P_1 \,||\, \ldots \,||\, P_n$ in the form $(||_{P_i \in \mathcal{P}_1} P_i) \,||\, (||_{P_j \in \mathcal{P}_2} P_j)$ using appropriate action mappings, even for non-associative parallel composition operators. For instance, the parallel composition operator $|[\ldots]|$ is not associative, as $P_1 \,|[a]|\, (P_2 \,|[]|\, P_3)$ is

not equal (in terms of LTS) to $(P_1 \,|[a]|\, P_2) \,|[]|\, P_3$. However, the former parallel composition expression is equal to $\rho_0((\rho_1(P_1) \,|[a_1]|\, \rho_2(P_2)) \,|[a_2]|\, \rho_3(P_3))$, where ρ_1 maps a onto $\{a_1, a_2\}$, ρ_2 renames a into a_1, ρ_3 renames a into a_2, and ρ_0 renames a_1 and a_2 into a.

LTS can be compared and reduced with respect to well-known bisimilarity relations. In this paper, we consider strong bisimulation [43] and divbranching bisimulation, which itself derives from branching bisimulation [56, 57].

Definition 5 (Strong bisimulation) A strong bisimulation is a symmetric relation $R \subseteq \Sigma \times \Sigma$ such that if $(p_1, p_2) \in R$ then for all $p_1 \xrightarrow{a} p_1'$, there exists p_2' such that $p_2 \xrightarrow{a} p_2'$ and $(p_1', p_2') \in R$.

Definition 6 (Branching bisimulation) A branching bisimulation is a symmetric relation $R \subseteq \Sigma \times \Sigma$ such that if $(p_1, p_2) \in R$ then for all $p_1 \xrightarrow{a} p_1'$, either:

- $a = \tau$ and $(p_1', p_2) \in R$, or
- there exists a sequence $p_2 \xrightarrow{\tau^*} p_2' \xrightarrow{a} p_2''$ such that $(p_1, p_2') \in R$ and $(p_1', p_2'') \in R$.

Definition 7 (Divbranching bisimulation) A divergence-preserving branching bisimulation (divbranching bisimulation for short) is a branching bisimulation R such that if $(p_1^0, p_2^0) \in R$ and there exists an infinite sequence $p_1^0 \xrightarrow{\tau} p_1^1 \xrightarrow{\tau} p_1^2 \xrightarrow{\tau} \ldots$ with $(p_1^i, p_2^0) \in R$ for all $i \geq 0$, then there exists an infinite sequence $p_2^0 \xrightarrow{\tau} p_2^1 \xrightarrow{\tau} p_2^2 \xrightarrow{\tau} \ldots$ such that $(p_1^i, p_2^j) \in R$ for all $i, j \geq 0$.

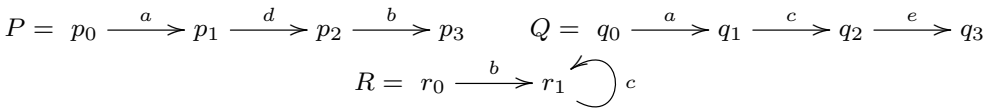
Definition 8 (Strong, branching, divbranching bisimilarity) Two states p_1 and p_2 are strongly (resp. branching, divbranching) bisimilar, written $p_1 \sim p_2$ (resp. $p_1 \sim_{br} p_2$, $p_1 \sim_{dbr} p_2$), if there exists a strong (resp. branching, divbranching) bisimulation R such that $(p_1, p_2) \in R$. Two LTS P_1 and P_2 are strongly (resp. branching, divbranching) bisimilar, written $P_1 \sim P_2$ (resp. $P_1 \sim_{br} P_2$, $P_1 \sim_{dbr} P_2$), if their initial states are strongly (resp. branching, divbranching) bisimilar.

Reduction with respect to (div)branching bisimulation usually yields much smaller LTS than reduction with respect to strong bisimulation when the LTS contains transitions labelled by τ. In particular, (div)branching bisimulation reduction compresses the so-called *inert* τ-transitions, which are the transitions labelled by τ, whose source and target state are (div)branching bisimilar. An effective way to fight against state explosion therefore consists in hiding as many labels as possible and then applying (div)branching bisimulation reduction whenever possible. However, some properties satisfied by an LTS may no longer hold true on the LTS obtained after label hiding and/or (div)branching bisimulation reduction. It is therefore essential to characterize which sets of properties are preserved by which reductions. This is a major subject of this paper.

Strong, branching, and divbranching bisimilarities are congruences for parallel composition and admissible action mapping. This allows reductions to be applied at any intermediate step during the state space construction, thus potentially reducing the overall cost of reduction. However, since processes may constrain each other by synchronization, composing LTS two by two following the algebraic structure of the composition expression and applying reduction after each composition can be orders of magnitude less efficient than other strategies in terms of the largest

intermediate LTS. Finding an optimal strategy is difficult. One generally relies on heuristics to select a subset of LTS to compose at each step of the compositional reduction. In this paper, we will use the *smart reduction* heuristic [8,15], which is implemented within the SVL [14] tool of CADP [16]. This heuristic tries to find an efficient composition order by analysing the synchronization and hiding structure of the composition expression.

Example 1 Consider the system defined as **hide** a **in** $(P \,|[a]|\, Q) \,|[b,c]|\, R$, where P, Q, and R are defined below (throughout this paper, in every graphical representation of an LTS, we use index 0 to denote its initial state):



The order consisting in first computing the minimal LTS corresponding to **hide** a **in** $(P \,|[a]|\, Q)$, then composing the result with R produces a largest intermediate LTS with 10 states and 13 transitions. On the other hand, computing the minimal LTS corresponding to **hide** a **in** $(P \,|[a]|\, Q) \,|[b,c]|\, R$ at once (which is the order chosen by smart reduction in this case) produces a (largest) LTS with 6 states and 5 transitions. Another order consisting in first composing Q and R, then reducing the result and composing it with P produces a largest intermediate LTS with 6 states and 6 transitions.

Note that composing Q and R is possible using, e.g., the composition expression $\rho_1(Q) \,|[b',c']|\, \rho_0(R)$, considering that **hide** a **in** $(P \,|[a]|\, Q) \,|[b,c]|\, R$ is equal to **hide** a **in** $\rho_2(P \,|[a,b,c]|\, (\rho_1(Q) \,|[b',c']|\, \rho_0(R)))$, where ρ_0 maps b onto $\{b,b'\}$, c onto $\{c,c'\}$, and a onto $\{a'\}$ (leaving all other actions unchanged), ρ_1 renames b into b' and c into c' (leaving all other actions unchanged), ρ_2 renames a' into a, b' into b, and c' into c (leaving all other actions unchanged), and a', b', c' are fresh actions that do not occur in P, Q, and R. In practice, we do not rewrite composition expressions in this way, but instead translate composition expressions into networks of LTS [31] and use an operator of aggregation defined in [8,15].

2.2 Temporal logics

Definition 9 (Modal μ-calculus [29]) The modal μ-calculus (L_μ) is built from action formulas α and state formulas φ, whose syntax and semantics w.r.t. an LTS $P = (\Sigma, A, \longrightarrow, p_{init})$ are defined as follows:

$$
\begin{array}{lll}
\alpha ::= a & & [\![a]\!]_A = \{a\} \\
\quad | \ \mathsf{false} & & [\![\mathsf{false}]\!]_A = \emptyset \\
\quad | \ \alpha_1 \vee \alpha_2 & & [\![\alpha_1 \vee \alpha_2]\!]_A = [\![\alpha_1]\!]_A \cup [\![\alpha_2]\!]_A \\
\quad | \ \neg\alpha_0 & & [\![\neg\alpha_0]\!]_A = A \setminus [\![\alpha_0]\!]_A \\
\\
\varphi ::= \mathsf{false} & & [\![\mathsf{false}]\!]_P\delta = \emptyset \\
\quad | \ \varphi_1 \vee \varphi_2 & & [\![\varphi_1 \vee \varphi_2]\!]_P\delta = [\![\varphi_1]\!]_P\delta \cup [\![\varphi_2]\!]_P\delta \\
\quad | \ \neg\varphi_0 & & [\![\neg\varphi_0]\!]_P\delta = \Sigma \setminus [\![\varphi_0]\!]_P\delta \\
\quad | \ \langle\alpha\rangle\,\varphi_0 & & [\![\langle\alpha\rangle\,\varphi_0]\!]_P\delta = \{p \in \Sigma \mid \exists p \xrightarrow{a} p'.a \in [\![\alpha]\!]_A \wedge p' \in [\![\varphi_0]\!]_P\delta \ \} \\
\quad | \ X & & [\![X]\!]_P\delta = \delta(X) \\
\quad | \ \mu X.\varphi_0 & & [\![\mu X.\varphi_0]\!]_P\delta = \bigcap\{U \subseteq \Sigma \mid \Phi_{0\,P,\delta}(U) \subseteq U\}
\end{array}
$$

where:

- $X \in \mathcal{X}$ are propositional variables, which denote sets of states;
- $\delta : \mathcal{X} \mapsto 2^{\Sigma}$ is a context mapping propositional variables to sets of states; $[\,]$ denotes the empty context and $\delta[U/X]$ the context identical to δ except for variable X, which is mapped to state set U;
- and $\Phi_{0\,P,\delta} : 2^{\Sigma} \to 2^{\Sigma}$ is the functional associated to the formula $\mu X.\varphi_0$, defined as $\Phi_{0\,P,\delta}(U) = [\![\varphi_0]\!]_P \delta[U/X]$.

A closed formula is a formula in which every occurrence of X occurs in the context φ_0 of a fixed point operator $\mu X.\varphi_0$ or $\nu X.\varphi_0$. For a closed formula φ, we write $P \models \varphi$ (read P satisfies φ) for $p_{init} \in [\![\varphi]\!]_P[\,]$.

Action formulas α are built from actions and Boolean operators. State formulas φ are built from Boolean operators, the possibility modality $\langle\alpha\rangle\,\varphi_0$ denoting the states with an outgoing transition labelled by an action satisfying α and leading to a state satisfying φ_0, and the minimal fixed point operator $\mu X.\varphi_0$ denoting the least solution of the equation $X = \varphi_0$ interpreted over 2^{Σ}.

The usual derived operators are defined as follows: Boolean connectors $\mathsf{true} = \neg\mathsf{false}$ and $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$; necessity modality $[\alpha]\,\varphi_0 = \neg\langle\alpha\rangle\,\neg\varphi_0$; and maximal fixed point operator $\nu X.\varphi_0 = \neg\mu X.\neg\varphi_0[\neg X/X]$, where $\varphi_0[\neg X/X]$ is the syntactic substitution of X by $\neg X$ in φ_0. Syntactically, $\langle\rangle$ and $[]$ have the highest precedence, followed by $\wedge$, then $\vee$, and finally μ and ν, so that e.g., the formula $\mu X.\langle a\rangle\,\mathsf{true} \wedge [b]\,X \vee \nu Y.\langle c\rangle\,Y$ is parsed as $\mu X.(((\langle a\rangle\,\mathsf{true}) \wedge [b]\,X) \vee \nu Y.\langle c\rangle\,Y)$. To have a well-defined semantics, state formulas are syntactically monotonic [29], i.e., in every subformula $\mu X.\varphi_0$, all occurrences of X in φ_0 fall in the scope of an even number of negations. Thus, negations can be eliminated by downward propagation.

Although L_μ subsumes most action-based logics, its operators are rather low-level and lead to complex formulas. In practice, temporal logics or extensions of L_μ with higher-level operators are used, avoiding (or at least reducing) the use of fixed point operators and modalities. We review informally some of these logics (whose operators can be translated to L_μ), which will be useful in the sequel.

Propositional Dynamic Logic with Looping The logic PDL-Δ [50] introduces the modalities $\langle\beta\rangle\,\varphi_0$ and $\langle\beta\rangle\,@$, where β is a regular formula defined as follows:

$$\beta ::= \varphi? \mid \alpha \mid \beta_1.\beta_2 \mid \beta_1\,|\,\beta_2 \mid \beta_0^*$$

Regular formulas β denote sets of transition sequences in an LTS: the testing operator $\varphi?$ denotes all zero-step sequences consisting of states satisfying φ; α denotes all one-step sequences consisting of a transition labelled by an action satisfying α; the concatenation $\beta_1.\beta_2$, choice $\beta_1\,|\,\beta_2$, and transitive-reflexive closure β_0^* operators have their usual semantics transposed to transition sequences.

The regular diamond modality $\langle\beta\rangle\,\varphi_0$ denotes the states with an outgoing transition sequence satisfying β and leading to a state satisfying φ_0. The infinite looping operator $\langle\beta\rangle\,@$ denotes the states having an outgoing transition sequence consisting of an infinite concatenation of subsequences satisfying β. An encoding of PDL-Δ in L_μ is formally defined in [11].

Action Computation Tree Logic The logic ACTL\X (ACTL without next operator) [42] introduces four temporal operators, whose semantics can be found in terms of L_μ formulas in [12,40], where α_1, α_2 are action formulas interpreted over visible actions:

$$\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2), \mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2), \mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2), \mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2)$$

A transition sequence satisfies the path formula $\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2$ if it contains a visible transition whose action satisfies α_2 and whose target state satisfies φ_2, whereas at any moment before this transition, φ_1 holds and all visible actions satisfy α_1. A sequence satisfies $\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2$ if it contains a state satisfying φ_2 and at any moment before, φ_1 holds and all visible actions satisfy α_1. A state satisfies $\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2)$ (resp. $\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2)$) if it has an outgoing sequence satisfying $\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2$ (resp. $\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2$). It satisfies $\mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2)$ (resp. $\mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2)$) if all its outgoing sequences satisfy the corresponding path formula. An encoding of ACTL\X in L_μ is formally defined in [40].

The following abbreviations are often used:

$$\mathsf{EF}_\alpha(\varphi_0) = \mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}_\alpha\,\varphi_0) \qquad \mathsf{AG}_\alpha(\varphi_0) = \neg\mathsf{EF}_{\neg\alpha}(\mathsf{true}) \wedge \neg\mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\neg\varphi_0)$$

A state satisfies $\mathsf{EF}_\alpha(\varphi_0)$ if it has an outgoing sequence leading to a transition whose action satisfies α and target state satisfies φ_0. A state satisfies $\mathsf{AG}_\alpha(\varphi_0)$ if none of its outgoing sequences leads to a transition labelled by an action not satisfying α or to a state not satisfying φ_0.

Computation Tree Logic The logic CTL [6] contains the following operators:

$$\mathsf{E}(\varphi_1\,\mathsf{U}\,\varphi_2), \mathsf{A}(\varphi_1\,\mathsf{U}\,\varphi_2), \mathsf{E}(\varphi_1\,\mathsf{W}\,\varphi_2), \mathsf{A}(\varphi_1\,\mathsf{W}\,\varphi_2), \mathsf{EF}(\varphi_0), \mathsf{AG}(\varphi_0), \mathsf{AF}(\varphi_0), \mathsf{EG}(\varphi_0)$$

A state satisfies $\mathsf{E}(\varphi_1\,\mathsf{U}\,\varphi_2)$ (resp. $\mathsf{A}(\varphi_1\,\mathsf{U}\,\varphi_2)$) if some of (resp. all) its outgoing sequences lead to states satisfying φ_2 after passing only through states satisfying φ_1. It satisfies $\mathsf{E}(\varphi_1\,\mathsf{W}\,\varphi_2)$ (resp. $\mathsf{A}(\varphi_1\,\mathsf{W}\,\varphi_2)$) if some of (resp. all) its outgoing sequences either contain only states satisfying φ_1, or lead to states satisfying φ_2 after passing only through states satisfying φ_1. A state satisfies $\mathsf{EF}(\varphi_0)$ (resp. $\mathsf{AF}(\varphi_0)$) if some of (resp. all) its outgoing sequences lead to states satisfying φ_0. A state satisfies $\mathsf{EG}(\varphi_0)$ (resp. $\mathsf{AG}(\varphi_0)$) if some of (resp. all) its outgoing sequences contain only states satisfying φ_0.

The above CTL operators can be encoded in ACTL\X (hence in L_μ) using the following equations:

$$\begin{aligned}
\mathsf{E}(\varphi_1\,\mathsf{U}\,\varphi_2) &= \mathsf{E}(\varphi_1\ {}_{\mathsf{true}}\mathsf{U}\,\varphi_2) && \text{(Ctl1)}\\
\mathsf{A}(\varphi_1\,\mathsf{U}\,\varphi_2) &= \mathsf{A}(\varphi_1\ {}_{\mathsf{true}}\mathsf{U}\,\varphi_2) && \text{(Ctl2)}\\
\mathsf{EF}(\varphi_0) &= \mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\varphi_0) && \text{(Ctl3)}\\
\mathsf{AG}(\varphi_0) &= \neg\mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\neg\varphi_0) && \text{(Ctl4)}\\
\mathsf{AF}(\varphi_0) &= \mathsf{A}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\varphi_0) && \text{(Ctl5)}\\
\mathsf{EG}(\varphi_0) &= \neg\mathsf{A}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\neg\varphi_0) && \text{(Ctl6)}\\
\mathsf{E}(\varphi_1\,\mathsf{W}\,\varphi_2) &= \mathsf{E}(\varphi_1\,\mathsf{U}\,\varphi_2) \vee \mathsf{EG}(\varphi_1) && \text{(Ctl7)}\\
\mathsf{A}(\varphi_1\,\mathsf{W}\,\varphi_2) &= \neg\mathsf{E}(\neg\varphi_2\,\mathsf{U}\,\neg\varphi_1 \wedge \neg\varphi_2) && \text{(Ctl8)}
\end{aligned}$$

2.3 Compositional property-dependent LTS reductions

Given a formula $\varphi \in L_\mu$ and a composition of processes $P_1 \,||\, \ldots \,||\, P_n$, [40] shows two results that allow $P_1 \,||\, \ldots \,||\, P_n$ to be reduced compositionally, while preserving the truth value of φ. The first result is a procedure, called *maximal hiding*, which extracts systematically from φ a set of actions $H(\varphi)$ that are not discriminated by any action formula occurring in φ. It is shown that $P_1 \,||\, \ldots \,||\, P_n \models \varphi$ if and only if **hide** $H(\varphi)$ **in** $(P_1 \,||\, \ldots \,||\, P_n) \models \varphi$. The second result is the identification of a fragment of L_μ, called L_μ^{dbr}, which is strictly more expressive than μACTL\X[3] and adequate with divbranching bisimulation. This fragment is defined as follows.

