HAL
open science

Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems

Frédéric Lang, Radu Mateescu

To cite this version:

Frédéric Lang, Radu Mateescu. Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems. Tools and Algorithms for the Construction and Analysis of Systems, Mar 2012, Tallinn, Estonia. hal-00684471

HAL Id: hal-00684471

https://inria.hal.science/hal-00684471v1

Submitted on 2 Apr 2012

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

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

Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems

Frédéric Lang and Radu Mateescu

Vasy project team, Inria Grenoble Rhône-Alpes/Lig, Montbonnot, France
{Frederic.Lang,Radu.Mateescu}@inria.fr

Abstract. Partial model checking was proposed by Andersen in 1995 to verify a temporal logic formula compositionally on a composition of processes. It consists in incrementally incorporating into the formula the behavioural information taken from one process — an operation called quotienting — to obtain a new formula that can be verified on a smaller composition from which the incorporated process has been removed. Simplifications of the formula must be applied at each step, so as to maintain the formula at a tractable size. In this paper, we revisit partial model checking. First, we extend quotienting to the network of labelled transition systems model, which subsumes most parallel composition operators, including m among n synchronisation and parallel composition using synchronisation interfaces, available in the E-Lotos standard. Second, we reformulate quotienting in terms of a simple synchronous product between a graph representation of the formula (called formula graph) and a process, thus enabling quotienting to be implemented efficiently and easily, by reusing existing tools dedicated to graph compositions. Third, we propose simplifications of the formula as a combination of bisimulations and reductions using Boolean equation systems applied directly to the formula graph, thus enabling formula simplifications also to be implemented easily and efficiently. Finally, we describe an implementation in the Cadp (*Construction and Analysis of Distributed Processes*) toolbox and present some experimental results in which partial model checking uses hundreds of times less memory than on-the-fly model checking.

1 Introduction

Concurrent safety critical systems can be verified using *model checking* [14], i.e., automatic evaluation of a temporal property against a model of the system. Although successful in many applications, model checking may face state explosion, particularly when the number of concurrent processes grows.

State explosion can be tackled by *divide-and-conquer* approaches regrouped under the vocable *compositional verification*, which take advantage of the compositional structure of the concurrent system. One such approach, which we call *compositional model generation* in this paper, consists in building the model

of the system — usually an LTS (*Labelled Transition System*) — in a stepwise manner, by successive compositions and minimisations modulo equivalence relations, possibly using *interface constraints* [23, 27] to avoid explosion of intermediate compositions. Tools using this approach [19, 28, 29, 16] are available in the CADP (*Construction and Analysis of Distributed Processes*) [20] toolbox.

In this paper, we explore a dual approach named *partial model checking*, proposed by Andersen [2, 4] for concurrent processes running asynchronously and composed using CCS parallel composition and restriction operators. For a modal μ-calculus [26] formula φ and a process composition $P_1||\ldots||P_n$, Andersen uses an operation $\varphi /\!\!/ P_1$ called *quotienting* of the formula φ w.r.t. the process P_1, so that $P_1||\ldots||P_n$ satisfies φ if and only if the smaller composition $P_2||\ldots||P_n$ satisfies $\varphi /\!\!/ P_1$. In addition, simplifications can (must) be applied to $\varphi /\!\!/ P_1$ to reduce its size. Partial model checking is the incremental application of quotienting and simplifications, so that state explosion is avoided if the size of intermediate formulas can be kept sufficiently small.

Partial model checking has been adapted and used successfully in various contexts, such as state-based models [5, 6], synchronous state/event systems [10], and timed systems [9, 12, 31–33]. It has also been specialised for security properties [34]. More recently, it has been generalised to the full CCS process algebra, with an application to the verification of parameterised systems [8].

In this paper, we focus on partial model checking of the modal μ-calculus applied to (untimed) concurrent asynchronous processes. By considering only binary associative composition operators, previous works [2, 4, 8] are not directly applicable to more general operators, such as m among n synchronisation and parallel composition by synchronisation interfaces [21], present in the E-LOTOS standard and variants [13, 25]. Our first contribution in this paper is thus a generalisation of partial model checking to networks of LTSs [28], a general model that subsumes parallel composition, hiding, cutting, and renaming operators of standard process languages (CCS, CSP, μCRL, LOTOS, E-LOTOS, etc.).

In realistic cases, partial model checking handles huge formulas and processes, thus requiring efficient implementations. Our second contribution is a reformulation of quotienting as a simple synchronous product, which can itself be represented in the network model, between a graph representing the formula (called a *formula graph*) and the behaviour graph of a process, thus enabling efficient implementation using existing tools dedicated to graph manipulations. Our third contribution is the reformulation of formula simplifications as a combination of graph reductions and partial evaluation of the formula graph using a BES (*Boolean Equation System*) [1]. Verifying modal μ-calculus formulas of arbitrary alternation depth is generally exponential in the size of the process graph, while verifying the alternation-free fragment remains of linear complexity. Our fourth contribution is a specialisation of the technique to alternation-free μ-calculus formulas. Finally, we present an implementation in CADP and a case-study that illustrates the complementarity between partial and on-the-fly model checking.

Paper Overview. The modal μ-calculus is presented in Sect. 2, networks of LTSs in Sect. 3, the generalisation of quotienting to networks and its reformulation as

a synchronous product in Sect. 4, simplification rules in Sect. 5, rules specific to alternation-free μ-calculus formulas in Sect. 6, our implementation in Sect. 7, a case study in Sect. 8, and concluding remarks in Sect. 9.

2 The Modal μ-Calculus

An LTS (*Labelled Transition System*) is a tuple $(\Sigma, A, \longrightarrow, s_0)$, with Σ a set of states, A a set of labels, $\longrightarrow \subseteq \Sigma \times A \times \Sigma$ the (labelled) transition relation, and $s_0 \in \Sigma$ the initial state. Properties of LTSs can be expressed in the modal μ-calculus [26], whose syntax and semantics are defined in the table below.

$\varphi ::=$	$\mathbf{ff}$	$[\![\mathbf{ff}]\!]\,\rho = \emptyset$
$\mid$	$\neg\varphi_0$	$[\![\neg\varphi_0]\!]\,\rho = \Sigma \setminus [\![\varphi_0]\!]\,\rho$
$\mid$	$\varphi_1 \vee \varphi_2$	$[\![\varphi_1 \vee \varphi_2]\!]\,\rho = [\![\varphi_1]\!]\,\rho \cup [\![\varphi_2]\!]\,\rho$
$\mid$	$\langle a \rangle\,\varphi_0$	$[\![\langle a \rangle\,\varphi_0]\!]\,\rho = \{s \in \Sigma \mid s \xrightarrow{a} s' \wedge s' \in [\![\varphi_0]\!]\,\rho\}$
$\mid$	X	$[\![X]\!]\,\rho = \rho(X)$
$\mid$	$\mu X.\varphi_0$	$[\![\mu X.\varphi_0]\!]\,\rho = \bigcap\{U \subseteq \Sigma \mid [\![\varphi_0]\!]\,(\rho \oslash [U/X]) \subseteq U\}$

