Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems

Radu Mateescu

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems

Radu Mateescu

N° 4430

Avril 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4430--FR+ENG

INRIA
RHÔNE-ALPES

Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems

Radu Mateescu*

Thème 1 — Réseaux et systèmes
Projet VASY

Rapport de recherche n° 4430 — Avril 2002 — 36 pages

Abstract: Model-checking is a popular technique for verifying finite-state concurrent systems, the behaviour of which can be modeled using Labeled Transition Systems (LTSs). In this report, we study the model-checking problem for the modal μ-calculus on acyclic LTSs. This has various applications of practical interest such as trace analysis, log information auditing, run-time monitoring, etc. We show that on acyclic LTSs, the full μ-calculus has the same expressive power as its alternation-free fragment. We also present two new algorithms for local model-checking of μ-calculus formulas on acyclic LTSs. Our algorithms are based upon a translation to boolean equation systems and exhibit a better performance than existing model-checking algorithms applied to acyclic LTSs. The first algorithm handles μ-calculus formulas φ with alternation depth $ad(\varphi) \geq 2$ and has time complexity $O(|\varphi|^2 \cdot (|S| + |T|))$ and space complexity $O(|\varphi|^2 \cdot |S|)$, where $|S|$ and $|T|$ are the number of states and transitions of the acyclic LTS and $|\varphi|$ is the number of operators in φ. The second algorithm handles formulas φ with alternation depth $ad(\varphi) = 1$ and has time complexity $O(|\varphi| \cdot (|S| + |T|))$ and space complexity $O(|\varphi| \cdot |S|)$.

Key-words: labeled transition system, model-checking, mu-calculus, specification, temporal logic, verification

A short version of this research report is also available as "Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems", in Joost-Pieter Katoen and Perdita Stevens, editors, Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2002 (Grenoble, France), April 2002.

* Radu.Mateescu@inria.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Vérification à la volée du mu-calcul modal sur des systèmes de transitions étiquetées sans circuit

Résumé : La vérification énumérative (*model-checking*) est une technique largement utilisée pour valider les systèmes concurrents à nombre fini d'états, le comportement desquels peut être modélisé au moyen de systèmes de transitions étiquetées (STEs). Dans ce rapport, nous étudions le problème de la vérification des formules du μ-calcul modal sur des STEs sans circuit. Ce problème a diverses applications d'intérêt pratique, comme l'analyse de traces, l'audit d'informations de sécurité, le suivi en temps-réel, etc. Nous montrons que, sur des STEs sans circuit, le μ-calcul complet possède la même expressivité que son fragment d'alternance 1. Nous présentons également deux nouveaux algorithmes pour la vérification à la volée des formules du μ-calcul sur des STEs sans circuit. Nos algorithmes sont basés sur une traduction vers des systèmes d'équations booléennes et présentent de meilleures performances que les algorithmes de vérification existants appliqués à des STEs sans circuit. Le premier algorithme permet de traiter des formules φ du μ-calcul d'alternance $ad(\varphi) \geq 2$ avec une complexité $O(|\varphi|^2 \cdot (|S| + |T|))$ en temps d'éxecution et $O(|\varphi|^2 \cdot |S|)$ en espace mémoire, où $|S|$ et $|T|$ dénotent le nombre d'états et de transitions du STE sans circuit et $|\varphi|$ dénote le nombre d'opérateurs de φ. Le deuxième algorithme permet de traiter des formules φ d'alternance $ad(\varphi) = 1$ avec une complexité $O(|\varphi| \cdot (|S| + |T|))$ en temps et $O(|\varphi| \cdot |S|)$ en mémoire.

Mots-clés : logique temporelle, mu-calcul, spécification, système de transitions étiquetées, vérification

1 Introduction

Model-checking [4] is a popular approach for efficiently verifying the correctness of concurrent finite-state systems. This approach proceeds by translating the system into a finite *model*, represented as a state transition graph, also known as a Labeled Transition System (LTS), on which the desired correctness properties, expressed in temporal logic, are verified using specific model-checking algorithms. Depending on the way in which they handle the construction of the LTS, model-checking algorithms can be divided in two classes: *global* algorithms, which require to construct the LTS completely before starting the verification, and *local* algorithms, which allow to construct the LTS in a demand-driven way during verification. The latter algorithms are able to detect errors even if the LTS cannot be entirely constructed (e.g., because of insufficient computing resources).

Although model-checking has been mainly used for verifying concurrent systems (communication protocols, distributed applications, hardware architectures), the underlying techniques and algorithms are useful in other contexts as well. In particular, several problems related to the analysis of sequential systems can be formulated as model-checking problems on single trace LTSs: *intrusion detection* by auditing of log file information, as in the USTAT rule-based expert system [14], in which security properties of log files are encoded as state transition diagrams; *trace analysis* for program debugging, as in the OPIUM trace analyzer for Prolog [9], which uses a dedicated language to describe trace queries; and *run-time monitoring* by observation of event traces in real-time, as in the MOTEL monitoring system [7], which uses LTL [20] to express test requirements and to synthesize observers. When analyzing sequential systems, it appears that existing model-checking algorithms can be optimized significantly, especially by reducing their memory consumption, which is crucial for scaling up to larger systems. Therefore, optimizing the performance of model-checking algorithms on sequential systems becomes an interesting issue, with applications in all the aforementioned domains.

In this report, we consider the problem of model-checking temporal properties on acyclic LTSs (ALTSs), which contain traces as a particular case. As regards the property specification language, we adopt the modal μ-calculus [16], a powerful fixed point-based temporal logic that subsumes virtually all temporal logics defined in the literature (in-depth presentations of modal μ-calculus can be found in [19, 3, 23]). Various global [10, 5, 6] and local [17, 1, 25, 18, 22] algorithms have been proposed for model-checking μ-calculus formulas on arbitrary LTSs. However, as far as we know, no attempt has been made to optimize these algorithms in the case of ALTSs.

Our results concern both the expressiveness of μ-calculus interpreted on ALTSs and the underlying model-checking algorithms. We first show that the full modal μ-calculus interpreted on ALTSs has the same expressive power as its alternation-free fragment [10]. Our result is based upon a succinct translation (quadratic blow-up

of the formula size) from full μ-calculus to guarded μ-calculus [16], followed by a reduction to alternation-free μ-calculus. Together with the linear-time complexity results for alternation-free μ-calculus [6], this yields a model-checking procedure for the full μ-calculus on ALTSs which is quadratic in the size of the formula (number of operators) and linear in the size of the ALTS (number of states and transitions).

We also propose two local model-checking algorithms for μ-calculus on ALTSs based upon a translation to boolean equation systems (BESs) [19]. The first algorithm handles full μ-calculus formulas and has a time complexity $O(|\varphi|^2 \cdot (|S|+|T|))$ and a space complexity $O(|\varphi|^2 \cdot |S|)$, where $|\varphi|$ is the size of the formula and $|S|$, $|T|$ are the number of states and transitions in the ALTS. The second algorithm handles only alternation-free formulas and has a time complexity $O(|\varphi| \cdot (|S| + |T|))$ and a space complexity $O(|\varphi| \cdot |S|)$. Both algorithms exploit the particular structure of the underlying BES to avoid storing ALTS transitions and thus to achieve a lower space complexity than existing local model-checking algorithms [1, 25, 18, 8] executed on ALTSs.

The report is organized as follows. Section 2 defines the modal μ-calculus and its guarded fragment, and presents the simplification results for μ-calculus formulas on ALTSs. Section 3 describes the local model-checking algorithms for full μ-calculus and alternation-free μ-calculus on ALTSs. Section 4 gives some concluding remarks and directions for future work. Finally, Annex A contains proofs of the technical results.

2 Modal Mu-Calculus and Acyclic LTSs

In this section we study the expressiveness of μ-calculus formulas interpreted on acyclic LTSs, our goal being to simplify formulas as much as possible in order to increase the efficiency of model-checking algorithms. We first define the syntax and semantics of the modal μ-calculus, then we propose a succinct translation of the μ-calculus to its guarded fragment, and finally we present the simplification results obtained.