Definition 10 (Modal μ-calculus fragment L_μ^{dbr} [40]) By convention, we use the symbols α_τ and α_a to denote action formulas such that $\tau \in [\![\alpha_\tau]\!]_A$ and $\tau \notin [\![\alpha_a]\!]_A$. The fragment L_μ^{dbr} of L_μ is defined as the set of formulas that are semantically equivalent to some formula of the following language (whose modalities are a subset of PDL-Δ regular modalities):

$$\begin{array}{l} \varphi ::= \mathsf{false} \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_0 \mid X \mid \mu X.\varphi_0 \\ \quad \mid \ \langle (\varphi_1?.\alpha_\tau)^* \rangle\, \varphi_2 \mid \langle (\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a \rangle\, \varphi_2 \mid \langle \varphi_1?.\alpha_\tau \rangle\, @ \end{array}$$

The ultra-weak modality $\langle (\varphi_1?.\alpha_\tau)^* \rangle\, \varphi_2$, weak modality $\langle (\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a \rangle\, \varphi_2$, and weak infinite looping modality $\langle \varphi_1?.\alpha_\tau \rangle\, @$ are shorthand notations defined upon L_μ as follows:

$$\begin{array}{c} \langle (\varphi_1?.\alpha_\tau)^* \rangle\, \varphi_2 = \mu X.\varphi_2 \vee (\varphi_1 \wedge \langle \alpha_\tau \rangle\, X) \\ \langle (\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a \rangle\, \varphi_2 = \mu X.\varphi_1 \wedge (\langle \alpha_a \rangle\, \varphi_2 \vee \langle \alpha_\tau \rangle\, X) \\ \langle \varphi_1?.\alpha_\tau \rangle\, @ = \nu X.\varphi_1 \wedge \langle \alpha_\tau \rangle\, X \end{array}$$

Derived operators are also defined in terms of L_μ^{dbr} as follows:

$$\begin{array}{c} [(\varphi_1?.\alpha_\tau)^*]\, \varphi_2 = \neg\langle (\varphi_1?.\alpha_\tau)^* \rangle\, \neg\varphi_2 \\ [\varphi_1?.\alpha_\tau]\, \dashv = \neg\langle \varphi_1?.\alpha_\tau \rangle\, @ \\ [(\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a]\, \varphi_2 = \neg\langle (\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a \rangle\, \neg\varphi_2 \end{array}$$

Depending on the L_μ fragment φ belongs to, it is thus possible to determine whether the system can or cannot be reduced for divbranching bisimulation[4]. We refine this result in the next section, where we show that even if φ does not belong to L_μ^{dbr}, some part of the system might still be minimized for divbranching bisimulation.

3 Combining Bisimulations Compositionally

The above approach is a *mono-bisimulation* approach: either the formula is in L_μ^{dbr} and then the system is entirely reduced for divbranching bisimulation, or it is not and then the system is entirely reduced for strong bisimulation: the presence in

[3] μACTL\X denotes ACTL\X plus fixed points. The authors of [40] claim that L_μ^{dbr} is as expressive as μACTL\X, but they omit that the $\langle \varphi_1?.\alpha_\tau \rangle\, @$ weak infinite looping modality cannot be expressed in μACTL\X.

[4] Note however that there is no proof that the test $\varphi \in L_\mu^{dbr}$ is decidable. So far, we can only conjecture it, based on the fact that testing equivalence between L_μ formulas is decidable [51].

the formula of a single operator that is not encodable in L_μ^{dbr} wipes the possibility to use divbranching bisimulation reduction out. In this section, we refine this approach by showing that, even if the formula is not in L_μ^{dbr}, it may still be possible to reduce some processes among the parallel processes $P_1, \ldots, P_n$ for divbranching instead of strong bisimulation. This approach relies on the fact that, in general, an arbitrary temporal logic formula φ may be rewritten in a form that contains both weak modalities, as those present in L_μ^{dbr}, and non-weak modalities of L_μ (called *strong modalities* in this context).

To do so, we characterize a family of fragments of L_μ, each of which is written $L_\mu^{strong}(A_s)$, where A_s is the set of actions that can be matched by strong modalities. We then prove that if φ belongs to $L_\mu^{strong}(A_s)$ and some process P_i does not contain any action from the set A_s, then P_i can be reduced for divbranching bisimulation. Throughout this section, we assume that the concurrent system $P_1 \mid\mid \ldots \mid\mid P_n$ is fixed, and we write A for the set of actions occurring in the system.

3.1 The $L_\mu^{strong}(A_s)$ fragments of L_μ

Definition 11 (Fragment $L_\mu^{strong}(A_s)$ parameterized by A_s) Let $A_s \subseteq A$ be a fixed set of actions, called strong actions, and let α_s denote any action formula such that $[\![\alpha_s]\!]_A \subseteq A_s$, called a strong action formula. The fragment $L_\mu^{strong}(A_s)$ of L_μ is defined as the set of formulas that are semantically equivalent to some formula of the following language:

$$\begin{aligned}\varphi ::= {} & \mathsf{false} \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_0 \mid \langle \alpha_s \rangle \varphi_0 \mid X \mid \mu X.\varphi_0 \\ \mid {} & \langle (\varphi_1?.\alpha_\tau)^* \rangle \varphi_2 \mid \langle (\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a \rangle \varphi_2 \mid \langle \varphi_1?.\alpha_\tau \rangle @\end{aligned}$$

We call $\langle \alpha_s \rangle \varphi_0$ a *strong modality*. $L_\mu^{strong}(A_s)$ is the fragment of L_μ consisting of formulas expressible in a form where strong modalities match only actions in A_s. Its formal relationship with L_μ^{dbr} and L_μ is given in Theorem 1.

Theorem 1 *The following three propositions hold trivially:*

1. $L_\mu^{strong}(\emptyset) = L_\mu^{dbr}$
2. $L_\mu^{strong}(\mathcal{A}) = L_\mu$
3. *if* $A_s \subset A'_s$ *then* $L_\mu^{strong}(A_s) \subset L_\mu^{strong}(A'_s)$

Given $\varphi \in L_\mu$, there exists a (not necessarily unique, see Theorem 3 page 26) minimal set A_s such that $\varphi \in L_\mu^{strong}(A_s)$. Obviously, $L_\mu^{strong}(A_s)$ is not adequate with divbranching bisimulation when A_s is not empty, as illustrated by the following example.

Example 2 The PDL formula $\varphi = [\mathsf{true}^*.a_1.a_2]\,\mathsf{false}$ expresses that the system does not contain two successive transitions labelled by a_1 and a_2 respectively. It is in the fragment $L_\mu^{strong}(\{a_2\})$ as it is equivalent to $[(\mathsf{true}?.\mathsf{true})^*.\mathsf{true}?.a_1]\,[a_2]\,\mathsf{false}$, but it does not belong to L_μ^{dbr}. Indeed, consider the LTS P, P', Q, and Q' depicted in Figure 1, where P' (resp. Q') denotes the minimal LTS equivalent to P (resp. Q) for divbranching bisimulation. The LTS $P \mid[a_1]\mid Q$ satisfies φ because a_1 is necessarily followed by a τ transition, but $P' \mid[a_1]\mid Q'$ (which is isomorphic to Q') does not.

However, although the system cannot be fully reduced for divbranching, we will see in the sequel that it can be partly reduced, i.e., that $P' \,|[a_1]|\, Q$ preserves φ as well as any other property belonging to some fragment $L_\mu^{strong}(A_s)$, provided A_s does not contain any of the actions occurring in P, namely a_1 and τ.

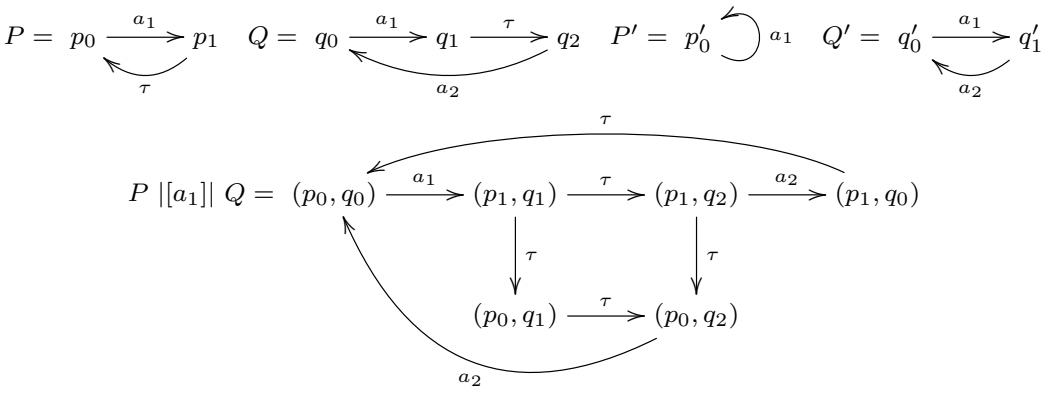


Fig. 1 LTS used in Examples 2 and 3

Actions that are not strong are called weak. Intuitively, weak actions are those actions that can always be delayed, i.e., the mere absence or presence of a weak action in a given state cannot validate (resp. invalidate) the property immediately, because there is still a chance that this action occurs (or stops to occur) in a subsequent state, such that it invalidates (resp. validates) the property. A strong action is thus an action that may potentially not be delayed, i.e., may have to occur *immediately*.

Theorem 1 indicates that it is always possible to consider as strong (i.e., immediate) some actions that could be considered as weak (i.e., delayable), whereas the converse is obviously wrong. We will see in the next section that having the most accurate knowledge as possible of which actions are strong and which are weak (i.e., not considering as strong an action that can actually be considered as weak) may help fighting against state explosion more effectively.

3.2 Applying divbranching bisimulation to selected components

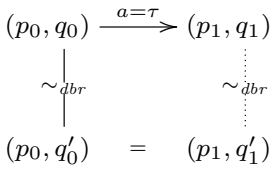
Theorem 2 below states the main result of this paper, namely that every component process containing only weak actions can be replaced by any divbranching equivalent process, without affecting the truth value of the formula.[5] To show Theorem 2, we first need Lemma 1, which simply lifts a standard property of branching and divbranching bisimulations up to the level of product states.

Lemma 1 *Let p_0, p_1 be states of an LTS P, and q_0, q_0', q_1 be states of an LTS Q, and consider the composition expression $P \,|[A_{sync}]|\, Q$. If $(p_0, q_0) \xrightarrow{a} (p_1, q_1)$ and $q_0' \sim_{dbr} q_0$, then there exists a state q_1' in Q such that $q_1' \sim_{dbr} q_1$ and either:*

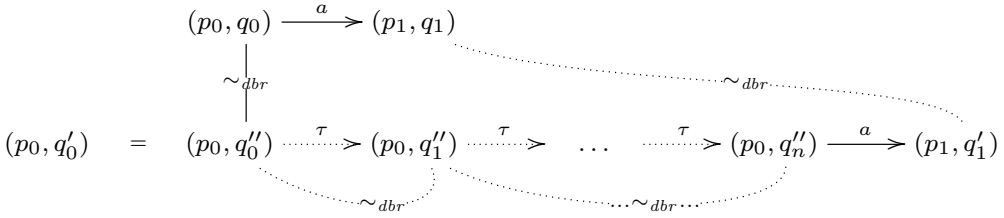
[5] Theorem 2 generalizes easily to more general compositions $P \,||\, Q$ (with admissible action mappings) if Q does not contain any action that maps onto a strong action.

- $a = \tau$, $p_0 = p_1$, $q_0 \sim_{dbr} q_1$, *and* $q'_1 = q'_0$, *or*
- *there exist* $q''_0, ..., q''_n$ $(n \geq 0)$ *such that* $q''_0 = q'_0$, *for all* $i \in 0..n-1$, $q''_{i+1} \sim_{dbr} q'_0$, $(p_0, q''_i) \xrightarrow{\tau} (p_0, q''_{i+1})$, *and* $(p_0, q''_n) \xrightarrow{a} (p_1, q'_1)$.

Graphically, this means that one of the diagrams below holds, where solid lines denote universal quantification whereas dotted lines denote existential quantification:



– or –



Proof From the definition of $P \,|[A_{sync}]|\, Q$ and the fact that $(p_0, q_0) \xrightarrow{a} (p_1, q_1)$, there are three possible cases:

1. $a \in A_{sync}$, $p_0 \xrightarrow{a}_P p_1$, and $q_0 \xrightarrow{a}_Q q_1$, or
2. $a \notin A_{sync}$, $p_0 \xrightarrow{a}_P p_1$, and $q_1 = q_0$, or
3. $a \notin A_{sync}$, $p_1 = p_0$, and $q_0 \xrightarrow{a}_Q q_1$

Instead of considering those three cases, we merge cases 1 and 3 (where $q_0 \xrightarrow{a}_Q q_1$, whatever p_0 does), which have a similar proof, into a single case. Thus we consider the following two cases, the first one corresponding to case 2 of the above enumeration, and the second one corresponding to cases 1 and 3:

- $q_1 = q_0$, $a \notin A_{sync}$, and $p_0 \xrightarrow{a}_P p_1$. Take $q'_1 = q'_0$. We have $q'_1 \sim_{dbr} q_1$ because $q'_1 = q'_0$, $q'_0 \sim_{dbr} q_0$, and $q_0 = q_1$. The second item of the lemma (bottom diagram) is verified with $n = 0$, $q''_0 = q''_n = q'_0$, and $(p_0, q''_0) = (p_0, q''_n) = (p_0, q'_0) \xrightarrow{a} (p_1, q'_0) = (p_1, q'_1)$.
- $q_0 \xrightarrow{a}_Q q_1$ (and either $a \notin A_{sync}$ and $p_1 = p_0$, or $a \in A_{sync}$ and $p_0 \xrightarrow{a}_P p_1$). Then since $q'_0 \sim_{dbr} q_0$, we have by definition of $\sim_{dbr}$:
  - Either $a = \tau$ and $q_0 \sim_{dbr} q_1$. Since $\tau \notin A_{sync}$, we have $p_0 = p_1$. Then, we can take $q'_1 = q'_0$. Indeed, $q'_1 \sim_{dbr} q_1$ because $q'_1 = q'_0$, $q'_0 \sim_{dbr} q_0$, and $q_0 \sim_{dbr} q_1$. The first item of the lemma (top diagram) is verified.
  - Or there exist $q''_0, ..., q''_n$ $(n \geq 0)$ such that $q''_0 = q'_0$, for all $i \in 0..n-1$, $q''_{i+1} \sim_{dbr} q'_0$ (by the well-known stuttering lemma of divbranching bisimilarity, see [57]), $q''_i \xrightarrow{\tau}_Q q''_{i+1}$, and $q''_n \xrightarrow{a}_Q q'_1$. In this case, we also have $(p_0, q''_i) \xrightarrow{\tau} (p_0, q''_{i+1})$ and $(p_0, q''_n) \xrightarrow{a} (p_1, q'_1)$, i.e., the second item of the lemma (bottom diagram) is verified. □

We draw the reader's attention to the fact that the top diagram of Lemma 1 concerns only the case $(p_0, q_0) \xrightarrow{\tau} (p_0, q_1)$ deriving from a transition $q_0 \xrightarrow{\tau}_Q$

q_1 where $q_0 \sim_{dbr} q_1$. The bottom diagram concerns all other cases, including $(p_0, q_0) \xrightarrow{\tau} (p_1, q_0)$ deriving from $p_0 \xrightarrow{\tau} p_1$ where $p_0 \sim_{dbr} p_1$.

Theorem 2 *Consider the finite LTS* $P = (\Sigma_P, A_P, \longrightarrow_P, p_{init})$, $Q = (\Sigma_Q, A_Q, \longrightarrow_Q, q_{init})$, *and* $Q' = (\Sigma_{Q'}, A_Q, \longrightarrow_{Q'}, q'_{init})$. *Let* $A_{sync} \subseteq \mathcal{A}$ *and* $\varphi \in L_\mu^{strong}(A_s)$. *If* $A_Q \cap A_s = \emptyset$ *and* $Q \sim_{dbr} Q'$, *then* $P \,|[A_{sync}]|\, Q \models \varphi$ *if and only if* $P \,|[A_{sync}]|\, Q' \models \varphi$.

Proof We prove that, given $p \in \Sigma_P$, $q \in \Sigma_Q$, and $q' \in \Sigma_{Q'}$ such that $q \sim_{dbr} q'$, (p, q) satisfies φ if and only if (p, q') satisfies φ. We show this claim by structural induction on the formula φ. Since we work on finite LTS, every formula containing a fixed point operator μ can be expanded into an equivalent formula by unfolding the fixed point a bounded number of times[6]. This way, we do not have to consider fixed points and propositional variables in this proof, and contexts δ mapping propositional variables to sets of states are always empty in the semantics of formulas. For conciseness, we write $[\![\varphi]\!]$ instead of $[\![\varphi]\!]_{P \,|[A_{sync}]|\, Q}[\,]$ and $[\![\varphi]\!]_{P \,|[A_{sync}]|\, Q'}[\,]$, respectively, as the LTS "$P \,|[A_{sync}]|\, Q$" or "$P \,|[A_{sync}]|\, Q'$" to which the semantics apply is clear from the context. Note that since the claim is symmetric, it is sufficient to prove one implication only. We thus assume that $(p, q) \in [\![\varphi]\!]$ and prove that $(p, q') \in [\![\varphi]\!]$.

- Case $\varphi = \mathsf{false}$. It is obvious that both $(p, q) \notin [\![\varphi]\!]$ and $(p, q') \notin [\![\varphi]\!]$.
- Case $\varphi = \varphi_1 \vee \varphi_2$. By definition, either $(p, q) \in [\![\varphi_1]\!]$ or $(p, q) \in [\![\varphi_2]\!]$. If $(p, q) \in [\![\varphi_1]\!]$ (resp. $[\![\varphi_2]\!]$) then $(p, q') \in [\![\varphi_1]\!]$ (resp. $[\![\varphi_2]\!]$) by the induction hypothesis. Therefore, $(p, q') \in [\![\varphi_1 \vee \varphi_2]\!]$.
- Case $\varphi = \neg\varphi_0$. By definition, $(p, q) \notin [\![\varphi_0]\!]$. By the induction hypothesis, $(p, q') \notin [\![\varphi_0]\!]$ and thus $(p, q') \in [\![\neg\varphi_0]\!]$.
- Case $\varphi = \langle\alpha_s\rangle\, \varphi_0$. By definition, there exists $a \in [\![\alpha_s]\!]_A$ such that $(p, q) \xrightarrow{a} (p_1, q_1)$ and $(p_1, q_1) \in [\![\varphi_0]\!]$. Let action a be such.
  Since $[\![\alpha_s]\!]_A \subseteq A_s$, $A_Q \cap A_s = \emptyset$, and $a \in [\![\alpha_s]\!]_A$, we have $a \notin A_Q$ (hence, $a \notin A_{sync}$). Therefore, $q = q_1$, $p \xrightarrow{a}_P p_1$, and $(p, q') \xrightarrow{a} (p_1, q')$. Since $q' \sim_{dbr} q$ and $(p_1, q) \in [\![\varphi_0]\!]$, we have by the induction hypothesis $(p_1, q') \in [\![\varphi_0]\!]$. As a consequence, $(p, q') \in [\![\langle\alpha_s\rangle\, \varphi_0]\!]$.
- Case $\varphi = \langle(\varphi_1?.\alpha_\tau)^*\rangle\, \varphi_2$. By definition, there exist $m \geq 0, p_i \in \Sigma_P, q_i \in \Sigma_Q$, and $a_i \in [\![\alpha_\tau]\!]_A$ $(i \in 0..m)$, such that $(p, q) = (p_0, q_0)$, $(p_m, q_m) \in [\![\varphi_2]\!]$, and for all $0 \leq i < m$, $(p_i, q_i) \xrightarrow{a_i} (p_{i+1}, q_{i+1})$ and $(p_i, q_i) \in [\![\varphi_1]\!]$. We show that the same holds from state (p, q'), i.e., there exists a sequence of transitions matching α_τ and passing through states satisfying φ_1, until reaching a state satisfying φ_2. By the induction hypothesis, $(p, q') = (p_0, q'_0) \in [\![\varphi_1]\!]$. Consider one transition $(p_i, q_i) \xrightarrow{a_i} (p_{i+1}, q_{i+1})$. We know that $(p_i, q_i) \in [\![\varphi_1]\!]$. Assume that there exists $q'_i \sim_{dbr} q_i$. Substituting a_i for a, (p_i, q_i) for (p_0, q_0), and (p_{i+1}, q_{i+1}) for (p_1, q_1) in the two diagrams of Lemma 1, it appears clearly that there exists a state q'_{i+1} such that $q'_{i+1} \sim_{dbr} q_{i+1}$. Moreover, by the induction hypothesis, every state divbranching equivalent to (p_i, q_i) in the bottom sequence of the diagrams is in $[\![\varphi_1]\!]$. Finally, note that $\tau \in [\![\alpha_\tau]\!]$ by definition, so that each of the τ-transitions in the bottom sequence of the second diagram thus matches α_τ. Moreover, $(p_m, q_m) \in [\![\varphi_2]\!]$ and $q'_m \sim_{dbr} q_m$, so by the induction hypothesis, we have also $(p_m, q'_m) \in [\![\varphi_2]\!]$.