Formulas (φ) are built from Boolean connectors, the possibility modality ($\langle _ \rangle$), and the minimal fix-point operator (μ) over propositional variables X. We write $\mathsf{fv}\,(\varphi)$ (resp. $\mathsf{bv}\,(\varphi)$) for the set of variables free (resp. bound) in φ and call a *closed formula* any formula φ s.t. $\mathsf{fv}\,(\varphi) = \emptyset$. We assume that all bound variables have distinct names, and for $X \in \mathsf{bv}\,(\varphi)$, we write $\varphi[X]$ for the (unique) sub-formula of φ of the form $\mu X.\varphi_0$. Given φ_1 and φ_2, we write $\varphi_1[\varphi_2/X]$ for substituting all free occurrences of X in φ_1 by φ_2. Derived operators are defined as usual: $\mathbf{tt} = \neg\mathbf{ff}$, $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $[a]\,\varphi_0 = \neg\,\langle a \rangle\,\neg\varphi_0$ (necessity modality), and $\nu X.\varphi_0 = \neg\mu X.\neg\varphi_0[\neg X/X]$ (maximal fix-point operator).

A propositional context ρ is a partial function mapping propositional variables to sets of states and $\rho \oslash [U/X]$ stands for a propositional context identical to ρ except that X is mapped to U. The interpretation $[\![\varphi]\!]\,\rho$ (also written $[\![\varphi]\!]$ if ρ is empty) of a state formula on an LTS in a propositional context ρ (which maps each variable free in φ to a set of states) denotes the subset of states satisfying φ in that context. The Boolean connectors are interpreted as usual in terms of set operations. The possibility modality $\langle a \rangle\,\varphi_0$ (resp. the necessity modality $[a]\,\varphi_0$) denotes the states for which some (resp. all) of their outgoing transitions labelled by a lead to states satisfying φ_0. The minimal fix-point operator $\mu X.\varphi_0$ (resp. the maximal fix-point operator $\nu X.\varphi_0$) denotes the least (resp. greatest) solution of the equation $X = \varphi_0$ interpreted over the complete lattice $\langle 2^{\Sigma}, \emptyset, \Sigma, \cap, \cup, \subseteq \rangle$. A state s satisfies a closed formula φ if and only if $s \in [\![\varphi]\!]$.

To ensure a proper definition of fix-point operators, it suffices that formulas φ are *syntactically monotonic* [26], i.e., have an even number of negations on every path between a variable occurrence X and the μ or ν operator that binds X. Negations can then be eliminated from formulas using the identities defining the derived operators. We write $\hat{\varphi}$ the formula obtained after eliminating all negations in φ. A formula φ is *alternation-free* if there is no sub-formula of $\hat{\varphi}$ of

the form $\mu X.\varphi_1$ (resp. $\nu X.\varphi_1$) containing a sub-formula of the form $\nu Y.\varphi_2$ (resp. $\mu Y.\varphi_2$) such that $X \in \mathsf{fv}(\varphi_2)$. The *fix-point sign* of a variable X in φ is μ (resp. ν) if $\hat{\varphi}[X]$ has the form $\mu X.\varphi$ (resp. $\nu X.\varphi$).

In this paper, we consider *block-labelled* formulas φ in which each propositional variable X is labelled by a unique natural number k, called its *block number*. Initially, we require that in every sub-formula of $\hat{\varphi}$ of the form $\mu X^k.\varphi_0$ (resp. $\nu X^k.\varphi_0$), every sub-formula $\mu Y^{k'}.\varphi_1$ (resp. $\nu Y^{k'}.\varphi_1$) satisfies $k' \geq k$, and every sub-formula $\nu Y^{k'}.\varphi_1$ (resp. $\mu Y^{k'}.\varphi_1$) satisfies $k' > k$. In addition, variables bound in disjoint sub-formulas may have the same block number only if they have the same fix-point sign, and by convention, block number 0 must be a μ-block (so that $k > 0$ in any formula $\nu X^k.\varphi$). We write $\mathsf{blocks}(\varphi)$ the set of block numbers occurring in φ. A block-labelled formula φ is *alternation-free* if $k' \geq k$ for all $X^k \in \mathsf{bv}(\varphi)$ and all $Y^{k'} \in \mathsf{fv}(\varphi[X^k])$. Any unlabelled formula is alternation-free if and only if it can be block-labelled to satisfy that constraint.

In the remainder of this paper, we will consider block-labelled formulas φ in *disjunctive form*, i.e., built only using the operators shown in the table above.

3 Networks of LTSs

Networks of LTSs (or *networks* for short) are inspired from the Mec [7] and Fc2 [11] synchronisation vectors and were introduced in [28] as an intermediate model to represent compositions of Ltss using various operators.

Background. We write $n..m$ for the set of integers ranging from n to m, or the empty set if $n > m$. A *vector* $\mathbf{v}$ of size n is a total function on $1..n$. For $i \in 1..n$, we write $\mathbf{v}[i]$ for $\mathbf{v}$ applied to i, denoting the element of $\mathbf{v}$ stored at index i. We write $(e_1, \ldots, e_n)$ for the vector $\mathbf{v}$ of size n such that $(\forall i \in 1..n)\ \mathbf{v}[i] = e_i$. In particular, $()$ is a vector of size 0. Given $n \geq 1$ and $i \in 1..n$, $\mathbf{v}_{\setminus i}$ denotes the projection of $\mathbf{v}$ on to the set of indices $1..n \setminus \{i\}$, defined as the vector of size $n-1$ such that $(\forall j \in 1..i-1)\ \mathbf{v}_{\setminus i}[j] = \mathbf{v}[j]$ and $(\forall j \in i..n-1)\ \mathbf{v}_{\setminus i}[j] = \mathbf{v}[j+1]$.

A *network of LTSs* N of size n is a pair $(\mathbf{S}, V)$, where $\mathbf{S}$ is a vector of Ltss (called *individual LTSs*) of size n, and V is a set of *synchronisation rules*, each rule having the form $(\mathbf{t}, a)$ with a a label and $\mathbf{t}$ a vector of size n, called the *synchronisation vector*, of labels and occurrences of a special symbol $\bullet$ distinct from any label. We write Σ_i, A_i, $\longrightarrow_i$, and s_i^0 for the sets of states and labels, the transition relation, and the initial state of $\mathbf{S}[i]$. N can be associated to a (global) Lts $\mathsf{lts}(N)$ which is the parallel composition of individual Ltss. Each $(\mathbf{t}, a) \in V$ defines transitions labelled by a, obtained either by synchronisation (if more than one index i is such that $\mathbf{t}[i] \neq \bullet$) or by interleaving (otherwise) of individual Lts transitions. Formally, $\mathsf{lts}(N) = (\Sigma, A, \longrightarrow, \mathbf{s}_0)$, where $\Sigma = \Sigma_1 \times \ldots \times \Sigma_n$, $A = \{a \mid (\mathbf{t}, a) \in V\}$, $\mathbf{s}_0 = (s_1^0, \ldots, s_n^0)$, and $\longrightarrow$ is the smallest relation satisfying:

$$(\mathbf{t}, a) \in V \wedge (\forall i \in 1..n) \begin{pmatrix} (\mathbf{t}[i] = \bullet \wedge \mathbf{s}'[i] = \mathbf{s}[i]) \vee \\ (\mathbf{t}[i] \neq \bullet \wedge \mathbf{s}[i] \xrightarrow{\mathbf{t}[i]}_i \mathbf{s}'[i]) \end{pmatrix} \Rightarrow \mathbf{s} \xrightarrow{a} \mathbf{s}'$$

$A(\mathbf{t})$ denotes the set of *active* Lts (indices), defined by $\{i \mid i \in 1..n \wedge \mathbf{t}[i] \neq \bullet\}$.

