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

Frédéric Lang and Radu Mateescu
Convecs team,
Inria Grenoble – Rhône-Alpes and LIG,
Montbonnot, France

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 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* [13], i.e., automatic evaluation of a temporal property against a formal 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 name *compositional verification*, which take advantage of the compositional structure of the concurrent system under verification. 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* [26, 30] to avoid explosion of intermediate compositions. Tools using this approach [21, 31, 32, 15] are available in the Cadp (*Construction and Analysis of Distributed Processes*) [22, 23] toolbox.

In this paper, we explore a dual approach named *partial model checking*, proposed by Andersen [2, 3] for concurrent processes running asynchronously and composed using Ccs parallel composition and restriction operators. For a modal μ-calculus [29] 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 (and 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, 4], synchronous state/event systems [9], and timed systems [8, 11, 36, 37, 38]. It has also been specialised for security properties [40]. More recently, it has been generalised to the full Ccs process algebra, with an application to the verification of parameterised systems [7]. These various developments of partial model checking, although successful, were relatively scarce, which may be explained by the complexity of the method: obtaining a fully operational partial model checker requires a significant implementation effort and extensive experiments for fine-tuning and optimization.

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 parallel composition operators (such as Ccs and Csp parallel compositions), previous works [2, 3, 7] are not directly applicable to more general operators, such as m-among-n synchronisation (where among n processes executing in parallel, any m of them must synchronise on a given action) and parallel composition by synchronisation interfaces (where all processes containing a given action in their synchronisation interface must synchronise on that action) [24], present in the E-Lotos standard and variants [12, 28]. Our first contribution in this paper is thus a generalisation of partial model checking to networks of Ltss [31], a general model that subsumes parallel composition, hiding, cutting, and renaming operators of standard process languages (Ccs, Csp, μCrl, Lotos, E-Lotos, etc.), including the above-mentioned parallel composition operators. Regarding the communication of data values, our approach is applicable to classical (i.e., with static communication) value-passing process algebras equipped with early operational semantics. This framework encompasses a significant fragment of the π-calculus (containing channel mobility and bounded process creation), which can be translated into classical value-passing process algebras [44].

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 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. We prove that this reformulation is sound. Our third contribution is the reformulation of formula simplifications as a combination of graph reductions (including minimisations modulo equivalence relations and bisimulations) 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. We also present how this specialisation can be again generalised to handle also useful fairness operators of alternation 2 in linear time without developing the complex machinery to evaluate general alternation-2 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 Section 2. The network of Ltss model is presented in Section 3. The generalisation of quotienting to networks and its reformulation as a synchronous product is presented in Section 4. The simplification rules are presented in Section 5. The rules specific to alternation-free μ-calculus formulas are presented in Section 6. The way we handle fairness operators is presented in Section 7. Our implementation of partial model checking of the regular alternation-free μ-calculus extended with fairness operators is presented in Section 8. Experimental results are presented in Section 9. Concluding remarks are given in Section 10. This paper is an extended version of an earlier paper [34].

2 The Modal μ-Calculus

We consider systems whose behavioural semantics can be represented using an Lts (*Labelled Transition System*), and whose properties can be expressed in the modal μ-calculus [29].

Definition 1 (Lts) *An* Lts *is a tuple* $(\Sigma, A, \longrightarrow, s_0)$*, where:*

- Σ *is a set of states,*

- A *is a set of labels,*

- $\longrightarrow \subseteq \Sigma \times A \times \Sigma$ *is the (labelled) transition relation,*

$$\begin{aligned}
\llbracket \mathbf{ff} \rrbracket\, \rho &= \emptyset \\
\llbracket \mathbf{tt} \rrbracket\, \rho &= \Sigma \\
\llbracket \varphi_1 \vee \varphi_2 \rrbracket\, \rho &= \llbracket \varphi_1 \rrbracket\, \rho \cup \llbracket \varphi_2 \rrbracket\, \rho \\
\llbracket \varphi_1 \wedge \varphi_2 \rrbracket\, \rho &= \llbracket \varphi_1 \rrbracket\, \rho \cap \llbracket \varphi_2 \rrbracket\, \rho \\
\llbracket \langle a \rangle\, \varphi_0 \rrbracket\, \rho &= \{ s \in \Sigma \mid (\exists s' \in \Sigma)\ s \xrightarrow{a} s' \wedge s' \in \llbracket \varphi_0 \rrbracket\, \rho \} \\
\llbracket [a]\, \varphi_0 \rrbracket\, \rho &= \{ s \in \Sigma \mid (\forall s' \in \Sigma)\ s \xrightarrow{a} s' \implies s' \in \llbracket \varphi_0 \rrbracket\, \rho \} \\
\llbracket \mu X . \varphi_0 \rrbracket\, \rho &= \bigcap \{ U \subseteq \Sigma \mid \llbracket \varphi_0 \rrbracket\, (\rho \oslash [U/X]) \subseteq U \} \\
\llbracket \nu X . \varphi_0 \rrbracket\, \rho &= \bigcup \{ U \subseteq \Sigma \mid U \subseteq \llbracket \varphi_0 \rrbracket\, (\rho \oslash [U/X]) \} \\
\llbracket \neg \varphi_0 \rrbracket\, \rho &= \Sigma \setminus \llbracket \varphi_0 \rrbracket\, \rho \\
\llbracket X \rrbracket\, \rho &= \rho(X)
\end{aligned}$$

Figure 1: Semantics of the modal μ-calculus

- *and $s_0 \in \Sigma$ is the initial state.*

For an Lts $S = (\Sigma, A, \longrightarrow, s_0)$, *we may also write $s \xrightarrow{a} s' \in S$ (or simply $s \xrightarrow{a} s'$ when S is clear from the context) instead of $(s, a, s') \in \rightarrow$.*

Definition 2 (Syntax of the modal μ-calculus) *The modal μ-calculus formulas (φ) are terms built from Boolean constants ($\mathbf{ff}, \mathbf{tt}$), Boolean connectors (disjunction $\vee$, conjunction $\wedge$, and negation $\neg$), modalities (possibility $\langle \_ \rangle$ and necessity $[\_]$), and fix-point operators (minimal μ and maximal ν) over propositional variables X, generated by the following grammar:*

$$\begin{array}{lcl}
\varphi & ::= & \mathbf{ff} \ \mid\ \varphi_1 \vee \varphi_2 \ \mid\ \langle a \rangle\, \varphi_0 \ \mid\ \mu X . \varphi_0 \\
 & \mid & \mathbf{tt} \ \mid\ \varphi_1 \wedge \varphi_2 \ \mid\ [a]\, \varphi_0 \ \mid\ \nu X . \varphi_0 \\
 & \mid & \neg \varphi_0 \ \mid\ X
\end{array}$$

To ensure a proper definition of fix-point operators, a commonly adopted and sufficient condition is that formulas φ are syntactically monotonic *[29], 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. Therefore, we will only consider syntactically monotonic formulas. We write $L\mu$ for the set of μ-calculus formulas.*

We write $\mathsf{fv}\,(\varphi)$ for the set of variables free in φ, and $\mathsf{bv}\,(\varphi)$ for the set of variables bound in φ. We call a closed formula *any formula φ such that $\mathsf{fv}\,(\varphi) = \emptyset$. 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 either form $\mu X . \varphi_0$ or $\nu X . \varphi_0$. 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 (while implicitly applying α-conversion to maintain the unicity of bound variables).*

Definition 3 (Semantics of the modal μ-calculus) *The semantics of the modal μ-calculus are formally defined by the equations of Figure 1. 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 $\llbracket \varphi \rrbracket\, \rho$ (also written $\llbracket \varphi \rrbracket$ 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 $s \in \llbracket \varphi \rrbracket$.*

Proposition 1 *The modal μ-calculus satisfies the following identities:*

$$\begin{array}{rcl}
\neg\mathbf{tt} & = & \mathbf{ff} \\
\neg\mathbf{ff} & = & \mathbf{tt} \\
\neg(\varphi_1 \wedge \varphi_2) & = & \neg\varphi_1 \vee \neg\varphi_2 \\
\neg(\varphi_1 \vee \varphi_2) & = & \neg\varphi_1 \wedge \neg\varphi_2 \\
\neg\,[a]\,\varphi_0 & = & \langle a\rangle\,\neg\varphi_0 \\
\neg\,\langle a\rangle\,\varphi_0 & = & [a]\,\neg\varphi_0 \\
\neg\nu X.\varphi_0 & = & \mu X.\neg\varphi_0[\neg X/X] \\
\neg\mu X.\varphi_0 & = & \nu X.\neg\varphi_0[\neg X/X]
\end{array}$$

Definition 4 (Positive form and disjunctive form) *Every modal μ-calculus formula φ can be rewritten in both of the following forms:*

- *A formula is in* positive form *if it contains any of the modal μ-calculus operators but the negation operator $\neg$. Note that syntactic monotonicity implies that every negation can be eliminated using the identities of Proposition 1. Given a modal μ-calculus formula φ, we write φ^+ the corresponding formula in positive form.*
- *A formula is in* disjunctive form *if it contains only the constant $\mathbf{ff}$, disjunctions, possibility modalities, minimal fix-points, propositional variables and negations. Every formula can be put in disjunctive form using the identities of Proposition 1. Note that a formula in disjunctive form is not necessarily a disjunctive formula due to the presence of negations.*

Definition 5 *A formula φ is* alternation-free *if φ^+ does not contain any sub-formula 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 $\varphi^+[X]$ has the form $\mu X.\varphi_0$ (resp. $\nu X.\varphi_0$). We write $L\mu_1$ for the set of alternation-free μ-calculus formulas, and more generally $L\mu_n$ for the set of μ-calculus formulas of alternation up to n (for some n).*

Definition 6 (Block-labelled formula) *In this paper, we consider* block-labelled *formulas φ in disjunctive form, in which each propositional variable X is labelled by a natural number k, called its* block number.

Intuitively, a block-labelling is well-formed *if the μ-calculus formula can be converted into an equivalent set of μ-calculus equations partitioned into blocks, so that all variables having the same block number are defined in the same block and if $k < k'$ then the equations within block number k occur before the equations within block number k'. The proof is beyond the scope of this paper. The well-formedness conditions are the following:*

1. *All occurrences of a given variable X are labelled by the same block number k.*
2. *All variables sharing the same block number have the same fix-point sign.*
3. *For all $X^k \in \mathsf{bv}\,(\varphi), Y^{k'} \in \mathsf{fv}(\varphi[X^k])$ it holds that $k' \leq k$.*

By convention, we assume without loss of generality that the even block numbers are associated to variables of sign μ and odd block numbers are associated to variables of sign ν.

Initially, every unlabelled formula φ in disjunctive form can be turned into the well-formed block-labelled formula $\mathsf{bl}\,(\varphi, \mathbf{tt}, 0, [])$, where $\mathsf{bl}\,(\psi, b, k, \gamma)$ is defined as follows, γ denoting a mapping from variables to block numbers:

$$\begin{array}{rcl}
\mathsf{bl}\,(\mathbf{ff}, b, k, \gamma) & = & \mathbf{ff} \\
\mathsf{bl}\,(X, b, k, \gamma) & = & X^{\gamma\,(X)} \\
\mathsf{bl}\,(\neg\varphi_0, b, k, \gamma) & = & \neg\mathsf{bl}\,(\varphi_0, \neg b, k, \gamma) \\
\mathsf{bl}\,(\varphi_1 \vee \varphi_2, b, k, \gamma) & = & \mathsf{bl}\,(\varphi_1, b, k, \gamma) \vee \mathsf{bl}\,(\varphi_2, b, k, \gamma) \\
\mathsf{bl}\,(\langle a\rangle\,\varphi_0, b, k, \gamma) & = & \langle a\rangle\,\mathsf{bl}\,(\varphi_0, b, k, \gamma) \\
\mathsf{bl}\,(\mu X.\varphi_0, b, k, \gamma) & = & \left\{\begin{array}{ll} \mu X^k.\mathsf{bl}\,(\varphi_0, \mathbf{tt}, k, \gamma[X \mapsto k]) & \textit{if } b = \mathbf{tt} \\ \mu X^{k+1}.\mathsf{bl}\,(\varphi_0, \mathbf{tt}, k+1, \gamma[X \mapsto k+1]) & \textit{otherwise} \end{array}\right.
\end{array}$$

We write $\mathsf{blocks}(\varphi)$ *the set of block numbers occurring in φ. A block-labelled formula φ in disjunctive form is* alternation-free *if $k' = k$ for all $X^k \in \mathsf{bv}(\varphi), Y^{k'} \in \mathsf{fv}(\varphi[X^k])$.*

A well-known result of the μ-calculus is that the variables of an alternation-free formula can be partitioned into blocks that have no cyclic dependencies. Another way to state this result is that any unlabelled formula in disjunctive form is alternation-free if and only if it can be block-labelled so that it satisfies the definition of alternation-free block-labelled formula.