[6] Note that this number may be different for $P \,|[A_{sync}]|\, Q$ and for $P \,|[A_{sync}]|\, Q'$. We have to take the maximum.

- Case $\varphi = \langle(\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a\rangle\,\varphi_2$. Since $\langle(\varphi_1?.\alpha_\tau)^*.\varphi_1?.\alpha_a\rangle\,\varphi_2$ is equivalent to $\langle(\varphi_1?.\alpha_\tau)^*\rangle\,\langle(\varphi_1?.\tau)^*.\varphi_1?.\alpha_a\rangle\,\varphi_2$ and since we have already considered the case $\varphi = \langle(\varphi_1?.\alpha_\tau)^*\rangle\,\varphi_2$ above, we only have to consider $\varphi = \langle(\varphi_1?.\tau)^*.\varphi_1?.\alpha_a\rangle\,\varphi_2$, i.e., $\alpha_\tau = \tau$. By definition, there exist $m \geq 0, p_i \in \Sigma_P, q_i \in \Sigma_Q$ $(i \in 0..m+1)$ and $a \in [\![\alpha_a]\!]_A$, such that $(p,q) = (p_0,q_0)$, $(p_m,q_m) \xrightarrow{a} (p_{m+1},q_{m+1})$, $(p_{m+1},q_{m+1}) \in [\![\varphi_2]\!]$, and for all $0 \leq i < m$, $(p_i,q_i) \xrightarrow{\tau} (p_{i+1},q_{i+1})$ and $(p_i,q_i) \in [\![\varphi_1]\!]$. The reasoning is similar to the previous case. It is important to note that the transition $(p_m,q_m) \xrightarrow{a} (p_{m+1},q_{m+1})$ necessarily has a counterpart in the built sequence, since $a \in [\![\alpha_a]\!]_A$ and $\tau \notin [\![\alpha_a]\!]$. Therefore this transition cannot be an inert transition and the first case of Lemma 1 does not apply here.
- Case $\varphi = \langle\varphi_1?.\alpha_\tau\rangle\,@$. By definition, there exists an infinite sequence of states $(p_0,q_0) \xrightarrow{a_0} (p_1,q_1) \xrightarrow{a_1} \ldots (p_i,q_i) \xrightarrow{a_i} \ldots$, such that $p_0 = p, q_0 = q$, and for all $i > 0$, $(p_i,q_i) \in [\![\varphi_1]\!]$ and $a_i \in [\![\alpha_\tau]\!]$. The reasoning is similar to the previous case. It is important to note that an infinite sequence of τ-transitions (when $a_i = \tau$ for all $i > 0$) cannot collapse into an empty sequence, as guaranteed by divbranching bisimulation. □

Note that τ may belong to A_s. For instance, the formula $[\mathsf{true}^*.a]\,\langle\tau\rangle\,\mathsf{true}$ (which expresses that the target state of every transition labelled by a is also the source state of a transition labelled by τ) belongs to $L_\mu^{strong}(\{\tau\})$. If $\tau \in A_s$, following Theorem 2, every P_i not containing τ can be reduced for divbranching bisimulation, but in the case of a process not containing τ, divbranching bisimulation coincides with strong bisimulation. On the other hand, processes containing the strong action τ cannot be reduced for divbranching bisimulation. In the end, all τ-transitions of the system are preserved, as expected for the truth value of formulas containing strong modalities matching τ to be preserved.

Example 3 In Example 2, P does not contain a_2, the only strong action of the system. Thus, φ can be checked on P' $|[a_1]|$ Q (which is isomorphic to Q and has only 3 states) instead of P $|[a_1]|$ Q (6 states), while preserving its truth value.

4 Identifying Strong Actions in Formulas

4.1 Identifying strong actions in L_μ, ACTL, and CTL operators

In the general case, identifying a minimal set of strong actions is not easy, if at all feasible. One cannot reasonably assume that formulas are written in the obscure $L_\mu^{strong}(A_s)$ syntax (see Ex. 2) and that the strong modalities cannot be turned to weak ones. Instead, users shall continue to use "syntactic sugar" extensions of L_μ, with operators of e.g., CTL, ACTL, PDL, or PDL-Δ. In Lemmas 2 to 4 below we provide patterns that can be used to prove that a formula written using one of those operators belongs to a particular instance of $L_\mu^{strong}(A_s)$.

Lemma 2 (Modal μ-calculus) *Assume $\varphi_i \in L_\mu^{strong}(A_s)$ $(i = 0,1,2)$ and $[\![\alpha_s]\!]_A \subseteq A_s$. Then the following hold:*

1. $\langle\alpha_s\rangle\,\varphi_0 \in L_\mu^{strong}(A_s)$
2. $[\alpha_s]\,\varphi_0 \in L_\mu^{strong}(A_s)$

3. $\neg\varphi_0 \in L_\mu^{strong}(A_s)$
4. $\varphi_1 \vee \varphi_2 \in L_\mu^{strong}(A_s)$
5. $\varphi_1 \wedge \varphi_2 \in L_\mu^{strong}(A_s)$
6. $\varphi_1 \Rightarrow \varphi_2 \in L_\mu^{strong}(A_s)$

Proof Immediate from the definition of $L_\mu^{strong}(A_s)$ (see Def. 11) and the well-known identities $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$ and $\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$. □

Lemma 3 (Action Computation Tree Logic) *Assume that $\varphi_i \in L_\mu^{strong}(A_s)$ ($i = 1, 2$). Then the following hold:*

1. $\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2) \in L_\mu^{strong}(A_s)$
2. $\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2) \in L_\mu^{strong}(A_s)$
3. $\mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_2) \in L_\mu^{strong}(A_s)$
4. $\mathsf{A}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2) \in L_\mu^{strong}(A_s)$
5. $\mathsf{AG}_{\alpha_0}(\varphi_0) \in L_\mu^{strong}(A_s)$
6. $\mathsf{EF}_{\alpha_0}(\varphi_0) \in L_\mu^{strong}(A_s)$

Proof It was shown in [40] that the operators of ACTL\X can be encoded using the weak modalities of L_μ^{dbr}. Therefore, the only strong modalities that remain in these formulas are those occurring in φ_1 and φ_2, which by definition match only labels in A_s. Therefore, these formulas belong to $L_\mu^{strong}(A_s)$. □

Lemma 4 (Computation Tree Logic) *Assume that $\varphi_i \in L_\mu^{strong}(A_s)$ ($i = 0, 1, 2$) and $\tau \notin [\![\alpha_a]\!]_A$. Then the following hold:*

1. $\mathsf{E}(\varphi_1\,\mathsf{U}\,\varphi_2) \in L_\mu^{strong}(A_s)$
2. $\mathsf{A}(\varphi_1\,\mathsf{U}\,\varphi_2) \in L_\mu^{strong}(A_s)$
3. $\mathsf{AG}(\varphi_0) \in L_\mu^{strong}(A_s)$
4. $\mathsf{EF}(\varphi_0) \in L_\mu^{strong}(A_s)$
5. $\mathsf{AF}(\varphi_0) \in L_\mu^{strong}(A_s)$
6. $\mathsf{EG}(\varphi_0) \in L_\mu^{strong}(A_s)$
7. $\mathsf{E}(\varphi_1\,\mathsf{W}\,\varphi_2) \in L_\mu^{strong}(A_s)$
8. $\mathsf{A}(\varphi_1\,\mathsf{W}\,\varphi_2) \in L_\mu^{strong}(A_s)$
9. $\mathsf{A}([\alpha_a]\,\varphi_1\,\mathsf{U}\,\varphi_2) \in L_\mu^{strong}(A_s)$
10. $\mathsf{A}([\alpha_a]\,\varphi_1\,\mathsf{W}\,\varphi_2) \in L_\mu^{strong}(A_s)$
11. $\mathsf{AG}([\alpha_a]\,\varphi_0) \in L_\mu^{strong}(A_s)$
12. $\mathsf{EF}(\langle\alpha_a\rangle\,\varphi_0) \in L_\mu^{strong}(A_s)$
13. $\mathsf{AG}(\varphi_1 \vee [\alpha_a]\,\varphi_2) \in L_\mu^{strong}(A_s)$
14. $\mathsf{EF}(\varphi_1 \wedge \langle\alpha_a\rangle\,\varphi_2) \in L_\mu^{strong}(A_s)$

Proof We first prove the following identity, which will be used in the sequel:

$$\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2) = \mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_1 \wedge \langle\alpha_2\rangle\,\varphi_2) \qquad \text{if } \tau \notin [\![\alpha_2]\!]_A \qquad \text{(Actl1)}$$

This identity is easily proven by showing that the modal μ-calculus definitions of the left-hand-side and right-hand-side formulas are equivalent. First note that

since $\tau \notin [\![\alpha_2]\!]_A$, we have $[\![\alpha_2]\!]_A = [\![\alpha_2]\!]_A \setminus \{\tau\} = [\![\alpha_2 \wedge \neg\tau]\!]_A$ and thus $\langle\alpha_2\rangle\,\varphi = \langle\alpha_2 \wedge \neg\tau\rangle\,\varphi$ for any formula φ. We then have the following:

$$
\begin{array}{ll}
\mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}\,\varphi_1 \wedge \langle\alpha_2\rangle\,\varphi_2) & \\
\quad = \mu X.(\varphi_1 \wedge \langle\alpha_2\rangle\,\varphi_2) \vee (\varphi_1 \wedge \langle\alpha_1 \vee \tau\rangle\,X) & \text{by definition of } \mathsf{E}(_\,_\mathsf{U}\,_) \\
\quad = \mu X.\varphi_1 \wedge (\langle\alpha_2\rangle\,\varphi_2 \vee \langle\alpha_1 \vee \tau\rangle\,X) & \text{by factorization of } \varphi_1 \\
\quad = \mu X.\varphi_1 \wedge (\langle\alpha_2 \wedge \neg\tau\rangle\,\varphi_2 \vee \langle\alpha_1 \vee \tau\rangle\,X) & \text{from } \langle\alpha_2\rangle\,\varphi_2 = \langle\alpha_2 \wedge \neg\tau\rangle\,\varphi_2 \\
\quad = \mathsf{E}(\varphi_1\ {}_{\alpha_1}\mathsf{U}_{\alpha_2}\,\varphi_2) & \text{by definition of } \mathsf{E}(_\,_\mathsf{U}_{_}\,_)
\end{array}
$$

In addition to the equations (Ctl1) to (Ctl8) defined in Section 2.2, we will use the following identity, which is easily proven by showing that the modal μ-calculus definitions of the left-hand-side and right-hand-side formulas are equivalent:

$$\mathsf{A}(\varphi_1\,\mathsf{U}\,\varphi_2) = \neg\mathsf{E}(\neg\varphi_2\,\mathsf{W}\,\neg\varphi_1 \wedge \neg\varphi_2) \qquad \text{(Ctl9)}$$

We also use the following standard identity of the modal μ-calculus:

$$\neg[\alpha]\,\varphi_0 = \langle\alpha\rangle\,\neg\varphi_0 \qquad \text{(Mcl1)}$$

Items 1 to 8 are a direct consequence of Lemma 3 and the fact that these operators can be translated to ACTL$\setminus X$, as shown by equations (Ctl1) to (Ctl8). The proof of Items 9 to 14 follows:

9. $\mathsf{A}([\alpha_a]\,\varphi_1\,\mathsf{U}\,\varphi_2)$
   $= \neg\mathsf{E}(\neg\varphi_2\,\mathsf{W}\,\neg[\alpha_a]\,\varphi_1 \wedge \neg\varphi_2)$ By (Ctl9)
   $= \neg(\mathsf{E}(\neg\varphi_2\,\mathsf{U}\,\neg[\alpha_a]\,\varphi_1 \wedge \neg\varphi_2) \vee \mathsf{EG}(\neg\varphi_2))$ By (Ctl7)
   $= \neg(\mathsf{E}(\neg\varphi_2\,\mathsf{U}\,\langle\alpha_a\rangle\,\neg\varphi_1 \wedge \neg\varphi_2) \vee \mathsf{EG}(\neg\varphi_2))$ By (Mcl1)
   $= \neg(\mathsf{E}(\neg\varphi_2\ {}_{\mathsf{true}}\mathsf{U}\,\langle\alpha_a\rangle\,\neg\varphi_1 \wedge \neg\varphi_2) \vee \mathsf{EG}(\neg\varphi_2))$ By (Ctl1)
   $= \neg(\mathsf{E}(\neg\varphi_2\ {}_{\mathsf{true}}\mathsf{U}_{\alpha_a}\,\neg\varphi_1) \vee \mathsf{EG}(\neg\varphi_2))$ By (Actl1) and $\tau \notin [\![\alpha_a]\!]_A$
   which is in $L_\mu^{strong}(A_s)$ following Item 6 and Lemmas 2 and 3.
10. $\mathsf{A}([\alpha_a]\,\varphi_1\,\mathsf{W}\,\varphi_2)$
   $= \neg\mathsf{E}(\neg\varphi_2\,\mathsf{U}\,\neg[\alpha_a]\,\varphi_1 \wedge \neg\varphi_2)$ By (Ctl8)
   $= \neg\mathsf{E}(\neg\varphi_2\,\mathsf{U}\,\langle\alpha_a\rangle\,\neg\varphi_1 \wedge \neg\varphi_2)$ By (Mcl1)
   $= \neg\mathsf{E}(\neg\varphi_2\ {}_{\mathsf{true}}\mathsf{U}\,\langle\alpha_a\rangle\,\neg\varphi_1 \wedge \neg\varphi_2)$ By (Ctl1)
   $= \neg\mathsf{E}(\neg\varphi_2\ {}_{\mathsf{true}}\mathsf{U}_{\alpha_a}\,\neg\varphi_1)$ By (Actl1) and $\tau \notin [\![\alpha_a]\!]_A$
   which is in $L_\mu^{strong}(A_s)$ following Lemmas 2 and 3.
11. $\mathsf{AG}([\alpha_a]\,\varphi_0)$
   $= \neg\mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\neg[\alpha_a]\,\varphi_0)$ By (Ctl4)
   $= \neg\mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\langle\alpha_a\rangle\,\neg\varphi_0)$ By (Mcl1)
   $= \neg\mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}_{\alpha_a}\,\neg\varphi_0)$ By (Actl1) and $\tau \notin [\![\alpha_a]\!]_A$
   which is in $L_\mu^{strong}(A_s)$ following Lemmas 2 and 3. This is also a consequence of $\mathsf{AG}(\varphi_1 \vee [\alpha_a]\,\varphi_2) \in L_\mu^{strong}(A_s)$ (shown below), replacing φ_1 by false and φ_2 by φ_0.
12. $\mathsf{EF}(\langle\alpha_a\rangle\,\varphi_0)$
   $= \mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}\,\langle\alpha_a\rangle\,\varphi_0)$ By (Ctl3)
   $= \mathsf{E}(\mathsf{true}\ {}_{\mathsf{true}}\mathsf{U}_{\alpha_a}\,\varphi_0)$ By (Actl1) and $\tau \notin [\![\alpha_a]\!]_A$
   which is in $L_\mu^{strong}(A_s)$ following Lemmas 2 and 3. This is also a consequence of $\mathsf{EF}(\varphi_1 \wedge \langle\alpha_a\rangle\,\varphi_2) \in L_\mu^{strong}(A_s)$ (shown below), replacing φ_1 by true and φ_2 by φ_0.
13. $\mathsf{AG}(\varphi_1 \vee [\alpha_a]\,\varphi_2) = \neg\mathsf{EF}(\neg\varphi_1 \wedge \langle\alpha_a\rangle\,\neg\varphi_2)$, which is shown to belong to $L_\mu^{strong}(A_s)$ below.

14. $\mathsf{EF}(\varphi_1 \wedge \langle \alpha_a \rangle \varphi_2)$ is semantically equivalent to $\mathsf{EF}(\langle (\varphi_1?.\mathsf{true})^*.\varphi_1?.\alpha_a \rangle \varphi_2)$, which belongs to $L_\mu^{strong}(A_s)$. Indeed, if $\mathsf{EF}(\varphi_1 \wedge \langle \alpha_a \rangle \varphi_2)$ is true, then there is a reachable state satisfying $\varphi_1 \wedge \langle \alpha_a \rangle \varphi_2$, by definition of EF. This state also satisfies $\langle (\varphi_1?.\mathsf{true})^*.\varphi_1?.\alpha_a \rangle \varphi_2$, after an empty sequence of steps matching $(\varphi_1?.\mathsf{true})^*$. On the other hand, if a reachable state satisfies $\langle (\varphi_1?.\mathsf{true})^*.\varphi_1?.\alpha_a \rangle \varphi_2$, then a reachable state satisfies both φ_1 and $\langle \alpha_a \rangle \varphi_2$, by definition of the weak modality, i.e, the LTS satisfies $\mathsf{EF}(\varphi_1 \wedge \langle \alpha_a \rangle \varphi_2)$. □

More complex formula patterns can be easily shown to be in $L_\mu^{strong}(A_s)$ by extension of Lemma 4. For instance, if $\varphi_1, \ldots, \varphi_n \in L_\mu^{strong}(A_s)$, since we have $\mathsf{AG}(\varphi_1 \wedge \ldots \wedge \varphi_n) = \mathsf{AG}(\varphi_1) \wedge \ldots \wedge \mathsf{AG}(\varphi_n)$, then $\mathsf{AG}([\alpha_{a,1}] \varphi_1 \wedge \ldots \wedge [\alpha_{a,n}] \varphi_n) \in L_\mu^{strong}(A_s)$. Similarly, $\mathsf{EF}(\langle \alpha_{a,1} \rangle \varphi_1 \vee \ldots \vee \langle \alpha_{a,n} \rangle \varphi_n) \in L_\mu^{strong}(A_s)$.