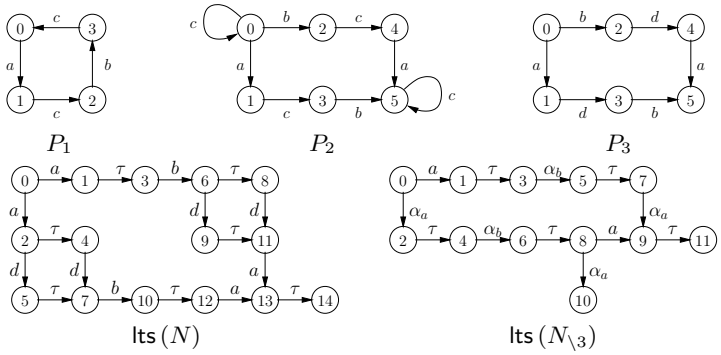

Fig. 1. Labelled Transition Systems for N defined in Ex. 1

Example 1. Let a, b, c, and d be labels, and P_1, P_2, and P_3 be the processes defined in Fig. 1 (top), where 0's denote initial states. Let $N = ((P_1, P_2, P_3), V)$ with $V = \{((a, a, \bullet), a), ((a, \bullet, a), a), ((b, b, b), b), ((c, c, \bullet), \tau), ((\bullet, \bullet, d), d)\}$, whose global LTS is in Fig. 1 (bottom left). The first two rules express a nondeterministic synchronisation on a between either P_1, P_2 or P_1, P_3. The third rule expresses a multiway synchronisation on b. The fourth rule yields an internal (τ) transition. The fifth rule expresses full interleaving of transitions labelled by d.

The network of LTSS model subsumes most hiding, renaming, cutting, and parallel composition operators present in process algebras (CCS, CSP, LOTOS, μCRL, etc.), but also more expressive operators, such as m among n synchronisation and parallel composition using synchronisation interfaces [21] present in E-LOTOS [25] and LOTOS NT [13]. For instance, the rules $\{((a, a, \bullet), a), ((a, \bullet, a), a), ((\bullet, a, a), a)\}$ realize 2 among 3 synchronisation on a.

Sub-network extraction. Computing the interactions of a process P_i with its environment in a composition of processes $||_{j \in 1..n} P_j$ is easy when $||$ is a binary and associative parallel composition operator, since $||_{j \in 1..n} P_j = P_i \,||\, (||_{j \in 1..n \setminus \{i\}} P_j)$. However, as argued in [21], binary and associative parallel composition operators are of limited use when considering, e.g., m among n synchronisation. A more involved operation named *sub-network extraction* is necessary for networks. $N = (\mathbf{S}, V)$ being a network of size n, we assume a function $\alpha\,(\mathbf{t}, a)$ that assigns an unused label to each $(\mathbf{t}, a) \in V$. Given $i \in 1..n$, we define $N_{\setminus i} = (\mathbf{S}_{\setminus i}, V_{\setminus i})$ the sub-network of N modeling the environment of $\mathbf{S}[i]$ in N, where $V_{\setminus i} = \{(\mathbf{t}_{\setminus i}, a) \mid (\mathbf{t}, a) \in V \wedge i \notin A(\mathbf{t})\} \cup \{(\mathbf{t}_{\setminus i}, \alpha\,(\mathbf{t}, a)) \mid (\mathbf{t}, a) \in V \wedge \{i\} \subset A(\mathbf{t})\}$. N is semantically equivalent to the network $((\mathbf{S}[i], \mathsf{lts}\,(N_{\setminus i})), V')$ with V' the following set of rules, which define the interactions between $\mathbf{S}[i]$ and $N_{\setminus i}$:

$$
\begin{array}{l}
\{\ ((\bullet, \quad a), \qquad a) \mid (\mathbf{t}, a) \in V \wedge i \notin A(\mathbf{t}) \quad \} \cup \\
\{\ ((\mathbf{t}[i], \alpha\,(\mathbf{t}, a)), a) \mid (\mathbf{t}, a) \in V \wedge \{i\} \subset A(\mathbf{t})\ \} \cup \\
\{\ ((a, \quad \bullet), \qquad a) \mid (\mathbf{t}, a) \in V \wedge \{i\} = A(\mathbf{t})\ \}
\end{array}
$$

Each $\alpha(\mathbf{t}, a)$ is a unique interaction label between $\mathbf{S}[i]$ and $N_{\setminus i}$, which aims at avoiding erroneous interactions in case of nondeterministic synchronisation.

Example 2. N being defined in Ex. 1, $N_{\backslash 3}$ has vector of LTSs (P_1, P_2) and rules $\{((a,a),a),((a,\bullet),\alpha_a),((b,b),\alpha_b),((c,c),\tau)\}$ with $\alpha_a = \alpha\,((a,\bullet),a),a)$ and $\alpha_b = \alpha\,((b,b,b),b)$; $\mathsf{lts}(N_{\backslash 3})$ is depicted in Fig. 1 (bottom right); Composing it with P_3 using $\{((\bullet,a),a),((a,\alpha_a),a),((b,\alpha_b),b),((\bullet,\tau),\tau),((d,\bullet),d)\}$ yields $\mathsf{lts}(N)$.

Note that if a had been used instead of α_a in the above synchronisation rules, then the composition of $N_{\backslash 3}$ with P_3 would have enabled, in addition to the (correct) binary synchronisations on a between P_1 and P_2 and between P_1 and P_3, the (incorrect) multiway synchronisation on a between the three of P_1, P_2, and P_3. Indeed, the label a resulting from the synchronisation between P_1 and P_2 in $N_{\backslash 3}$ — rule $((a,a),a)$ in $N_{\backslash 3}$ — could synchronise with the label a in P_3 — rule $((a,a),a)$ in the composition between $N_{\backslash 3}$ and P_3. Note however that $\mathbf{t}[i]$ can be used instead of $\alpha(\mathbf{t},a)$ when the network does not have nondeterministic synchronisation on $\mathbf{t}[i]$, as is the case for b and α_b in this example. In this paper we use $\alpha(\mathbf{t},a)$ uniformly to avoid complications.