In the remainder of this paper, we will consider block-labelled formulas in disjunctive form. At last, we consider the following notion of formula equivalence, which is a slight generalisation of syntactic equality to enclose also the semantic notions of renaming, commutativity, and idempotence.

Definition 7 *Let f be a bijective function from the set of propositional variables to itself, called a renaming. For formulas in disjunctive form, we define syntactic equality modulo commutativity, idempotence and f-renaming as the smallest relation, written $=_f$, such that if $\varphi_i =_f \varphi'_i$ $(i \in 0..2)$ then:*

- $\mathbf{ff} =_f \mathbf{ff}$, $\neg\varphi_0 =_f \neg\varphi'_0$, $\langle a\rangle\,\varphi_0 =_f \langle a\rangle\,\varphi'_0$, $\varphi_1 \vee \varphi_2 =_f \varphi'_1 \vee \varphi'_2$, $X =_f f(X)$, *and* $\mu X.\varphi_0 =_f \mu f(X).\varphi'_0$ *for each propositional variable X (*syntactic equality modulo renaming*),*
- $\varphi_1 \vee \varphi_2 =_f \varphi'_2 \vee \varphi'_1$ *(*commutativity*),*
- $\varphi_0 \vee \varphi_0 =_f \varphi'_0$ *and* $\varphi_0 =_f \varphi'_0 \vee \varphi'_0$ *(*idempotence*).*

3 Networks of LTSs

Networks of LTSs (or *networks* for short) are inspired from the Mec [6] and Fc2 [10] synchronisation vectors and were introduced in [31] as an intermediate model to represent compositions of Ltss using various operators.

Definition 8 (Vector and vector projection) *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, $()$ denotes 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 $(\forall j \in i..n-1)$ $\mathbf{v}_{\setminus i}[j] = \mathbf{v}[j+1]$.

Definition 9 (Network of LTSs) *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 synchronisation rule has 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. Let $\mathbf{S}[i] = (\Sigma_i, A_i, \longrightarrow_i, s_i^0)$ $(i \in 1..n)$. 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 relation satisfying* $\mathbf{s} \xrightarrow{a} \mathbf{s}'$ *if and only if there exists $(\mathbf{t}, a) \in V$ such that for all $i \in 1..n$:*

$$\begin{cases} \mathbf{s}'[i] = \mathbf{s}[i] & \text{if } \mathbf{t}[i] = \bullet \\ \mathbf{s}[i] \xrightarrow{\mathbf{t}[i]}_i \mathbf{s}'[i] & \text{otherwise} \end{cases}$$

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

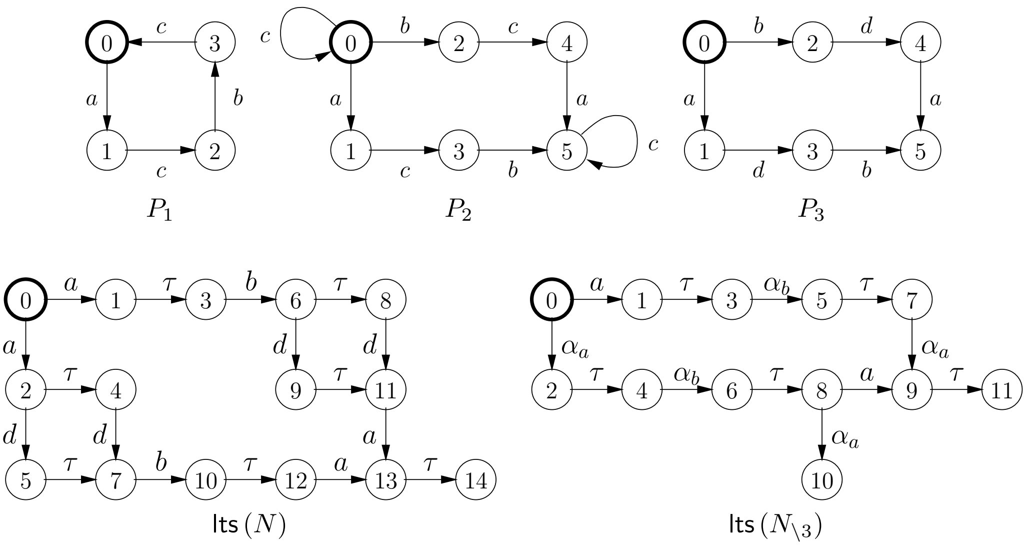


Figure 2: Labelled Transition Systems for N defined in Example 1

Example 1 *Let a, b, c, and d be labels, and P_1, P_2, and P_3 be the processes defined in Figure 2 (top), where the initial states are denoted by bold circles. 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 depicted in Figure 2 (bottom left). The first two rules express a nondeterministic synchronisation on a between either P_1 and P_2, or P_1 and 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 is used in the tool EXP.OPEN [31] of CADP as an intermediate model for representing LTSs composed using the hiding, renaming, cutting, and parallel composition operators present in the process algebras CCS, CSP, LOTOS, and μCRL, but also more expressive operators, such as m-among-n synchronisation and parallel composition using synchronisation interfaces [24] present in E-LOTOS [28] and LOTOS NT [12]. For instance, the rules $\{((a, a, \bullet), a), ((a, \bullet, a), a), ((\bullet, a, a), a)\}$ realize 2-among-3 synchronisation on a.

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 $||_{j \in 1..n} P_j = P_i \mathbin{||} (||_{j \in 1..n \setminus \{i\}} P_j)$. However, as argued in [24], 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.

Definition 10 (Sub-network extraction) *$N = (\mathbf{S}, V)$ being a network of size n, we assume a function $\alpha\,(\mathbf{t}, a)$ that assigns a unique 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 $N_{\setminus i}$:*

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

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 Example 1, $N_{\setminus 3}$ has vector of* LTSs *(P_1, P_2), P_1 and P_2 being defined in Figure 2 (top left and top middle), 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 Figure 2 (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 individual 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 $N_{\backslash i}$.

Definition 11 (Quotient formula) *The quotient formula is written $\varphi \,/\!\!/_i^{\emptyset}\, s_0^i$ and defined as follows for closed formulas in disjunctive form:*

$$\begin{array}{rcl}
\mathbf{ff} \,/\!\!/_i^B\, s & = & \mathbf{ff} \\
X^k \,/\!\!/_i^B\, s & = & \varphi[X^k] \,/\!\!/_i^B\, s \\
(\neg\varphi_0) \,/\!\!/_i^B\, s & = & \neg(\varphi_0 \,/\!\!/_i^B\, s) \\
(\varphi_1 \vee \varphi_2) \,/\!\!/_i^B\, s & = & (\varphi_1 \,/\!\!/_i^B\, s) \vee (\varphi_2 \,/\!\!/_i^B\, s) \\
(\mu X^k.\varphi_0) \,/\!\!/_i^B\, s & = & \left\{ \begin{array}{ll} X_s^k & \textit{if } X_s^k \in B \\ \mu X_s^k.(\varphi_0 \,/\!\!/_i^{B\cup\{X_s^k\}}\, s) & \textit{otherwise} \end{array} \right. \\
(\langle a\rangle\varphi_0) \,/\!\!/_i^B\, s & = & \bigvee_{(\mathbf{t},a)\in V} \left(\begin{array}{llrll} (\quad i \notin A(\mathbf{t}) & \wedge & \langle a\rangle & (\varphi_0 \,/\!\!/_i^B\, s)\,) & \vee \\ (\{i\} \subset A(\mathbf{t}) & \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} & \langle \alpha(\mathbf{t},a)\rangle & (\varphi_0 \,/\!\!/_i^B\, s')) & \vee \\ (\{i\} = A(\mathbf{t}) & \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} & & (\varphi_0 \,/\!\!/_i^B\, s')) & \end{array} \right)
\end{array}$$

This definition follows and generalises Andersen's [2] (specialised for Ccs) to networks. The main difference is the definition of $(\langle a\rangle\varphi_0) \,/\!\!/_i^B\, s$, 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 Ccs *co-action*, making the use of special labels $\alpha(\mathbf{t},a)$ not necessary. A minor difference is that we use μ-calculus terms instead of equations[1]. 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 definition. 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 $\mu X^k.\varphi_0$ (because the formula is in disjunctive form), and the size of the set B is bounded by $|\,\mathsf{bv}(\varphi)| \times |\Sigma_i|$. Note that well-formedness of the block-labelling is preserved by quotienting, because for every variable $X_s^k \in \mathsf{bv}(\varphi \,/\!\!/_i^{\emptyset}\, s_0)$ we have $X^k \in \mathsf{bv}(\varphi)$ and for every variable $Y_{s'}^{k'} \in \mathsf{fv}((\varphi \,/\!\!/_i^{\emptyset}\, s_0)[X_s^k])$ we have $Y^{k'} \in \mathsf{fv}(\varphi[X^k])$, and therefore $k' \leq k$.

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 Example 1 (page 6) yields the formula $\mu X_0^0.\langle a\rangle\neg\mathbf{ff} \vee \langle\alpha_a\rangle\neg\mathbf{ff} \vee \langle\alpha_b\rangle\mu X_2^0.\langle a\rangle\neg\mathbf{ff} \vee \mathbf{ff}$. In other words, an action a can be reached after a (possibly empty) sequence of b actions in the network N if and only if an action a, or an action α_a, or an action α_b followed by an action a, can be reached immediately in $N_{\backslash 3}$, given the behaviour of P_3 depicted in Figure 2 (page 6).*

[1] Note that terms will be compiled into graphs, thus enabling the sharing of sub-formulas that is also possible using equations.

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.

Definition 12 (Circuit) *Let $S = (\Sigma, A, \rightarrow, s_0)$ be an* Lts *and $T \subseteq \ \rightarrow$ be a subset of its transitions. The* states *of T are defined as the set* $\mathrm{st}\,(T) = \{s, s' \in \Sigma \mid (s, \sigma, s') \in T\}$*. T is a* circuit *of S if for all $s, s' \in \mathrm{st}\,(T)$ there is a sequence of transitions belonging to T from s to s'. A state $s \in \mathrm{st}\,(T)$ is a* root *of the circuit T if there is a sequence of transitions from s_0 to s that does not traverse any transition of T.*

Definition 13 (Formula graph) *A* formula graph *is an* Lts $(\Sigma, A, \rightarrow, s_0)$ *such that:*

1. *Every label $\sigma \in A$ has either form $\vee$, $\neg$, $\langle a \rangle$ (for some a belonging to a fixed set of action names), or μ^k (for some $k \in \mathbb{N}$).*
2. *If $s_0 \xrightarrow{\delta} s \xrightarrow{\mu^k} s'$ for some $\delta \in A^*$ and $k \in \mathbb{N}$, then k is even if and only if δ contains an even number of occurrences of the label $\neg$.*
3. *If $s \in \Sigma$ is a root of a circuit then (a) the circuit contains a μ^k-transition and (b) if the first μ^k-transition traversed on the circuit starting in s has block number k' then every μ^k-transition belonging to the circuit satisfies $k \geq k'$.*

Every formula graph can be decoded into a closed formula as follows.

Definition 14 (Decoding a formula graph) *A formula graph $P = (\Sigma, A, \rightarrow, s_0)$ encodes the modal μ-calculus formula* $\mathsf{dec_s}\,(P, s_0, \emptyset)$*, where* $\mathsf{dec_s}\,(P, s, E)$ *is defined as follows ($E \subseteq \Sigma$). In our decoding every variable is uniquely identified by the source state s and the block number k of the μ-transition, which we write $\overline{s}^k$.*

$$\mathsf{dec_s}\,(P, s, E) = \bigvee_{s \xrightarrow{\sigma} s' \in P} \mathsf{dec_t}\,(P, s \xrightarrow{\sigma} s', E)$$

where

$$\begin{array}{rcl}
\mathsf{dec_t}\,(P, s \xrightarrow{\vee} s', E) & = & \mathsf{dec_s}\,(P, s', E) \\
\mathsf{dec_t}\,(P, s \xrightarrow{\neg} s', E) & = & \neg\,\mathsf{dec_s}\,(P, s', E) \\
\mathsf{dec_t}\,(P, s \xrightarrow{\langle a \rangle} s', E) & = & \langle a \rangle\,\mathsf{dec_s}\,(P, s', E) \\
\mathsf{dec_t}\,(P, s \xrightarrow{\mu^k} s', E) & = & \begin{cases} \overline{s}^k & \text{if } s \in E \\ \mu \overline{s}^k .\,\mathsf{dec_s}\,(P, s', E \cup \{s\}) & \text{otherwise} \end{cases}
\end{array}$$

This definition implies that a deadlock state decodes as **ff** (empty disjunction). Function $\mathsf{dec_s}$ is well-defined. In particular, it terminates because every cyclic path contains a label of the form μ^k. By recording in the set E the source states of traversed μ^k-transitions, we thus avoid infinite traversals of cycles. In practice (see next section), formula graphs need not be decoded except for correctness proofs.