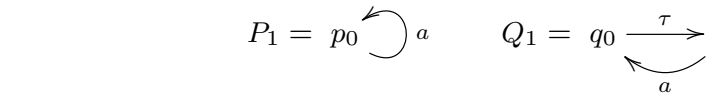
Example 4 Let a_1, a_2, and a_3 be visible actions and recall that A denotes the set of all actions of the system. Lemmas 2 to 4 allow the following to be shown (this is left as an exercise):

$$\begin{array}{rr}
\langle \mathsf{true}^*.a_1.(\neg a_2)^*.a_3 \rangle \mathsf{true} \in L_\mu^{strong}(\emptyset) & [\mathsf{true}] \mathsf{false} \in L_\mu^{strong}(A) \\
\mathsf{A}(\langle a_1 \rangle \mathsf{true}\, \mathsf{U}_{\neg a_2 a_3}\, \mathsf{true}) \in L_\mu^{strong}(\{a_1\}) & \mathsf{AG}([a_1] \mathsf{false}) \in L_\mu^{strong}(\emptyset) \\
\mathsf{A}([a_1] \mathsf{false}\, \mathsf{W}_{\neg a_2 a_3}\, \mathsf{true}) \in L_\mu^{strong}(\emptyset) & \langle a_1^*.a_2 \rangle \mathsf{true} \in L_\mu^{strong}(\{a_1, a_2\}) \\
\mathsf{E}(\mathsf{true}\, \mathsf{U}_{\mathsf{true}}\, \langle \tau \rangle \mathsf{true}) \in L_\mu^{strong}(\{\tau\}) & [a_1.a_2] \mathsf{false} \in L_\mu^{strong}(\{a_1, a_2\}) \\
\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true}) \in L_\mu^{strong}(\{a_1\}) & \mathsf{EF}(\langle a_1 \rangle \langle a_2 \rangle \mathsf{true}) \in L_\mu^{strong}(\{a_2\}) \\
\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true}) \in L_\mu^{strong}(\{a_2\}) & \mathsf{EF}(\langle \neg a_1 \rangle \mathsf{true}) \in L_\mu^{strong}(A \setminus \{a_1\})
\end{array}$$

4.2 Combinations of CTL operators and modalities that are not in $L_\mu^{strong}(A_s)$

We have given in Section 4.1 a set of formula patterns obtained by combination of CTL operators and L_μ modalities, which belong to $L_\mu^{strong}(A_s)$. In this section, we review other combinations, which do not belong to $L_\mu^{strong}(A_s)$. For each combination, we explain why. Below, we assume $\varphi_0, \varphi_1, \varphi_2 \in L_\mu^{strong}(A_s)$ and $a, a_1, a_2 \notin A_s$.

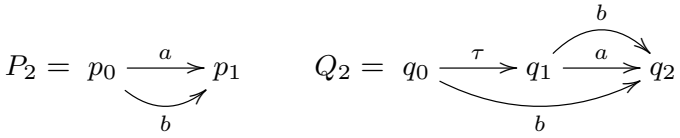
– $\mathsf{AG}(\langle a \rangle \varphi_0)$ and $\mathsf{EG}(\langle a \rangle \varphi_0)$ generally[7] do not belong to $L_\mu^{strong}(A_s)$, because these formulas require that a occurs immediately. They become false if a is delayed by a τ-transition. For instance below, $\mathsf{AG}(\langle a \rangle \mathsf{true})$ and $\mathsf{EG}(\langle a \rangle \mathsf{true})$ do not belong to $L_\mu^{strong}(\emptyset)$ because these formulas are satisfied by LTS P_1 but not by the divbranching bisimilar LTS Q_1 (where a is delayed by a τ-transition from q_0 to q_1). As a consequence, $\mathsf{EF}([a] \varphi_0) = \neg \mathsf{AG}(\langle a \rangle \neg \varphi_0)$ and $\mathsf{AF}([a] \varphi_0) = \neg \mathsf{EG}(\langle a \rangle \neg \varphi_0)$ generally do not belong to $L_\mu^{strong}(A_s)$ either.



– $\mathsf{A}(\langle a \rangle \varphi_1\ \mathsf{U}\ \varphi_2)$, $\mathsf{A}(\langle a \rangle \varphi_1\ \mathsf{W}\ \varphi_2)$, $\mathsf{E}(\langle a \rangle \varphi_1\ \mathsf{U}\ \varphi_2)$, and $\mathsf{E}(\langle a \rangle \varphi_1\ \mathsf{W}\ \varphi_2)$ generally do not belong to $L_\mu^{strong}(A_s)$, for similar reasons. In particular, $\mathsf{A}(\langle a \rangle \varphi_0\ \mathsf{W}\ \mathsf{false}) = \mathsf{AG}(\langle a \rangle \varphi_0)$ and $\mathsf{E}(\langle a \rangle \varphi_0\ \mathsf{W}\ \mathsf{false}) = \mathsf{EG}(\langle a \rangle \varphi_0)$, which have been shown to be outside $L_\mu^{strong}(A_s)$ in the previous item.

[7] Obviously, there are particular instances of such formulas that belong to $L_\mu^{strong}(A_s)$, e.g., $\mathsf{AG}(\langle a \rangle \mathsf{false})$ and $\mathsf{EG}(\langle a \rangle \mathsf{false})$, which are equivalent to false.

– $\mathsf{AF}(\langle a \rangle \varphi_0)$ generally does not belong to $L_\mu^{strong}(A_s)$ because even though $\langle a \rangle \varphi_0$ is satisfied in a state, $\mathsf{AF}(\langle a \rangle \varphi_0)$ is not necessarily satisfied in divbranching bisimilar states where a is delayed. For instance below, $\mathsf{AF}(\langle a \rangle \mathsf{true})$ does not belong to $L_\mu^{strong}(\emptyset)$ because this formula is satisfied by LTS P_2 but not by the divbranching bisimilar LTS Q_2 (where a is delayed by a τ-transition from q_0 to q_1). As a consequence, $\mathsf{EG}([a] \varphi_0) = \neg\mathsf{AF}(\langle a \rangle \neg\varphi_0)$ generally does not belong to $L_\mu^{strong}(A_s)$ either.



– $\mathsf{E}([a] \varphi_1 \ \mathsf{U} \ \varphi_2)$ and $\mathsf{E}([a] \varphi_1 \ \mathsf{W} \ \varphi_2)$ generally do not belong to $L_\mu^{strong}(A_s)$ for similar reasons. For instance, the formula $\mathsf{E}([a] \mathsf{false} \ \mathsf{U} \ [\mathsf{true}^*.b] \mathsf{false})$ does not belong to $L_\mu^{strong}(\emptyset)$, because this formula is false in P_2 but true in Q_2. Also, $\mathsf{E}([a] \varphi_0 \ \mathsf{W} \ \mathsf{false}) = \mathsf{EG}([a] \varphi_0)$, which has been shown to be outside $L_\mu^{strong}(A_s)$ in the previous item.
– At last, unlike $\mathsf{EF}(\varphi_1 \wedge \langle a \rangle \varphi_2)$ (Theorem 4, item 14), it should be clear that $\mathsf{EF}(\langle a_1 \rangle \varphi_1 \wedge \langle a_2 \rangle \varphi_2)$ generally does not belong to $L_\mu^{strong}(A_s)$ if neither a_1 nor a_2 belong to A_s, as formalized by Lemma 5 below.

Lemma 5 *There are action formulas α_{a1}, α_{a2} and state formulas $\varphi_1, \varphi_2 \in L_\mu^{strong}(A_s)$ such that $\mathsf{EF}(\langle \alpha_{a1} \rangle \varphi_1 \wedge \langle \alpha_{a2} \rangle \varphi_2) \notin L_\mu^{strong}(A_s)$.*

Proof Let $A_s = \emptyset$ and consider the formula $\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true})$, i.e., $\alpha_{a1} = a_1$, $\alpha_{a2} = a_2$, and $\varphi_1 = \varphi_2 = \mathsf{true}$. We have $\varphi_1 \in L_\mu^{strong}(\emptyset)$ and $\varphi_2 \in L_\mu^{strong}(\emptyset)$. We know that if $\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true}) \in L_\mu^{strong}(\emptyset)$, then it has the same truth value when evaluated on divbranching equivalent LTS. This is not the case as shown by the divbranching equivalent LTS P and Q depicted below. Indeed, $\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true})$ evaluates to true on P, whereas it evaluates to false on Q. Therefore, $\mathsf{EF}(\langle a_1 \rangle \mathsf{true} \wedge \langle a_2 \rangle \mathsf{true}) \notin L_\mu^{strong}(\emptyset)$.

□

So far, we did not mention explicitly the next operators of ACTL. We do it here for completeness:

– $\mathsf{AX}_{\alpha_a} \varphi$ is equivalent to $\langle \alpha_a \rangle \varphi \wedge [\alpha_a] \varphi \wedge [\neg\alpha_a] \mathsf{false}$. This operator does not belong to any fragment $L_\mu^{strong}(A_s)$ but $L_\mu^{strong}(\mathcal{A})$, i.e., it makes all actions strong.
– Similarly, $\mathsf{AX}_\tau \varphi$ is equivalent to $\langle \tau \rangle \varphi \wedge [\tau] \varphi \wedge [\neg\tau] \mathsf{false}$ and makes all actions strong.
– $\mathsf{EX}_{\alpha_a} \varphi$ is equivalent to $\langle \alpha_a \rangle \varphi$ and thus makes all actions satisfying α_a strong, unless it occurs in the immediate context of an EF operator.
– Finally, $\mathsf{EX}_\tau \varphi$ is equivalent to $\langle \tau \rangle \varphi$ and thus makes τ strong.

4.3 Identifying strong actions in PDL modalities

PDL modalities are a useful means to describe transition sequences in an LTS. We consider here a fragment of PDL named *test-free* PDL [26, Chap. 10], consisting of modalities on regular action sequences (without the testing operator $\varphi?$) and standard Boolean connectors $\wedge, \vee, \neg$, etc., which can be encoded in the full syntax of PDL defined in [13]. Using this fragment of PDL, one can describe sequences of transitions only in terms of their actions, and not of their intermediate states; however, this is often sufficient in the action-based setting for the usual classes of properties: safety (e.g., absence of undesirable sequences), liveness (e.g., presence of desirable sequences), and fairness (e.g., absence of unfair infinite sequences).

We propose in this section a method for identifying a reduced set of strong actions present in test-free PDL modalities. In Section 4.3.1, we show how to translate these modalities into an equivalent equational representation, which is more convenient for reasoning about strong actions. Then, in Section 4.3.2, we provide the lemmas for identifying the strong actions in a test-free PDL modality via its corresponding equational representation. Finally, in Section 4.3.3, we give a procedure to obtain a reduced (although not necessarily minimal) set of strong actions for a test-free PDL modality.

4.3.1 Translating PDL modalities to HMLR

In general, every test-free PDL modality $\langle \beta \rangle \, \varphi_0$ can be rewritten in several ways (by replacing the regular formula β with another, equivalent one). Each of these variants may have a different set of strong actions, which can be identified by translating the modality into an L_μ formula using the encoding given in [11]. For example, the modality $\langle b| \ (\neg a.(\neg a)^*.b) \rangle \, \varphi_0$ is translated into $\langle b \rangle \, \varphi_0 \vee \langle \neg a \rangle \, \mu X.(\langle b \rangle \, \varphi_0 \vee \langle \neg a \rangle \, X)$, which belongs to $L_\mu^{strong}(\llbracket \neg a \rrbracket \cup \{b\}) = L_\mu^{strong}(A \setminus \{a\})$ according to Definition 11. However, if the modality was written in the equivalent form $\langle (\neg a)^*.b \rangle \, \varphi_0$, it would belong to $L_\mu^{strong}(\emptyset)$, since it is a weak modality of L_μ^{dbr}.

Searching for an equivalent regular formula with a minimal set of strong actions is more convenient by representing regular formulas as finite automata, which have a canonical form (minimal deterministic automata). In the sequel, we use a modal logic counterpart of finite automata, namely HMLR (*Hennessy-Milner Logic with Recursion*) [35] specifications, which are equational variants of fixed point L_μ formulas. For our purpose, we need only a simple kind of HMLR specifications.

Definition 12 (HMLR specification) A HMLR specification D is a set of minimal fixed point equations of the following form:

$$D = \left\{ X_i = \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle \, X_{ij} \vee \varphi_i \right\}_{1 \leq i \leq n}$$

where $X_1, ..., X_n$ are propositional variables, α_{ij} are action formulas, and φ_i is an arbitrary state formula with free variables among $X_1, \ldots, X_n$. The semantics of D w.r.t. an LTS $P = (\Sigma, A, \longrightarrow, p_{init})$ is defined as the solution of variable X_1 in the minimal fixed point equation system, i.e., $\llbracket D \rrbracket_P = (\mu \Phi_P)_1$, where $(.)_1$ denotes the projection of an n-ary tuple on its first component, and the functional $\Phi_P : (2^\Sigma)^n \rightarrow (2^\Sigma)^n$ is defined as follows:

$$\Phi_P(U_1, ..., U_n) = (\llbracket \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle \, X_{ij} \vee \varphi_i \rrbracket_P [U_1/X_1, ..., U_n/X_n])_{1 \leq i \leq n}$$

where $(W_i)_{1 \leq i \leq n}$ denotes the tuple $(W_1, ..., W_n)$. D is called *guarded* if all the state formulas φ_i (for $1 \leq i \leq n$) are closed.

Intuitively, a PDL modality $\langle \beta \rangle \varphi_0$ expresses the existence of a transition sequence matching a regular formula β, and its corresponding HMLR specification provides an equivalent equational counterpart similar to Brzozowski's derivatives for regular expressions [3]. The translation from PDL to guarded HMLR [37] uses an intermediate form called PDLR (similar to HMLR, but having regular formulas instead of action formulas in modalities), and works by eliminating the regular operators contained in β using classical PDL identities [13].

The HMLR specifications of Definition 12 (and also the formalization in the remainder of this subsection) are suited for possibility modalities $\langle \beta \rangle \varphi_0$. Necessity modalities $[\beta] \varphi_0$ can be handled either by a dual formalization (replacing disjunctions by conjunctions and minimal fixed points by maximal ones), or simply by observing that negations on state formulas do not change the nature of strong actions in formulas, and therefore $[\beta] \varphi_0 = \neg \langle \beta \rangle \neg \varphi_0$ has the same set of strong actions as $\langle \beta \rangle \varphi_0$.

Example 5 Consider the following (overly complicated) PDL modality:

$$\psi = \langle \mathsf{true}^*.snd.(rec|\neg rec.rec|\neg rec.(\neg rec)^*.\neg rec.rec).ack \rangle \varphi_0$$

To translate ψ to HMLR, we convert it into an equivalent PDLR specification (left), that we refine by eliminating the . and the | operators (right):

$$\left\{ \begin{array}{l} X_1 = \langle \mathsf{true}^*.snd.(rec|\neg rec.rec| \\ \qquad \neg rec.(\neg rec)^*.\neg rec.rec).ack \rangle X_2 \\ X_2 = \varphi_0 \end{array} \right\} \quad \left\{ \begin{array}{l} X_1 = \langle \mathsf{true}^* \rangle X_3 \\ X_2 = \varphi_0 \\ X_3 = \langle snd \rangle X_4 \\ X_4 = \langle rec \rangle X_5 \vee \langle \neg rec \rangle X_6 \vee \\ \qquad \langle \neg rec \rangle X_7 \\ X_5 = \langle ack \rangle X_2 \\ X_6 = \langle rec \rangle X_5 \\ X_7 = \langle (\neg rec)^* \rangle X_8 \\ X_8 = \langle \neg rec \rangle X_9 \\ X_9 = \langle rec \rangle X_5 \end{array} \right\}$$

Then, we eliminate the * operators (left) and the unguarded occurrences of variables X_3 and X_8 in the equations defining X_1 and X_7 (right):

$$\left\{ \begin{array}{l} X_1 = X_3 \vee \langle \mathsf{true} \rangle X_1 \\ X_2 = \varphi_0 \\ X_3 = \langle snd \rangle X_4 \\ X_4 = \langle rec \rangle X_5 \vee \langle \neg rec \rangle X_6 \vee \\ \qquad \langle \neg rec \rangle X_7 \\ X_5 = \langle ack \rangle X_2 \\ X_6 = \langle rec \rangle X_5 \\ X_7 = X_8 \vee \langle \neg rec \rangle X_7 \\ X_8 = \langle \neg rec \rangle X_9 \\ X_9 = \langle rec \rangle X_5 \end{array} \right\} \quad \left\{ \begin{array}{l} X_1 = \langle snd \rangle X_4 \vee \langle \mathsf{true} \rangle X_1 \\ X_2 = \varphi_0 \\ X_3 = \langle snd \rangle X_4 \\ X_4 = \langle rec \rangle X_5 \vee \langle \neg rec \rangle X_6 \vee \\ \qquad \langle \neg rec \rangle X_7 \\ X_5 = \langle ack \rangle X_2 \\ X_6 = \langle rec \rangle X_5 \\ X_7 = \langle \neg rec \rangle X_9 \vee \langle \neg rec \rangle X_7 \\ X_8 = \langle \neg rec \rangle X_9 \\ X_9 = \langle rec \rangle X_5 \end{array} \right\}$$

Finally, we eliminate the unreachable equations defining X_3, X_8 and rename the variables, yielding the guarded HMLR specification below:

$$D = \left\{ \begin{array}{l} X_1 = \langle snd \rangle\, X_2 \vee \langle \mathsf{true} \rangle\, X_1 \\ X_2 = \langle rec \rangle\, X_3 \vee \langle \neg rec \rangle\, X_4 \vee \langle \neg rec \rangle\, X_5 \\ X_3 = \langle ack \rangle\, X_7 \\ X_4 = \langle rec \rangle\, X_3 \\ X_5 = \langle \neg rec \rangle\, X_6 \vee \langle \neg rec \rangle\, X_5 \\ X_6 = \langle rec \rangle\, X_3 \\ X_7 = \varphi_0 \end{array} \right\}$$

Note that equation defining X_7 is the only one containing a state formula (φ_0), which specifies the ending states of the regular sequence denoted by ψ.

Any HMLR specification D can be converted into an equivalent L_μ formula $\varphi(D)$ using the classical translation [7], recalled below.

Definition 13 (L_μ formula $\varphi(D)$ [7]) Let $D = \{X_i = rhs(X_i)\}_{1 \leq i \leq n}$ be a HMLR specification, where we note $rhs(X_i) = \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle\, X_{ij} \vee \varphi_i$ the formulas in the right-hand sides of equations. We also note $D_{1,i}$ the fragment of D consisting of its first i equations. The L_μ formula $\varphi(D) = \varphi(D_{1,n})$ associated to D is defined as follows:

$$\begin{array}{l} \varphi(D_{1,n}) = \varphi(D_{1,n-1}[\mu X_n.rhs(X_n)/X_n]) \\ \varphi(D_{1,1}) = \mu X_1.rhs(X_1) \end{array}$$

where $D_{1,n-1}[\mu X_n.rhs(X_n)/X_n]$ denotes the syntactic substitution of X_n by the formula $\mu X_n.rhs(X_n)$ in all equations of $D_{1,n-1}$.

It was shown in [7] that $[\![\varphi(D)]\!]_P = [\![D]\!]_P$ for any LTS P. Note that the translation in Definition 13 can be also applied to PDLR specifications.