4 Quotienting for Networks using Networks

To check a closed formula φ on a network $N = (\mathbf{S}, V)$, one can choose an LTS $\mathbf{S}[i]$, compute the quotient of the formula φ with respect to $\mathbf{S}[i]$, and check the resulting quotient formula on the smaller (at least in number of individual LTSs, but also hopefully in global LTS size) network $N_{\backslash i}$. The quotient formula is written $\varphi \mathbin{/\!\!/}_i^{\emptyset}\, s_0^i$ and defined as follows for formulas in disjunctive form:

$$\begin{array}{rcl}
\mathbf{ff} \mathbin{/\!\!/}_i^B\, s = \mathbf{ff} & & X^k \mathbin{/\!\!/}_i^B\, s = \varphi[X^k] \mathbin{/\!\!/}_i^B\, s \\
(\neg\varphi_0) \mathbin{/\!\!/}_i^B\, s = \neg(\varphi_0 \mathbin{/\!\!/}_i^B\, s) & & (\varphi_1 \vee \varphi_2) \mathbin{/\!\!/}_i^B\, s = (\varphi_1 \mathbin{/\!\!/}_i^B\, s) \vee (\varphi_2 \mathbin{/\!\!/}_i^B\, s) \\
(\mu X^k.\varphi_0) \mathbin{/\!\!/}_i^B\, s = \begin{cases} X_s^k & \text{if } X_s^k \in B \\ \mu X_s^k.(\varphi_0 \mathbin{/\!\!/}_i^{B\cup\{X_s^k\}}\, s) & \text{otherwise} \end{cases} & &
\end{array}$$

$$(\langle a\rangle\varphi_0) \mathbin{/\!\!/}_i^B\, s = \bigvee_{(\mathbf{t},a)\in V} \begin{pmatrix} (\quad i \notin A(\mathbf{t}) \wedge & \langle a\rangle\,(\varphi_0 \mathbin{/\!\!/}_i^B\, s)\,) & \vee \\ (\{i\} \subset A(\mathbf{t}) \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} & \langle \alpha\,(\mathbf{t},a)\rangle\,(\varphi_0 \mathbin{/\!\!/}_i^B\, s')) & \vee \\ (\{i\} = A(\mathbf{t}) \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} & (\varphi_0 \mathbin{/\!\!/}_i^B\, s')) & \end{pmatrix}$$

This definition generalises Andersen's [2], specialised for CCS, to networks. The major difference is the definition of $(\langle a\rangle\varphi_0) \mathbin{/\!\!/}_i^B\, s$, CCS composition corresponding to vectors $((a,\bullet),a)$, $((\bullet,a),a)$, or $((a,\overline{a}),\tau)$, a and $\overline{a}$ being an action and its *co-action*, making the use of special labels $\alpha(\mathbf{t},a)$ not necessary. A slightly minor difference is that we use μ-calculus terms instead of equations. Any sub-formula produced by quotienting has the same block number as the original sub-formula, reflecting the order of equation blocks in Andersen's work. The set B keeps track of new variables already introduced in the quotient formula. Quotienting is well-defined, because formulas are finite, every $\varphi[X^k]$ has the form $\mu X^k.\varphi_0$, and the size of the set B is bounded by $|\,\mathsf{bv}\,(\varphi)| \times |\Sigma_i|$.

Example 3. The μ-calculus formula $\mu X^0.\langle a\rangle\mathbf{tt} \vee \langle b\rangle X^0$ (existence of a path of zero or more b leading to an a) can be rewritten to disjunctive form as $\mu X^0.\langle a\rangle\neg\mathbf{ff} \vee \langle b\rangle X^0$. Quotienting of this formula with respect to P_3 in the network N introduced in Ex. 1 yields the formula $\mu X_0^0.\langle a\rangle\neg\mathbf{ff} \vee \langle\alpha_a\rangle\neg\mathbf{ff} \vee \langle\alpha_b\rangle\mu X_2^0.\langle a\rangle\neg\mathbf{ff} \vee \mathbf{ff}$.

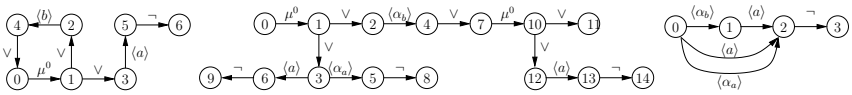

Fig. 2. Examples of formula graphs

We now show that quotienting can be implemented as a network that realises a product between an LTS encoding the formula (called a *formula graph*) and an individual LTS of the network under verification. The formula graph corresponding to a formula φ in disjunctive form is an LTS whose states are identified with sub-formulas of φ and whose transitions are labelled by $\vee$, $\neg$, μ^k (k being a block number), and $\langle a \rangle$ (a being any action of the network under verification). The initial state of the formula graph is φ, **ff** is a deadlock state, and each sub-formula has transitions as follows:

$$X^k \xrightarrow{\vee} \varphi[X^k] \qquad \neg\varphi_0 \xrightarrow{\neg} \varphi_0 \qquad \langle a \rangle \varphi_0 \xrightarrow{\langle a \rangle} \varphi_0$$
$$\varphi_1 \vee \varphi_2 \xrightarrow{\vee} \varphi_1 \qquad \varphi_1 \vee \varphi_2 \xrightarrow{\vee} \varphi_2 \qquad \mu X^k.\varphi_0 \xrightarrow{\mu^k} \varphi_0$$

Formula graphs are finite, connected, and every circular path (i.e., from one state to itself) contains at least one transition that is labelled by μ^k. We write $\mathsf{enc}\,(\varphi)$ the formula graph of φ. Conversely, every formula graph $P = (S, A, \rightarrow, s_0)$ can be decoded into the closed formula $\mathsf{dec}\,(P, s_0, \emptyset)$ as follows, where E is a mapping of the form $\{s \mapsto k \mid s \in \Sigma \wedge k \in \mathbb{N}\}$:

$$\mathsf{dec}\,(P, s, E) = \begin{cases} \mathsf{X}^k_s & \text{if } s \mapsto k \in E \\ \bigvee_{s \xrightarrow{\sigma} s'} \delta^s_\sigma(P, s', E) & \text{otherwise} \end{cases} \qquad \text{where}$$
$$\delta^s_\vee(P, s', E) = \mathsf{dec}\,(P, s', E) \qquad \delta^s_\neg(P, s', E) = \neg\,\mathsf{dec}\,(P, s', E)$$
$$\delta^s_{\langle a \rangle}(P, s', E) = \langle a \rangle\,\mathsf{dec}\,(P, s', E) \quad \delta^s_{\mu^k}(P, s', E) = \mu \mathsf{X}^k_s.\,\mathsf{dec}\,(P, s', E \cup \{s \mapsto k\})$$

This definition implies that a deadlock state decodes as **ff** (empty disjunction). dec is well-defined, the mapping E ensuring termination. Although the states of a formula graph are identified by formulas, only the transition labels are required for decoding. In figures, states will be be simply identified by numbers.

Example 4. The formula graph corresponding to the formula $\mu X^0.(\langle a \rangle \mathbf{tt}) \vee \langle b \rangle X^0$ introduced in Ex. 3 is depicted in Fig. 2 (left), where 0 denotes the initial state.