Definition 15 (Encoding a formula into a formula graph) *The formula graph corresponding to a formula φ in disjunctive form is an* Lts *written* $\mathsf{enc}\,(\varphi)$*, whose states are identified with sub-formulas of φ. The initial state of the formula graph is φ,* **ff** *is a deadlock state, and each sub-formula has transitions as follows:*

$$\begin{array}{ccc}
X^k \xrightarrow{\vee} \varphi[X^k] & \neg\varphi_0 \xrightarrow{\neg} \varphi_0 & \langle a \rangle \varphi_0 \xrightarrow{\langle a \rangle} \varphi_0 \\
\varphi_1 \vee \varphi_2 \xrightarrow{\vee} \varphi_1 & \varphi_1 \vee \varphi_2 \xrightarrow{\vee} \varphi_2 & \mu X^k.\varphi_0 \xrightarrow{\mu^k} \varphi_0
\end{array}$$

Although the states of a formula graph are identified by formulas, only the transition labels are required for decoding. In figures, states will be simply identified by numbers.

Note that the formula graph obtained by encoding a formula satisfies the conditions given in Definition 13. Condition (2) is a direct consequence of the block-labelling convention stated in Definition 6. Condition (3) comes from the fact that the roots of the circuits are the states associated to formulas of the form $\mu X^k.\psi$ such that X^k occurs free in ψ. In particular, subcondition (b) is a consequence on the third well-formedness condition given in Definition 6.

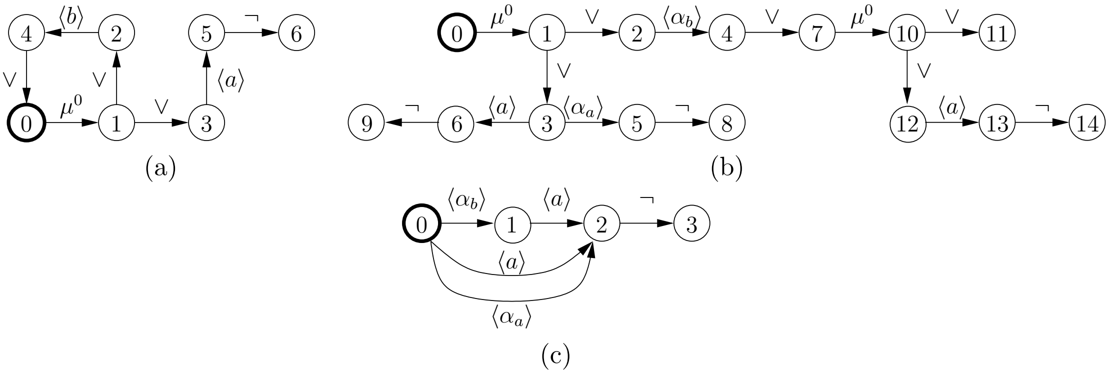


Figure 3: Examples of formula graphs

Example 4 *The formula graph corresponding to the formula $\mu X^0.(\langle a\rangle \mathtt{tt}) \vee \langle b\rangle X^0$ introduced in Example 3 is depicted in Figure 3 (a).*

We now prove that our encoding of closed formulas into formula graphs is sound, in the sense that the formula can be recovered from the formula graph into which the formula is encoded. This is stated formally in Proposition 2 below, which is a corollary of the following Lemma:

Lemma 1 *Let φ be a closed formula in disjunctive form and f be a renaming that maps each propositional variable $X^k \in \mathsf{bv}(\varphi)$ to $\overline{\varphi[X^k]}^k$. For every sub-formula ψ of φ, if $\{\varphi[Y^k] \mid Y^k \in \mathsf{fv}(\psi)\} \subseteq E$ and $E \cap \{\varphi[Y^k] \mid Y^k \in \mathsf{bv}(\psi)\} = \emptyset$, then $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi, E) =_f \psi$.*

Proof. We proceed by structural induction on ψ:

Case $\psi = \mathbf{ff}$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has no outgoing transition. Therefore by definition of $\mathsf{dec_s}$, we have $\mathsf{dec_s}(\mathsf{enc}(\varphi), \mathbf{ff}, E) = \mathbf{ff}$.

Case $\psi = X^k$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has a single transition $X^k \xrightarrow{\vee} \varphi[X^k]$. Therefore by definition of $\mathsf{dec_s}$, we have $\mathsf{dec_s}(\mathsf{enc}(\varphi), X^k, E) = \mathsf{dec_s}(\mathsf{enc}(\varphi), \varphi[X^k], E)$. Since $X^k \in \mathsf{fv}(X^k)$, by the hypothesis $\varphi[X^k] \in E$. It follows by definition of $\mathsf{dec_s}$ that $\mathsf{dec_s}(\mathsf{enc}(\varphi), X^k, E) = \overline{\varphi[X^k]}^k =_f X^k$.

Case $\psi = \neg\psi_0$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has a single transition $\neg\psi_0 \xrightarrow{\neg} \psi_0$. Therefore by definition of $\mathsf{dec_s}$, we have $\mathsf{dec_s}(\mathsf{enc}(\varphi), \neg\psi_0, E) = \neg\,\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_0, E)$. Since $\mathsf{fv}(\psi_0) = \mathsf{fv}(\psi)$ and $\mathsf{bv}(\psi_0) = \mathsf{bv}(\psi)$, the induction hypothesis holds and then $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_0, E) =_f \psi_0$. It follows immediately that $\mathsf{dec_s}(\mathsf{enc}(\varphi), \neg\psi_0, E) =_f \neg\psi_0$.

Case $\psi = \psi_1 \vee \psi_2$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has two transitions $\psi_1 \vee \psi_2 \xrightarrow{\vee} \psi_1$ and $\psi_1 \vee \psi_2 \xrightarrow{\vee} \psi_2$. Therefore by definition of $\mathsf{dec_s}$, we have $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_1 \vee \psi_2, E) = \mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_1, E) \vee \mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_2, E)$ (modulo commutativity if the transitions are enumerated in the opposite order, and idempotence if the transitions are identical). Since $\mathsf{fv}(\psi_1) \cup \mathsf{fv}(\psi_2) = \mathsf{fv}(\psi)$ and $\mathsf{bv}(\psi_1) \cup \mathsf{bv}(\psi_2) = \mathsf{bv}(\psi)$, the induction hypothesis holds and then we have both $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_1, E) =_f \psi_1$ and $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_2, E) =_f \psi_2$. It follows that $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_1 \vee \psi_2, E) =_f \psi_1 \vee \psi_2$.

Case $\psi = \langle a\rangle\psi_0$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has a single transition $\langle a\rangle\psi_0 \xrightarrow{\langle a\rangle} \psi_0$. Therefore by definition of $\mathsf{dec_s}$, we have $\mathsf{dec_s}(\mathsf{enc}(\varphi), \langle a\rangle\psi_0, E) = \langle a\rangle\,\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_0, E)$. Since $\mathsf{fv}(\psi_0) = \mathsf{fv}(\psi)$ and $\mathsf{bv}(\psi_0) = \mathsf{bv}(\psi)$, the induction hypothesis holds and then $\mathsf{dec_s}(\mathsf{enc}(\varphi), \psi_0, E) =_f \psi_0$. It follows immediately that $\mathsf{dec_s}(\mathsf{enc}(\varphi), \langle a\rangle\psi_0, E) =_f \langle a\rangle\psi_0$.

Case $\psi = \mu X^k.\psi_0$: By definition of $\mathsf{enc}(\varphi)$, the state ψ has a single transition $\mu X^k.\psi_0 \xrightarrow{\mu^k} \psi_0$. Also, $\mu X^k.\psi_0 \notin E$ because $\mu X^k.\psi_0 = \varphi[X^k]$, $X^k \in \mathsf{bv}(\psi)$ and, by hypothesis, $E \cap \{\varphi[Y^k] \mid Y^k \in \mathsf{bv}(\psi)\} = \emptyset$. As a consequence and by definition of dec_s, we have $\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \mu X^k.\psi_0, E) = \mu\overline{\mu X^k.\psi_0}^k.\,\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \psi_0, E \cup \{\mu X^k.\psi_0\})$. Since $\mu X^k.\psi_0 = \varphi[X^k]$, the latter formula is also equal to $\mu\overline{\varphi[X^k]}^k.\,\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \psi_0, E \cup \{\varphi[X^k]\})$. To apply the induction hypothesis, we must show that $\{\varphi[Y^k] \mid Y^k \in \mathsf{fv}(\psi_0)\} \subseteq E \cup \{\varphi[X^k]\}$ and that $(E \cup \{\varphi[X^k]\}) \cap \{\varphi[Y^k] \mid Y^k \in \mathsf{bv}(\psi_0)\} = \emptyset$. This is true by hypothesis and because $\mathsf{fv}(\psi_0) = \mathsf{fv}(\psi) \cup \{X^k\}$ and $\mathsf{bv}(\psi_0) = \mathsf{bv}(\psi) \setminus \{X^k\}$. Therefore, $\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \psi_0, E) =_f \psi_0$. It follows immediately that $\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \mu X^k.\psi_0, E) =_f \mu X^k.\psi_0$. □

Proposition 2 *If φ is a closed formula in disjunctive form, then $\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \varphi, \emptyset) =_f \varphi$ where f maps each propositional variable $X^k \in \mathsf{bv}(\varphi)$ to $\overline{\varphi[X^k]}^k$.*

Proof. If φ is a closed formula, then $\mathsf{fv}(\varphi) = \emptyset$. We have $\{\varphi[Y^k] \mid Y^k \in \mathsf{fv}(\varphi)\} = \emptyset$. Therefore, the hypotheses of Lemma 1 are satisfied, which implies $\mathsf{dec}_\mathsf{s}(\mathsf{enc}(\varphi), \varphi, \emptyset) =_f \varphi$. □

Using this encoding, the quotient of a formula with respect to the ith Lts of a network can be computed as a synchronous product using a network called *quotient formula network*.

Definition 16 (Quotient formula network) *Let φ be a modal μ-calculus formula in disjunctive form, $N = (\mathbf{S}, V)$ be a network of size n, and $i \in 1..n$. The* quotient formula network *of φ with respect to $\mathbf{S}[i]$ is defined as the network $((\mathsf{enc}(\varphi), \mathbf{S}[i]), V/\!\!/_i)$, where $V/\!\!/_i$ denotes the following set of rules:*

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

Note that the Lts corresponding to the quotient formula network is a formula graph. This can easily be shown by observing that, if $(\psi_1, s_1) \xrightarrow{\delta} (\psi_n, s_n)$ is a transition sequence of the quotient formula network, then there exists a transition sequence of the form $\psi_1 \xrightarrow{\delta'} \psi_n$ in the input formula graph, such that the μ-projection of δ' (i.e., the sequence obtained from δ' by keeping only the μ^k-labels) and the μ-projection of δ are identical. In addition, if the transition sequence labelled by δ is a circuit, then δ' can be found such that the transition sequence labelled by δ' is also a circuit. This ensures that conditions (2) and (3) of Definition 13 are preserved in the Lts corresponding to the quotient formula network.

We now prove that the Lts corresponding to the quotient formula network indeed encodes the quotient correctly. This is stated formally in Proposition 1 below, which is a corollary of the following Lemma:

Lemma 2 *Let φ be a closed formula in disjunctive form, $N = (\mathbf{S}, V)$ be a network of size n, $i \in 1..n$, $P = \mathsf{lts}((\mathsf{enc}(\varphi), \mathbf{S}[i]), V/\!\!/_i)$ be the quotient formula network of φ with respect to $\mathbf{S}[i]$, s be a state of $\mathbf{S}[i]$, and f be a renaming that maps each propositional variable $Y_t^k \in \mathsf{bv}(\varphi /\!\!/_i^B s_0^i)$ to $\overline{(\varphi[Y^k], t)}^k$. If $E = \{(\varphi[Y^k], t) \mid Y_t^k \in B\}$ then for every sub-formula ψ of φ, $\mathsf{dec}_\mathsf{s}(P, (\psi, s), E) =_f \psi /\!\!/_i^B s$.*

Proof. We proceed by case on ψ and by structural induction on the formula $\psi /\!\!/_i^B s$ (which is finite):

Case $\psi = \mathbf{ff}$: By definition of P, the state $(\mathbf{ff}, s)$ has no outgoing transition, because by definition of $\mathsf{enc}(\varphi)$ the state $\mathbf{ff}$ has no outgoing transition, and $V/\!\!/_i$ contains no synchronisation rule of the form $((\bullet, a), b)$. Therefore, by definition of dec_s we have $\mathsf{dec}_\mathsf{s}(P, (\mathbf{ff}, s), E) = \mathbf{ff}$ and by definition of quotienting we have $\mathbf{ff} /\!\!/_i^B s = \mathbf{ff}$. It follows immediately that $\mathsf{dec}_\mathsf{s}(P, (\mathbf{ff}, s), E) =_f \mathbf{ff} /\!\!/_i^B s$.