Example 6 Consider the following HMLR specification $D_{1,3}$ obtained by translating the PDL modality $\langle \mathsf{true}^*.snd.(\neg rec)^*.rec \rangle\, \varphi_0$:

$$D_{1,3} = \left\{ \begin{array}{l} X_1 = \langle snd \rangle\, X_2 \vee \langle \mathsf{true} \rangle\, X_1 \\ X_2 = \langle rec \rangle\, X_3 \vee \langle \neg rec \rangle\, X_2 \\ X_3 = \varphi_0 \end{array} \right\}$$

We can retrieve its equivalent L_μ formula $\varphi(D_{1,3})$ by applying Definition 13 (we simplify $\mu X_3.\varphi_0$ into φ_0 because X_3 does not occur free in φ_0):

$$\begin{array}{rl} \varphi(D_{1,3}) & = \varphi(D_{1,2}[\mu X_3.\varphi_0/X_3]) \\ & = \varphi(D_{1,1}[\varphi_0/X_3][\mu X_2.(\langle rec \rangle\, \varphi_0 \vee \langle \neg rec \rangle\, X_2)/X_2]) \\ & = \varphi(\{X_1 = \langle snd \rangle\, \mu X_2.(\langle rec \rangle\, \varphi_0 \vee \langle \neg rec \rangle\, X_2) \vee \langle \mathsf{true} \rangle\, X_1\}) \\ & = \mu X_1.(\langle snd \rangle\, \mu X_2.(\langle rec \rangle\, \varphi_0 \vee \langle \neg rec \rangle\, X_2) \vee \langle \mathsf{true} \rangle\, X_1) \end{array}$$

The formula $\varphi(D_{1,3})$ contains two minimal fixed points X_1 and X_2, corresponding to the two $*$-operators in the initial PDL modality.

4.3.2 Identifying strong actions in HMLR

Given a guarded HMLR specification, we can identify its set of strong actions by inspecting its equations as follows.

Definition 14 (Strong actions for HMLR) Let $P = (\Sigma, A, \longrightarrow, p_{init})$ be an LTS and $D = \left\{ X_i = \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle X_{ij} \vee \varphi_i \right\}_{1 \leq i \leq n}$ be a guarded HMLR specification. The set of strong actions associated to D is defined as follows:

$$A_{str}(D) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{n_i} \left\{ [\![\alpha_{ij}]\!]_A \;\middle|\; \begin{array}{l} (\tau \in [\![\alpha_{ij}]\!]_A \wedge X_{ij} \neq X_i) \\ \vee \\ (\tau \notin [\![\alpha_{ij}]\!]_A \wedge \forall 1 \leq k \leq n_i . (\tau \notin [\![\alpha_{ik}]\!]_A \vee X_{ik} \neq X_i)) \end{array} \right\}.$$

Intuitively, the strong actions $A_{str}(D)$ are obtained by collecting, for each equation X_i of D, the actions matched by the strong modalities $\langle \alpha_{ij} \rangle X_{ij}$ present in the equation. Two cases are possible: (i) either α_{ij} captures τ without being a self-loop on variable X_i, or (ii) α_{ij} captures only visible actions, and there is no other action formula α_{ik} in the equation capturing τ and being a self-loop on variable X_i.

Note that Definition 14 identifies strong actions by inspecting individual equations only, and does not consider the cycles of τ-transitions induced by $\langle \tau \rangle X_{ij}$ modalities occurring in mutual recursive equations. Therefore, several semantically equivalent, but syntactically different HMLR specifications may have different sets of strong actions according to Definition 14. To overcome this issue, one can reason in terms of the canonical representation of a HMLR specification (determinized and minimized), as it is discussed in Section 4.3.3.

Lemma 6 (Test-free PDL modalities) *Let $\langle \beta \rangle \varphi_0$ be a test-free PDL modality, D its corresponding guarded HMLR specification, and assume that $\varphi_0 \in L_\mu^{strong}(A_s)$. The following holds:*

$$\langle \beta \rangle \varphi_0 \in L_\mu^{strong}(A_s \cup A_{str}(D)).$$

Proof Let $D = \left\{ X_i = \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle X_{ij} \vee \varphi_i \right\}_{1 \leq i \leq n}$ be the guarded HMLR specification obtained by translating $\langle \beta \rangle \varphi_0$. We prove the statement by constructing a formula of $L_\mu^{strong}(A_s \cup A_{str}(D))$ equivalent to D. To do so, we first transform D into D' containing only strong modalities on action formulas α such that $[\![\alpha]\!]_A \subseteq A_{str}(D)$, and then we show that $\varphi(D') \in L_\mu^{strong}(A_s \cup A_{str}(D))$.

Consider an equation $X_i = \bigvee_{j=1}^{n_i} \langle \alpha_{ij} \rangle X_{ij} \vee \varphi_i$ for some $1 \leq i \leq n$. We can rewrite its right-hand side as follows:

$$X_i = \bigvee_{j \in J} \langle \alpha_{ij} \rangle X_i \vee \bigvee_{k \in K} \langle \alpha_{\tau ik} \rangle X_{ik} \vee \bigvee_{l \in L} \langle \alpha_{ail} \rangle X_{il} \vee \varphi_i$$

where the index sets J, K, L form a partition of $\{1, ..., n_i\}$, $\tau \in [\![\alpha_{\tau ik}]\!]_A$ and $X_{ik} \neq X_i$ for all $k \in K$, $\tau \notin [\![\alpha_{ail}]\!]_A$ and $X_{il} \neq X_i$ for all $l \in L$. The three disjuncts indexed by j, k, l represent the "self-loops" leading to X_i, the "potentially invisible successors" of X_i, and the "visible successors" of X_i, respectively. Two cases are possible.
(i) There exists a potentially invisible self-loop, i.e., $\tau \in [\![\alpha_{ij}]\!]_A$ for some $j \in J$. Using the HML identity $\langle \alpha_1 \rangle \varphi \vee \langle \alpha_2 \rangle \varphi = \langle \alpha_1 \vee \alpha_2 \rangle \varphi$, the equation becomes:

$$X_i = \Big\langle \bigvee_{j \in J} \alpha_{ij} \Big\rangle X_i \vee \bigvee_{k \in K} \langle \alpha_{\tau ik} \rangle X_{ik} \vee \bigvee_{l \in L} \langle \alpha_{ail} \rangle X_{il} \vee \varphi_i$$

Finally, using the PDL identities $\langle \alpha^* \rangle \varphi = \varphi \vee \langle \alpha \rangle \langle \alpha^* \rangle \varphi$, $\langle \alpha_1 \rangle \langle \alpha_2 \rangle \varphi = \langle \alpha_1.\alpha_2 \rangle \varphi$ and the distributivity of $\vee$ over $\langle\ \rangle$, the equation reaches the form:

$$X_i = \bigvee_{k \in K} \overbrace{\Big\langle \big(\bigvee_{j \in J} \alpha_{ij} \big)^* \Big\rangle}^{\text{ultra weak}} \overbrace{\big\langle \alpha_{\tau ik} \big\rangle X_{ik}}^{\text{strong}} \vee \bigvee_{l \in L} \overbrace{\Big\langle \big(\bigvee_{j \in J} \alpha_{ij} \big)^* . \alpha_{ail} \Big\rangle X_{il}}^{\text{weak}} \vee \overbrace{\Big\langle \big(\bigvee_{j \in J} \alpha_{ij} \big)^* \Big\rangle \varphi_i}^{\text{ultra weak}}$$

Since $\tau \in [\![\alpha_{ij}]\!]_A$ for some $j \in J$, also $\tau \in [\![\bigvee_{j \in J} \alpha_{ij}]\!]_A$, and therefore the modalities above belong to $L_\mu^{strong}(\bigcup_{k \in K} [\![\alpha_{\tau ik}]\!]_A)$, where $[\![\alpha_{\tau ik}]\!]_A$ are precisely the strong actions identified in Definition 14 and are therefore included in $A_{str}(D)$.
(ii) There is no potentially invisible self-loop, i.e., $\tau \notin [\![\alpha_{ij}]\!]_A$ for any $j \in J$. In this case, all the modalities present in the equation are considered as strong, according to Definition 14, and the equation is left unchanged.

After modifying all equations of D that match case (i), we obtain a HMLR specification D' whose strong modalities contain the action formulas satisfied by the actions in $A_{str}(D)$. It remains to show that $\varphi(D') \in L_\mu^{strong}(A_s \cup A_{str}(D))$. By hypothesis, we know that $\varphi_0 \in L_\mu^{strong}(A_s)$. The translation from PDL to guarded HMLR [37] propagates φ_0 in all equations, i.e., $\varphi_i = \varphi_0$ for all $1 \leq i \leq n$. Thus, the right-hand side formulas of all equations in D' belong to $L_\mu^{strong}(A_s \cup A_{str}(D))$. By applying the translation in Definition 13, which does not change the structure of the formulas in the right-hand sides but only introduces fixed point operators, we obtain a formula $\varphi(D') \in L_\mu^{strong}(A_s \cup A_{str}(D))$. □

Example 7 Consider the guarded HMLR specification D in Example 5. We update the equations of D according to Lemma 6 and obtain the specification D' below:

$$D' = \left\{ \begin{array}{l} X_1 = \langle \mathsf{true}^*.snd \rangle\, X_2 \\ X_2 = \langle rec \rangle\, X_3 \vee \langle \neg rec \rangle\, X_4 \vee \langle \neg rec \rangle\, X_5 \\ X_3 = \langle ack \rangle\, X_7 \\ X_4 = \langle rec \rangle\, X_3 \\ X_5 = \langle (\neg rec)^* \rangle \langle \neg rec \rangle\, X_6 \\ X_6 = \langle rec \rangle\, X_3 \\ X_7 = \varphi_0 \end{array} \right\}$$

We see that the strong actions of D' are $[\![rec]\!]_A$ (imposed by the equations defining X_2, X_4, and X_6), $[\![\neg rec]\!]_A$ (imposed by the equations defining X_2 and X_5), and $[\![ack]\!]_A$ (imposed by the equation defining X_3), and thus $A_{str}(D') = [\![rec]\!]_A \cup [\![\neg rec]\!]_A \cup [\![ack]\!]_A = A$, which is equal to $A_{str}(D)$ according to Definition 14.

Test-free PDL modalities without nesting of $*$-operators can be handled by applying a simpler criterion, stated below.

Lemma 7 (Simple PDL modalities) *Let A_s be a set of strong actions and assume that $\varphi_0 \in L_\mu^{strong}(A_s)$, $\tau \in [\![\alpha_\tau]\!]_A$, and $\tau \notin [\![\alpha_a]\!]_A$. Then the following hold:*

1. $\langle \alpha_\tau^*.\alpha_a \rangle\, \varphi_0 \in L_\mu^{strong}(A_s)$
2. $[\alpha_\tau^*.\alpha_a]\, \varphi_0 \in L_\mu^{strong}(A_s)$
3. $\langle \alpha_\tau^* \rangle\, \varphi_0 \in L_\mu^{strong}(A_s)$
4. $[\alpha_\tau^*]\, \varphi_0 \in L_\mu^{strong}(A_s)$
5. $\langle \alpha_\tau \rangle\, @ \in L_\mu^{strong}(A_s)$

6. $[\alpha_\tau] \dashv \in L_\mu^{strong}(A_s)$

Proof It was shown in [40] that these weak PDL-Δ modalities can be encoded in L_μ^{dbr} when $\varphi_0 \in L_\mu^{dbr}$. The proposed encoding also works when $\varphi_0 \in L_\mu^{strong}(A_s)$. As this encoding does not add more strong modalities than already present in φ_0, these properties belong to $L_\mu^{strong}(A_s)$. □

4.3.3 Reducing the number of strong actions

Lemma 6 identifies the strong actions in a guarded HMLR specification D corresponding to a PDL modality $\langle\beta\rangle\,\varphi_0$. However, in general there are several such HMLR specifications D (and even an infinite number of them if D contains cyclic dependencies between variables), each one with its own set $A_{str}(D)$. Identifying, among all these HMLR specifications, the one having the minimal set of strong actions is a non-trivial problem. We sketch below a procedure to obtain a D with a reduced (not necessarily minimal) number of strong actions.

All guarded HMLR specifications D corresponding to a PDL modality $\langle\beta\rangle\,\varphi_0$ represent the regular language denoted by β. Every such specification D can be brought to a canonical form by determinizing and minimizing it similarly to a finite automaton. For the sake of simplicity, we use a *syntactic* notion of determinization, by considering each action formula in a modality of D as a literal. Thus, a guarded HMLR specification is deterministic if, for each of its equations $X_i = \bigvee_{j=1}^{n_i} \langle\alpha_{ij}\rangle\, X_{ij} \vee \varphi_i$, all action formulas α_{ij} are syntactically different[8]. The determinization of a guarded HMLR specification D can be done using the classical subset construction, as shown in [37].

Example 8 Consider the guarded HMLR specification D of Example 5, whose equations defining X_2 and X_5 are nondeterministic because each one contains two modalities on $\neg rec$. By applying the subset construction, we obtain (sets of propositional variables are represented by sets of indexes):

$$\left\{\begin{array}{rl} X_{\{1\}} &= \langle snd\rangle\, X_{\{2\}} \vee \langle \mathsf{true}\rangle\, X_{\{1\}} \\ X_{\{2\}} &= \langle rec\rangle\, X_{\{3\}} \vee \langle \neg rec\rangle\, X_{\{4,5\}} \\ X_{\{3\}} &= \langle ack\rangle\, X_{\{7\}} \\ X_{\{4,5\}} &= \langle rec\rangle\, X_{\{3\}} \vee \langle \neg rec\rangle\, X_{\{5,6\}} \\ X_{\{5,6\}} &= \langle rec\rangle\, X_{\{3\}} \vee \langle \neg rec\rangle\, X_{\{5,6\}} \\ X_{\{7\}} &= \varphi_0 \end{array}\right\}$$

After eliminating the equations defining $X_{\{4,5\}}$ and $X_{\{5,6\}}$ (both equal to $X_{\{2\}}$) and renaming the variables, we obtain a deterministic HMLR specification (left), which can be further refined by identifying the weak modalities in the equations defining X_1 and X_2 (right):

$$\left\{\begin{array}{l} X_1 = \langle snd\rangle\, X_2 \vee \langle \mathsf{true}\rangle\, X_1 \\ X_2 = \langle rec\rangle\, X_3 \vee \langle \neg rec\rangle\, X_2 \\ X_3 = \langle ack\rangle\, X_4 \\ X_4 = \varphi_0 \end{array}\right\} \qquad \left\{\begin{array}{l} X_1 = \langle \mathsf{true}^*.snd\rangle\, X_2 \\ X_2 = \langle (\neg rec)^*.rec\rangle\, X_3 \\ X_3 = \langle ack\rangle\, X_4 \\ X_4 = \varphi_0 \end{array}\right\}$$

[8] Any action formula can be reduced to one of two forms: either a disjunction $a_1 \vee ... \vee a_n$, or a conjunction of negations $\neg b_1 \wedge ... \wedge \neg b_m$. Two action formulas are syntactically equal if they are of the same form and contain the same set of actions.

This last PDLR specification has the strong actions $\{ack\}$ (induced by the strong modality in the equation defining X_3), and this is the set of strong actions in the initial PDL modality ψ of Example 5.

Determinization can simplify the HMLR specifications considerably and uncover that some action formulas initially appearing as strong (like the actions rec and $\neg rec$ in the HMLR specification of Example 7) are in fact weak. Reasoning on the determinized version $det(D)$ instead of D allows to spot all the potentially invisible self-loops: if such a loop exists somewhere in D, then it must also exist in $det(D)$, since both specifications denote the same regular language.

However, syntactic determinization alone followed by the application of Lemma 6 may artificially increase the set of strong actions, as shown by the example below.

Example 9 Consider the PDL modality $\langle \mathsf{nil} | \neg a.(\neg a)^* | a \rangle \, \varphi_0$, where $\mathsf{nil} = \mathsf{false}^*$ denotes the empty sequence. Its deterministic HMLR specification is:

$$D = \left\{ \begin{array}{l} X_1 = \langle \neg a \rangle \, X_2 \vee \langle a \rangle \, X_3 \vee \varphi_0 \\ X_2 = \langle \neg a \rangle \, X_2 \vee \varphi_0 \\ X_3 = \varphi_0 \end{array} \right\}$$

Lemma 6 identifies $\neg a$ as weak (because of the self-loop in the equation defining X_2), but also as strong (because of the equation defining X_1). Since a is also strong (because of the $\langle a \rangle \, X_3$ modality), this yields $A_{str}(D) = [\![\neg a]\!]_A \cup [\![a]\!]_A = A$.

But D can be simplified by observing that $rhs(X_2)$ is included in $rhs(X_1)$, and therefore $rhs(X_2)$ can be replaced by X_2 in $rhs(X_1)$. This produces an unguarded HMLR specification (left), subsequently transformed in PDLR (right):

$$\left\{ \begin{array}{l} X_1 = X_2 \vee \langle a \rangle \, X_3 \\ X_2 = \langle \neg a \rangle \, X_2 \vee \varphi_0 \\ X_3 = \varphi_0 \end{array} \right\} \quad \left\{ \begin{array}{l} X_1 = X_2 \vee \langle a \rangle \, X_3 \\ X_2 = \langle (\neg a)^* \rangle \, \varphi_0 \\ X_3 = \varphi_0 \end{array} \right\}$$

Using Definition 13, we translate this last PDLR specification into the L_μ formula $\langle (\neg a)^* \rangle \, \varphi_0 \vee \langle a \rangle \, \varphi_0$, in which the only strong action is a.

To preserve as much as possible the weak modalities identified in an equation $X_i = rhs(X_i)$, one must replace all occurrences of $rhs(X_i)$ by X_i in all the other $rhs(X_j)$; otherwise, as shown in Example 9, the modalities contained in $rhs(X_i)$ may be considered as strong in another equation $X_j = rhs(X_j)$ which includes $rhs(X_i)$. For a minimal deterministic HMLR specification D, the inclusion relation between right-hand sides of equations is acyclic (two equations $X_i = rhs(X_i)$ and $X_j = rhs(X_j)$ with mutual inclusion between $rhs(X_i)$ and $rhs(X_j)$ would imply $X_i = X_j$, and therefore D would not be minimal). Thus, one can carry out the substitutions following an inverse topological order of the inclusion relation, leading to a (unique) unguarded PDLR specification in which all weak modalities identified were kept as much as possible.

This syntactic procedure considers action formulas as literals, and is therefore less precise than a semantic one, in which action formulas denote subsets of actions in A. We leave the investigation of a semantic procedure for future work.

4.4 Minimal sets of strong actions

In this section, we present two results about minimal sets of strong actions. First, we show that in general, there may be several minimal sets of strong actions associated to an L_μ formula. Second, we show that computing a minimal set of strong actions, if feasible, is at least as hard as checking the satisfiability of an L_μ formula.