Proposition 1. *If φ is a closed formula, then $\mathsf{dec}\,(\mathsf{enc}\,(\varphi), \varphi, \emptyset) = \varphi$, modulo commutativity ($\phi_1 \vee \phi_2 = \phi_2 \vee \phi_1$), idempotence ($\phi \vee \phi = \phi$), and renaming of each propositional variable $X^k \in \mathsf{bv}\,(\varphi)$ into $\mathsf{X}^k_{\varphi[X^k]}$.*

Proof. This is a corollary of the more general property stating that for every sub-formula ϕ of φ, if $\{\varphi[Y^k] \mapsto k \mid Y^k \in \mathsf{fv}\,(\phi)\} \subseteq E$ and $E \cap \{\varphi[Y^k] \mapsto k \mid Y^k \in \mathsf{bv}\,(\phi)\} = \emptyset$, then $\mathsf{dec}\,(\mathsf{enc}\,(\varphi), \phi, E) = \phi$ (structural induction on ϕ).

Using this encoding, the quotienting of a formula φ with respect to the ith LTS of a network $N = (\mathbf{S}, V)$ can be realised as a synchronous product, using

the network $((\mathsf{enc}\,(\varphi), \mathbf{S}[i]), V /\!\!/_i)$, where $V /\!\!/_i$ denotes the following set of rules:

$$\begin{array}{lll} \{\ ((\sigma, \ \bullet), \ \ \sigma) & \mid \sigma \in \{\neg, \vee\} \cup \{\mu^k \mid k \in \mathsf{blocks}(\varphi)\}\ \} & \cup \\ \{\ ((\langle a\rangle, \bullet), \ \ \langle a\rangle) & \mid (\mathbf{t}, a) \in V \wedge i \notin A(\mathbf{t})\ \} & \cup \\ \{\ ((\langle a\rangle, \mathbf{t}[i]), \langle \alpha\,(\mathbf{t}, a)\rangle) & \mid (\mathbf{t}, a) \in V \wedge \{i\} \subset A(\mathbf{t})\ \} & \cup \\ \{\ ((\langle a\rangle, \mathbf{t}[i]), \vee) & \mid (\mathbf{t}, a) \in V \wedge \{i\} = A(\mathbf{t})\ \} & \end{array}$$

Proposition 2. *If* $P = \mathsf{lts}\,((\mathsf{enc}\,(\varphi), \mathbf{S}[i]), V /\!\!/_i)$ *then* $\mathsf{dec}\,(P, (\varphi, s_0^i), \emptyset) = \varphi /\!\!/_i^{\emptyset}\ s_0^i$*, modulo commutativity, idempotence, and renaming of each propositional variable* $Y_t^k \in \mathsf{bv}\,(\varphi /\!\!/_i^B\ s_0^i)$ *into* $\mathsf{X}^k_{(\varphi[Y^k], t)}$

Proof. A state of P has the form (ϕ, s), where ϕ is a sub-formula of φ and s is a state of $\mathbf{S}[i]$. The proof uses a slighty more general lemma: if $E = \{(\varphi[Y^k], t) \mapsto k \mid Y_t^k \in B\}$ then $\mathsf{dec}\,(P, (\phi, s), E) = \phi /\!\!/_i^B\ s$ (structural induction on $\phi /\!\!/_i^B\ s$).

Example 5. Consider the network N of Ex. 1 and the formula of Ex. 4. Quotienting of the formula with respect to P_3 involves the following set of rules:
$\{((\neg, \bullet), \neg), ((\vee, \bullet), \vee), ((\mu^0, \bullet), \mu^0), ((\langle a\rangle, \bullet), \langle a\rangle), ((\langle a\rangle, a), \langle \alpha_a\rangle), ((\langle b\rangle, b), \langle \alpha_b\rangle)\}$
It yields the formula graph depicted in Fig. 2 (middle). This graph encodes as expected the quotient formula of Ex. 3, which can be evaluated on $N_{\backslash 3}$.

Working with formulas in disjunctive form is crucial: branches in the formula graph denote disjunctions between sub-formulas (*or-nodes*). During composition between the formula graph and an individual LTS, the impossibility to synchronise on a modality $\langle a\rangle$ (no transition labelled by $\mathbf{t}[i]$ in the current state of the individual LTS) denotes invalidation of the corresponding sub-formula, which merely disappears, in conformance with the equality $\mathbf{ff} \vee \varphi_0 = \varphi_0$.

5 Formula Graph Simplifications

The size (number of states) of a formula graph of size n quotiented with respect to an LTS of size m is bounded by $n \times m$. Hence, as observed by Andersen [2], simplifications are needed to keep intermediate quotiented formulas at a reasonable size. We present in Fig. 3 several simplifications applying to formula graphs, as conditional rules of the form "$l \rightsquigarrow r$ (*cond*)" where l and r are subsets of transition relations, such that every variable representing a state (written $s, s_1, s_2, \ldots$) or a label (written $\sigma, \sigma_1, \sigma_2, \ldots$) in r or in the condition *cond* must occur in l. It means that all transitions matching the left-hand side so that *cond* is satisfied can be replaced by the transitions of the right-hand side.
Elimination of $\vee$*-transitions* (1). This rule is essential to eliminate the transitions labelled by $\vee$ introduced by synchronisation rules of the form $((\langle a\rangle, \mathbf{t}[i]), \vee)$ during quotienting. It can be achieved efficiently by applying reduction modulo $\tau^*.a$ equivalence [17], $\vee$-transitions being interpreted as internal (τ) transitions.
Elimination of double-negations (2). This rule can be used after the previous one to simplify formulas of the form $\neg\neg\varphi$, which may appear, e.g., in the quotienting of $\neg\langle a\rangle\neg\varphi'$ with an LTS that offers an action synchronising with a.

(1)	$s_1 \xrightarrow{\vee} s_2 \xrightarrow{\sigma} s_3 \rightsquigarrow s_1 \quad s_2 \xrightarrow{\sigma} s_3$, $s_1 \xrightarrow{\sigma} s_3$		
(2)	$s_1 \xrightarrow{\neg} s_2 \xrightarrow{\neg} s_3 \rightsquigarrow s_1 \quad s_2 \xrightarrow{\neg} s_3$, $s_1 \xrightarrow{\vee} s_3$		(s_2 has a single outgoing transition)
(3)	$s_1 \xrightarrow{\mu^k} s_2 \rightsquigarrow s_1 \xrightarrow{\vee} s_2$		(decoding of s_2 does not contain $\mathsf{X}^k_{s_1}$)
(4)	$s_1 \xrightarrow{\neg} s_2 \rightsquigarrow s_1 \quad s_2$		(s_2 evaluates to $\mathbf{tt}$)
(5)	$s_1 \xrightarrow{\sigma_2} s_2 \cdots \xrightarrow{\sigma_n} s_n \rightsquigarrow s_1 \quad s_2 \cdots s_n$, $s_1 \xrightarrow{\neg} \mathbf{ff}$		(s_1 evaluates to $\mathbf{tt}$)
(6)	$s_1 \xrightarrow{\sigma} s_2 \rightsquigarrow s_1 \quad s_2$		($\sigma \neq \neg$ and s_2 evaluates to $\mathbf{ff}$)
(7)	$s_1 \xrightarrow{\sigma_2} s_2 \cdots \xrightarrow{\sigma_n} s_n \rightsquigarrow s_1 \quad s_2 \cdots s_n$		(s_1 evaluates to $\mathbf{ff}$)