Case $\psi = X^k$: By definition of P, the state (X^k, s) has a transition $(X^k, s) \xrightarrow{\vee} (\varphi[X^k], s)$, because by definition of $\mathsf{enc}(\varphi)$ the state X^k has a transition $X^k \xrightarrow{\vee} \varphi[X^k]$ and $V/\!\!/_i$ contains the synchronisation rule $((\vee, \bullet), \vee)$. The state (X^k, s) has no other transition in P, because the state X^k has no other transition and $V/\!\!/_i$ does not contain other synchronisation rules of either form $((\bullet, a), b)$ or $((\vee, a), b)$. Therefore, we have $\mathsf{dec}_\mathsf{s}(P, (X^k, s), E) = \mathsf{dec}_\mathsf{s}(P, (\varphi[X^k], s), E)$ by definition of dec_s. As formulas are in disjunctive form, $\varphi[X^k]$ has the form $\mu X^k.\psi_0$. The rest of the proof for this case is identical to the case $\psi = \mu X^k.\psi_0$ detailed below.

Case $\psi = \neg\psi_0$: By definition of P, the state $(\neg\psi_0, s)$ has a transition $(\neg\psi_0, s) \xrightarrow{\neg} (\psi_0, s)$, because by definition of $\mathsf{enc}\,(\varphi)$ the state $\neg\psi_0$ has a transition $\neg\psi_0 \xrightarrow{\neg} \psi_0$ and $V/\!\!/_i$ contains the synchronisation rule $((\neg, \bullet), \neg)$. The state $(\neg\psi_0, s)$ has no other transition in P, because the state $\neg\psi_0$ has no other transition and $V/\!\!/_i$ does not contain other synchronisation rules of either form $((\bullet, a), b)$ or $((\neg, a), b)$. On the one hand, we thus have $\mathsf{dec_s}\,(P, (\neg\psi_0, s), E) = \neg\, \mathsf{dec_s}\,(P, (\psi_0, s), E)$ by definition of $\mathsf{dec_s}$. On the other hand, we have $(\neg\psi_0) /\!\!/_i^B\ s = \neg(\psi_0 /\!\!/_i^B\ s)$ by definition of quotienting. Also $\psi_0 /\!\!/_i^B\ s$ is a proper sub-formula of $\psi /\!\!/_i^B\ s$. Therefore, by induction hypothesis we have $\mathsf{dec_s}\,(P, (\psi_0, s), E) =_f \psi_0 /\!\!/_i^B\ s$. It follows immediately that $\mathsf{dec_s}\,(P, (\neg\psi_0, s), E) =_f (\neg\psi_0) /\!\!/_i^B\ s$.

Case $\psi = \psi_1 \vee \psi_2$: By definition of P, the state $(\psi_1 \vee \psi_2, s)$ has transitions $(\psi_1 \vee \psi_2, s) \xrightarrow{\vee} (\psi_1, s)$ and $(\psi_1 \vee \psi_2, s) \xrightarrow{\vee} (\psi_2, s)$, because by definition of $\mathsf{enc}\,(\varphi)$ the state $\psi_1 \vee \psi_2$ has transitions $\psi_1 \vee \psi_2 \xrightarrow{\vee} \psi_1$ and $\psi_1 \vee \psi_2 \xrightarrow{\vee} \psi_2$ and $V/\!\!/_i$ contains the synchronisation rule $((\vee, \bullet), \vee)$. The state $(\psi_1 \vee \psi_2, s)$ has no other transition in P, because the state $\psi_1 \vee \psi_2$ has no other transition and $V/\!\!/_i$ does not contain other synchronisation rules of either form $((\bullet, a), b)$ or $((\vee, a), b)$. On the one hand, we thus have $\mathsf{dec_s}\,(P, (\psi_1 \vee \psi_2, s), E) = \mathsf{dec_s}\,(P, (\psi_1, s), E) \vee \mathsf{dec_s}\,(P, (\psi_2, s), E)$ by definition of $\mathsf{dec_s}$. On the other hand, we have $(\psi_1 \vee \psi_2) /\!\!/_i^B\ s = (\psi_1 /\!\!/_i^B\ s) \vee (\psi_2 /\!\!/_i^B\ s)$ by definition of quotienting. Also $\psi_1 /\!\!/_i^B\ s$ and $\psi_2 /\!\!/_i^B\ s$ are proper sub-formulas of $\psi /\!\!/_i^B\ s$. Therefore, by induction hypothesis we have $\mathsf{dec_s}\,(P, (\psi_1, s), E) =_f \psi_1 /\!\!/_i^B\ s$ and $\mathsf{dec_s}\,(P, (\psi_2, s), E) =_f \psi_2 /\!\!/_i^B\ s$. It follows immediately that $\mathsf{dec_s}\,(P, (\psi_1 \vee \psi_2, s), E) =_f (\psi_1 \vee \psi_2) /\!\!/_i^B\ s$.

Case $\psi = \langle a \rangle \psi_0$: By definition of $\mathsf{enc}\,(\varphi)$, the state $\langle a \rangle \psi_0$ has a transition $\langle a \rangle \psi_0 \xrightarrow{\langle a \rangle} \psi_0$. By definition of P, the state $(\langle a \rangle \psi_0, s)$ has three kinds of transitions:

- A transition of the form $(\langle a \rangle \psi_0, s) \xrightarrow{\langle a \rangle} (\psi_0, s)$ for each $(\mathbf{t}, a) \in V$ such that $i \notin A(\mathbf{t})$, because $V/\!\!/_i$ contains the synchronisation rule $((\langle a \rangle, \bullet), \langle a \rangle)$. This corresponds to a disjunct of the form $i \notin A(\mathbf{t}) \wedge \langle a \rangle (\psi_0 /\!\!/_i^B\ s)$ in the definition of $(\langle a \rangle \psi_0) /\!\!/_i^B\ s$.
- A transition of the form $(\langle a \rangle \psi_0, s) \xrightarrow{\langle \alpha\,(\mathbf{t}, a) \rangle} (\psi_0, s')$ for each $(\mathbf{t}, a) \in V$ such that $\{i\} \subset A(\mathbf{t})$ and for each transition $s \xrightarrow{\mathbf{t}[i]}_i s'$ in $\mathbf{S}[i]$, because $V/\!\!/_i$ contains the synchronisation rule $((\langle a \rangle, \mathbf{t}[i]), \langle \alpha\,(\mathbf{t}, a) \rangle)$. This corresponds to a disjunct of the form $\{i\} \subset A(\mathbf{t}) \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} \langle \alpha\,(\mathbf{t}, a) \rangle (\psi_0 /\!\!/_i^B\ s')$ in the definition of $(\langle a \rangle \psi_0) /\!\!/_i^B\ s$.
- A transition of the form $(\langle a \rangle \psi_0, s) \xrightarrow{\vee} (\psi_0, s')$ for each $(\mathbf{t}, a) \in V$ such that $\{i\} = A(\mathbf{t})$ and for each transition $s \xrightarrow{\mathbf{t}[i]}_i s'$ in $\mathbf{S}[i]$, because $V/\!\!/_i$ contains the synchronisation rule $((\langle a \rangle, \mathbf{t}[i]), \vee)$. This corresponds to a disjunct of the form $\{i\} = A(\mathbf{t}) \wedge \bigvee_{s \xrightarrow{\mathbf{t}[i]}_i s'} (\psi_0 /\!\!/_i^B\ s')$ in the definition of $(\langle a \rangle \psi_0) /\!\!/_i^B\ s$.

The state $(\langle a \rangle \psi_0, s)$ has no other transitions in P, because the state $\langle a \rangle \psi_0$ has no other transition and $V/\!\!/_i$ does not contain other synchronisation rules of either form $((\bullet, b), c)$ or $((\langle a \rangle, b), c)$. Also, $\psi_0 /\!\!/_i^B\ s$ and $\psi_0 /\!\!/_i^B\ s'$ are proper sub-formulas of $\psi /\!\!/_i^B\ s$. By induction hypothesis, we have $\mathsf{dec_s}\,(P, (\psi_0, s), E) =_f \psi_0 /\!\!/_i^B\ s$ and $\mathsf{dec_s}\,(P, (\psi_0, s'), E) =_f \psi_0 /\!\!/_i^B\ s'$. It then follows immediately that $\mathsf{dec_s}\,(P, (\langle a \rangle \psi_0, s), E) =_f (\langle a \rangle \psi_0) /\!\!/_i^B\ s$.

Case $\psi = \mu X^k.\psi_0$: By definition of P, the state $(\mu X^k.\psi_0, s)$ has a transition $(\mu X^k.\psi_0, s) \xrightarrow{\mu^k} (\psi_0, s)$, because by definition of $\mathsf{enc}\,(\varphi)$ the state $\mu X^k.\psi_0$ has a transition $\mu X^k.\psi_0 \xrightarrow{\mu^k} \psi_0$ and $V/\!\!/_i$ contains the synchronisation rule $((\mu^k, \bullet), \mu^k)$. The state $(\mu X^k.\psi_0, s)$ has no other transition in P, because the state $\mu X^k.\psi_0$ has no other transition and $V/\!\!/_i$ does not contain other synchronisation rules of either form $((\bullet, a), b)$ or $((\mu, a), b)$. We consider two cases:

- If $(\mu X^k.\psi_0, s) \in E$ then by hypothesis $X_s^k \in B$. On the one hand, we thus have $\mathsf{dec_s}\,(P, (\mu X^k.\psi_0, s), E) = \overline{(\mu X^k.\psi_0, s)}^k$ by definition of $\mathsf{dec_s}$. On the other hand, we have $(\mu X^k.\psi_0) /\!\!/_i^B\ s = X_s^k$ by definition of quotienting. We also have $\overline{(\mu X^k.\psi_0, s)}^k =_f X^k$ by definition of $=_f$ and because $\mu X^k.\psi_0 = \varphi[X^k]$. It follows immediately that $\mathsf{dec_s}\,(P, (\mu X^k.\psi_0, s), E) =_f (\mu X^k.\psi_0) /\!\!/_i^B\ s$.
- If $(\mu X^k.\psi_0, s) \notin E$ then by hypothesis $X_s^k \notin B$. On the one hand, we thus have $\mathsf{dec_s}\,(P, (\mu X^k.\psi_0, s), E) = \mu\overline{(\mu X^k.\psi_0, s)}^k . \,\mathsf{dec_s}\,(P, (\psi_0, s), E')$ where $E' = E \cup \{\mu X^k.\psi_0\}$, by definition

of $\mathsf{dec_s}$. On the other hand, we have $(\mu X^k.\psi_0) /\!\!/_i^B\ s = \mu X^k_s.(\psi_0 /\!\!/_i^{B\cup\{X^k_s\}}\ s)$ by definition of quotienting. Also, $\psi_0 /\!\!/_i^{B\cup\{X^k_s\}}\ s$ is a proper sub-formula of $\psi /\!\!/_i^B\ s$. By induction hypothesis, we thus have $\mathsf{dec_s}\,(P, (\psi_0, s), E') =_f \psi_0 /\!\!/_i^{B\cup\{X^k_s\}}\ s$ using $E' = E \cup \{(\mu X^k.\psi_0, s)\} = \{(\varphi[Y^k], t) \mid Y^k_t \in B \cup \{X^k_s\}\}$. It then follows immediately that $\mathsf{dec_s}\,(P, (\mu X^k.\psi_0, s), E) =_f (\mu X^k.\psi_0) /\!\!/_i^B\ s$.