Theorem 3 *There is not a unique minimal set A_s such that $\varphi \in L_\mu^{strong}(A_s)$.*

Proof Following Lemma 4 (item 14), $\mathsf{EF}(\langle a_1\rangle\,\mathsf{true} \wedge \langle a_2\rangle\,\mathsf{true})$ belongs to both fragments $L_\mu^{strong}(\{a_1\})$ and $L_\mu^{strong}(\{a_2\})$ as it is equivalent both to the formula $\langle(\mathsf{true}?.\mathsf{true})^*\rangle\,(\langle(\langle a_1\rangle\,\mathsf{true}?.\mathsf{true})^*.\langle a_1\rangle\,\mathsf{true}?.a_2\rangle\,\mathsf{true})$ and to the symmetric formula $\langle(\mathsf{true}?.\mathsf{true})^*\rangle\,(\langle(\langle a_2\rangle\,\mathsf{true}?.\mathsf{true})^*.\langle a_2\rangle\,\mathsf{true}?.a_1\rangle\,\mathsf{true})$ where a_1 and a_2 are swapped. Yet, from Lemma 5 (page 18), we know that it is not in $L_\mu^{strong}(\emptyset)$. Thus, $\{a_1\}$ and $\{a_2\}$ are both minimal sets of strong actions for the formula $\mathsf{EF}(\langle a_1\rangle\,\mathsf{true}\wedge\langle a_2\rangle\,\mathsf{true})$. □

Theorem 3 illustrates that when the formula checks for the presence or absence of multiple actions in the same state (such as a_1 and a_2 in the formula $\mathsf{EF}(\langle a_1\rangle\,\mathsf{true}\wedge\langle a_2\rangle\,\mathsf{true})$ used to prove Theorem 3), then not all such actions can be considered as weak. This seems to break the intuition that weak actions are those actions that can always be delayed, because one might argue that both a_1 and a_2 can always be delayed in this example. In fact, this is not true, as a_1 cannot be delayed whenever a_2 occurs (and conversely) without having an effect on the truth value of the formula: at least once, a_1 and a_2 have to occur *simultaneously*. In this case, one can choose whether the delayable action is a_1 or a_2, the occurrence of the other action then being subject to the occurrence of the first. For instance, if a_1 is chosen as always delayable (i.e., if a_1 is weak), then at least once in a state where a_1 occurs, a_2 has to occur immediately (i.e., a_2 is strong).

Besides the fact that minimal sets of strong actions are generally not unique, computing such sets also requires to address the potential presence of so-called *vacuous* modalities, where vacuity is defined as follows.

Definition 15 (Vacuous subformula) Let φ be an L_μ formula and φ' be a subformula of φ. The subformula φ' is vacuous in φ if it occurs in a sub-formula of φ whose truth value is constant (either false or true on all models).

Example 10 The subformulas $\langle\alpha\rangle\,X$ and X are vacuous in $\mu X.\langle\alpha\rangle\,X$ (which is equivalent to false). The subformula φ_0 is vacuous in $\mu X.X \wedge \varphi_0$ (which is equivalent to false). The subformula $\langle\mathsf{false}\rangle\,\varphi_0$ (which is equivalent to false) is vacuous in any formula containing it, and φ_0 is vacuous in $\langle\mathsf{false}\rangle\,\varphi_0$. The subformulas false and $\mathsf{false} \wedge \varphi_1$ are vacuous in any formula containing them, and φ_1 is vacuous in $\langle\alpha_0\rangle\,\varphi_0 \vee (\mathsf{false} \wedge \varphi_1)$.

Detecting vacuity is an interesting issue on its own, as users generally do not intend to write formulas containing vacuous subformulas, except in the form of the constants true and false. We consider here vacuity as a tool to give a lower-bound on the complexity of the quest for minimality. Obviously, the actions matched by a vacuous strong modality do not have to be considered as strong (unless they are

matched by another, non-vacuous strong modality), because there exists an equivalent formula (the one with the constant subformula replaced by its truth value), which does not contain this modality. Theorem 4 below shows that computing a minimal set of strong actions is at least as hard as checking L_μ-(un)satisfiability, by reasoning on vacuous modalities.

Theorem 4 *If there exists a procedure for extracting a minimal set of strong actions from an arbitrary L_μ formula, then this algorithm can also be used to determine the (un)satisfiability of any L_μ formula φ.*

Proof Let φ be an L_μ formula and let a be an action that is not occurring in the action formulas of φ. Without loss of generality, we assume that no action formula α occurring in φ matches a. Otherwise, we can consider a new formula φ' obtained from φ by replacing every action formula α by $\alpha \wedge \neg a$ (e.g., replacing true by $\neg a$), such that φ' has the same satisfiability as φ. Indeed, $\alpha \wedge \neg a$ and α either match the same finite sets of actions (which do not contain a), or match infinite sets (which differ only by the presence of a). Therefore, there exists a model satisfying φ if and only if there exists a model (obtained by renaming the a-transitions), which satisfies φ'.

We show that checking the unsatisfiability of φ (meaning that φ is equivalent to false) can be reduced to computing a minimal set of strong actions for the formula $\varphi \wedge \langle a \rangle \mathsf{true}$. The formula φ is unsatisfiable if and only if this minimal set of strong actions does not contain a. Indeed, since in general the modality $\langle a \rangle \mathsf{true}$ cannot be weak in $\varphi \wedge \langle a \rangle \mathsf{true}$ (clearly, if a does not occur immediately then $\varphi \wedge \langle a \rangle \mathsf{true}$ does not hold), the procedure should return a set containing a unless $\langle a \rangle \mathsf{true}$ is vacuous in $\varphi \wedge \langle a \rangle \mathsf{true}$. Since the truth value of $\langle a \rangle \mathsf{true}$ itself is not constant, its vacuity can only be due to the fact that $\varphi \wedge \langle a \rangle \mathsf{true}$ is either constantly true or constantly false. It obviously cannot be constantly true, as its value depends on the presence or not of a transition labelled by a. It is constantly false if and only if φ and $\langle a \rangle \mathsf{true}$ are contradictory, i.e., $\varphi \Rightarrow \neg \langle a \rangle \mathsf{true}$ (or, equivalently, $\langle a \rangle \mathsf{true} \Rightarrow \neg \varphi$). However, this is possible if and only if φ is false (since no action formula of φ matches a, it is not possible that φ explicitly forbids the existence of a transition labelled by a), i.e., if and only if φ is unsatisfiable. □

Corollary 1 *Computing a minimal set of strong actions from an arbitrary L_μ formula is EXPTIME-hard.*

Proof This is a direct consequence of Theorem 4 and the proof in [51] that the theoretical worst-case complexity of L_μ-satisfiability is EXPTIME. □

Even though formulas are usually small, the complexity of computing minimal sets of strong actions is rather high. We believe that it is better to invest in algorithms that do not try to compute a minimal set of strong actions, but one that is close enough to minimality and still allows significant state space reductions. For instance, we would be happy with an algorithm whose resulting set of strong actions is minimal under the condition that the formula does not contain any vacuous modality, while not guaranteeing minimality in the presence of vacuity (although this does not hold on the algorithm for PDL presented in Section 4.3). This is an ambitious enough quest, as eliminating (or at least reducing the number of) vacuous modalities can be handled as an orthogonal problem. Building such an algorithm is still far from trivial and left for future work.

Regarding ACTL\X and CTL\X, we showed in Section 4.1 that all operators of those two logics (thus not considering fixed points and L_μ modalities) are part of $L_\mu^{strong}(A_s)$ for any A_s, in particular $L_\mu^{strong}(\emptyset)$. Therefore, for a formula φ expressed using a combination of those operators and L_μ modalities, the set of strong actions is included in the actions matched by the L_μ modalities of φ. It is therefore safe to take as set A_s exactly the set of actions matched by the L_μ modalities occurring in φ. If at least one process does not contain one of such actions, this safe approach enables the state space to be reduced more than with the mono-bisimulation approach. One can do better by excluding the actions matched only by modalities occurring in specific contexts, as expressed by Lemmas 3 and 4.

5 Applications

We consider two examples to illustrate our new verification approach combining strong and divbranching bisimulation and show how it can reduce both time and memory usage when associated to the smart reduction heuristic. In both examples, the aim is to perform a set of verification tasks, each consisting in checking a formula φ on a system of parallel processes $P_1 \mid\mid \ldots \mid\mid P_n$. Since our approach can only improve the verification of formulas containing both strong and weak modalities, we consider only the pairs of formulas and systems such that the formula is part of $L_\mu^{strong}(A_s)$ for some minimal A_s that is not empty and that is strictly included in the set of visible actions of the system.[9] For each verification task, we compare the largest intermediate LTS size, the verification time, and the memory peak obtained using the following two approaches:

Mono-bisimulation approach: φ is verified on **hide** $H(\varphi)$ **in** $(P_1 \mid\mid \ldots \mid\mid P_n)$ (where $H(\varphi)$ is the maximal hiding set mentioned in Sect. 2.3) reduced compositionally for strong bisimulation (since φ is not in L_μ^{dbr}) using the smart reduction heuristic.

Refined approach combining bisimulations: The set $\{P_1, \ldots, P_n\}$ is partitioned in two groups $\mathcal{P}_s$ and $\mathcal{P}_w$ such that $P_i \in \mathcal{P}_s$ if it contains actions in A_s and $P_i \in \mathcal{P}_w$ otherwise. $P_1 \mid\mid \ldots \mid\mid P_n$ is then rewritten in the equivalent form $(||_{P_i \in \mathcal{P}_s} P_i) \mid\mid (||_{P_j \in \mathcal{P}_w} P_j)$. The set A_I of actions on which at least one process of $\mathcal{P}_s$ and one process of $\mathcal{P}_w$ synchronize (*inter-group* synchronization) is then identified. Using the smart reduction heuristic, **hide** $H(\varphi) \setminus A_I$ **in** $||_{P_i \in \mathcal{P}_s} P_i$ (corresponding to the processes containing strong actions) is reduced compositionally for strong bisimulation, leading to a first LTS P_s, and **hide** $H(\varphi) \setminus A_I$ **in** $||_{P_j \in \mathcal{P}_w} P_j$ (corresponding to the processes containing only weak actions) is reduced compositionally for divbranching bisimulation, leading to a second LTS P_w. Finally, φ is verified on **hide** $H(\varphi) \cap A_I$ **in** $(P_s \mid[A_I]\mid P_w)$ reduced for strong bisimulation.

All experiments were done on a 3GHz/12GB RAM/8-core Intel Xeon computer running Linux, using the specification languages and 32-bit versions of tools provided in the CADP toolbox [16] version 2019-a "Pisa".

[9] Otherwise, our approach coincides with the mono-bisimulation approach of [40]. In all the examples addressed in this section, there is always a unique minimal set A_s, whose identification is made easy using Lemmas 2 to 4.

Nr.	Property
08	$[\mathsf{true}^*.a_1.\boxed{a_2}]\ \mathsf{false}$
09	$[\mathsf{true}^*.a_1.\boxed{a_2}.((\boxed{a_3}.(\neg a_4)^*.a_5)\|(\boxed{a_6}.(\neg a_7)^*.a_8))]\ \mathsf{false}$
14	$[\mathsf{true}^*.a_1.\boxed{a_2}.(\neg a_3)^*.a_4.\boxed{a_5}]\ \mathsf{false}$
16	$[(\neg a_1)^*.a_2.(\neg a_3)^*.a_4]\ \langle((\neg a_5)^*.a_6.\boxed{a_7}).((\neg a_5)^*.a_6.\boxed{a_7})\rangle\ \mathsf{true}$
17	Same shape as property Nr. 16

Table 1 TFTP properties (strong action formulas are highlighted)

5.1 Trivial File Transfer Protocol

The TFTP (*Trivial File Transfer Protocol*) case-study[10] addresses the verification of an avionic communication protocol between a plane and the ground [18]. It comprises two instances (A and B) of a process named TFTP, connected through a FIFO buffer. Since the state space is very large in the general case, the authors defined five scenarios named A, B, C, D, and E, depending on whether each instance may write and/or read a file. The system corresponding to each scenario is a parallel composition of eight processes. The requirements consist of 29 properties parameterized by the identity of a TFTP instance, defined in MCL [38] (an implementation of the alternation-free modal μ-calculus including PDL-Δ modalities and macro definitions enabling the construction of libraries of operators), 24 of which belong to L_μ^{dbr}. The remaining five, namely properties 08, 09, 14, 16, and 17, contain both weak and strong modalities. The shape of these properties is described in Table 1, where we do not provide the details of the action formulas, but instead denote them by letters $a_1, a_2, \ldots$, where $\tau \notin [\![a_i]\!]_A$ for all i. Strong action formulas are highlighted and one shows easily that the other are weak using the patterns of Lemma 7. This identification of strong action formulas was done by hand.

We consider 31 among a potential of 50 verification tasks (five properties, five scenarios, and two instances) as some properties are not relevant to every TFTP instance and scenario (e.g., in a scenario where one TFTP instance only receives messages, checking a property concerning a message emission is irrelevant). All 31 verification tasks return true and the strong actions occur in only three (although not the same three) out of the eight parallel processes.

Figure 2 shows that the refined approach always reduces LTS size (for both intermediate and final LTS), memory and time following similar curves, up to a factor 7 (the vertical axis is on a logarithmic scale). Detailed data is found in Table 2, where parameters P, S, and I correspond respectively to the property number, the scenario, and the TFTP instance. LTS sizes are given in kilostates, memory in megabytes, and time in seconds. Time does not include LTS generation of the component processes from their LNT specification, which takes only a few seconds and is common to both approaches. The column "Ratio Strong/Combined" shows the gain obtained by applying the combined bisimulations approach rather than the mono-bisimulation approach with respect to largest LTS size, memory peak, and time. In these experiments, time is dominated by the last step of generation and minimization, whereas memory usage is dominated by minimization.

[10] Specification available at `ftp://ftp.inrialpes.fr/pub/vasy/demos/demo_05`

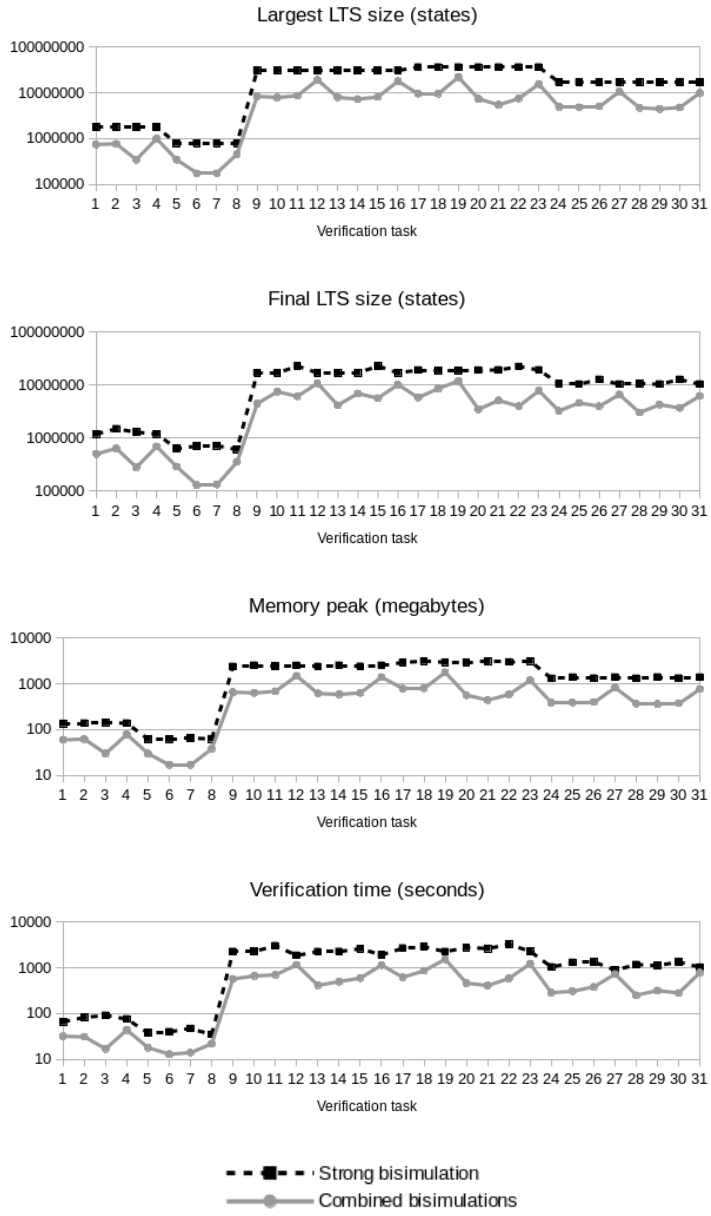


Fig. 2 Results of the TFTP case-study (vertical axis is on a logarithmic scale)

5.2 Parallel benchmark of the RERS 2018 challenge

The RERS (Rigorous Examination of Reactive Systems)[11] challenge is an international competition on a benchmark of verification tasks. Since 2018 (8th edition), the challenge features a set of parallel problems where systems are synchronizing LTS and properties are expressed using CTL and modalities. This section illustrates the benefits of our approach on these problems.