2.1 Syntax and Semantics

As semantic models, we consider labeled transition systems (LTSs), which are suitable for action-based description languages such as process algebras. An LTS is a tuple $M = (S, A, T, s_0)$, where: S is a (finite) set of *states*; A is a (finite) set of *actions*; $T \subseteq S \times A \times S$ is the *transition relation*; and $s_0 \in S$ is the *initial state*. A transition $(s_1, a, s_2) \in T$ (also noted $s_1 \xrightarrow{a} s_2$) means that the system can move from state s_1 to state s_2 by performing action a. The notation $s_1 \xrightarrow{*} s_2$ means that there exists a sequence of (0 or more) transitions leading from s_1 to s_2. All states in S are reachable from s_0 via sequences of transitions in T ($s_0 \xrightarrow{*} s$ for all $s \in S$). If T does not contain cycles, M is called an *acyclic* LTS (ALTS). In the sequel, we assume the existence of an LTS $M = (S, A, T, s_0)$ on which all temporal logic formulas will be interpreted.

The μ-calculus variant we consider here (see Table 1) consists of *action formulas* (noted α) and *state formulas* (noted φ). Action formulas are built from actions $a \in A$ and standard boolean operators (this is a slight extension w.r.t. the original definition of μ-calculus [16], in which action formulas were restricted to simple actions). State formulas φ are built from propositional variables X belonging to a set $\mathcal{X}$, standard boolean operators, possibility and necessity modal operators $\langle\alpha\rangle\,\varphi$ and $[\alpha]\,\varphi$, minimal and maximal fixed point operators $\mu X.\varphi$ and $\nu X.\varphi$. The μ and ν operators act as binders for propositional variables in the same way as quantifiers in first-order logic. In the sequel, we will use the symbol σ to denote μ or ν. For a fixed point formula $\varphi = \sigma X.\varphi'$, the subformula φ' is called the *body* of φ.

Table 1: Syntax and semantics of the modal μ-calculus

Syntax ($X \in \mathcal{X}$ are propositional variables):

$$\begin{array}{rcl} \alpha & ::= & \mathsf{F} \mid \mathsf{T} \mid a \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \wedge \alpha_2 \mid \neg\alpha \\ \varphi & ::= & \mathsf{F} \mid \mathsf{T} \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\,\varphi \mid [\alpha]\,\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi \end{array}$$

Semantics ($\rho : \mathcal{X} \rightarrow 2^S$ are propositional contexts):

$$\begin{array}{rcl@{\qquad}rcl} [\![\mathsf{F}]\!] & = & \emptyset & [\![\alpha_1 \vee \alpha_2]\!] & = & [\![\alpha_1]\!] \cup [\![\varphi_2]\!] \\ [\![\mathsf{T}]\!] & = & A & [\![\alpha_1 \wedge \alpha_2]\!] & = & [\![\alpha_1]\!] \cap [\![\alpha_2]\!] \\ [\![a]\!] & = & \{a\} & [\![\neg\alpha]\!] & = & A \setminus [\![\alpha]\!] \end{array}$$

$$\begin{array}{rcl} [\![\mathsf{F}]\!]\,\rho & = & \emptyset \\ [\![\mathsf{T}]\!]\,\rho & = & S \\ [\![\varphi_1 \vee \varphi_2]\!]\,\rho & = & [\![\varphi_1]\!]\,\rho \cup [\![\varphi_2]\!]\,\rho \\ [\![\varphi_1 \wedge \varphi_2]\!]\,\rho & = & [\![\varphi_1]\!]\,\rho \cap [\![\varphi_2]\!]\,\rho \\ [\![\langle\alpha\rangle\,\varphi]\!]\,\rho & = & \{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi]\!]\,\rho\} \\ [\![[\alpha]\,\varphi]\!]\,\rho & = & \{s \in S \mid \forall s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \Rightarrow s' \in [\![\varphi]\!]\,\rho\} \\ [\![X]\!]\,\rho & = & \rho(X) \\ [\![\mu X.\varphi]\!]\,\rho & = & \bigcap\{U \subseteq S \mid [\![\varphi]\!]\,\rho[U/X] \subseteq U\} \\ [\![\nu X.\varphi]\!]\,\rho & = & \bigcup\{U \subseteq S \mid U \subseteq [\![\varphi]\!]\,\rho[U/X]\} \end{array}$$

The interpretation $[\![\alpha]\!] \subseteq A$ of an action formula α yields the set of Lts actions satisfying α. The interpretation $[\![\varphi]\!]\,\rho \subseteq S$ of a state formula φ, where $\rho : \mathcal{X} \rightarrow 2^S$ is a propositional context assigning state sets to variables, yields the set of Lts states satisfying φ in the context ρ ($\rho[U/X]$ denotes a context identical to ρ except for variable X, which is assigned state set U). The $\langle\alpha\rangle\,\varphi$ and $[\alpha]\,\varphi$ modalities characterize the states for which some (resp. all) outgoing transitions whose actions satisfy α lead to states satisfying φ. The $\mu X.\varphi$ and $\nu X.\varphi$ formulas characterize the states satisfying the minimal (resp. maximal) solution over 2^S of the equation $X = \varphi$.

Definition 1 (free and bound variables) *Let φ be a state formula. The sets $fv(\varphi)$ and $bv(\varphi)$ defined inductively below denote the* free variables *and the* bound variables *of φ, respectively.*

φ	$fv(\varphi)$	$bv(\varphi)$
F, T	$\emptyset$	$\emptyset$
$\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$	$fv(\varphi_1) \cup fv(\varphi_2)$	$bv(\varphi_1) \cup bv(\varphi_2)$
$\langle \alpha \rangle \varphi$, $[\alpha] \varphi$	$fv(\varphi)$	$bv(\varphi)$
X	$\{X\}$	$\emptyset$
$\mu X.\varphi$, $\nu X.\varphi$	$fv(\varphi) \setminus \{X\}$	$bv(\varphi) \cup \{X\}$

A formula φ such that $fv(\varphi) = \emptyset$ is said *closed*. A formula φ such that $fv(\varphi) \cap bv(\varphi) = \emptyset$ is said to be in *normal form*, i.e., it does not contain variables which are both free and bound. In the sequel, we consider only state formulas in normal form.

Let φ, φ' be state formulas and X be a propositional variable. The expression $\varphi[\varphi'/X]$ denotes the syntactic substitution of every free occurrence of X in φ by φ'. The following lemma (proven in Annex A.1) states that if φ' does not contain free variables that may become bound in $\varphi[\varphi'/X]$, the interpretation of $\varphi[\varphi'/X]$ can be calculated using the interpretations of φ and φ'.

Lemma 1 *Let φ, φ' be state formulas such that $bv(\varphi) \cap fv(\varphi') = \emptyset$. Then:*

$$[\![\varphi[\varphi'/X]]\!]\,\rho = [\![\varphi]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

for any propositional variable X and any propositional context ρ.

Let $\sigma X.\varphi$ be a fixed point state formula. The expression $\varphi[\sigma X.\varphi/X]$ is called the *unfolding* of $\sigma X.\varphi$. The following proposition states that unfolding of fixed point formulas preserves their interpretation.

Proposition 1 *Let $\sigma X.\varphi$ be a state formula. Then:*

$$[\![\varphi[\sigma X.\varphi/X]]\!]\,\rho = [\![\sigma X.\varphi]\!]\,\rho$$

for any propositional context ρ.

Proof By Definition 1, we have $bv(\varphi) \cap fv(\sigma X.\varphi) = bv(\varphi) \cap (fv(\varphi) \setminus \{X\}) \subseteq bv(\varphi) \cap fv(\varphi)$, which is empty because we assumed that φ is in normal form. By applying Lemma 1, we obtain $[\![\varphi[\sigma X.\varphi/X]]\!]\,\rho = [\![\varphi]\!]\,[[\![\sigma X.\varphi]\!]\,\rho/X]$, which by interpretation of fixed point formulas (Table 1) and Tarski's theorem [24] is equal to $[\![\sigma X.\varphi]\!]\,\rho$. □

In the sequel, for closed state formulas φ we will simply write $[\![\varphi]\!]$, since the interpretation of these formulas does not depend upon any propositional context. An LTS $M = (S, A, T, s_0)$ satisfies a closed state formula φ (noted $M \models \varphi$) iff $[\![\varphi]\!] = S$.