□

Proposition 3 *The* Lts *corresponding to the quotient formula network of φ with respect to $\mathbf{S}[i]$ encodes the quotient of φ with respect to $\mathbf{S}[i]$.*

Proof. Let P be the quotient formula network of φ with respect to $\mathbf{S}[i]$, i.e., $P = \mathsf{lts}\,((\mathsf{enc}\,(\varphi), \mathbf{S}[i]), V/\!\!/_i)$. Since $\{(\varphi[Y^k], t) \mid Y^k_t \in \emptyset\} = \emptyset$, then we have by Lemma 2 that $\mathsf{dec_s}\,(P, (\varphi, s^i_0), \emptyset) =_f \varphi /\!\!/_i^{\emptyset}\ s^i_0$, where f maps each propositional variable $Y^k_t \in \mathsf{bv}\,(\varphi /\!\!/_i^B\ s^i_0)$ to $\overline{(\varphi[Y^k], t)}^k$. In other words P, the quotient formula network of φ with respect to $\mathbf{S}[i]$, encodes $\varphi /\!\!/_i^{\emptyset}\ s^i_0$, which is the quotient of φ with respect to $\mathbf{S}[i]$. □

Example 5 *Consider the network N of Example 1 (page 6) and the formula of Example 4 (page 9). 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 Figure 3 (b), page 9. This graph encodes as expected the quotient formula of Example 3 (page 7), 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 quotient of a formula graph with n states with respect to an Lts with m states may have up to $n \times m$ states. Hence, as observed by Andersen [2], simplifications are needed to keep intermediate quotiented formulas at a reasonable size. We present in Figure 4 several simplifications applying to formula graphs, as conditional rules of the form "$l \rightsquigarrow r$ (*cond*)" where l and r are transition relations and *cond* is a Boolean condition. l, r, and *cond* are expressed using variables representing either states (written $s, s_1, s_2, \ldots$) or labels (written $\sigma, \sigma_1, \sigma_2, \ldots$), such that every variable occurring in r or in *cond* must also 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 allows transitions generated by synchronisation rules of the form $((\langle a\rangle, \mathbf{t}[i]), \vee)$ in the quotient formula network to be eliminated. This elimination can be achieved efficiently by applying reduction modulo $\tau^*.a$ equivalence [17], $\vee$-transitions being interpreted as internal (τ) transitions.

Elimination of unguarded variables (2)

When combined with the previous rule, this rule allows unguarded variable occurrences to be eliminated. Indeed, an unguarded variable is characterized by a (possibly empty) sequence of $\vee$-transitions connecting the target and source of a μ-transition. The elimination of this sequence of $\vee$ transitions then produces a self-looping transition labelled by μ, which can be thereafter eliminated using the current rule.

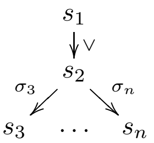
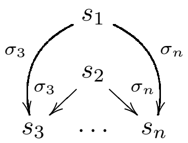
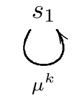
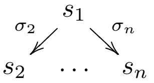
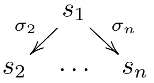
(1)		$\rightsquigarrow$		($s_3, \ldots, s_n$ are all the successors of s_2)
(2)		$\rightsquigarrow$	s_1	
(3)	$s_1 \xrightarrow{\neg} s_2 \xrightarrow{\neg} s_3$	$\rightsquigarrow$	$$\begin{tikzcd} s_1 \arrow[rr, bend right, "\vee"'] & s_2 \arrow[r, "\neg"] & s_3 \end{tikzcd}$$	(s_2 has a single outgoing transition)
(4)	$s_1 \xrightarrow{\mu^k} s_2$	$\rightsquigarrow$	$s_1 \xrightarrow{\vee} s_2$	(decoding of s_2 does not contain $\overline{s_1}^k$)
(5)	$s_1 \xrightarrow{\neg} s_2$	$\rightsquigarrow$	$s_1 \qquad s_2$	(s_2 evaluates to $\mathbf{tt}$)
(6)		$\rightsquigarrow$	$$\begin{tikzcd} & s_1 \arrow[r, "\neg"] & \mathbf{ff} \\ s_2 & \cdots & s_n \end{tikzcd}$$	(s_1 evaluates to $\mathbf{tt}$)
(7)	$s_1 \xrightarrow{\sigma} s_2$	$\rightsquigarrow$	$s_1 \qquad s_2$	($\sigma \neq \neg$ and s_2 evaluates to $\mathbf{ff}$)
(8)		$\rightsquigarrow$	$$\begin{tikzcd} & s_1 & \\ s_2 & \cdots & s_n \end{tikzcd}$$	(s_1 evaluates to $\mathbf{ff}$)

Figure 4: Simplification rules applying to formula graphs

Elimination of double-negations (3)

This rule can be used to simplify formulas of the form $\neg\neg\varphi$, which often occur in quotient formulas. For instance, a double-negation is introduced in the quotient of the formula $\neg\langle a\rangle\neg\varphi'$ with respect to an Lts that offers an action synchronising with a (thus having the modality disappear if the synchronisation is binary).

Elimination of μ-transitions (4)

In this rule, the transition from s_1 to s_2 denotes the binder of a propositional variable $\overline{s_1}^k$. If this variable does not occur free in the sub-formula denoted by state s_2, then the μ-transition can be replaced by an $\vee$-transition, which can be subsequently eliminated using rule (1). Determining whether $\overline{s_1}^k$ occurs free would require to decode the formula graph, which should be avoided in practice. For this reason, we only consider the following sufficient conditions, which can be checked in linear-time:

- s_1 and s_2 are not in the same strongly connected component (i.e., there is no path from s_2 to s_1), or
- s_1 is not the initial state and has a single predecessor p, and either p has a single outgoing transition (which necessarily goes to s_1) and this transition is labelled by $\mu^{k'}$, or p satisfies the same condition as s_1, recursively (this recursive condition is well-founded as long as it is applied to states reachable from the initial state)

Evaluation of constant sub-formulas (5–8)

These four rules apply when some state denotes a sub-formula that evaluates to a constant in any context. This can be determined by using the following Bes, which implements partial evaluation of the formula. This Bes consists of 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.

$F^{k'}$):

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

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 $\mathbf{ff}$ 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 $\mathbf{ff}$ and all its successors following transitions labelled by $\neg$ evaluate to $\mathbf{tt}$. Regarding fix-point signs, observe that for the formula $\mu X^k . X^k$ (which is equivalent to the constant $\mathbf{ff}$), $F^k_{\mu X^k . X^k}$ and $T^k_{\mu X^k . X^k}$ are defined respectively by the greatest fix-point equation $F^k_{\mu X^k . X^k} =_\nu F^k_{\mu X^k . X^k}$ and the least fix-point equation $T^k_{\mu X^k . X^k} =_\mu T^k_{\mu X^k . X^k}$. This Bes has the solution $F^k_{\mu X^k . X^k} = \mathbf{tt}, T^k_{\mu X^k . X^k} = \mathbf{ff}$, reflecting the constant value false of $\mu X^k . X^k$ as expected.

Repeated application of quotienting progressively eliminates 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 eight 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.

Example 6 *After applying the above simplifications to the formula graph of Example 5 (page 12), we obtain the (smaller) formula graph depicted in Figure 3 (c), page 9, 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* $\mathbf{ff}$ *(left as an exercise).*

Note that the simplification of a formula graph produces a formula graph. In particular, the parity of the number of occurrences of the label $\neg$ on paths leading to a μ^k-transition is not changed by any rule, including rule (3) which eliminates negations by pair. Also, the simplifications do not create new circuits and every μ^k-transition eliminated by rule (4) cannot be the first μ^k-transition occurring on any circuit.

All the simplifications that we propose in this paper correspond more or less to simplifications already proposed by Andersen [2], but we apply them directly on formula graphs instead of systems of μ-calculus equations. For the interested reader, we review below the simplifications proposed by Andersen and detail how they map to our simplification rules:

- *Reachability analysis* is included in our setting, due to our definition of the quotient on formulas (instead of systems of equations), which necessarily yields connected formulas (or formula graphs). In practice, reachability analysis is achieved using *on-the-fly* graph traversals, in particular on-the-fly generation of the Lts corresponding to the quotient network.

- *Simple evaluation*, *constant propagation*, and *trivial equation elimination* are implemented by rules 5–8. The Bes that we have proposed for partial evaluation seems however slightly more general than Andersen's simplification rules, which do not seem to provide means to evaluate X to $\mathbf{ff}$ in the system of equations "$X =_\mu \langle a \rangle Y \vee \langle c \rangle X, Y =_\mu \langle b \rangle X$", whereas the corresponding formula (see Example 7) evaluates as expected to $\mathbf{ff}$ in our setting.

- The approximation of *equivalence reduction* proposed by Andersen, which relies on a heuristic, is the same as our sharing of equivalent sub-formulas, implemented by strong bisimulation reduction. This can be seen easily as the definition of the heuristic in [2] looks very similar to the definition of strong bisimulation on Ltss.
- *Unguarded equations elimination* is implemented by the combination of rules 1–3.

About correctness of the simplifications

The eight simplification rules preserve the semantics of the encoded formula. We do not provide the formal proof of this statement, but we give the intuitions behind this result. Intuitively, every rule defines a rather simple transformation on a set of equations. Rule (1) replaces the set $\{s_1 = s_2, s_2 = \psi\}$ by $\{s_1 = \psi, s_2 = \psi\}$, which is correct independently of the fix-point sign. Rule (2) replaces the equation $\{s_1 =_\mu s_1 \vee \psi\}$ by $\{s_1 =_\mu \psi\}$, which is a well-known transformation of the μ-calculus. Rule (3) replaces $\{s_1 = \neg s_2 \vee \psi, s_2 = \neg s_3\}$ by $\{s_1 = s_3 \vee \psi, s_2 = \neg s_3\}$. Rule (4) reflects the fact that the fix-point sign of an equation does not influence the result of its resolution if the bound variable has no free occurrence in the set of equations. Rules (5) to (8) express that any variable can be replaced by its solution. At last, the sharing of equivalent formulas reflect that two variables can be merged if they are defined in the same block and if they have the same definition modulo variable names. The correctness of a similar transformation has been proven in [2].

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 ($L\mu_1$), which has a linear-time model checking complexity [14] 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 Figure 5, 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 [41].

$$T^k : \left\{ \begin{array}{lll} T^{+k}_s & =_\mu & \bigvee_{s \xrightarrow{\vee} s'} T^{+k}_{s'} \vee \bigvee_{s \xrightarrow{\neg} s'} T^{-k}_{s'} \vee \bigvee_{s \xrightarrow{\mu k'} s'} T^{+k'}_{s'} \\ T^{-k}_s & =_\mu & \bigwedge_{s \xrightarrow{\vee} s'} T^{-k}_{s'} \wedge \bigwedge_{s \xrightarrow{\langle\beta\rangle} s'} T^{-k}_{s'} \wedge \bigwedge_{s \xrightarrow{\neg} s'} T^{+k}_{s'} \wedge \bigwedge_{s \xrightarrow{\mu k'} s'} F^{+k'}_{s'} \end{array} \right\}_{s \in \Sigma}$$

$$F^k : \left\{ \begin{array}{lll} F^{+k}_s & =_\nu & \bigwedge_{s \xrightarrow{\vee} s'} F^{+k}_{s'} \wedge \bigwedge_{s \xrightarrow{\langle\beta\rangle} s'} F^{+k}_{s'} \wedge \bigwedge_{s \xrightarrow{\neg} s'} F^{-k}_{s'} \wedge \bigwedge_{s \xrightarrow{\mu k'} s'} F^{+k'}_{s'} \\ F^{-k}_s & =_\nu & \bigvee_{s \xrightarrow{\vee} s'} F^{-k}_{s'} \vee \bigvee_{s \xrightarrow{\neg} s'} F^{+k}_{s'} \vee \bigvee_{s \xrightarrow{\mu k'} s'} T^{+k'}_{s'} \end{array} \right\}_{s \in \Sigma}$$

Figure 5: Bes for the evaluation of constant alternation-free formulas

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 $k' > k$.

7 Handling fairness operators

In the previous sections, we described a partial model checking procedure for the full modal μ-calculus $L\mu$, which we then specialised to the alternation-free fragment $L\mu_1$. This fragment allows to express certain simple fairness operators, such as the fair reachability of actions (i.e., potential reachability by skipping cycles), originally proposed in the state-based setting [48]. The fair reachability of an action a is expressed by the following $L\mu_1$ formula (where $\neg a$ denotes all actions except a), stating that as long as a has not been encountered, it is still possible to reach it: $\nu X.(\mu Y.(\langle a\rangle\, \mathsf{tt} \vee \langle \mathsf{tt}\rangle\, Y) \wedge [\neg a]\, X)$. An equivalent, more concise, formulation of this property using the operators of PDL [18] is $[(\neg a)^*]\, \langle \mathsf{tt}^*.a\rangle\, \mathsf{tt}$.