The benchmark comprises three specifications of concurrent systems, numbered 101, 102, and 103, each accompanied by three properties to be checked, numbered

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

			Strong bisimulation				Combined bisimulations				Ratio Strong/Combined			
			Kstates		verif.		Kstates		verif.		Kstates		verif.	
P	S	I	largest	final	MB	sec.	largest	final	MB	sec.	largest	final	MB	sec.
08	A	A	1783	1197	136	66	746	503	60	32	2.4	2.4	2.3	2.1
14	A	A	1783	1495	138	83	770	639	62	31	2.3	2.3	2.2	2.7
08	A	B	1783	1295	143	91	347	281	30	17	5.1	4.6	4.8	5.4
17	A	B	1783	1191	138	76	1000	694	79	44	1.8	1.7	1.7	1.7
08	B	A	792	636	62	38	347	289	30	18	2.3	2.2	2.1	2.1
08	B	B	792	707	62	40	174	130	17	13	4.6	5.4	3.6	3.1
14	B	B	792	720	65	47	174	134	17	14	4.6	5.4	3.8	3.4
16	B	B	792	613	63	35	456	356	38	22	1.7	1.7	1.7	1.6
08	C	A	30926	16924	2393	2274	8448	4471	660	575	3.7	3.8	3.4	4.0
09	C	A	30965	16935	2540	2305	7940	7516	636	666	3.9	2.3	4.0	3.5
14	C	A	30965	22772	2417	3033	8679	6141	688	706	3.6	3.7	3.5	4.3
17	C	A	30992	16935	2527	1873	19269	10833	1502	1164	1.6	1.6	1.7	1.6
08	C	B	30926	16853	2391	2257	7995	4176	617	417	3.9	4.0	3.9	5.4
09	C	B	30953	16998	2538	2271	7326	6931	589	502	4.2	2.5	4.3	4.5
14	C	B	30953	22710	2410	2635	8146	5683	637	601	3.8	4.0	3.8	4.4
17	C	B	30992	16998	2529	1913	18033	10173	1406	1155	1.7	1.7	1.8	1.7
08	D	A	36447	19136	2934	2703	9655	5824	783	623	3.8	3.3	3.7	4.3
09	D	A	36447	18731	3152	2903	9636	8607	808	865	3.8	2.1	3.9	3.4
17	D	A	36447	18731	2971	2262	22203	11926	1793	1545	1.6	1.6	1.7	1.5
08	D	B	36447	19062	2933	2787	7405	3498	568	464	4.9	5.4	5.2	6.0
09	D	B	36478	19243	3110	2650	5495	5153	443	412	6.6	3.7	7.0	6.4
14	D	B	36478	22237	3002	3264	7533	3986	588	589	4.8	5.6	5.1	5.5
16	D	B	36478	19243	3161	2303	15576	7830	1204	1232	2.3	2.5	2.6	1.9
08	E	A	17035	10646	1315	1046	4970	3257	388	287	3.4	3.3	3.4	3.6
09	E	A	17035	10375	1397	1312	4856	4620	392	310	3.5	2.2	3.6	4.2
14	E	A	17035	12759	1327	1346	5081	3983	400	384	3.4	3.2	3.3	3.5
16	E	A	17035	10376	1399	897	10723	6643	832	736	1.6	1.6	1.7	1.2
08	E	B	17035	10529	1318	1177	4735	3033	369	253	3.6	3.5	3.6	4.7
09	E	B	17035	10401	1402	1130	4471	4267	363	320	3.8	2.4	3.9	3.5
14	E	B	17035	12725	1329	1342	4810	3714	377	283	3.5	3.4	3.5	4.7
16	E	B	17035	10402	1396	1033	9987	6202	773	793	1.7	1.7	1.8	1.3

Table 2 Detailed results of the TFTP case-study

p#21, p#22, and p#23, where p is the system number. Thus, nine verification tasks have to be solved. The properties are presented in Table 3, where the strong action formulas are highlighted. One easily shows that all other action formulas are weak using Lemma 4. However, for 103#22 and 103#23, the identity $(\langle\alpha\rangle \mathsf{true} \Rightarrow [\alpha]\,\varphi) = ([\alpha]\,\mathsf{false} \vee [\alpha]\,\varphi) = [\alpha]\,\varphi$ (because $[\alpha]\,\mathsf{false} \Rightarrow [\alpha]\,\varphi$ for all φ) was applied to obtain the simplified formulas occurring after the = sign in the table. For 102#23,

Nr.	Property	Result
101#21	AG([$A21$] [$A23$] [$A4$] [true] false)	false
101#22	AG([$A3$] AF(⟨$A2$⟩ true))	false
101#23	AG(⟨$A20$⟩ true ⇒ ⟨$A20$⟩ A([$A23$] false W ⟨$A8$⟩ true))	true
102#21	EF(AG([$A5$] false))	true
102#22	EG([$A35$] E([$A23$] false U ⟨$A35$⟩ true))	false
102#23	AG([$A22$] A([$A8$] false U ⟨$A22$⟩ true))	false
103#21	AG([$A11$] A([$A2$] false W ⟨$A6$⟩ true) ⇒ [$A11$] A([$A5$] false W ⟨$A6$⟩ true))	true
103#22	EG([$A14$] false ∧ (⟨$A18$⟩ true ⇒ [$A18$] EG([$A21$] false ∧ EF(⟨$A19$⟩ true)))) = EG([$A14$] false ∧ [$A18$] EG([$A21$] false ∧ EF(⟨$A19$⟩ true)))	true
103#23	AG(⟨$A34$⟩ true ⇒ [$A34$] A([$A68$] false W ⟨$A59$⟩ true)) = AG([$A34$] A([$A68$] false W ⟨$A59$⟩ true))	false

Table 3 RERS 2018 properties (strong action formulas are highlighted)

this simplification allowed us to prove that $A34$ is not a strong action, unlike what appears at first sight. This identification of strong actions was done initially by hand. We later implemented a script dedicated to CTL formulas of similar forms, which automatically returns a set of strong actions by identifying the patterns of Lemma 4. This script does not apply simplifications of the formula as the one described above for 103#22 and 103#23.

Task	#act	#hide	#sact	#proc	#sproc	#sync	relation
101#21	24	21	24	9	9	-	strong
101#22	24	22	1	9	4	11	combination
101#23	24	21	2	9	3	9	combination
102#21	28	27	0	20	0	-	divbranching
102#22	28	26	2	20	10	14	combination
102#23	28	26	1	20	4	12	combination
103#21	70	66	2	34	8	12	combination
103#22	70	66	3	34	6	18	combination
103#23	70	67	1	34	7	10	combination

Table 4 Some numbers about the RERS 2018 parallel benchmark

Table 4 gives, for each of the nine verification tasks, the number #act of actions in the system, the number #hide of actions in the maximal hiding set, the number #sact of strong actions, the number #proc of parallel processes, the number #sproc of processes in the strong group, the number #sync of inter-group actions, and the best reduction relation among strong bisimulation, divbranching bisimulation, or a combination of both. We observe that:

– The set of weak actions of 101#21 is empty due to the presence of the "true" strong action formula, whereas the set of strong actions of 102#21 is empty, i.e., the property belongs to L_μ^{dbr}. In both cases, our approach coincides with the mono-bisimulation approach. The verification of 101#21 (reduced for strong bisimulation) takes 75 seconds, with a memory peak of 11 MB and a largest LTS of 83,964 states and 374,809 transitions. The verification of 102#21 (reduced

for divbranching bisimulation) takes 261 seconds, with a memory peak of 22 MB and a largest LTS of 243 states and 975 transitions.
– 101#22, 101#23, 102#22, 102#23, 103#21, 103#22, and 103#23 contain both weak and strong actions. They are used to evaluate our approach.

Table 5 compares the performance of verifying the latter seven verification tasks using the approaches described above. LTS sizes are given in kilostates, memory in megabytes, and time in seconds. Tasks using more than 3 GB of memory were aborted. We see that our approach reduces both time and memory usage and allows all problems of the challenge to be solved, whereas using strong bisimulation alone fails in five out of those seven tasks.[12]

	Strong bisimulation				Combined bisimulations			
	Kstates		verif.		Kstates		verif.	
Task	largest	final	MB	sec.	largest	final	MB	sec.
101#22	84	77	10	77	1.4	1.4	10	72
101#23	84	77	11	80	0.5	0.5	8	73
102#22	-	-	-	-	611	585	57	295
102#23	-	-	-	-	17	9.8	22	260
103#21	-	-	-	-	734	313	101	604
103#22	-	-	-	-	14,143	14,141	1575	2533
103#23	-	-	-	-	122	122	35	566

Table 5 Experimental results of the RERS 2018 parallel benchmark

The negligible reductions in time and memory usage observed for tasks 101#22 and 101#23 are due to the fact that time and memory usage are dominated by the algorithm in charge of selecting a subset of processes to be composed and reduced (implemented in smart reduction). The complexity of this algorithm does not depend on the state space size, but on the number of actions and parallel processes, which is almost the same using both approaches. When considering larger examples, memory usage gets dominated by minimisation. In particular, for tasks 102#22, 102#23, 103#21, and 103#23 (and likely also 103#22), memory usage is reduced by several orders of magnitude.

5.3 Parallel benchmark of the RERS 2019 challenge

We tried the approach on the benchmark of parallel CTL problems given in the RERS 2019 challenge[13]. The principle of the challenge was the same as in 2018,

[12] For problem 103#23, we tried smart strong reduction using the Grid'5000 [2] high-performance computing platform and the distributed LTS generation tool DISTRIBUTOR [17] available in CADP. We could not get the final LTS (it would have required us to ask for a larger disk quota than the current 100 GB, which is already huge), but we obtained an intermediate LTS containing about 4.5 billion states and 36 billion transitions, whose generation took about 50 minutes using 84 distributed processes running on 7 multi-core computers. The largest intermediate LTS obtained using combined bisimulations is thus at least 36,800 times smaller than using strong bisimulation in this example. Experiments on Grid'5000 are courtesy of Wendelin Serwe.

[13] `http://rers-challenge.org/2019`

but the number of CTL formulas was raised from 9 in 2018 to 180 in 2019, the number of systems was raised from 3 in 2018 to 9 in 2019, and the size of systems was increased from up to 34 parallel process and 70 actions in 2018 to 70 parallel processes and 234 actions in 2019. Table 6 gives details on the system sizes. For each system (numbered from 101 to 109), column #proc indicates the number of parallel processes, column #avg-size indicates the average process size in number of states, column #act indicates the number of actions (including the invisible action τ), column #states gives the (unreduced) state space size of the system in number of states, and column #strong gives the size of the state space once reduced for strong bisimilarity. Unsurprisingly, the state space size grows exponentially with the number of processes and indeed, while we can rather easily generate the state space for the first (smallest) problems, it becomes quickly infeasible for the largest problems. Each system is accompanied by 20 CTL formulas numbered from 01 to 20.

System	#proc	#avg-size	#act	#states	#strong
101	8	12	30	26,996	6,600
102	10	19	59	290,149	149,040
103	12	12	66	791,862	300,000
104	15	19	54	1,667,045	589,680
105	20	19	78	52,946,151	6,036,800
106	25	22	106	$> 300,000,000$	_
107	50	19	180	_	_
108	60	19	218	_	_
109	70	16	234	_	_

Table 6 Some numbers about systems of the RERS 2019 parallel challenge

As in 2018, we analysed the 180 formulas to extract a set of strong actions, then partitioned the processes and refactored their composition into a weak and a strong group, and finally used the smart reduction heuristic to reduce both groups, using strong and divbranching bisimulation where appropriate. Figure 3 shows how such a strategy can be scripted using the SVL [14] language of CADP for problem 101#01, whose property is $\mathsf{A}([A20]\,\mathsf{false}\ \mathsf{W}\ \langle A25\rangle\,\mathsf{true} \vee \langle A21\rangle\,\mathsf{true})$. When verifying this property, all actions but $A20, A21$, and $A25$ can be hidden, whereas $A21$ and $A25$ are the only strong actions.

Using the combined bisimulations approach, we were able to verify all problems concerning systems 101 to 107, except problem 107#08, whose state space remained too large. Tables 7 to 13 indicate for each problem, the truth value, the maximal intermediate LTS size (in number of states), and the final LTS size, using both the mono-bisimulation approach (strong) and the combined bisimulations approach. They show that the combined bisimulations approach really increases our capabilities to verify large concurrent systems, even though the complete exploration of the state space of systems 108 and 109 remains problematic.

It is interesting to see that in some cases (e.g., problems 107#01 and 107#07), the final problem has just one state. In problem 107#01, the formula has the form $\mathsf{AG}([A108]\,\mathsf{false})$, which belongs to $L_\mu^{strong}(\emptyset)$ (i.e., the strong group is empty), and all the paths are loops that contain the $A108$ action (which is weak). The final LTS, obtained by divbranching minimization, encodes the process $P = A108.P$,

```
-- not hideable actions: A20, A21, and A25
-- strong actions: A21 and A25
-- inter-group actions: A4, A9, A10, A11, A14, A23, and A24

-- weak group: processes not containing A21 nor A25
"problem101_01_weak.bcg" = smart divbranching reduction of
   -- hiding all but inter-group actions and actions visible in the formula
   hide all but A4, A9, A10, A11, A14, A20, A21, A23, A24, A25 in
      par
         A9, A14 -> "P1.bcg"
      || A4, A7, A12, A13, A15, A23, A24 -> "P2.bcg"
      || A7, A12, A13 -> "P3.bcg"
      || A8, A27 -> "P4.bcg"
      || A26, A29, A30, A31, A32 -> "P5.bcg"
      || A8, A10, A11, A26, A27, A28, A29 -> "P8.bcg"
      end par
   end hide;

-- strong group: processes containing A21 and/or A25
"problem101_01_strong.bcg" = smart strong reduction of
   -- hiding all but inter-group actions and actions visible in the formula
   hide all but A4, A9, A10, A11, A14, A20, A21, A23, A24, A25 in
      par
         A1, A2, A3, A4, A5, A6, A16, A17, A18, A19, A20, A21, A22,
         A23, A24, A25 -> "P6.bcg"
      || A2, A9, A10, A11, A14, A20, A21 -> "P7.bcg"
      end par
   end hide;

-- group composition
"problem101_01_combined.bcg" = strong reduction of
   -- hiding all but actions visible in the formula
   hide all but A20, A21, A25 in
      par
         A4, A9, A10, A11, A14, A23, A24 -> "problem101_01_weak.bcg"
      || A4, A9, A10, A11, A14, A23, A24 -> "problem101_01_strong.bcg"
      end par
   end hide
```

Fig. 3 SVL encoding of combined bisimulations approach for problem 101#01

101	strong	combined	strong	combined	strong	mixed	strong	combined	strong	combined
property	1		2		3		4		5	
value	True		False		True		False		False	
max LTS size	6750	117	6750	37	6750	125	6750	37	6750	37
final LTS size	3750	46	3750	21	3750	46	3750	10	3750	9
property	6		7		8		9		10	
value	True		True		True		False		True	
max LTS size	6750	125	6750	455	6675	125	6750	117	3847	125
final LTS size	3750	51	3825	255	4050	50	3825	50	3750	51
property	11		12		13		14		15	
value	False		True		True		True		True	
max LTS size	6675	37	6750	455	4155	125	6750	37	6750	37
final LTS size	3750	10	4050	255	4050	50	3750	10	3825	9
property	16		17		18		19		20	
value	True		True		True		False		True	
max LTS size	6750	125	6750	117	6750	125	6750	117	6750	125
final LTS size	4050	50	3750	46	3750	46	3750	46	3750	51

Table 7 Results for problem 101 of the RERS 2019 parallel CTL challenge

102	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	True		False		True		True		False	
max LTS size	62952	164	62952	156	62952	151	59856	156	66048	164
final LTS size	62352	21	62362	116	62352	18	59256	110	65448	127
property	6		7		8		9		10	
value	False		False		False		False		True	
max LTS size	69000	164	62436	151	62436	151	62952	164	69000	164
final LTS size	59256	115	61836	17	61836	20	62352	121	59256	115
property	11		12		13		14		15	
value	False		False		False		True		True	
max LTS size	69000	164	65016	151	68112	151	69600	156	59856	156
final LTS size	59256	115	64416	18	67512	19	59856	110	59256	110
proposition	16		17		18		19		20	
value	False		True		True		True		False	
max LTS size	69000	164	62952	942	59856	151	62952	151	59856	151
final LTS size	59256	115	62352	702	59256	17	62352	18	59256	7

Table 8 Results for problem 102 of the RERS 2019 parallel CTL challenge

103	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	True		False		True		True		False	
max LTS size	158325	198	151725	184	151725	72	158325	71	151725	194
final LTS size	102600	126	98325	100	98325	19	102600	21	98325	105
property	6		7		8		9		10	
value	True		False		True		True		True	
max LTS size	151725	184	151725	72	151725	72	158325	198	158325	72
final LTS size	98325	105	98325	20	98325	21	102600	126	102600	20
property	11		12		13		14		15	
value	True		False		False		True		True	
max LTS size	151725	1536	151725	194	151725	194	151725	72	158325	194
final LTS size	98325	660	98325	105	98325	105	98325	20	102600	115
property	16		17		18		19		20	
value	False		True		True		False		False	
max LTS size	158325	72	102600	72	158325	72	158325	72	151725	72
final LTS size	102600	20	102600	21	102600	21	102600	20	98325	20

Table 9 Results for problem 103 of the RERS 2019 parallel CTL challenge

104	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	True		True		False		False		False	
max LTS size	503139	756	117390	212	117390	3024	117390	212	108360	3024
final LTS size	117390	92	117390	12	117390	100	117390	12	108360	92
property	6		7		8		9		10	
value	True		False		True		True		True	
max LTS size	117390	212	117390	212	464499	212	283102	212	282360	212
final LTS size	117390	12	117390	10	108360	10	117390	12	108360	11
property	11		12		13		14		15	
value	True		False		False		True		False	
max LTS size	283102	24192	464499	3024	117390	24192	117390	212	503139	212
final LTS size	117390	819	108360	92	117390	819	117390	12	117390	13
property	16		17		18		19		20	
value	True		False		True		False		True	
max LTS size	261367	3024	283102	756	108360	24192	261367	24192	108360	480
final LTS size	108360	83	117390	92	108360	756	108360	756	108360	71

Table 10 Results for problem 104 of the RERS 2019 parallel CTL challenge

105	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	True		True		False		True		True	
max LTS size	5926635	1696	4943745	1296	5669301	1696	4946125	451	5685961	1696
final LTS size	2568496	896	2242912	168	2459968	854	2242912	160	2459968	854
property	6		7		8		9		True	
value	True		False		True		False		True	
max LTS size	5943295	202	5188713	202	5926635	1296	5188205	202	5926635	202
final LTS size	2568496	18	2351440	19	2568496	168	2242912	18	2568496	19
property	11		12		13		14		15	
value	True		False		False		True		True	
max LTS size	2568496	20	5428795	202	5171545	202	5188205	1296	5428795	540
final LTS size	2242912	128	2351440	18	2242912	19	2242912	168	2351440	121
property	16		17		18		19		20	
value	True		False		False		True		True	
max LTS size	5154885	3157	5412093	202	4946125	202	5171545	202	6441051	540
final LTS size	2242912	1136	2242912	18	2242912	128	2242912	19	2785552	133

Table 11 Results for problem 105 of the RERS 2019 parallel CTL challenge

106	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	False		False		True		True		False	
max LTS size	-	172	-	275	-	172	-	172	-	172
final LTS size	-	18	-	180	-	19	-	18	-	19
property	6		7		8		9		10	
value	True		False		True		True		False	
max LTS size	-	172	-	172	-	172	-	289	-	172
final LTS size	-	18	-	18	-	19	-	190	-	19
property	11		12		13		14		15	
value	False		False		False		True		True	
max LTS size	-	172	-	172	-	172	-	275	-	275
final LTS size	-	17	-	17	-	17	-	180	-	180
property	16		17		18		19		20	
value	False		False		True		True		False	
max LTS size	-	172	-	172	-	172	-	172	-	172
final LTS size	-	17	-	17	-	17	-	17	-	18

Table 12 Results for problem 106 of the RERS 2019 parallel CTL challenge

107	strong	combined	strong	combined	strong	combined	strong	combined	strong	combined
property	1		2		3		4		5	
value	False		True		True		True		True	
max LTS size	-	258	-	6531840	-	58786560	-	231472080	-	26453952
final LTS size	-	1	-	340	-	340	-	5090	-	328
property	6		7		8		9		10	
value	False		False		False		True		False	
max LTS size	-	244944	-	258	-	XXX	-	244944	-	13122
final LTS size	-	26	-	1	-	XXX	-	27	-	378
property	11		12		13		14		15	
value	True		False		False		True		False	
max LTS size	-	81648	-	19595520	-	79361856	-	1458	-	6531840
final LTS size	-	26	-	340	-	302	-	364	-	340
property	16		17		18		19		20	
value	True		True		False		False		True	
max LTS size	-	734832	-	4374	-	244944	-	1417176	-	81648
final LTS size	-	26	-	364	-	26	-	4228	-	26

Table 13 Results for problem 107 of the RERS 2019 parallel CTL challenge

which is indeed a valid property preserving minimization of the system. Problem 107#07 is similar.