2.2 Reduction of Full Mu-Calculus to Guarded Mu-Calculus

In order to simplify the interpretation of modal μ-calculus on ALTSs, we must first translate all fixed point formulas $\sigma X.\varphi$ to *guarded form* [16, 26]: all free occurrences of X in φ must be guarded, i.e., in the scope of a $\langle . \rangle$ or [.] modality. The translations proposed in [16, 26] require repeated transformations of subformulas to conjunctive normal form, leading to an exponential blow-up of the formula. We present in this section a more succinct translation to guarded form, which (by using factorization of common subformulas) yields only a quadratic blow-up. This translation is purely syntactic, i.e., it does not depend upon the structure of the LTS $M = (S, A, T, s_0)$ on which formulas are interpreted.

The definition below introduces the notions of guardedness and weakly guardedness of a state formula φ w.r.t. a set of propositional variables $\mathcal{X}$. Intuitively, a formula φ weakly guarded w.r.t. $\{X\}$ allows unguarded occurrences of X only at top-level, i.e., outside any fixed point subformula of φ.

Definition 2 (guarded and weakly guarded formulas) *Let φ be a state formula and $\mathcal{X}$ be a set of variables. φ is called* guarded *(resp.* weakly guarded*) w.r.t. $\mathcal{X}$ iff it satisfies the predicate $g(\varphi, \mathcal{X})$ (resp. $wg(\varphi, \mathcal{X})$) defined inductively below. φ is called* guarded *iff it satisfies $g(\varphi, \emptyset)$.*

φ	$g(\varphi, \mathcal{X})$	$wg(\varphi, \mathcal{X})$
F, T	T	T
$\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$	$g(\varphi_1, \mathcal{X}) \wedge g(\varphi_2, \mathcal{X})$	$wg(\varphi_1, \mathcal{X}) \wedge wg(\varphi_2, \mathcal{X})$
$\langle \alpha \rangle \varphi$, $[\alpha] \varphi$	$g(\varphi, \emptyset)$	$g(\varphi, \emptyset)$
X	$X \notin \mathcal{X}$	T
$\mu X.\varphi$, $\nu X.\varphi$	$g(\varphi, \mathcal{X} \cup \{X\})$	$g(\varphi, \mathcal{X} \cup \{X\})$

Examples of unguarded formulas can be obtained by translating in μ-calculus certain regular modalities of PDL [12]. For instance, the μ-calculus formula below, produced by translating the PDL formula $\langle (a|b^*)^*.c \rangle \mathsf{T}$, is unguarded w.r.t. $\{X\}$:

$$\varphi_1 = \mu X.(\langle c \rangle \mathsf{T} \vee \langle a \rangle X \vee \mu Y.(X \vee \langle b \rangle Y))$$

To make this formula guarded, we must eliminate the unguarded occurrence of X contained in the μY-subformula. The first step is to bring this occurrence at the top-level of the body of φ_1. This is done by unfolding the μY-subformula, resulting in the formula below, whose body is weakly guarded w.r.t. $\{X\}$:

$$\varphi_2 = \mu X.(\langle c \rangle \mathsf{T} \vee \langle a \rangle X \vee (X \vee \langle b \rangle \mu Y.(X \vee \langle b \rangle Y)))$$

The second step is to eliminate the top-level unguarded occurrence of X. This is done using the syntactic transformation below, which replaces, in a formula $\mu X.\varphi$ (resp. $\nu X.\varphi$), all unguarded occurrences of X at the top-level of φ by F (resp. T).

Definition 3 (flattening) *Let φ be a state formula, X be a variable, and $\sigma \in \{\mu, \nu\}$. The formula $f(\varphi, X, \sigma)$ defined inductively below is called the* flattening *of φ w.r.t. X.*

φ	$f(\varphi, X, \sigma)$	φ	$f(\varphi, X, \sigma)$
F	F	T	T
$\varphi_1 \vee \varphi_2$	$f(\varphi_1, X, \sigma) \vee f(\varphi_2, X, \sigma)$	$\varphi_1 \wedge \varphi_2$	$f(\varphi_1, X, \sigma) \wedge f(\varphi_2, X, \sigma)$
$\langle \alpha \rangle \varphi$	$\langle \alpha \rangle \varphi$	$[\alpha] \varphi$	$[\alpha] \varphi$
Y	Y *(if* $Y \neq X$*)*	X	*if* $\sigma = \mu$ *then* F *else* T
$\mu Y.\varphi$	$\mu Y.\varphi$	$\nu Y.\varphi$	$\nu Y.\varphi$

By flattening the formula φ_2, we obtain the guarded formula below:

$$\varphi_3 = \mu X.(\langle c \rangle \mathsf{T} \vee \langle a \rangle X \vee (\mathsf{F} \vee \langle b \rangle \mu Y.(X \vee \langle b \rangle Y)))$$

This formula can be translated back to the PDL modality $\langle (a|(b.b^*))^*.c \rangle \mathsf{T}$, which is equivalent to the initial formula $\langle (a|b^*)^*.c \rangle \mathsf{T}$.

The following results (Lemma 2 is proven in Annex A.2) establish that flattening of fixed point formulas preserves their interpretation.

Lemma 2 *Let φ be a state formula, X be a variable, and $U \subseteq S$. Then:*

$$[\![f(\varphi, X, \nu)]\!] \rho[U/X] \cap U \subseteq [\![\varphi]\!] \rho[U/X] \subseteq [\![f(\varphi, X, \mu)]\!] \rho[U/X] \cup U$$

for any propositional context ρ.

Proposition 2 *Let $\sigma X.\varphi$ be a state formula. Then:*

$$[\![\sigma X.\varphi]\!] \rho = [\![\sigma X.f(\varphi, X, \sigma)]\!] \rho$$

for any propositional context ρ.

Proof We prove only the case $\sigma = \mu$, the other case being similar. Since some occurrences of X in φ have been replaced by F in $f(\varphi, X, \mu)$, it follows by monotonicity that $[\![\mu X.f(\varphi, X, \mu)]\!] \rho \subseteq [\![\mu X.\varphi]\!] \rho$. To show the converse, let $U \subseteq S$ such that $[\![f(\varphi, X, \mu)]\!] \rho[U/X] \subseteq U$. Using Lemma 2, this implies $[\![\varphi]\!] \rho[U/X] \subseteq [\![f(\varphi, X, \mu)]\!] \rho[U/X] \cup U = U$. This further implies $\{U \subseteq S \mid [\![f(\varphi, X, \mu)]\!] \rho[U/X] \subseteq U\} \subseteq \{U \subseteq S \mid [\![\varphi]\!] \rho[U/X] \subseteq U\}$, which by interpretation of formulas (Table 1) yields $[\![\mu X.\varphi]\!] \rho = \bigcap \{U \subseteq S \mid [\![\varphi]\!] \rho[U/X] \subseteq U\} \subseteq \bigcap \{U \subseteq S \mid [\![f(\varphi, X, \mu)]\!] \rho[U/X] \subseteq U\} = [\![\mu X.f(\varphi, X, \mu)]\!] \rho$. □

The syntactic transformation defined below, consisting of two mutually recursive functions t and t', reduces state formulas to guarded form. These functions implement the transformation outlined in the previous example when φ_1 was translated to φ_3: for every fixed point formula $\sigma X.\varphi$, the unguarded occurrences of X are brought to the top-level of φ using $t'(\varphi)$ and then they are eliminated by flattening using $f(t'(\varphi), X, \sigma)$. By applying $t'(\varphi)$, all fixed point subformulas of φ that are not in the scope of a modality are unfolded.