Fig. 3. Simplification rules applying to formula graphs

Elimination of μ-transitions (3). The transition from s_1 to s_2 denotes a propositional variable $\mathsf{X}^k_{s_1}$, which does not occur free in the formula if at least one of the following sufficient (and checkable in linear time) conditions holds: (i) s_1 and s_2 are not in the same strongly connected component; (ii) s_1 satisfies the recursive condition "*s_1 has a single predecessor p, distinct from the initial state, and either p has a single μ-transition to s_1 or p satisfies this condition, recursively*". This condition is well-founded as long as it is applied to reachable states.
Evaluation of constant sub-formulas (4–7). To decide whether a state denotes a sub-formula that evaluates to a constant in any context, we consider the following BES, consisting in blocks T^k and F^k ($k \in 0..n$) of respective signs μ and ν, n being the greatest block number in the formula graph. Blocks are ordered so that $k < k'$ implies T^k (resp. F^k) is before $T^{k'}$ (resp. $F^{k'}$):

$$T^k : \left\{ T^k_s =_\mu \bigvee_{s \xrightarrow{\vee} s'} T^k_{s'} \vee \bigvee_{s \xrightarrow{\neg} s'} F^k_{s'} \vee \bigvee_{s \xrightarrow{\mu^{k'}} s'} T^{k'}_{s'} \right\}_{s \in \Sigma}$$
$$F^k : \left\{ F^k_s =_\nu \bigwedge_{s \xrightarrow{\vee} s'} F^k_{s'} \wedge \bigwedge_{s \xrightarrow{\langle\beta\rangle} s'} F^k_{s'} \wedge \bigwedge_{s \xrightarrow{\neg} s'} T^k_{s'} \wedge \bigwedge_{s \xrightarrow{\mu^{k'}} s'} F^{k'}_{s'} \right\}_{s \in \Sigma}$$

We consider only the variables reachable from $T^0_{s_0}$ or $F^0_{s_0}$, s_0 being the initial state of the formula graph. A state s denotes $\mathbf{tt}$ (resp. $\mathbf{ff}$) if the Boolean variables T^k_s (resp. F^k_s) evaluate to $\mathbf{tt}$ in all (reachable) blocks k. Due to the presence of modalities, there may be states s and blocks k such that T^k_s and F^k_s are both false, indicating that the corresponding sub-formula is not constant. Intuitively, T^k_s expresses that s evaluates to $\mathbf{tt}$ in block k if one of its successors following a transition labelled by $\vee$ or $\mu^{k'}$ evaluates to $\mathbf{tt}$, or one of its successors following a transition labelled by $\neg$ evaluates to $\mathbf{ff}$. Variable F^k_s expresses that state s

evaluates to **ff** in block k if all its successors following transitions labelled by $\vee$, $\mu^{k'}$, or modalities (by applying the identity $\langle a \rangle \mathbf{ff} = \mathbf{ff}$) evaluate to **ff** and all its successors following transitions labelled by $\neg$ evaluate to **tt**. Regarding fix-point signs, observe that $F^k_{\mu X^k.X^k} =_\nu F^k_{\mu X^k.X^k}$ and $T^k_{\mu X^k.X^k} =_\mu T^k_{\mu X^k.X^k}$ respectively evaluate to **tt** and **ff**, reflecting that $\mu X^k.X^k$ evaluates to **ff** as expected.

Repeated applications of quotienting progressively eliminate modalities, until none of them remains in the formula graph, which then necessarily evaluates to a constant equal to the result of evaluating the formula on the whole network.

Sharing of equivalent sub-formulas. In addition to the above rules, reducing a formula graph modulo strong bisimulation does not change its decoding, modulo idempotence, renaming of propositional variables, and unification of equivalent variables defined in the same block. Strong bisimulation reduction can thus decrease the size of intermediate formula graphs. The reader may note that the heuristic to determine that two variables denote equivalent sub-formulas given in Andersen's work [2] is similar to the definition of strong bisimulation on LTSs.

A careful comparison between the simplifications proposed by Andersen [2] and ours would be useful and is left for further work.

Example 6. After applying the above simplifications to the formula graph of Ex. 5, we obtain the (smaller) formula graph depicted in Fig. 2 (right), which corresponds to the formula $(\langle a \rangle \mathbf{tt}) \vee (\langle \alpha_a \rangle \mathbf{tt}) \vee (\langle \alpha_b \rangle \langle a \rangle \mathbf{tt})$.

Example 7. The graph corresponding to $\mu X^0.(\langle a \rangle \mu Y^0.\langle b \rangle X^0) \vee \langle c \rangle X^0$ reduces as expected to a deadlock state representing the constant **ff** (left as an exercise).

6 Simplification of Alternation-Free Formula Graphs

Simplifications apply to μ-calculus formulas of arbitrary alternation depth. We focus here on the alternation-free μ-calculus fragment, which has a linear-time model checking complexity [15] and is therefore more suitable for scaling up to large LTSs. We propose a variant of constant sub-formula evaluation specialised for alternation-free formulas, using alternation-free BESs [1].

Even in the case of alternation-free formulas, the above BES is not alternation-free due to the cyclic dependency between T^k and F^k, e.g., when evaluating sequences of $\neg$-transitions. In Fig. 4, we propose a refinement of this BES, which splits each variable T^k_s of sign μ into two variables T^{+k}_s of sign μ and F^{-k}_s of sign ν, which evaluate to true iff the sub-formula corresponding to state s is preceded by an even (for T^{+k}_s) or odd (for F^{-k}_s) number of negations and evaluates to true. Variable F^k_s is split similarly. This BES is a generalisation, for formula graphs containing negations and modalities, of the BES characterising the solution of alternation-free Boolean graphs outlined in [35].

For general formulas, this BES is not alternation-free due to the cyclic dependencies between T^k and $F^{k'}$, of different fix-point signs. Yet, for alternation-free block-labelled formulas, it is alternation-free, since each dependency from T^k to $F^{k'}$ (or from F^k to $T^{k'}$) always traverses a μ-transition preceded by an odd number of negations, which switches to a different block number $k' > k$.

$$T^k : \left\{ \begin{array}{l} T_s^{+k} =_\mu \bigvee_{s \xrightarrow{\vee} s'} T_{s'}^{+k} \vee \bigvee_{s \xrightarrow{\neg} s'} T_{s'}^{-k} \vee \bigvee_{s \xrightarrow{\mu k'} s'} T_{s'}^{+k'} \\ T_s^{-k} =_\mu \bigwedge_{s \xrightarrow{\vee} s'} T_{s'}^{-k} \wedge \bigwedge_{s \xrightarrow{\langle\beta\rangle} s'} T_{s'}^{-k} \wedge \bigwedge_{s \xrightarrow{\neg} s'} T_{s'}^{+k} \wedge \bigwedge_{s \xrightarrow{\mu k'} s'} F_{s'}^{+k'} \end{array} \right\}_{s \in \Sigma}$$

$$F^k : \left\{ \begin{array}{l} F_s^{+k} =_\nu \bigwedge_{s \xrightarrow{\vee} s'} F_{s'}^{+k} \wedge \bigwedge_{s \xrightarrow{\langle\beta\rangle} s'} F_{s'}^{+k} \wedge \bigwedge_{s \xrightarrow{\neg} s'} F_{s'}^{-k} \wedge \bigwedge_{s \xrightarrow{\mu k'} s'} F_{s'}^{+k'} \\ F_s^{-k} =_\nu \bigvee_{s \xrightarrow{\vee} s'} F_{s'}^{-k} \vee \bigvee_{s \xrightarrow{\neg} s'} F_{s'}^{+k} \vee \bigvee_{s \xrightarrow{\mu k'} s'} T_{s'}^{+k'} \end{array} \right\}_{s \in \Sigma}$$