More elaborate fairness properties can be conveniently expressed by characterizing unfair cycles using the infinite looping operator ΔR of PDL-Δ [49], which states the existence of an infinite transition sequence made by concatenation of subsequences that satisfy the regular expression R. The ΔR operator can be translated into the fix-point formula $\nu X.\, \langle R\rangle\, X$, which can be further expanded into a plain μ-calculus formula [16]. This operator can encode the existence of accepting cycles in Büchi automata, and therefore it is able to capture LTL properties; in fact, this operator brings significant expressive power to PDL, making PDL-Δ more expressive than CTL* [50]. When the regular expression R contains Kleene star operators, the operator ΔR yields a formula of $L\mu_2$, the μ-calculus fragment of alternation depth 2. Although this fragment has a quadratic worst-case model checking complexity [16], the ΔR operator can be checked on-the-fly in linear-time by formulating the problem as a BES resolution and applying the $\mathrm{A4}_{cyc}$ algorithm [46]. This algorithm generalizes the resolution algorithm A_4 for disjunctive BESs [43] by enabling the detection of cycles in the underlying Boolean graphs that pass through marked Boolean variables, in a way similar to the detection of accepting cycles in Büchi automata. However, this does not yield a linear-time model checking for LTL (resp. CTL*) because the translations from LTL model checking problems to Büchi automata (resp. from CTL* formulas to PDL-Δ) are not succinct.

We propose a way to evaluate the ΔR operator on a network of LTSs using partial model checking, without developing the complex (and quadratic-time) machinery needed to evaluate general $L\mu_2$ formulas. We rely instead on the approach proposed in [46], which transforms the evaluation of ΔR into the resolution of an alternation-free BES containing marked Boolean variables. We first illustrate this approach using an example of ΔR operator where R contains star operators, and then we show its application in the partial model checking framework.

Consider the formula $\Delta((a|b)^*.c)$, which is equivalent to the $L\mu$ formula $\nu X.\, \langle (a|b)^*.c\rangle\, X$. The regular diamond modality can be further expanded by repeatedly applying the classical PDL identities ($\langle R_1.R_2\rangle\, \varphi = \langle R_1\rangle\, \langle R_2\rangle\, \varphi$, $\langle R_1|R_2\rangle\, \varphi = \langle R_1\rangle\, \varphi \vee \langle R_2\rangle\, \varphi$, and $\langle R^*\rangle\, \varphi = \mu Y.(\varphi \vee \langle R\rangle\, Y)$) until all regular operators have been eliminated:

$$\begin{array}{rcl} \nu X.\, \langle (a|b)^*.c\rangle\, X & = & \nu X.\, \langle (a|b)^*\rangle\, \langle c\rangle\, X \\ & = & \nu X.\mu Y.(\langle c\rangle\, X \vee \langle a|b\rangle\, Y) \\ & = & \nu X.\mu Y.(\langle c\rangle\, X \vee \langle a\rangle\, Y \vee \langle b\rangle\, Y) \end{array}$$

The resulting $L\mu_2$ formula can be written equivalently as a modal equation system containing two mutually recursive blocks with opposite fix-point signs:

$$\{X \stackrel{}{=}_\nu Y\}, \{Y \stackrel{}{=}_\mu \langle c\rangle\, X \vee \langle a\rangle\, Y \vee \langle b\rangle\, Y\}$$

The evaluation of variable X on a state s is reformulated as the resolution of the Boolean variable X_s of the following BES:

$$\{X_s =_\nu Y_s\}_{s\in S}, \{Y_s =_\mu \bigvee_{s \xrightarrow{c} s'} X_{s'} \vee \bigvee_{s \xrightarrow{a} s'} Y_{s'} \vee \bigvee_{s \xrightarrow{b} s'} Y_{s'}\}_{s\in S}$$

We observe that the ν-block contains only singular equations, the μ-block is disjunctive (i.e., all right-hand sides of equations contain only disjunctions), and does not contain tt constants but possibly ff constants (which correspond to empty disjunctions). This structure, which is guaranteed by construction for every BES encoding the evaluation of a ΔR operator, enables to obtain a linear-time resolution procedure in the following way: (a) The ν-block is merged into the μ-block by changing the fix-point sign of its equations (this operation is abusive, since it changes the semantics of the BES); (b) In the resulting μ-block, the X_s Boolean variables are marked (with the superscript $^@$) in order to retrieve the original semantics of the BES during resolution. For the example considered, this procedure yields the following single-block BES:

$$\{X_s^@ =_\mu Y_s, Y_s =_\mu \bigvee_{s \xrightarrow{c} s'} X_{s'}^@ \vee \bigvee_{s \xrightarrow{a} s'} Y_{s'} \vee \bigvee_{s \xrightarrow{b} s'} Y_{s'}\}_{s\in S}$$

If the LTS does not contain any infinite sequence belonging to the ω-regular language $((a|b)^*.c)^\omega$, the initial formula $\Delta((a|b)^*.c)$ evaluates to ff, which is also the result of evaluating variable X_{s_0} in the μ-block above.

If there exists such an infinite sequence going out of the initial state s_0, the initial formula evaluates to $\mathbf{tt}$, whereas variable X_{s_0} in the μ-block above does still evaluate to $\mathbf{ff}$ (given the absence of $\mathbf{tt}$ constants in this Bes). The existence of such an infinite sequence in the Lts corresponds to a cycle in the Boolean graph associated to the Bes, which passes through some X_s variable. Therefore, to retrieve the original semantics of the two-block Bes the resolution algorithm must mark the X_s variables and detect whether one of these variables $X_s^@$ belongs to a cycle; if this is the case, then the variable is replaced by a $\mathbf{tt}$ constant, which forces (by back-propagation through the disjunctive operators) the variable X_{s_0} to evaluate to $\mathbf{tt}$.

This kind of resolution is carried out in linear-time by the $\mathrm{A4}_{cyc}$ algorithm [46], based on a depth-first search of the Boolean graph with detection of cycles containing marked variables by computing the strongly connected components. This algorithm is robust w.r.t. repeated invocations, i.e., a sequence of calls has a cumulated linear-time complexity, which enables the evaluation of ΔR operators nested with (alternation-free) fix-point operators without losing the overall linear-time complexity in the size of the Bes.

This evaluation procedure for ΔR operators can be applied in the partial model checking setting by abusively merging the two equation blocks into a single one, producing the formula graph in which the X variable is marked (using an outgoing transition labeled by a special action $\mu@$), carrying out the projection steps, obtaining in the last step a modality-free formula graph corresponding to a Bes with marked variables, and solving this Bes using the $\mathrm{A4}_{cyc}$ algorithm. During the projection steps, partial evaluation is carried out on the formula graph by using the same Bes as in Section 6, slightly extended to take into account the transitions labeled by $\mu_@^k$ corresponding to marked variables. Every μ-block corresponding to a ΔR operator (with marked variables) is assigned a unique block number. Partial evaluation is carried out using algorithm $\mathrm{A4}_{cyc}$ every time a variable Y belonging to such a block is encountered: if the algorithm detects a modality-free cycle containing a marked variable of that block, the variable Y evaluates to $\mathbf{tt}$.

Figure 6 illustrates the partial model checking of a formula containing an infinite looping operator on a network representing a semaphore-based mutual exclusion protocol. The network $N = ((P_0, S, P_1), V)$, shown in Figure 6(a), consists of two processes P_0 and P_1 competing for a shared resource, and a semaphore S guarding the access to the resource. Each process P_i (for $i \in \{0, 1\}$) cyclically executes the following sequence: first it performs its non-critical section ncs_i, then it requests the access to the resource by synchronising with the semaphore on req_i, then it accesses the resource during its critical section cs_i, and finally it releases the semaphore by synchronising on rel_i. The three processes interact via the following set of synchronisation vectors:

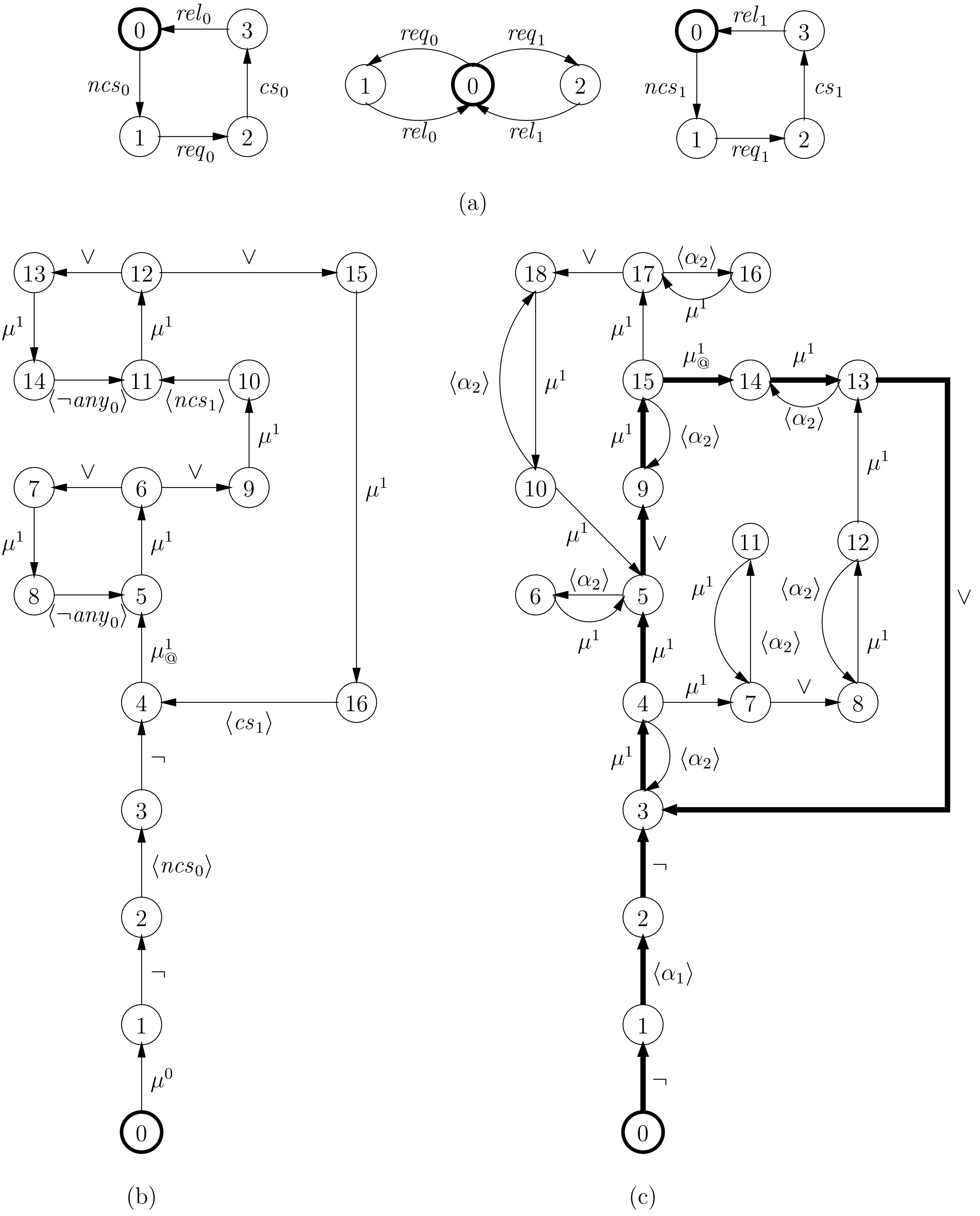
$$\begin{aligned} V = \{\ & ((req_0, req_0, \bullet), req_0), ((rel_0, rel_0, \bullet), rel_0), \\ & ((\bullet, req_1, req_1), req_1), ((\bullet, rel_1, rel_1), rel_1), \\ & ((ncs_0, \bullet, \bullet), ncs_0), ((cs_0, \bullet, \bullet), cs_0), \\ & ((\bullet, \bullet, ncs_1), ncs_1), ((\bullet, \bullet, cs_1), cs_1) \qquad \} \end{aligned}$$

The Pdl-Δ formula checked on the network N is $[ncs_0]\, \Delta((\neg any_0)^*.ncs_1.(\neg any_0)^*.cs_1)$, stating that after P_0 executes its non-critical section, it may never access the shared resource because of a systematic overtaking by P_1 (the action formula $\neg any_0$ denotes the set of actions not executed by P_0, i.e., $\{ncs_1, req_1, cs_1, rel_1\}$). This formula can be expressed in $L\mu$ as $\nu X.\mu Y.(\langle ncs_1 \rangle\, \mu Z.(\langle cs_1 \rangle\, X \vee \langle \neg any_0 \rangle\, Z) \vee \langle \neg any_0 \rangle\, Y)$, or equivalently as the modal equation system $\{U =_\mu \neg \langle ncs_0 \rangle \neg X^@, X^@ =_\mu Y, Y =_\mu \langle ncs_1 \rangle\, Z \vee \langle \neg any_0 \rangle\, Y, Z =_\mu \langle cs_1 \rangle\, X^@ \vee \langle \neg any_0 \rangle\, Z\}$, in which the equation defining $X^@$ has been abusively merged into the minimal fix-point block. The graph corresponding to this formula, where $X^@$ is marked by means of an outgoing transition labeled by $\mu_@^1$, is shown in Figure 6(b).