Definition 4 (translation to guarded μ-calculus) *Let φ be a state formula. The functions t and t' defined inductively below translate φ to guarded form.*

φ	$t(\varphi)$	$t'(\varphi)$
F	F	F
T	T	T
$\varphi_1 \vee \varphi_2$	$t(\varphi_1) \vee t(\varphi_2)$	$t'(\varphi_1) \vee t'(\varphi_2)$
$\varphi_1 \wedge \varphi_2$	$t(\varphi_1) \wedge t(\varphi_2)$	$t'(\varphi_1) \wedge t'(\varphi_2)$
$\langle\alpha\rangle\,\varphi$	$\langle\alpha\rangle\,t(\varphi)$	$\langle\alpha\rangle\,t(\varphi)$
$[\alpha]\,\varphi$	$[\alpha]\,t(\varphi)$	$[\alpha]\,t(\varphi)$
X	X	X
$\sigma X.\varphi$	$\sigma X.f(t'(\varphi), X, \sigma)$	$f(t'(\varphi), X, \sigma)[\sigma X.f(t'(\varphi), X, \sigma)/X]$

The interested reader can easily check that $t(\varphi_1) = \varphi_3$. The syntactic and semantic properties below (Proposition 3 is proven in Annex A.3) establish that the transformation t (resp. t') indeed reduces formulas to guarded (resp. weakly guarded) form while preserving their interpretation.

Proposition 3 *Let φ be a state formula. Then:*

$$g(t(\varphi), \emptyset) \wedge wg(t'(\varphi), fv(\varphi)).$$

Proposition 4 *Let φ be a state formula. Then:*

$$[\![t(\varphi)]\!]\,\rho = [\![t'(\varphi)]\!]\,\rho = [\![\varphi]\!]\,\rho$$

for any propositional context ρ.

Proof By structural induction on φ. We prove only the case $\varphi = \sigma X.\varphi_1$, the other cases being straightforward. By Definition 4 and Proposition 1, we have $[\![t'(\sigma X.\varphi_1)]\!]\,\rho = [\![f(t'(\varphi_1), X, \sigma)[\sigma X.f(t'(\varphi_1), X, \sigma)/X]]\!]\,\rho = [\![\sigma X.f(t'(\varphi_1), X, \sigma)]\!]\,\rho = [\![t(\sigma X.\varphi_1)]\!]\,\rho$. From Proposition 2 and the induction hypothesis, it follows that $[\![\sigma X.f(t'(\varphi_1), X, \sigma)]\!]\,\rho = [\![\sigma X.t'(\varphi_1)]\!]\,\rho = [\![\sigma X.\varphi_1]\!]\,\rho$. □

We conclude this section by an estimation of the size $|t(\varphi)|$ (number of operators and variables in $t(\varphi)$). The application of $t(\varphi)$ consists of flattening and unfolding steps performed in a bottom-up manner on the fixed point subformulas of φ. Flattening does not change the size of formulas, since it simply replaces some occurrences of variables by constant boolean operators. A direct implementation of fixed point unfolding would yield, for each $\sigma X.\varphi'$ subformula of φ, a size $|f(t'(\varphi'), X, \sigma)[\sigma X.f(t'(\varphi'), X, \sigma)/X]| \leq |f(t'(\varphi'), X, \sigma)|^2 = |t'(\varphi')|^2$, leading to an overall size $|t(\varphi)| \leq |\varphi|^{2\cdot|\varphi|}$.

A refined implementation of unfolding by using factorization of common subformulas would yield, for each $\sigma X.\varphi'$ subformula of φ, a size

$|f(t'(\varphi'), X, \sigma)[\sigma X.f(t'(\varphi'), X, \sigma)/X]| = |f(t'(\varphi'), X, \sigma)| + |\sigma X.f(t'(\varphi'), X, \sigma)|$. Note that the second summand above appears only once for each subformula $\sigma X.\varphi'$: since $f(t'(\varphi'), X, \sigma)$ is guarded w.r.t. $\{X\}$ (see the proof of Proposition 3 in Annex A.3), all fixed point subformulas $\sigma X.f(t'(\varphi'), X, \sigma)$ substituted for X in $f(t'(\varphi'), X, \sigma)$ will occur in the scope of modalities and will remain unchanged during later flattening steps. In this way, each fixed point subformula of φ will be duplicated only once (when it is unfolded) and reused in later steps, leading to an overall size $|t(\varphi)| \leq |\varphi|^2$.

The translations to guarded form proposed in [16, 26] perform the flattening of a formula $\sigma X.\varphi$ by converting φ to conjunctive normal form (considering modal and fixed point subformulas as literals) before replacing the top-level unguarded occurrences of X in φ with F or T. This yields a worst-case exponential size of the final formula, even if a factorization scheme is applied.

2.3 Simplification of Guarded Mu-Calculus on ALTSs

We show in this section that guarded μ-calculus formulas can be considerably simplified when interpreted on ALTSs $M = (S, A, T, s_0)$, thus increasing the efficiency of model-checking. For technical reasons, we need to use the negation operator ($\neg$) on state formulas. The negation of a state formula φ, noted $\neg\varphi$, is interpreted in a context ρ as $[\![\neg\varphi]\!]\,\rho = S \setminus [\![\varphi]\!]\,\rho$. For the sake of simplicity, we did not use the negation operator in the definition of μ-calculus given in Section 2.1. In the presence of negation, to ensure a well-defined interpretation of fixed point formulas, we must impose the *syntactic monotonicity* condition [16]: in any fixed point formula $\mu X.\varphi$ or $\nu X.\varphi$, all free occurrences of X in φ must be in the scope of an even number of negations. Using *duality* (see below), any syntactically monotonic formula can be converted in *positive form*, i.e., an equivalent formula without negation operators.

The *dual* of a state formula φ w.r.t. a set of propositional variables $\{X_1, ..., X_n\}$ is the negation of φ in which all free occurrences of $X_1, ..., X_n$ are replaced by $\neg X_1, ..., \neg X_n$, respectively. However, to facilitate the reasoning by structural induction, we prefer an inductive definition of dual formulas.

Definition 5 (dual formulas) *Let φ be a state formula and let $\mathcal{X} = \{X_1, ..., X_n\}$ be a set of propositional variables. The formula $d(\varphi, \mathcal{X})$ defined inductively below is called the* dual *of φ w.r.t. $\mathcal{X}$.*

φ	$d(\varphi, \mathcal{X})$	φ	$d(\varphi, \mathcal{X})$
F	T	T	F
$\varphi_1 \vee \varphi_2$	$d(\varphi_1, \mathcal{X}) \wedge d(\varphi_2, \mathcal{X})$	$\varphi_1 \wedge \varphi_2$	$d(\varphi_1, \mathcal{X}) \vee d(\varphi_2, \mathcal{X})$
$\langle\alpha\rangle\,\varphi$	$[\alpha]\,d(\varphi, \mathcal{X})$	$[\alpha]\,\varphi$	$\langle\alpha\rangle\,d(\varphi, \mathcal{X})$
X	X *if* $X \in \mathcal{X}$	X	$\neg X$ *if* $X \notin \mathcal{X}$
$\mu X.\varphi$	$\nu X.d(\varphi, \mathcal{X} \cup \{X\})$	$\nu X.\varphi$	$\mu X.d(\varphi, \mathcal{X} \cup \{X\})$

Proposition 5 *Let $M = (S, A, T, s_0)$ be an* LTS*, φ be a state formula, and $X_1, ..., X_n$ be propositional variables. Then:*

$$S \setminus [\![\varphi]\!]\,\rho = [\![d(\varphi, \{X_1, ..., X_n\})]\!]\,\rho[(S \setminus \rho(X_1))/X_1, ..., (S \setminus \rho(X_n))/X_n]$$

for any propositional context ρ.

Proposition 5 (proven in Annex A.4) allows in particular to derive the duality between minimal and maximal fixed point formulas: $S \setminus [\![\mu X.\varphi]\!]\,\rho = [\![d(\mu X.\varphi, \emptyset)]\!]\,\rho = [\![\nu X.d(\varphi, \{X\})]\!]\,\rho$. Lemma 3 (proven in Annex A.5) states a more involved property about dual formulas interpreted on ALTSs: every state s satisfying both a formula and its dual is the origin of a particular transition sequence $s \xrightarrow{*} s'$.

Lemma 3 *Let $M = (S, A, T, s_0)$ be an* ALTS*, φ be a state formula, $X_1, ..., X_n$ be propositional variables, $A_1, B_1, ..., A_n, B_n \subseteq S$ be state sets, and $\mathcal{Y} \subseteq \{X_1, ..., X_n\}$ be a set of variables such that $g(\varphi, \mathcal{Y})$. Then:*

$$\forall s \in [\![\varphi]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![d(\varphi, \{X_1, ..., X_n\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n]\ .$$
$$\exists i \in [1, n]\ .\ \exists s' \in A_i \cap B_i\ .\ s \xrightarrow{*} s' \wedge (X_i \in \mathcal{Y} \Rightarrow s' \neq s)$$

for any propositional context ρ.