Fig. 4. BES for the evaluation of constant alternation-free formulas

7 Implementation

We have implemented partial model checking of alternation-free μ-calculus formulas using CADP, which provided much of what was needed:

- Individual processes can be described in the language LOTOS [24], or in the LOTOS NT variant of E-LOTOS [25], among others, for which CADP contains tools to generate LTSs automatically.
- Process compositions can be described in the EXP.OPEN 2.0 language [28], which provides various parallel composition operators, such as synchronisation vectors [7], process algebra operators (LOTOS, CCS, CSP, μCRL), and the generalised parallel composition operator of E-LOTOS [21]. It also provides generalised operators for hiding, renaming, and cutting labels based on a representation of label sets using regular expressions. The EXP.OPEN 2.0 tool compiles its input into a network of LTSs. It then generates C code for representing the transition relation [18], so that the LTS can be either generated or traversed on-the-fly using various libraries. For partial model checking, the EXP.OPEN 2.0 tool has been slightly extended both to implement sub-network extraction and to generate the network representing the parallel composition between the formula graph and a chosen individual LTS.
- Alternation-free μ-calculus formulas can be handled by the EVALUATOR 3.5 on-the-fly model checker [38], in which an option has been added for compiling a formula into a formula graph.
- Reductions modulo $\tau^*.a$ equivalence and strong bisimulation are achieved using respectively the REDUCTOR and BCG_MIN tools of CADP.

Elimination of double-negations, of μ-transitions, and evaluation of constant formulas have been implemented in a new prototype tool (1,000 lines of C code), which relies on the CAESAR_SOLVE library [37] for solving alternation-free BES. Finally, selection of the LTS w.r.t. which the formula is quotiented at each step is done using the principles described in [16] for networks of LTSs.

8 Experimentation

We have used partial model checking in a case-study in avionics, namely the verification of a communication protocol between a plane and the ground, based on TFTP (*Trivial File Transfer Protocol*)/UDP (*User Datagram Protocol*) [22].

The system consists in two instances of the TFTP connected by UDP using a FIFO buffer. We considered five scenarios, named A to E, depending whether each instance may write and/or read a file. We also checked the (alternation-free) μ-calculus (branching-time) properties named $A01$ to $A28$, studied in [22], both using the well-established on-the-fly model checker EVALUATOR 3.5 [38] of CADP and using the partial model checking approach described in this paper. These experiments were done on a 64-bit computer with 148 gigabytes of memory.

The results summarized in Tab. 1 give, for each scenario, the LTS size in kilostates (ks), and for each property, the peak of memory in megabytes (MB) used by on-the-fly model checking (column fly) and partial model checking (column pmc). Some properties being irrelevant to some scenarios (e.g., they concern a read or write operation absent in the corresponding scenario), they have not been checked, explaining the shaded cells. The symbol "$\star$" corresponds to unfinished verifications that used too much memory. For lack of space, times are not reported but each partial model checking experiment that used less than 100 MB of memory took from a few seconds to less than a minute. Note that the major part of time and memory are used by formula simplifications, as compared to the low complexity of the synchronous product operation used for quotienting.

These results confirm that partial model checking may be much more efficient (up to 600 times less memory in this example) than on-the-fly model checking. For several properties, we observe that partial model checking sometimes allows complete evaluation of formulas before they have been quotiented with respect to all individual LTSs, because the truth value of the formula is independent of some individual LTS. However, in a few cases, partial model checking leads to combinatorial explosion (properties $A12$, $A13$, $A15$, and $A17$) while on-the-fly model checking is efficient. This is inherent to the structure of the system, intermediate quotients needing to capture a large part of the behaviour before the truth value of the formula can be computed. This shows that both approaches are complementary and worthy of being used concurrently.

9 Conclusion

The original contributions of this paper are the following: (1) Partial model checking has been generalised to the network model, which subsumes many parallel composition operators. (2) An efficient implementation of quotienting with respect to an individual LTS has been proposed, using a simple synchronous product between this LTS and a graph representation of the formula. A key is the representation of the formula in a disjunctive form (using negations), which turns every node of the formula graph into an *or-node*. (3) An efficient implementation of formula simplifications has also been proposed, using a combination of existing algorithms (such as reductions modulo equivalence relations), simple transformations, and traversals of the formula graph using a BES. Using a graph equivalence relation to simplify the formula was already proposed in [8], where the formula was translated into an *and-or-graph* and then reduced modulo strong bisimulation. We use a weaker relation ($\tau^*.a$ equivalence) that

	Scenario A 1,963 ks		Scenario B 867 ks		Scenario C 35,024 ks		Scenario D 40,856 ks		Scenario E 19,436 ks	
Prop	fly	pmc	fly	pmc	fly	pmc	fly	pmc	fly	pmc
$A01$	199	6	89	6	2,947	24	3,351	27	1,530	23
$A02$	207	6	93	6	3,156	25	3,631	28	1,612	10
$A03$	182	6	80	6	2,737	6	3,162	6	1,386	6
$A04$	199	6	89	6	2,947	6	3,351	29	1,530	7
$A05$	10	6	7	6	7	6	7	6	10	10
$A06$	187	6	85	6	2,808	6	3,249	7	1,428	6
$A07$	187	6	85	6	2,808	6	3,249	6	1,428	6
$A08$	186	6	80	6	2,745	6	3,170	6	1,390	6
$A09a$							3,290	28	1,488	6
$A09b$					2,955	6				
$A10$					3,354	6			1,674	6
$A11$					3,206	6	4,444	7	1,711	6
$A12$					620	$\star$	133	$\star$	101	$\star$
$A13$							4,499	$\star$	2,094	$\star$
$A14$	267	6			3,988	23			2,107	15
$A15$			118	15	521	$\star$	156	$\star$	1,524	59
$A16$									186	8
$A17$					667	$\star$	569	2,702		
$A18$			85	6	476	11	255	6	1,391	6
$A19$			207	6	6,352	90	8,753	13	3,104	55
$A20$	31	9			837	21			261	25
$A21$	374	6			4,958	25			2,817	25
$A22$			35	7			427	1,271	191	650
$A23$			170	6			6,909	9	3,039	40
$A24$	41	9			427	1,786				
$A25$	391	6			5,480	40				
$A26$	195	6			2,857	15			1,477	10
$A27$	228	6			3,534	6			1,871	6
$A28$			102	6	3,654	22	4,032	6	1,821	6

Table 1. Experimental results for the TFTP/UDP case study

enables more reduction of the formula graph, and we apply it directly on simple LTSs, thus allowing efficient LTS reduction tools to be used without any modification. Our simplifications integrate smoothly in the approach, both quotienting and simplifications applying to the same graph representation, without encoding and decoding formulas back and forth. (4) A specialisation to the case of alternation-free formulas (using alternation-free BES) has also been presented, showing that partial model checking may result in much better performance than complementary approaches, such as on-the-fly model checking. Only small software developments were required, thanks to the wealth of functionalities available in CADP. The approach would be also applicable to formulas of arbitrary alternation depth using a solver for BES of arbitrary alternation depth.