At the last step of the partial model checking procedure (i.e., after quotienting w.r.t. processes P_1 and S), the formula graph obtained contains a modality-free cycle passing through X, indicated with thick arrows in Figure 6(c). This cycle is detected in linear-time by applying the simplification procedure, which invokes the Bes resolution algorithm $\mathrm{A4}_{cyc}$. We observe that the quotienting w.r.t. process P_0 was not necessary (and not done), since the presence of the cycle containing X was detected as soon as processes P_1 and S were taken into account.

8 Implementation

We have implemented partial model checking of the alternation-free μ-calculus extended with the ΔR fairness operator. We used Cadp, which provided much of what was needed:



(a)

(b)

(c)

Figure 6: Partial model checking of a fairness property on a network

- Individual processes can be described in one of the numerous formats and languages available in Cadp: directly as Ltss in, e.g. the Bcg file format[2], or as high-level processes in the Lotos [27], Lotos NT [12] (a variant of E-Lotos [28]), or Fsp [39] languages. Cadp contains tools to generate Ltss in the Bcg format automatically from those three languages. For the latter two, this is done via an automated generation of intermediate Lotos code using translators [35, 12]. Other languages can easily be connected to Cadp using either the same approach (for instance a connection of the applied π-calculus [44]), or through the Open/Cæsar [19] Api of Cadp.

- Process compositions can be described in the Exp.Open 2.0 language [31], which provides various parallel composition operators, such as synchronisation vectors [6], process algebra operators (Lotos, Ccs, Csp, μCrl), and the generalised parallel composition operator of E-Lotos/Lotos NT [24]. 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 using the Open/Cæsar interface [19], 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.

- Regular alternation-free μ-calculus formulas (i.e., an extension of the alternation-free μ-calculus with action formulas and regular expressions inside modalities to represent actions and sequences of actions) extended with the ΔR fairness operator can be handled by the Evaluator on-the-fly model checker [45, 46]. Regular expressions inside modalities are eliminated by Evaluator and replaced by ordinary fix-point formulas with mere action formulas inside the modalities.

  An option has been added for compiling the formula into a formula graph represented in the Bcg format. This option also takes as input the set of actions potentially occurring in the process composition (which can be obtained using Exp.Open 2.0), so that the action formulas can be replaced by finite sets of actions.

- Reductions modulo $\tau^*.a$ equivalence and strong bisimulation are achieved using respectively the Reductor and Bcg_Min tools of Cadp, without any modification.

Elimination of double-negations, of μ-transitions, and evaluation of constant formulas (for $L\mu_1$ extended with the ΔR operator) have been implemented in a new prototype tool[3] ($1,000$ lines of C code), which relies on the Caesar_Solve library [43] for solving alternation-free Bes (extended to handle fairness as explained in Section 7). Finally, the Lts w.r.t. which the formula is quotiented at each step is selected automatically using the *smart* heuristic, described in [15].

9 Experimentation

We have used partial model checking in two case studies, one in avionics addressing the verification of a communication protocol between a plane and the ground, based on Tftp (*Trivial File Transfer Protocol*)/Udp (*User Datagram Protocol*) and the other one in hardware, addressing the verification of the bus arbitration protocol used in the Scsi-2 standard.

9.1 Trivial File Transfer Protocol/User Datagram Protocol

The Tftp/Udp case-study has been described by Garavel & Thivolle in [25]. In this section, we consider the same specifications and compare our new partial model checking approach with on-the-fly model checking.

[2] `http://cadp.inria.fr/man/bcg.html`

[3] This prototype tool, accompanied with a shell-script implementing partial model checking, a manual, and examples, can be downloaded at `http://convecs.inria.fr/software/pmc`. Cadp is required to be installed for the script and the prototype tool to run. Cadp licenses are free for academic users.

Scenario	TFTP A		TFTP B	
	read	write	read	write
A		✓		
B	✓			
C		✓		✓
D	✓			✓
E	✓		✓	

Table 1: The five scenarios of the TFTP/UDP case study

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E	
	States	Trans.	States	Trans.	States	Trans.	States	Trans.	States	Trans.
TFTP_A	704	4,542	719	4,610	704	4,542	719	4,610	719	4,610
TFTP_B	504	3,421	504	3,421	1,058	7,164	1,058	7,164	1,058	7,164
MEDIUM_{A,B}	801	5,440	801	5,440	801	5,440	801	5,440	801	5,440
SND_A, RCV_B	1	4	1	4	1	7	1	5	1	6
SND_B, RCV_A	1	4	1	3	1	7	1	6	1	6
Product ($\times 10^3$)	$1,963$	$8,527$	867	$3,737$	$35,024$	$151,810$	$40,856$	$189,068$	$19,436$	$83,921$

Table 2: Individual Lts sizes (in states and transitions) and product Lts size (in kilostates and kilotransitions) for each scenario

The system consists of two instances (A and B) of the Tftp connected by Udp using a Fifo buffer. Since the state space of the specification is very large in the general case, Garavel & Thivolle have defined five scenarios named A to E, depending on whether each instance may write and/or read a file (see Table 1). We have considered the same five scenarios in our study. All of them are specified in Lotos, as the parallel composition of eight processes named TFTP_A, TFTP_B, MEDIUM_A, MEDIUM_B, RCV_A, RCV_B, SND_A, and SND_B. The Ltss corresponding to those eight processes are generated automatically from their Lotos specification using the Caesar tool of Cadp. Their parallel composition is translated into a network of Ltss using the Exp.Open tool of Cadp. Table 2 provides the sizes after reduction of the Ltss corresponding to the eight processes for each scenario, as well as the size of their composition.

We considered the (alternation-free) μ-calculus (branching-time) properties named $A01$ to $A28$, studied in [25], as well as an additional alternation-2 fairness property $A29$ not checked in [25]. We checked all properties both using the well-established on-the-fly model checker Evaluator [45, 46] 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 Table 3 give, for each scenario and 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, which explains the shaded cells. The symbol "$\star$" corresponds to verifications that have been stopped because they took too long and used too much memory. The execution times are given in Table 4. Note that the major part of time and memory are used by formula simplifications, as compared to the rather 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. This is particularly the case of some formulas of either form $[R]\,\mathbf{ff}$ or $\langle R\rangle\,\mathbf{tt}$, where R is a regular expression, which denote the absence, respectively the existence, of a sequence of transitions that matches R. The quotient evaluates to true (in the case of formulas of the form $[R]\,\mathbf{ff}$) or false (in the case of formulas of the form $\langle R\rangle\,\mathbf{tt}$) before all individual Ltss have been taken into account in the quotient, because it has been possible to determine that none of the paths possible in the parts of the system already taken into account in the quotient may yield a path satisfying R in the global system. We illustrate this by giving details on the verification of formula $A09b$ on Scenario C. This formula has the form $[R]\,\mathbf{ff}$ and evaluates to true after the partial model checking steps reported in the

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E	
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	$\star$		
$A18$			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
$A29$	198	7	88	7	2,942	9	3,350	7	1,525	9

Table 3: Experimental results for the TFTP/UDP case study: memory (in megabytes)

following table.

Step	States	Transitions
Initial formula graph	13	62
Simplification & reduction	7	56
Quotient wrt. TFTP_A	125	1,964
Simplification & reduction	60	1,512
Quotient wrt. TFTP_B	9,166	69,490
Simplification & reduction	5,308	50,799
Quotient wrt. MEDIUM_B (encodes **tt**)	2	1

The fairness formula $A29$ is also evaluated efficiently using partial model checking. This formula is specified in PDL as $\Delta\,(\mathbf{tt}^*.A_1.(\neg(A_1 \vee A_2))^*.A_3.(\neg A_1)^*.A_2)$ (or $\langle \mathbf{tt}^*.A_1.(\neg(A_1 \vee A_2))^*.A_3.(\neg A_1)^*.A_2 \rangle$ @ in the MCL input language of EVALUATOR) and denotes the existence of a cyclic sequence of transitions matching the regular expression $\mathbf{tt}^*.A_1.(\neg(A_1 \vee A_2))^*.A_3.(\neg A_1)^*.A_2$, where A_1, A_2, and A_3 are particular actions. It evaluates to false on all scenarios. The first steps of partial model checking for this formula on Scenario E are detailed in the following table.

Step	States	Transitions
Initial formula graph	19	151
Simplification & reduction	7	139
Quotient wrt. TFTP_B	903	20,388
Simplification & reduction	896	20,099
Quotient wrt. TFTP_A	26,369	197,480
Simplification & reduction (encodes **ff**)	1	0

In a few other cases, partial model checking leads to combinatorial explosion (properties $A12$, $A13$, $A15$, and $A17$) while on-the-fly model checking performs efficiently. We illustrate this with the verification of formula $A12$ on scenario C. This formula has the form $\langle R \rangle\, \mathbf{tt}$ and evaluates to true. The first steps of partial model checking are detailed in the following table, in which we provide the time and memory used to complete each step. The reduction step includes both the pre-reduction modulo $\tau^*.a$ equivalence (i.e.,

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E	
Prop	fly	pmc	fly	pmc	fly	pmc	fly	pmc	fly	pmc
$A01$	28	2	10	3	1,324	3	1,590	2	772	3
$A02$	31	3	12	3	1,640	6	2,010	7	883	6
$A03$	22	1	8	1	1,210	1	1,365	1	668	1
$A04$	26	3	10	3	1,400	3	1,598	3	770	3
$A05$	1	5	1	5	1	5	1	5	1	5
$A06$	23	3	9	3	1,306	3	1,540	3	667	3
$A07$	23	3	9	3	1,299	3	1,687	3	674	3
$A08$	22	3	8	3	1,220	3	1,620	3	625	3
$A09a$							1,679	7	695	3
$A09b$					1,415	8				
$A10$					2,112	3			929	3
$A11$					1,722	3	3,583	1	997	3
$A12$					76	⋆	8	⋆	6	⋆
$A13$							3,297	⋆	1,446	⋆
$A14$	54	3			2,681	3			1,443	3
$A15$			11	5	55	⋆	15	⋆	705	7
$A16$									40	1
$A17$					315	⋆	217	⋆		
$A18$			9	1	86	7	35	3	599	1
$A19$			53	3	6,159	3	9,393	3	2,697	3
$A20$	1	3			224	6			39	6
$A21$	131	3			4,004	3			2,293	3
$A22$			1	12			147	2,712	43	1,007
$A23$			39	3			5,605	9	2,345	6
$A24$	1	13			148	3,189				
$A25$	133	3			4,163	6				
$A26$	25	3			1,383	3			687	3
$A27$	38	3			2,323	3			1,196	3
$A28$			15	3	2,538	3	2,615	3	1,277	3
$A29$	26	2	11	2	1,524	6	1,738	3	700	5

Table 4: Experimental results for the TFTP/UDP case study: time (in seconds)

elimination of τ-transitions) and the reduction modulo strong bisimulation. Note that this may produce a graph that is not minimal in number of transitions, although always minimal in number of states.

Step	Time (s)	Memory (MB)	States	Transitions
Initial formula graph			8	56
Simplification	0	4	8	56
Reduction	0	66	4	52
Quotient wrt. TFTP_A	0	66	210	5,687
Simplification	0	4	136	3,665
Reduction	0	66	134	3,587
Quotient wrt. TFTP_B	0	66	21,172	168,172
Simplification	0	6	21,015	168,172
Reduction	1	66	14,042	119,789
Quotient wrt. MEDIUM_B	14	66	1,648,096	10,327,294
Simplification	35	267	1,648,089	10,327,294
Reduction	72	234	1,551,338	14,773,975
Quotient wrt. MEDIUM_A	686	540	40,572,824	229,050,227
...				

This explosion seems inherent to the structure of the system and the formula, 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 partial and on-the-fly model checking are complementary and worthy of being used concurrently.

9.2 The SCSI-2 Bus Arbitration Protocol

This case-study has been described by Garavel & Hermanns in [20]. It was originally designed to illustrate the combination of functional verification and performance evaluation features of CADP. In this section, we reuse the specification (a CADP demo available on-line at `ftp://ftp.inrialpes.fr/pub/vasy/demos/demo_31`) to compare on-the-fly verification of an alternation-2 fairness formula with its verification using partial model checking.

The case-study represents a storage system (developed by Bull in the early 90's) consisting of up to eight *devices* (up to seven hard disks and a disk controller) connected by a bus (which enables eight connections) implementing the Scsi-2 standard. Each device is assigned a unique Scsi-number ranging between 0 and 7, the device assigned the highest number having highest priority when several devices are ready to access the bus. Each disk is represented by a process named DISK_n, the controller by a process named CONTROLLER_n, and each unused connection to the bus by a process named NO_DEVICE_n, n corresponding to the assigned Scsi-number. The controller may send randomly to any disk of number n a message "`CMD !`n" (*command*) indicating a transfer request (read/write a block of data from/to the disk). After processing this command, the disk sends back to the controller a message "`REC !`n" (*reconnect*).