Theorem 1 gives a characterization of ALTSs by means of guarded μ-calculus formulas: ALTSs are precisely those LTSs on which minimal and maximal guarded fixed point formulas have the same interpretation.

Theorem 1 (characterization of acyclic LTSs) *Let $M = (S, A, T, s_0)$ be an* LTS*. M is acyclic iff, for every state formula φ and every propositional variable X such that $g(\varphi, \{X\})$, the following equality holds:*

$$[\![\mu X.\varphi]\!]\,\rho = [\![\nu X.\varphi]\!]\,\rho$$

for any propositional context ρ.

Proof **If.** Let $M = (S, A, T, s_0)$ be an LTS, and consider the formula $\varphi = [\mathsf{T}]\,X$, which obviously satisfies $g(\varphi, \{X\})$. By hypothesis, we have that $[\![\mu X.\,[\mathsf{T}]\,X]\!] = [\![\nu X.\,[\mathsf{T}]\,X]\!] = S$. This means $M \models \mu X.\,[\mathsf{T}]\,X$, i.e., for every state $s \in S$, all outgoing transition sequences are finite. Hence, M is acyclic.

Only if. Let $M = (S, A, T, s_0)$ be an ALTS, φ be a state formula, and X be a propositional variable such that $g(\varphi, \{X\})$. We show that $[\![\nu X.\varphi]\!]\,\rho \setminus [\![\mu X.\varphi]\!]\,\rho = \emptyset$ for an arbitrary propositional context ρ. Using the duality between minimal and maximal fixed points, this equality can be rewritten as $[\![\nu X.\varphi]\!]\,\rho \cap [\![\nu X.d(\varphi, \{X\})]\!]\,\rho = \emptyset$. We note $A = [\![\nu X.\varphi]\!]\,\rho$ and $B = [\![\nu X.d(\varphi, \{X\})]\!]\,\rho$. By unfolding the fixed point formulas and applying Proposition 1, the above equality becomes $[\![\varphi]\!]\,\rho[A/X] \cap [\![d(\varphi, \{X\})]\!]\,\rho[B/X] = \emptyset$.

Suppose this equality does not hold, and let $s \in [\![\varphi]\!]\,\rho[A/X] \cap [\![d(\varphi,\{X\})]\!]\,\rho[B/X]$. By applying Lemma 3, and since $g(\varphi,\{X\})$ by hypothesis, this implies $s \xrightarrow{*} s'$, where $s' \in A \cap B = [\![\varphi]\!]\,\rho[A/X] \cap [\![d(\varphi,\{X\})]\!]\,\rho[B/X]$ and $s' \neq s$. Thus, from every state $s \in A \cap B$, there is a non empty transition sequence to another state $s' \in A \cap B$. Since the state set $A \cap B$ is finite, this means there is a cycle between states in $A \cap B$ (contradiction with M acyclic). □

The practical consequence of Theorem 1 is to allow the simplification of guarded state formulas interpreted on ALTSs by converting all occurrences of maximal fixed points into minimal fixed points (or vice-versa, which would yield an equivalent simplification). The resulting formulas φ are *alternation-free* [10] because they do not contain mutually recursive minimal and maximal fixed point operators. Consequently, the model-checking of these formulas on ALTSs $M = (S, A, T, s_0)$ has a time and space complexity $O(|\varphi| \cdot (|S| + |T|))$ [6].

Together with the translation to guarded form given in Section 2.2, the reduction provided by Theorem 1 yields a time and space complexity $O(|\varphi|^2 \cdot (|S| + |T|))$ for the model-checking of arbitrary μ-calculus formulas on ALTSs. However, by exploiting the absence of cycles in the ALTS and the guardedness of the formulas, we can improve the space complexity of model-checking to $O(|\varphi|^2 \cdot |S|)$, as shown in the next section.

3 Local Model-Checking on Acyclic LTSs

After applying the translation to guarded form and the simplification defined in Sections 2.2 and 2.3, we obtain simplified guarded state formulas $\mu X.\varphi$ that contain only minimal fixed point variables[1]. We develop our local model-checking algorithms for these formulas by adopting an approach that has been used for model-checking the alternation-free μ-calculus [1, 25, 18, 8, 21]. The approach consists of translating the verification problem to the local resolution of a boolean equation system, which is performed by on-the-fly exploration of the dependency graph between boolean variables.

3.1 Translation to Boolean Equation Systems

We encode the model-checking problem of a guarded simplified formula $\mu X.\varphi$ on an ALTS by using boolean equation systems (BESs) [19]. A (simple) BES (see Table 2) is a system of minimal fixed point equations $\{z_i = op_i Z_i\}_{1 \leq i \leq n}$ whose left-hand sides are boolean variables $z_i \in \mathcal{Z}$ and whose right-hand sides are pure disjunctive or conjunctive boolean formulas. Empty disjunctions and conjunctions are equivalent to F and T, respectively. The interpretation of a BES w.r.t. a boolean context $\delta : \mathcal{Z} \to \{\mathsf{F},\mathsf{T}\}$ is the minimal fixed point of a vectorial functional $\overline{\Psi}_\delta$ defined over $\{\mathsf{F},\mathsf{T}\}^n$.

[1] Any formula φ can be converted to the form $\mu X.\varphi$, where X is a "fresh" variable.

Table 2: Syntax and semantics of BESs

$\{z_i = op_i Z_i\}_{1 \leq i \leq n}$
where $op_i \in \{\vee, \wedge\}$, $Z_i \subseteq \mathcal{Z}$ for all $1 \leq i \leq n$

$[\![op_i\{z_1, ..., z_p\}]\!]\delta = \delta(z_1)\ op_i ... op_i\ \delta(z_p)$
$[\![\{z_i = op_i Z_i\}_{1 \leq i \leq n}]\!]\ \delta = \mu\overline{\Psi}_\delta$
where $\overline{\Psi}_\delta : \{\mathsf{F}, \mathsf{T}\}^n \rightarrow \{\mathsf{F}, \mathsf{T}\}^n$,
$\overline{\Psi}_\delta(b_1, ..., b_n) = ([\![op_i Z_i]\!]\ \delta[b_1/z_1, ..., b_n/z_n])_{1 \leq i \leq n}$

The function h_1 defined below constructs a BES encoding the model-checking problem of a guarded simplified formula $\mu X.\varphi$ on an ALTS. For each subformula φ' of φ and each $s \in S$, the BES defines a boolean variable $Z_{\varphi',s}$ that is true iff s satisfies φ'. The auxiliary function $h_2(\varphi', s)$ produces the boolean formulas on the right-hand sides of the equations, by introducing extra variables in order to obtain purely disjunctive or conjunctive formulas.