Note that some of the RERS 2019 tasks can be evaluated more efficiently using non-compositional approaches, such as on-the-fly model checking (also available in CADP), in cases where proofs or counterexamples can be detected much before having explored the full state space. However, on-the-fly verification fails in many cases, in particular when the verification requires traversing the whole state space, such as the following problems, which all evaluate to true:

- 103#06: $\mathsf{AG}(\langle A5\rangle\,\mathsf{true} \Rightarrow \mathsf{A}([A45]\,\mathsf{false}\ \mathsf{W}\ \langle A46\rangle\,\mathsf{true})$
- 104#01: $\mathsf{AG}(\langle A17\rangle\,\mathsf{true} \Rightarrow \mathsf{AF}(\langle A16\rangle\,\mathsf{true}))$
- 105#06: $\mathsf{AG}(\langle A1\rangle\,\mathsf{true} \Rightarrow \mathsf{A}([A61]\,\mathsf{false}\ \mathsf{U}\ \langle A22\rangle\,\mathsf{true}))$
- 107#20: $\mathsf{AG}(\langle A175\rangle\,\mathsf{true} \Rightarrow \mathsf{A}([A76]\,\mathsf{false}\ \mathsf{W}\ \langle A59\rangle\,\mathsf{true}))$

This suggests a reasonable approach to tackle model checking problems, consisting in first applying on-the-fly model checking, setting a timeout and a memory quota to stop the verification when it becomes too eager, and then applying compositional reduction for the problems whose on-the-fly verification has been stopped.

The main drawback of our compositional approach with respect to on-the-fly verification is that, due to maximal hiding, the generated counterexamples are given only in terms of the actions visible in the formula, which abstracts out a lot of intermediate transitions. However, this is the price to pay for being able to verify most of the tasks.

6 Related Work

We make more precise in this section the links between our approach and two other approaches described in the literature, namely partial model checking and partial order reductions.

Given a temporal logic formula φ and a system described by a composition of processes $P_1||\dots||P_n$, partial model checking [1] is based on the computation of a quotient $\varphi//P_i$ of the formula φ with respect to some process P_i ($i \in 1..n$). This quotient is a new temporal logic formula, which holds true on the system from which P_i has been removed if and only if φ holds true on the whole system. This so-called quotienting approach can be applied repeatedly, until the formula evaluates (using partial evaluation rules) to either constant true or false, which happens at the latest when the formula has been quotiented with respect to all processes. At each step, simplifications are applied to the quotient, with the aim to keep it small enough. Successive quotienting of φ with respect to processes $P_1, \dots, P_i$ "consumes" the modalities on actions which do not have to be synchronized with the remaining process $P_{i+1}, \dots, P_n$. As shown in [32], quotienting can be assimilated to (and implemented as) a parallel composition between a graph representing the formula and the process P_i, partial evaluation of the formula can be implemented as the resolution of a Boolean Equation System, and simplifications can be obtained by a combination of strong bisimulation minimisation and compression of those sequences of transitions representing consumed modalities.

Both compositional minimization and partial model checking are iterative approaches, each step of which starts with a formula φ and a composition $P_1||\dots||P_n$. Thus virtually, steps of partial model checking and compositional minimization could be intertwined, one potentially having a choice between either generating the state space of (all or some part of) $P_1||\dots||P_n$ or computing the quotient of φ with respect to some P_i and then simplifying this quotient. Choosing state space generation enables hiding and bisimulation reduction, possibly taking into account the results presented in the current paper, which is a neat advantage when divbranching bisimulation can be applied. On the other hand, partial model checking, in addition to an overhead due to the representation of the formula, suffers from the fact that once the formula has been quotiented with respect to some process, the potential benefits of minimizing the composition of this process with other processes with respect to divbranching bisimulation is lost. Yet, quotienting may have different benefits in some cases, such as the ability to yield a definitive verdict of the verification before the formula has been quotiented with respect to all processes (when some processes have no influence on the verdict), or the ability to trim processes to only those parts that are actually needed to verify the formula (when the formula requires only a partial traversal of the state space). From our own experience, compositional minimization of the system often shows better performance than partial model checking in practice. Yet, there remain corner cases where partial model checking is the winning strategy.

Our approach (in particular Theorem 2, page 13) is consistent with partial model checking and the mono-bisimulation approach [40] in the following sense: If the system has the form $P||Q$ where process P contains strong actions and process Q does not, all the modalities on actions present only in P (thus strong modalities) are consumed in $\varphi//P$, which thus belongs to L_μ^{dbr}. Following [40], since $\varphi//P$ has

to be checked on Q, this confirms the main result presented in this paper, namely that Q can be reduced with respect to divbranching bisimulation.

Another interesting class of approaches when model checking compositions of processes $P_1||\ldots||P_n$ is partial order reductions [20], which consist in identifying in $P_1||\ldots||P_n$ sets of transitions that "commute" and whose interleaving can sometimes be avoided. Side criteria guarantee the preservation of some equivalence relation and/or temporal logic. These sets of transitions are computed in each state, either by using a structural analysis of the system [20, 47, 44, 54, 19], or from a local, on-the-fly exploration of the state space, such as in τ-confluence reduction [24, 25, 39]. To our knowledge, works on partial order reduction in the action-based setting preserve either logics containing only some forms of weak modalities, but no strong modalities as in $L_\mu^{strong}(A_s)$. Yet, partial order reductions preserving divbranching bisimulation can be used complementary to the approach presented in this paper, when generating compositions of processes that do not contain strong actions (thus providing an efficient pre-reduction of the resulting state space).

In a recent paper [34], we refined the approach presented in the current paper by devising a family of bisimilarity relations called sharp bisimilarities, which are weaker than strong bisimilarity and stronger than divbranching bisimilarity. Each of these relations is parameterized by a set A_s of strong actions. We showed that a formula is preserved (on all models) by sharp bisimilarity with respect to the set A_s of strong actions if and only if the formula belongs to the fragment $L_\mu^{strong}(A_s)$. Moreover, sharp bisimilarities are congruences for parallel composition and action mapping, and can thus be applied compositionally. The sharp bisimilarity approach is consistent with the combination of strong and divbranching bisimilarities presented in the current paper, in the sense that (1) sharp bisimilarity with respect to any set of actions that do not occur in a process is preserved by divbranching bisimilarity for this process and (2) sharp bisimilarity is also preserved by strong bisimilarity, which is a stronger relation. Although the sharp bisimilarity approach is generally more effective to reduce the state space (in particular it allowed us to verify successfully all problems of the RERS 2019 parallel CTL challenge, including those concerning the largest systems 108 and 109), the current approach has the advantage that it can be applied using standard bisimilarity relations (implemented in various toolsets), namely strong and divbranching, and does not require a specific implementation of sharp bisimilarity reduction.

7 Conclusion and Future Work

In this paper, we proposed a compositional verification approach that refines a previous approach [40] and consists of three steps: First, so-called strong actions are identified, corresponding to those actions of the system that the formula cannot match using weak modalities in the sense of the L_μ fragment L_μ^{dbr} adequate with divbranching bisimulation. These actions are used to partition the parallel processes into those containing strong actions and the others. Second, maximal hiding and compositional reduction are used to minimize the composition of processes not containing strong actions for divbranching bisimulation, and the other processes for strong bisimulation. Finally, the property is verified on the reduced system.

The originality of this approach is to combine strong and divbranching bisimulation, as opposed to the mono-bisimulation approach of [40]. We proved it correct by characterizing a family of fragments of the logic L_μ, called $L_\mu^{strong}(A_s)$, parameterized by the set A_s of strong actions. We also showed under which conditions action-based branching-time temporal logic formulas containing well-known operators from the logics CTL, ACTL, PDL, and PDL-Δ are part of $L_\mu^{strong}(A_s)$ when A_s is fixed. In the future, it might be worth investigating whether more operators can be considered, e.g., from the linear-time logic LTL.

This approach may significantly improve the verification performance for systems containing both processes with and without strong actions, as illustrated by two case-studies. In particular, it allowed the whole parallel CTL benchmark of the RERS 2018 challenge to be solved on a standard computer.

Identifying minimal sets of strong actions for arbitrary formulas manually is a cumbersome task, prone to errors. We shall investigate ways to compute such sets automatically. As illustrated by verification task 103#23 of RERS 2018, the problem is not purely syntactic: considering non-trivial semantic equivalences may prove useful to eliminate actions that appear strong at first sight. Yet, we trust that the presented approach has potential to be implemented in automated software tools, such as those available in the CADP toolbox.

Acknowledgements

This work has been partially funded by the 4SECURail project. The 4SECURail project received funding from the Shift2Rail Joint Undertaking under the European Union's Horizon 2020 research and innovation programme under grant agreement No 881775 in the context of the open call S2R-OC-IP2-01-2019, part of the "Annual Work Plan and Budget 2019", of the programme H2020-S2RJU-2019. The content of this paper reflects only the authors' view and the Shift2Rail Joint Undertaking is not responsible for any use that may be made of the included information.

The authors warmly thank Wendelin Serwe for his comments on earlier versions of this paper and his help to carry on experiments on Grid'5000. They also are extremely grateful to Hubert Garavel, who triggered the author's collaboration between Grenoble and Pisa and who provided the TFTP case-study (together with Damien Thivolle). They thank the organizers of the RERS challenge, who produced the numerous examples from which we could develop these ideas. Finally, they thank the anonymous referees of FM'2019 for their useful comments on the short version of this paper and of FMSD for the current version.

References

1. Henrik Reif Andersen. Partial model checking. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science LICS (San Diego, California, USA)*, pages 398–407. IEEE Computer Society Press, June 1995.
2. Raphael Bolze, Franck Cappello, Eddy Caron, Michel J. Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quétier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *IJHPCA*, 20(4):481–494, 2006.
3. Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the Association for Computing Machinery*, 11(4):481–494, 1964.

4. David Champelovier, Xavier Clerc, Hubert Garavel, Yves Guerte, Christine McKinty, Vincent Powazny, Frédéric Lang, Wendelin Serwe, and Gideon Smeding. Reference Manual of the LNT to LOTOS Translator (Version 6.7). INRIA, Grenoble, France, July 2017.
5. S. C. Cheung and J. Kramer. Enhancing Compositional Reachability Analysis with Context Constraints. In *Proceedings of the 1st ACM SIGSOFT International Symposium on the Foundations of Software Engineering (Los Angeles, CA, USA)*, pages 115–125. ACM Press, December 1993.
6. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
7. R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G. v. Bochmann and D. K. Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification CAV '92 (Montréal, Canada)*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422, Berlin, June 1992. Springer.
8. Pepijn Crouzen and Frédéric Lang. Smart Reduction. In Dimitra Giannakopoulou and Fernando Orejas, editors, *Proceedings of Fundamental Approaches to Software Engineering (FASE'11), Saarbrücken, Germany*, volume 6603 of *Lecture Notes in Computer Science*, pages 111–126. Springer, March 2011.
9. R. De Nicola and F.W Vaandrager. Three logics for branching bisimulation. *Journal of the Association for Computing Machinery*, 1990.
10. Sander de Putter, Anton Wijs, and Frédéric Lang. Compositional model checking is lively — extended version, 2018. Submitted to Science of Computer Programming.
11. E. Allen Emerson and C-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the 1st International Symposium on Logic in Computer Science LICS'86*, pages 267–278, 1986.
12. Alessandro Fantechi, Stefania Gnesi, and Gioia Ristori. From actl to μ-calculus (extended abstract). In *Proceedings of the Workshop on Theory and Practice in Verification*. ERCIM, 1992.
13. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, September 1979.
14. Hubert Garavel and Frédéric Lang. SVL: a Scripting Language for Compositional Verification. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'01), Cheju Island, Korea*, pages 377–392. Kluwer Academic Publishers, August 2001. Full version available as INRIA Research Report RR-4223.
15. Hubert Garavel, Frédéric Lang, and Radu Mateescu. Compositional Verification of Asynchronous Concurrent Systems Using CADP. *Acta Informatica*, 52(4):337–392, April 2015.
16. Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 15(2):89–107, April 2013.
17. Hubert Garavel, Radu Mateescu, Damien Bergamini, Adrian Curic, Nicolas Descoubes, Christophe Joubert, Irina Smarandache-Sturm, and Gilles Stragier. Distributor and bcg_merge: Tools for distributed explicit state space generation. In Holger Hermanns and Jens Palberg, editors, *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), Vienna, Austria*, volume 3920 of *Lecture Notes in Computer Science*, pages 445–449. Springer, March–April 2006.
18. Hubert Garavel and Damien Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In Corina Pasareanu, editor, *Proceedings of the 16th International SPIN Workshop on Model Checking of Software (SPIN'09), Grenoble, France*, volume 5578 of *Lecture Notes in Computer Science*, pages 241–260. Springer, June 2009.
19. R. Gerth, R. Kuiper, W. Penczek, and D. Peled. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2):132–152, 1999. A short version of this paper was previously published at the Third Israel Symposium on Theory of Computing and Systems ISTCS 1995.
20. Patrice Godefroid. Using partial orders to improve automatic verification methods. In R. P. Kurshan and E. M. Clarke, editors, *Proceedings of the 2nd Workshop on Computer-Aided Verification (Rutgers, New Jersey, USA)*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 321–340. AMS-ACM, June 1990.

21. Susanne Graf and Bernhard Steffen. Compositional Minimization of Finite State Systems. In Edmund M. Clarke and Robert P. Kurshan, editors, *Proceedings of the 2nd Workshop on Computer-Aided Verification (CAV'90), Rutgers, New Jersey, USA*, volume 531 of *Lecture Notes in Computer Science*, pages 186–196. Springer, June 1990.
22. Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.
23. Jan Friso Groote and Alban Ponse. The Syntax and Semantics of μCRL. CS-R 9076, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.
24. Jan Friso Groote and M. P. A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1–2):47–81, 1996.
25. J.F. Groote and J. van de Pol. State space reduction using partial τ-confluence. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'00), Bratislava, Slovakia*, volume 1893 of *Lecture Notes in Computer Science*, pages 383–393. Springer, August 2000. Also available as CWI Technical Report SEN-R0008, Amsterdam, March 2000.
26. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Berlin, 2000.
27. ISO/IEC. LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva, September 1989.
28. ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization – Information Technology, Geneva, September 2001.
29. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
30. Jean-Pierre Krimm and Laurent Mounier. Compositional State Space Generation from LOTOS Programs. In Ed Brinksma, editor, *Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), University of Twente, Enschede, The Netherlands*, volume 1217 of *Lecture Notes in Computer Science*. Springer, April 1997. Extended version with proofs available as Research Report VERIMAG RR97-01.
31. Frédéric Lang. EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods. In Judi Romijn, Graeme Smith, and Jaco van de Pol, editors, *Proceedings of the 5th International Conference on Integrated Formal Methods (IFM'05), Eindhoven, The Netherlands*, volume 3771 of *Lecture Notes in Computer Science*, pages 70–88. Springer, November 2005. Full version available as INRIA Research Report RR-5673.
32. Frédéric Lang and Radu Mateescu. Partial model checking using networks of labelled transition systems and boolean equation systems. In Cormac Flanagan and Barbara König, editors, *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12), Talinn, Estonia*, volume 7214 of *Lecture Notes in Computer Science*, pages 141–156. Springer, March 2012.
33. Frédéric Lang, Radu Mateescu, and Franco Mazzanti. Compositional verification of concurrent systems by combining bisimulations. In Annabelle McIver and Maurice ter Beek, editors, *Proceedings of the 23rd International Symposium on Formal Methods – 3rd World Congress on Formal Methods FM 2019 (Porto, Portugal)*, volume 11800 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2019.
34. Frédéric Lang, Radu Mateescu, and Franco Mazzanti. Sharp congruences adequate with temporal logics combining weak and strong modalities. In Armin Biere and Dave Parker, editors, *Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2020 (Dublin, Ireland)*, volume 12079 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2020.
35. K. G. Larsen. Proof systems for hennessy-milner logic with recursion. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming CAAP '88 (Nancy, France)*, volume 299 of *Lecture Notes in Computer Science*, pages 215–230, Berlin, March 1988. Springer.
36. J. Malhotra, S. A. Smolka, A. Giacalone, and R. Shapiro. A Tool for Hierarchical Design and Simulation of Concurrent Systems. In *Proceedings of the BCS-FACS Workshop on Specification and Verification of Concurrent Systems, Stirling, Scotland, UK*, pages 140–152. British Computer Society, July 1988.

37. Radu Mateescu and Jose Ignacio Requeno. On-the-Fly Model Checking for Extended Action-Based Probabilistic Operators. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 20(5):563–587, 2018.
38. Radu Mateescu and Damien Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In Jorge Cuellar, Tom Maibaum, and Kaisa Sere, editors, *Proceedings of the 15th International Symposium on Formal Methods (FM'08), Turku, Finland*, volume 5014 of *Lecture Notes in Computer Science*, pages 148–164. Springer, May 2008.
39. Radu Mateescu and Anton Wijs. Sequential and Distributed On-the-Fly Computation of Weak Tau-Confluence. *Science of Computer Programming*, 77(10–11):1075–1094, September 2012.
40. Radu Mateescu and Anton Wijs. Property-Dependent Reductions Adequate with Divergence-Sensitive Branching Bisimilarity. *Science of Computer Programming*, 96(3):354–376, 2014.
41. Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
42. R. De Nicola and F. W. Vaandrager. *Action versus State Based Logics for Transition Systems*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer, 1990.
43. David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, March 1981.
44. D. Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In Peled et al. [45].
45. D.A. Peled, V.R. Pratt, and G.J. Holzmann, editors. *Proceedings of the Workshop on Partial Order Methods in Verification*, volume 29 of *Dimacs Series in Discrete Mathematics*, 1997.
46. Amir Pnueli. In transition from global to modular temporal reasoning about programs. *Logic and Models of Concurrent Systems*, 13:123–144, 1984.
47. Y.S. Ramakrishna and S.A. Smolka. Partial-order reduction in the weak modal mu-calculus. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 1997.
48. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.
49. Krishan K. Sabnani, Aleta M. Lapone, and M. Ümit Uyar. An Algorithmic Procedure for Checking Safety Properties of Protocols. *IEEE Transactions on Communications*, 37(9):940–948, September 1989.
50. R. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1-2):121–141, 1982.
51. R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, June 1989.
52. Kuo-Chung Tai and Pramod V. Koppol. An Incremental Approach to Reachability Analysis of Distributed Programs. In *Proceedings of the 7th International Workshop on Software Specification and Design, Los Angeles, CA, USA*, pages 141–150, Piscataway, NJ, December 1993. IEEE Press.
53. Kuo-Chung Tai and Pramod V. Koppol. Hierarchy-Based Incremental Reachability Analysis of Communication Protocols. In *Proceedings of the IEEE International Conference on Network Protocols, San Francisco, CA, USA*, pages 318–325, Piscataway, NJ, October 1993. IEEE Press.
54. A. Valmari. Stubborn set methods for process algebras. In Peled et al. [45].
55. Antti Valmari. Compositional State Space Generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993 – Papers from the 12th International Conference on Applications and Theory of Petri Nets (ICATPN'91), Gjern, Denmark*, volume 674 of *Lecture Notes in Computer Science*, pages 427–457. Springer, 1993.
56. R. J. van Glabbeek and W. Peter 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.
57. Rob J. van Glabbeek and W. Peter Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
58. Wei Jen Yeh and Michal Young. Compositional Reachability Analysis Using Process Algebra. In *Proceedings of the ACM SIGSOFT Symposium on Testing, Analysis, and Verification (SIGSOFT'91), Victoria, British Columbia, Canada*, pages 49–59. ACM Press, October 1991.