The implementation of quotienting as a synchronous product opens the way for combining partial model checking with techniques originating from compositional model generation, such as (compositional) τ-confluence reduction [30, 36, 40], or restriction using interface constraints following the approach developed in [23] and refined in [19, 27, 29]. Note also that partial model checking and compositional model generation are complementary. Although it is difficult in

general to know which of them will be most efficient, a reasonable methodology is to try compositional model generation first (because one then obtains a single model on which all formulas of interest can be evaluated). In case of failure, partial model checking can then be used for each formula.

As future work, we also plan to study partial model checking of certain μ-calculus formulas of alternation depth 2 describing the existence of complex cycles (e.g., $\nu X.\mu Y.(\langle b\rangle X \vee \langle a\rangle Y)$, expressing the infinite repetition of sequences belonging to the regular language $a^*.b$), which can still be checked in linear-time using specialised BES resolution algorithms [39] generalising the detection of accepting cycles in Büchi automata.

References

1. H.R. Andersen. Model checking and Boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.
2. H.R. Andersen. Partial Model Checking. In *Proc. of Logic in Computer Science LICS*. IEEE Computer Society Press, 1995.
3. H.R. Andersen and J. Lind-Nielsen. MuDiv: A Tool for Partial Model Checking. In *Proc. of CONCUR*, 1996.
4. H.R. Andersen and J. Lind-Nielsen. Partial Model Checking of Modal Equations: A Survey. *STTT*, 2(1999):242–259, 1999.
5. H.R. Andersen, J. Staunstrup, and N. Maretti. Partial Model Checking with ROBDDs. In *Proc. of TACAS*, 1997.
6. H.R. Andersen, J. Staunstrup, and N. Maretti. A Comparison of Modular Verification. In *Proc. of FASE*, 1997.
7. A. Arnold. MEC: A System for Constructing and Analysing Transition Systems. In *Proc. of Autom. Verif. Methods for Finite State Systems*, *LNCS* 407. 1989.
8. S. Basu and C.R. Ramakrishnan. Compositional Analysis for Verification of Parameterized Systems. In *Proc. of TACAS*, *LNCS* 2619. Springer, 2003.
9. B. Berard and F. Laroussinie. Verification compositionnelle des p-automates. Tech. Report Lot 4.1, RNTL, projet AVERROES, 2003.
10. N. Bodentien, J. Vestergaard, J. Friis, K. Kristoffersen, and K. Larsen. Verification of State/Event Systems by Quotienting. Tech. Report RS-99-41, BRICS, 1999.
11. A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The Fc2Tools set: a Toolset for the Verification of Concurrent Systems. In *Proc. of CAV*, *LNCS* 1102. 1996.
12. F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *Proc. of CAV*, 2000.
13. D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, F. Lang, C. McKinty, V. Powazny, W. Serwe, and G. Smeding. Reference Manual of the LOTOS NT to LOTOS Translator (Version 5.4). INRIA/VASY, 2011.
14. E. Clarke, O. Grumberg, D. Peled. Model Checking. MIT Press, 2000.
15. R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. *FMSD*, 2(2):121–147, 1993.
16. P. Crouzen and F. Lang. Smart Reduction. In *Proc. of Fundamental Approaches to Software Engineering FASE*, *LNCS* 6603. Springer, 2011.
17. J.-C. Fernandez and L. Mounier. "On the Fly" Verification of Behavioural Equivalences and Preorders. In *Proc. of CAV*, *LNCS* 575. Springer, 1991.
18. H. Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In *Proc. of TACAS*, *LNCS* 1384. Springer, 1998.

19. H. Garavel and F. Lang. SVL: a Scripting Language for Compositional Verification. In *Proc. of FORTE*. IFIP, Kluwer Academic Publishers, 2001.
20. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of TACAS*, *LNCS* 6605. Springer, 2011.
21. H. Garavel and M. Sighireanu. A Graphical Parallel Composition Operator for Process Algebras. In *Proc. of FORTE/PSTV*. IFIP, Kluwer, 1999.
22. H. Garavel and D. Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In *Proc. of SPIN Workshop*, LNCS. Springer, 2009.
23. S. Graf and B. Steffen. Compositional Minimization of Finite State Systems. In *Proc. of Workshop on Computer-Aided Verification*, *LNCS* 531. Springer, 1990.
24. ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO International Standard 8807, 1989.
25. ISO/IEC. Enhancements to LOTOS (E-LOTOS). ISO International Standard 15437, 2001.
26. D. Kozen. Results on the Propositional μ-calculus. *TCS*, 27:333–354, 1983.
27. J.-P. Krimm and L. Mounier. Compositional State Space Generation from LOTOS Programs. In *Proc. of TACAS*, *LNCS* 1217. Springer, 1997.
28. F. Lang. EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods. In *Proc. of IFM*, *LNCS* 3771. Springer, 2005.
29. F. Lang. Refined Interfaces for Compositional Verification. In *Proc. of FORTE*, *LNCS* 4229. Springer, 2006.
30. F. Lang and R. Mateescu. Partial Order Reductions using Compositional Confluence Detection. In *Proc. of FM*, *LNCS* 5850. Springer, 2009.
31. F. Laroussinie and K. Larsen. Compositional Model Checking of Real Time Systems. In *Proc. of CONCUR*, 1995.
32. F. Laroussinie and K. Larsen. CMC: A Tool for Compositional Model Checking of Real-Time Systems. In *Proc. of FORTE*, 1998.
33. K. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model Checking of Real-Time Systems. In *Proc. of the IEEE Real-Time Symposium*, 1995.
34. F. Martinelli. Symbolic Partial Model Checking for Security Analysis. In *Proc. of Math. Methods, Models, and Architectures for Computer Network Security*, 2003.
35. R. Mateescu. Efficient Diagnostic Generation for Boolean Equation Systems. In *Proc. of TACAS*, *LNCS* 1785. Springer, 2000.
36. R. Mateescu. On-the-fly State Space Reductions for Weak Equivalences. In *Proc. of FMICS*. ERCIM, ACM Computer Society Press, 2005.
37. R. Mateescu. CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *STTT*, 8(1):37–56, Springer, 2006.
38. R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *SCP*, 46(3):255–281, 2003.
39. R. Mateescu and D. Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In *Proc. of FM*, number 5014 in LNCS. Springer, 2008.
40. G. Pace, F. Lang, and R. Mateescu. Calculating τ-Confluence Compositionally. In *Proc. of Computer Aided Verification CAV*, *LNCS* 2725. Springer, 2003.