Definition 6 (translation to BESs) *Let $M = (S, A, T, s_0)$ be an* ALTS *and $\mu X.\varphi$ be a closed simplified guarded formula. The translation $h_1(\mu X.\varphi, s)$ defined below yields a* BES *encoding the satisfaction of $\mu X.\varphi$ by a state $s \in S$.*

φ	$h_1(\varphi, s)$	$h_2(\varphi, s)$
F	$\{\}$	$\bigvee \emptyset$
T		$\bigwedge \emptyset$
$\varphi_1 \vee \varphi_2$	$\{Z_{\varphi_1,s} = h_2(\varphi_1, s), Z_{\varphi_2,s} = h_2(\varphi_2, s)\} \cup$	$Z_{\varphi_1,s} \vee Z_{\varphi_2,s}$
$\varphi_1 \wedge \varphi_2$	$h_1(\varphi_1, s) \cup h_1(\varphi_2, s)$	$Z_{\varphi_1,s} \wedge Z_{\varphi_2,s}$
$\langle\alpha\rangle\,\varphi$	$\{Z_{\varphi,s'} = h_2(\varphi, s') \mid s \xrightarrow{a} s' \wedge a \in [\![\alpha]\!]\} \cup$	$\bigvee\{Z_{\varphi,s'} \mid s \xrightarrow{a} s' \wedge a \in [\![\alpha]\!]\}$
$[\alpha]\,\varphi$	$h_1(\varphi, s)$	$\bigwedge\{Z_{\varphi,s'} \mid s \xrightarrow{a} s' \wedge a \in [\![\alpha]\!]\}$
X	$\{\}$	$Z_{X,s}$
$\mu X.\varphi$	$\{Z_{X,s} = h_2(\varphi, s)\} \cup h_1(\varphi, s)$	

The size of the resulting BES is linear in the size of the formula $\mu X.\varphi$ and the size of the ALTS: there are at most $|\varphi| \cdot |S|$ boolean variables and $|\varphi| \cdot (|S| + |T|)$ operators on the right-hand sides. The functions h_1 and h_2 are very similar to other translations from fixed point formulas to BESs given in [6, 1] or [19, chap. 2]. The following proposition (which can be proven using Bekić's theorem [2] and the isomorphism between $(2^S)^n$ and $\{\mathsf{F}, \mathsf{T}\}^{|S| \cdot n}$, see e.g., [1, 19]) states that the local model-checking problem of $\mu X.\varphi$ on the initial state of $M = (S, A, T, s_0)$ can be reduced to the resolution of variable Z_{X,s_0} in the BES $h_1(\mu X.\varphi, s_0)$.

Proposition 6 *Let $M = (S, A, T, s_0)$ be an* ALTS *and $\mu X.\varphi$ be a closed, simplified guarded formula. Then:*

$$[\![\mu X.\varphi]\!] = \{s \in S \mid ([\![h_1(\mu X.\varphi, s)]\!])_{X,s} = \mathsf{T}\}.$$

Note that during the translation given in Definition 6, the transitions $s \xrightarrow{a} s'$ are traversed forwards, which enables to construct the ALTS simultaneously with the BES in a demand-driven way.

3.2 Local Resolution of BESs

For developing our resolution algorithm, we use a representation of BESs as *boolean graphs* [1], which provide a more intuitive way of reasoning about dependencies between boolean variables. To each BES $\{z_i = op_i Z_i\}_{1 \leq i \leq n}$ corresponds a boolean graph $G = (V, E, L)$, where: $V = \{z_1, ..., z_n\}$ is the set of *vertices* (boolean variables), $E = \{(z_i, z_j) \mid 1 \leq i, j \leq n \wedge z_j \in Z_i\}$ is the set of *edges* (dependencies between boolean variables), and $L : V \rightarrow \{\vee, \wedge\}, L(z_i) = op_i$ is the *vertex labeling* (vertices are conjunctive or disjunctive according to the operator in the corresponding equation). The set of successors of a vertex z is noted $E(z)$. We assume that vertices in $E(z)$ are ordered from $(E(z))_0$ to $(E(z))_{|E(z)|-1}$. A boolean graph is represented *implicitly* by its successor function, which associates to each vertex z the set $E(z)$.

Note that the boolean graphs produced from BESs encoding the local model-checking of guarded μ-calculus formulas on ALTSs are acyclic (otherwise, there would be either a cycle in the ALTS, or an unguarded occurrence of a propositional variable in the formula).

The DAGSOLVE algorithm that we propose for the local resolution of a BES with acyclic boolean graph is shown in Figure 1. It takes as input a boolean graph $G = (V, E, L)$ represented implicitly and a vertex $x \in V$ denoting a boolean variable. Starting at vertex x, DAGSOLVE recursively performs a depth-first search of G and, for each vertex y encountered, it computes its truth value $v(y)$. A counter $p(x)$ indicates the next successor of x to be visited. The vertices already visited are stored in a global set $A \subseteq V$ (initially empty). The exploration of the successors of x stops as soon as its truth value can be decided (e.g., if $L(x) = \vee$, then $v(x)$ becomes T as soon as a successor $y \in E(x)$ with $v(y) = \mathsf{T}$ has been encountered). Upon termination of the call DAGSOLVE $(x, (V, E, L))$, vertex x is contained in A and its final truth value is given by $v(x)$.

DAGSOLVE is similar in spirit with other local BES resolution algorithms [1, 25, 18, 8, 21] based upon exploration of the dependency graph between boolean variables. Like these algorithms, DAGSOLVE has an $O(|V| + |E|)$ time complexity, equivalent to $O(|\varphi| \cdot (|S| + |T|))$ in terms of the state formula and the ALTS. However, by exploiting the absence of cycles in G, DAGSOLVE does not need to keep track of backward dependencies for updating the value of variables, and therefore it does not

```
A := ∅;
procedure DagSolve (x, (V, E, L)) is
  v(x) := if L(x) = ∨ then F else T endif;
  p(x) := 0; A := A ∪ {x};
  while p(x) < |E(x)| do
    y := (E(x))_{p(x)};
    if y ∉ A then
      DagSolve (y, (V, E, L))
    endif;
    if v(y) ≠ v(x) then
      v(x) := v(y); p(x) := |E(x)|
    else
      p(x) := p(x) + 1
    endif
  end
end
```

Figure 1: Local resolution of BESs with acyclic boolean graphs

store the transitions of G in memory. This yields for DAGSOLVE a space complexity $O(|V|)$, equivalent to $O(|\varphi| \cdot |S|)$, instead of $O(|V|+|E|)$ as for the algorithms proposed in [1, 25, 18, 8, 21] when they are applied to acyclic boolean graphs.

3.3 Handling of Unguarded Alternation-Free Mu-Calculus

To verify an alternation-free μ-calculus formula φ on an ALTS $M = (S, A, T, s_0)$, one could first transform φ to simplified guarded form by using the translations given in Sections 2.2 and 2.3, and then apply the DAGSOLVE model-checking algorithm given in Section 3.2. If φ is already guarded, this procedure would yield a time complexity $O(|\varphi| \cdot (|S| + |T|))$ and a space complexity $O(|\varphi| \cdot |S|)$.

However, if φ is unguarded, this would yield a worst-case time complexity $O(|\varphi|^2 \cdot (|S| + |T|))$ instead of the complexity $O(|\varphi| \cdot (|S| + |T|))$ obtained by using an existing linear-time model-checking algorithm [1, 25, 18, 8, 21]. Although unguarded formulas seldom occur in practice, and although the blow-up caused by translation to guarded form is often less than quadratic, the above procedure may be unacceptable in some cases. In this section, we seek to devise a linear-time, memory efficient algorithm for checking unguarded alternation-free formulas on ALTSs.

Unguarded propositional variables occurring in a formula $\mu X.\varphi$ will induce cyclic dependencies in the corresponding BES (i.e., cycles in the boolean graph) even if the LTS $M = (S, A, T, s_0)$ is acyclic. However, these cycles have a rather simple structure:

they contain only vertices of the form $Z_{\varphi_1,s}$, $Z_{\varphi_2,s}$, ..., where φ_1, φ_2, ... are subformulas of φ dominated by boolean or fixed point operators, and the state s remains unchanged along the cycle. Vertices belonging to a strongly connected component (SCC) of the boolean graph are reachable from the vertex of interest Z_{X,s_0} only through the root of the SCC, which is always a vertex $Z_{X',s}$ corresponding to a closed subformula $\mu X'.\varphi'$. Moreover, assuming there is no factorization of common subformulas in the initial formula $\mu X.\varphi$, a vertex of a SCC is reachable from the root of the SCC along a single path.

The AFMCSOLVE algorithm that we propose for solving these BESs is shown in Figure 2. We consider here only the case of minimal fixed point BESs, the other case being dual (every alternation-free formula can be checked by combining these two algorithms [1, 21]). Besides an implicit boolean graph $G = (V, E, L)$ and a vertex $x \in V$, AFMCSOLVE takes as input a predicate $root : V \rightarrow \{\mathsf{F}, \mathsf{T}\}$ indicating the roots of the SCCs of G.