We considered the alternation-2 fairness property expressing that after the controller (of number c) sends a data transfer request to disk number n such that $n < c$, then for each disk of number m such that $m > n$, there must exist a cyclic execution sequence matching the regular expression $(\neg\texttt{REC !}n)^*\cdot\texttt{CMD !}m\cdot(\neg\texttt{REC !}n)^*\cdot\texttt{REC !}m$, i.e., the processing of data transfer request with a disk that has not priority over the controller does not prevent other requests to be processed by disks of higher priority.

In a first step, we considered two different configurations (named A and B) of the storage system, each consisting of three disks, one controller and four unused connections. In configuration A, the controller is assigned number 7 and the disks are assigned numbers 0 to 2. In configuration B, the controller is assigned number 1 and the disks are assigned numbers 0, 2, and 3. In both configurations, the Lts corresponding to the system has $56,168$ states and $154,748$ transitions[4].

Configuration A satisfies the fairness property. On-the-fly model checking takes 1.67 seconds and 66 MB, whereas partial model checking takes 4 minutes and 107 MB. The largest intermediate formula graph has $489,983$ states and $4,336,623$ transitions. On the contrary, configuration B violates the property. On-the-fly model checking takes 1.12 seconds and 66 MB, whereas partial model checking takes 19.16 seconds and 66 MB. The largest intermediate formula graph has $22,171$ states and $198,467$ transitions.

The performance of partial model checking on configuration B is interesting, because intermediate formula graphs always remain smaller than the product Lts. To see how this scales up, we evaluated the property on larger configurations, still assigning number 1 to the controller, but progessively replacing the unused connections by additional disks (up to 6 disks, the configuration with 7 disks being too large for model checking). The results are given in Table 5. Note that partial model checking scales well on this example as, for configurations with five disks and more, it terminates faster than the product Lts generation. We summarize in the following table the sizes of intermediate formula graphs during the partial model checking of the configuration with 6 disks.

Step	Time (s)	Memory (MB)	States	Transitions
Initial formula graph			109	360
Simplification	0	4	9	28
Reduction	0	66	6	25
Quotient wrt. CONTROLLER_1	734	1,165	19,545,220	332,937,946
Simplification	1,021	7,630	19,072,829	332,937,946
Reduction	1,807	7,483	12,400,293	326,265,410
Quotient wrt. NO_DEVICE_6	489	1,472	12,400,293	320,065,265
Simplification	801	7,234	12,400,293	320,065,265
Reduction	2,219	4,673	12,400,293	547,718,843
Quotient wrt. DISK_0	1,073	2,657	29,367,067	710,452,069
Simplification	721	17,594	1,345,007	36,186,832
Reduction	145	479	1,285,959	36,127,784
Quotient wrt. DISK_7	145	297	3,101,185	51,160,987
Simplification	271	1,129	3,101,177	, 51,160,987
Reduction	285	744	3,101,169	51,160,979
Quotient wrt. DISK_5	125	283	7,124,779	78,762,466
Simplification	389	1,765	7,124,771	78,762,466
Reduction	652	1,398	6,024,459	103,247,732
Quotient wrt. DISK_4	276	623	13,770,325	152,237,584
Simplification	971	3,449	13,770,317	152,237,584
Reduction	1,717	2,632	12,201,825	223,819,978
Quotient wrt. DISK_3	680	1,330	27,557,019	290,881,082
Simplification	1,721	6,667	27,557,011	290,881,082
Reduction	5,099	5,571	25,967,207	442,140,277
Quotient wrt. DISK_2	1,002	2,521	44,137,283	343,601,116
Simplification	417	7,791	1	0

[4] Actually, a third configuration C is proposed in the on-line Cadp demo, with the controller assigned number 0. We have not considered this configuration as the fairness formula is trivially true in this case, the controller yielding priority to all disks.

	Number of disks			
	3	4	5	6
DISK_n size (states)	768	768	768	768
DISK_n size (transitions)	5, 119	9, 215	17, 407	33, 791
CONTROLLER_n size (states)	4, 617	53, 217	583, 929	6, 200, 145
CONTROLLER_n size (transitions)	32, 373	630 828	12, 237, 723	238, 990, 986
NO_DEVICE_n size (states)	1	1	1	1
NO_DEVICE_n size (transitions)	16	32	64	128
Product Lts size (states)	56, 168	1, 384, 021	32, 003, 282	708, 174, 559
Product Lts size (transitions)	154, 748	4, 499, 237	119, 691, 662	2, 992, 012, 087
Generation time (seconds)	1	17	884	31, 193
Memory peak (MB)	66	66	680	17, 594
	On-the-fly model checking			
Verification time (seconds)	1	17	1, 273	47, 532
Memory peak (MB)	66	95	1, 705	39, 236
	Partial model checking			
Verification time (seconds)	19	61	759	24, 276
Memory peak (MB)	66	66	1, 007	16, 239
Largest formula graph (states)	22, 171	253, 723	2, 773, 147	29, 367, 067
Largest formula graph (transitions)	198, 467	3, 023, 449	45, 639, 547	710, 452, 069

Table 5: Experimental results for Scsi-2 bus arbitration, 3 to 6 disks (configuration B)

10 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 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 solver. Using a graph equivalence relation to simplify the formula was already proposed in [7], 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 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) extended with the alternation-2 ΔR operator of Pdl-Δ has also been proposed, and experiments have been conducted, 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 [33, 42, 47], or restriction using interface constraints following the approach developed in [26] and refined in [21, 30, 32]. 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.

Acknowledgements

The authors warmly thank Hubert Garavel, Wendelin Serwe, and Damien Thivolle for providing the sources of case-studies. They thank the past and present developers of CADP, without which this work would not have been possible. They also thank the anonymous referees, whose remarks greatly helped to improve this paper.

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 *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science LICS*, IEEE Computer Society Press, 1995.

[3] H.R. Andersen and J. Lind-Nielsen. Partial Model Checking of Modal Equations: A Survey. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1999):242–259, 1999.

[4] H.R. Andersen, J. Staunstrup, and N. Maretti. A Comparison of Modular Verification Techniques. In *Proceedings of the 7th International Joint Conference CAAP/FASE*, volume 1214 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[5] H.R. Andersen, J. Staunstrup, and N. Maretti. Partial Model Checking with ROBDDs. In *Proceedings of the 3rd International Workshop on Tools and Algorithms for Construction and Analysis of Systems TACAS*, volume 1217 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[6] A. Arnold. MEC: A System for Constructing and Analysing Transition Systems. In *Proceedings of the 1st Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer Verlag, 1989.

[7] S. Basu and C.R. Ramakrishnan. Compositional Analysis for Verification of Parameterized Systems. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*, volume 2619 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[8] B. Berard and F. Laroussinie. Verification compositionnelle des p-automates. Technical Report Lot 4.1, Réseau National des Technologies Logicielles, projet AVERROES, 2003.

[9] N. Bodentien, J. Vestergaard, J. Friis, K. Kristoffersen, and K. Larsen. Verification of State/Event Systems by Quotienting. Technical Report RS-99-41, Basic Research in Computer Science, 1999.

[10] A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The Fc2Tools set: a Toolset for the Verification of Concurrent Systems. In *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[11] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *Proceedings of the 12th International Conference on Computer Aided Verification CAV*, volume 1855 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[12] 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 5.8). INRIA/VASY, 155 pages, 2013.

[13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 2000.

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

[15] P. Crouzen and F. Lang. Smart Reduction. In *Proceedings of Fundamental Approaches to Software Engineering FASE'2011*, volume 6603 of *Lecture Notes in Computer Science*. Springer Verlag, 2011.

[16] E. A. Emerson and C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of the 1st LICS*, 1986.

[17] J.-C. Fernandez and L. Mounier. "On the Fly" Verification of Behavioural Equivalences and Preorders. In *Proceedings of the 3rd Workshop on Computer-Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, Berlin, 1991. Springer Verlag.

[18] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, (18):194–211, 1979.

[19] H. Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, Berlin, 1998. Springer Verlag.

[20] H. Garavel and H. Hermanns. On Combining Functional Verification and Performance Evaluation using CADP. In *Proceedings of the 11th International Symposium of Formal Methods Europe FME'2002*, volume 2391 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[21] H. Garavel and F. Lang. SVL: a Scripting Language for Compositional Verification. In *Proceedings of the 21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2001*. IFIP, Kluwer Academic Publishers, 2001.

[22] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2011*, volume 6605 of *Lecture Notes in Computer Science*. Springer Verlag, 2011.

[23] H. Garavel, F. Lang, R. Mateescu, and W. 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, 2013.

[24] H. Garavel and M. Sighireanu. A Graphical Parallel Composition Operator for Process Algebras. In *Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'99*. IFIP, Kluwer Academic Publishers, 1999.

[25] H. Garavel and D. Thivolle. Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In *Model Checking Software, Proceedings of the 16th International SPIN Workshop on Model Checking of Software SPIN'2009*, Lecture Notes in Computer Science. Springer Verlag, 2009.

[26] S. Graf and B. Steffen. Compositional Minimization of Finite State Systems. In *Proceedings of the 2nd Workshop on Computer-Aided Verification*, volume 531 of *Lecture Notes in Computer Science*. Springer Verlag, 1990.

[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, Genève, 1989.

[28] ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization — Information Technology, Genève, 2001.

[29] D. Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[30] J.-P. Krimm and L. Mounier. Compositional State Space Generation from LOTOS Programs. In *Proceedings of TACAS'97 Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lecture Notes in Computer Science*, Berlin, 1997. Springer Verlag.

[31] F. Lang. EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods. In *Proceedings of the 5th International Conference on Integrated Formal Methods IFM'2005*, volume 3771 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[32] F. Lang. Refined Interfaces for Compositional Verification. In *Proceedings of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2006*, volume 4229 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.

[33] F. Lang and R. Mateescu. Partial Order Reductions using Compositional Confluence Detection. volume 5850 of *Lecture Notes in Computer Science*. Springer Verlag, 2009.

[34] F. Lang and R. Mateescu. Partial Model Checking using Networks of Labelled Transition Systems and Boolean Equation Systems. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*, volume 7214 of *Lecture Notes in Computer Science*. Springer Verlag, 2012.

[35] F. Lang, G. Salaün, R. Hérilier, J. Kramer, and J. Magee. Translating FSP into LOTOS and Networks of Automata. *Formal Aspects of Computing*, 22(6):681–711, 2010.

[36] F. Laroussinie and K. Larsen. Compositional Model Checking of Real Time Systems. In *Proceedings of the 6th International Conference on Concurrency Theory CONCUR*, volume 962 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.

[37] F. Laroussinie and K. Larsen. CMC: A Tool for Compositional Model Checking of Real-Time Systems. In *Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification FORTE/PSTV*, volume 135 of *IFIP Conference Proceedings*. Kluwer, 1998.

[38] K. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model Checking of Real-Time Systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*. IEEE Computer Society, 1995.

[39] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, 2006 edition, 2006.

[40] F. Martinelli. Symbolic Partial Model Checking for Security Analysis. In *Proceedings of the 2nd International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security MMM-ACNS*, volume 2776 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[41] R. Mateescu. Efficient Diagnostic Generation for Boolean Equation Systems. In *Proceedings of 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[42] R. Mateescu. On-the-fly State Space Reductions for Weak Equivalences. In *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems FMICS'05*. ERCIM, ACM Computer Society Press, 2005.

[43] R. Mateescu. CAESAR_SOLVE: A Generic Library for On-the-Fly Resolution of Alternation-Free Boolean Equation Systems. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 8(1):37–56, 2006.

[44] R. Mateescu and G. Salaün. Translating Pi-Calculus into LOTOS NT. In *Proceedings of the 8th International Conference on Integrated Formal Methods IFM'2010*, volume 6396 of *Lecture Notes in Computer Science*. Springer Verlag, 2010.

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

[46] R. Mateescu and D. Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In *Proceedings of the 15th International Symposium on Formal Methods FM'08*, number 5014 in Lecture Notes in Computer Science. Springer Verlag, 2008.

[47] G. Pace, F. Lang, and R. Mateescu. Calculating τ-Confluence Compositionally. In *Proceedings of the 15th International Conference on Computer Aided Verification CAV'2003*, volume 2725 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[48] J.-P. Queille and J. Sifakis. Fairness and Related Properties in Transition Systems — A Temporal Logic to Deal with Fairness. *Acta Informatica*, 19:195–220, 1983.

[49] R. Streett. Propositional Dynamic Logic of Looping and Converse. *Information and Control*, (54):121–141, 1982.

[50] P. Wolper. A Translation from Full Branching Time Temporal Logic to One Letter Propositional Dynamic Logic with Looping, 1982. Unpublished manuscript.