AFMCSOLVE is similar to DAGSOLVE, except that the value $v(x)$ of some vertices x (those with an outgoing path leading to an "ancestor" vertex currently on the call stack of AFMCSOLVE) cannot be decided upon termination of the call AFMCSOLVE $(x, (V, E, L), root)$. This information is recorded using an additional boolean $stable(x)$, which is set to T in two situations: (a) $v(x)$ has been decided after exploring a stable successor $y \in E(x)$; (b) $v(x)$ has not been decided after exploring all successors in $E(x)$, but either no unstable successor of x has been encountered, or x is the root of a SCC of G (in the latter case, $v(x)$ is set to F, since G corresponds to a minimal fixed point BES).

After deciding $v(x)$, there is no need to propagate this value to the "descendants" of x already explored, because these vertices will not be reached anymore in the SCC of x. Therefore, AFMCSOLVE does not need to store the transitions of G in memory, thus achieving a space complexity $O(|V|)$, which is equivalent to $O(|\varphi| \cdot |S|)$.

4 Conclusion and Future Work

We have shown that the full modal μ-calculus interpreted on acyclic LTSs has the same expressive power as its alternation-free fragment. We also proposed two local model-checking algorithms: DAGSOLVE, which handles formulas φ with alternation depth $ad(\varphi) \geq 2$ and has time complexity $O(|\varphi|^2 \cdot (|S| + |T|))$ and space complexity $O(|\varphi|^2 \cdot |S|)$; and AFMCSOLVE, which handles alternation-free formulas and has time complexity $O(|\varphi| \cdot (|S| + |T|))$ and space complexity $O(|\varphi| \cdot |S|)$. Both algorithms have been implemented using the generic OPEN/CÆSAR environment [13] for on-the-fly exploration of LTSs, and are integrated within the EVALUATOR 3.5 model-checker [21] of the CADP verification toolbox [11].

```
A := ∅;
procedure AFMCSOLVE (x, (V, E, L), root) is
  v(x) := if L(x) = ∨ then F else T endif;
  p(x) := 0;
  stable(x) := F;
  A := A ∪ {x};
  unstable := F;
  while p(x) < |E(x)| do
    y := (E(x))_{p(x)};
    if y ∉ A then
      AFMCSOLVE (y, (V, E, L), root)
    endif;
    if stable(y) then
      if v(y) ≠ v(x) then
        v(x) := v(y);
        p(x) := |E(x)|;
        stable(x) := T
      else
        p(x) := p(x) + 1
      endif
    else
      unstable := T
    endif
  end;
  if ¬stable(x) then
    stable(x) := ¬unstable ∨ root(x);
    if unstable ∧ root(x) then
      v(x) := F
    endif
  endif
end
```

Figure 2: Local resolution of BESs produced from unguarded alternation-free formulas

These results are currently applied in an industrial project involving the verification and testing of multiprocessor architectures designed by BULL. One of the verification tasks concerns the off-line analysis of large execution traces (about 100,000 events) obtained from intensive testing of a hardware cache coherency protocol. Several hundreds of correctness properties, derived from the formal specification of the protocol, were checked on the set of traces (grouped in one ALTS). Most properties were expressed as PDL [12] formulas of the form $[R_1]\langle R_2\rangle \mathsf{T}$, where R_1 and R_2 are complex regular expressions encoding sequences of request and response actions. These formulas are translated to guarded alternation-free μ-calculus, simplified by replacing maximal fixed points with minimal ones, and checked on the ALTS using DAGSOLVE. Compared to standard local algorithms for alternation-free μ-calculus [1, 25, 18, 8, 21], this procedure improves both execution time (the number of ALTS traversals is reduced) and memory consumption (ALTS transitions are not stored).

The model-checking approach we proposed can be directly applied to other forms of trace analysis (e.g., run-time monitoring, security log file auditing, etc.) by encoding these problems as model-checking of temporal formulas on single trace ALTSs. Moreover, our simplification of guarded μ-calculus interpreted on ALTSs (equivalence between minimal and maximal fixed points) can be useful as an intermediate optimization step in any model-checker, by allowing to simplify temporal formulas in a similar manner when verifying ALTSs. For instance, when checking CTL [4] formulas on ALTSs, our simplification makes valid the equality $\mathsf{A}[\varphi_1 \mathsf{U} \varphi_2] = \neg\mathsf{E}[\neg\varphi_2 \mathsf{U} \neg(\varphi_2 \vee (\varphi_1 \wedge \mathsf{EX}\,\mathsf{T}))]$, which allows to derive all CTL operators from $\mathsf{E}[.\mathsf{U}.]$ and to reduce the model-checking of CTL to the search of finite transition sequences satisfying $\mathsf{E}[\varphi_1 \mathsf{U} \varphi_2]$.

Two directions for future work seem promising. Firstly, one could devise a local model-checking algorithm for guarded μ-calculus that would work without backtracking in the ALTS (this is not guaranteed by DAGSOLVE, which uses a depth-first traversal of the boolean graph). When analyzing a trace in real-time during its generation, this algorithm would scan it only once, potentially leading to performance improvements. Secondly, our model-checking procedure could be used for comparing on-the-fly an LTS M_1 with an ALTS M_2 modulo a bisimulation or preorder relation. This problem has various practical applications (search for execution sequences, interactive replay of simulation trees and model-checking diagnostics, etc.) and could be solved by building a guarded characteristic formula [15] of M_1 and verifying it on M_2 using the DAGSOLVE algorithm.

Acknowledgements

We are grateful to Hubert Garavel for suggesting and encouraging the research on model-checking acyclic LTSs, and to him, Frédéric Lang, and Gordon Pace for their careful proofreading of the manuscript.

References

[1] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[2] H. Bekić. *Definable operations in general algebras, and the theory of automata and flowcharts*. volume 177 of *Lecture Notes in Computer Science*, pages 30–55. Springer Verlag, Berlin, 1984.

[3] J. C. Bradfield and C. Stirling. *Modal Logics and Mu-Calculi: An Introduction*. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 4, pages 293–330. Elsevier, 2001.

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

[5] R. Cleaveland, M. Klein, and B. Steffen. Faster Model Checking for the Modal Mu-Calculus. In G. v. Bochmann and D. K. Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification CAV '92 (Montréal, Canada)*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422, Berlin, June-July 1992. Springer Verlag.

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

[7] F. Dietrich, X. Logean, S. Koppenhoefer, and J-P. Hubaux. Testing Temporal Logic Properties in Distributed Systems. In A. Petrenko and N. Yevtushenko, editors, *Proceedings of the 11th International Workshop on Testing of Communicating Systems IWTCS'98*, pages 247–262. Kluwer Academic Publishers, 1998.

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

[9] M. Ducassé. OPIUM: An Extendable Trace Analyzer for Prolog. *Journal of Logic Programming*, 39(1–3):177–224, 1999.

[10] E. A. Emerson and C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of the 1st LICS*, pages 267–278, 1986.

[11] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In Rajeev Alur and

Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer Verlag, August 1996.

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

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

[14] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection Approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, 1995.

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

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

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

[18] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully Local and Efficient Evaluation of Alternating Fixed Points. In Bernhard Steffen, editor, *Proceedings of 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *LNCS*, pages 5–19, Berlin, March 1998. Springer Verlag.

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

[20] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume I (Specification). Springer Verlag, 1992.

[21] Radu Mateescu and Mihaela Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. In Stefania Gnesi, Ina Schieferdecker, and Axel Rennoch, editors, *Proceedings of the 5th International Workshop on Formal*

Methods for Industrial Critical Systems FMICS'2000 (Berlin, Germany), GMD Report 91, pages 65–86, Berlin, April 2000. Also available as INRIA Research Report RR-3899.

[22] Perdita Stevens and Colin Stirling. Practical Model-Checking using Games. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *Lecture Notes in Computer Science*, pages 85–101, Berlin, March 1998. Springer Verlag.

[23] C. Stirling. *Modal and Temporal Properties of Processes*. Springer Verlag, 2001.

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

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

[26] I. Walukiewicz. A Complete Deductive System for the μ-calculus. In *Proceedings of the International Conference on Logic in Computer Science LICS'93*, pages 136–147, 1993. Full version available as BRICS Research Report RS-95-6, University of Aarhus, 1995.

A Proofs

This annex contains the proofs of the lemmas and propositions stated in Sections 2.2 and 2.3 that were left unproved in the main text. For the sake of conciseness, we give only the most involved parts of the proofs, the remainder being left as exercise for the interested reader.

The notation $\varphi[\varphi'/X]$, representing the syntactic substitution of all free occurrences of X in φ by φ', has been informally introduced in Section 2.1. To facilitate the reasoning by structural induction, we give below an inductive definition of substitution.

Definition 7 (substitution) *Let φ, φ' be state formulas and X be a variable. The formula $\varphi[\varphi'/X]$ defined inductively below is the* substitution *of X in φ by φ'.*

φ	$\varphi[\varphi'/X]$	φ	$\varphi[\varphi'/X]$
F	F	T	T
$\varphi_1 \vee \varphi_2$	$\varphi_1[\varphi'/X] \vee \varphi_2[\varphi'/X]$	$\varphi_1 \wedge \varphi_2$	$\varphi_1[\varphi'/X] \wedge \varphi_2[\varphi'/X]$
$\langle\alpha\rangle\,\varphi$	$\langle\alpha\rangle\,(\varphi[\varphi'/X])$	$[\alpha]\,\varphi$	$[\alpha]\,(\varphi[\varphi'/X])$
Y	Y *(if* $Y \neq X$*)*	X	φ'
$\sigma Y.\varphi$	$\sigma Y.(\varphi[\varphi'/X])$ *(if* $Y \neq X$*)*	$\sigma X.\varphi$	$\sigma X.\varphi$

In the sequel, we assume the existence of an LTS $M = (S, A, T, s_0)$ on which all temporal formulas will be interpreted. We also assume that all state formulas φ are in normal form, i.e., they satisfy $fv(\varphi) \cap bv(\varphi) = \emptyset$.

A.1 Lemma 1

Proof Let φ, φ' be state formulas such that $bv(\varphi) \cap fv(\varphi') = \emptyset$, X be a propositional variable, and ρ be a propositional context. We must show that:

$$[\![\varphi[\varphi'/X]]\!]\,\rho = [\![\varphi]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

We proceed by structural induction on φ, using the interpretation of state formulas defined in Table 1.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$[\![\mathsf{F}[\varphi'/X]]\!]\,\rho = [\![\mathsf{F}]\!]\,\rho = \emptyset = [\![\mathsf{F}]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:

$$[\![Y[\varphi'/X]]\!]\,\rho = [\![Y]\!]\,\rho = \rho(Y) = (\rho[[\![\varphi']\!]\,\rho/X])(Y) = [\![Y]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

b) $Y = X$. We have:

$$[\![X[\varphi'/X]]\!]\,\rho = [\![\varphi']\!]\,\rho = (\rho[[\![\varphi']\!]\,\rho/X])(X) = [\![X]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). From hypothesis and Definition 1, it follows that $\emptyset = bv(\varphi_1 \vee \varphi_2) \cap fv(\varphi') = (bv(\varphi_1) \cup bv(\varphi_2)) \cap fv(\varphi') = (bv(\varphi_1) \cap fv(\varphi')) \cup (bv(\varphi_2) \cap fv(\varphi'))$, which is equivalent to $bv(\varphi_1) \cap fv(\varphi') = \emptyset$ and $bv(\varphi_2) \cap fv(\varphi') = \emptyset$. Using the induction hypothesis, we have:

$$[\![(\varphi_1 \vee \varphi_2)[\varphi'/X]]\!]\,\rho = [\![\varphi_1[\varphi'/X] \vee \varphi_2[\varphi'/X]]\!]\,\rho =$$
$$[\![\varphi_1[\varphi'/X]]\!]\,\rho \cup [\![\varphi_2[\varphi'/X]]\!]\,\rho = [\![\varphi_1]\!]\,\rho[[\![\varphi']\!]\,\rho/X] \cup [\![\varphi_2]\!]\,\rho[[\![\varphi']\!]\,\rho/X] =$$
$$[\![\varphi_1 \vee \varphi_2]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

Case $\varphi = \langle\alpha\rangle\,\varphi_1$ (similar proof for $\varphi = [\alpha]\,\varphi_1$). From hypothesis and Definition 1, it follows that $\emptyset = bv(\langle\alpha\rangle\,\varphi_1) \cap fv(\varphi') = bv(\varphi_1) \cap fv(\varphi')$. Using the induction hypothesis, we have:

$$[\![(\langle\alpha\rangle\,\varphi_1)[\varphi'/X]]\!]\,\rho = [\![\langle\alpha\rangle\,(\varphi_1[\varphi'/X])]\!]\,\rho =$$
$$\{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1[\varphi'/X]]\!]\,\rho\} =$$
$$\{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\,\rho[[\![\varphi']\!]\,\rho/X]\} =$$
$$[\![\langle\alpha\rangle\,\varphi_1]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). Two cases are possible.

a) $Y \neq X$. From hypothesis and Definition 1, it follows that $\emptyset = bv((\mu Y.\varphi_1)[\varphi'/X]) \cap fv(\varphi') = bv(\mu Y.(\varphi_1[\varphi'/X])) \cap fv(\varphi') = (\{Y\} \cup bv(\varphi_1[\varphi'/X])) \cap fv(\varphi') = (\{Y\} \cap fv(\varphi')) \cup (bv(\varphi_1[\varphi'/X]) \cap fv(\varphi'))$, which means $bv(\varphi_1[\varphi'/X]) \cap fv(\varphi') = \emptyset$ and $Y \notin fv(\varphi')$. Since the interpretation of a formula in a context depends only upon the values of its free variables assigned by the context, it follows that the interpretation of φ' in a context ρ does not depend upon the value of Y assigned by ρ, i.e., for any state set $U \subseteq S$, $[\![\varphi']\!]\,\rho[U/Y] = [\![\varphi']\!]\,\rho$. Using the induction hypothesis, we have, for any $U \subseteq S$:

$$[\![\varphi_1[\varphi'/X]]\!]\,\rho[U/Y] = [\![\varphi_1]\!]\,(\rho[U/Y])[[\![\varphi']\!]\,\rho[U/Y]/X] =$$
$$[\![\varphi_1]\!]\,(\rho[U/Y])[[\![\varphi']\!]\,\rho/X] = [\![\varphi_1]\!]\,(\rho[[\![\varphi']\!]\,\rho/X])[U/Y]$$

which by interpretation of fixed point formulas implies $[\![(\mu Y.\varphi_1)[\varphi'/X]]\!]\,\rho = [\![\mu Y.(\varphi_1[\varphi'/X])]\!]\,\rho = [\![\mu Y.\varphi_1]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$.

b) $Y = X$. We have:

$$[\![(\mu X.\varphi_1)[\varphi'/X]]\!]\,\rho = [\![\mu X.\varphi_1]\!]\,\rho = [\![\mu X.\varphi_1]\!]\,\rho[[\![\varphi']\!]\,\rho/X]$$

□

A.2 Lemma 2

Proof Let φ be a state formula, X be a propositional variable, ρ be a propositional context, and $U \subseteq S$. We must show that:

$$[\![f(\varphi, X, \nu)]\!]\, \rho[U/X] \cap U \subseteq [\![\varphi]\!]\, \rho[U/X] \subseteq [\![f(\varphi, X, \mu)]\!]\, \rho[U/X] \cup U$$

We proceed by structural induction on φ, using Definition 3 and the interpretation of state formulas defined in Table 1.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$[\![f(\mathsf{F}, X, \nu)]\!]\, \rho[U/X] \cap U = [\![\mathsf{F}]\!]\, \rho[U/X] \cap U = \emptyset \cap U = \emptyset = [\![\mathsf{F}]\!]\, \rho[U/X] \subseteq$$
$$[\![\mathsf{F}]\!]\, \rho[U/X] \cup U = [\![f(\mathsf{F}, X, \mu)]\!]\, \rho[U/X] \cup U$$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:

$$[\![f(Y, X, \nu)]\!]\, \rho[U/X] \cap U = [\![Y]\!]\, \rho[U/X] \cap U \subseteq [\![Y]\!]\, \rho[U/X] \subseteq$$
$$[\![Y]\!]\, \rho[U/X] \cup U = [\![f(Y, X, \mu)]\!]\, \rho[U/X] \cup U$$

b) $Y = X$. We have:

$$[\![f(X, X, \nu)]\!]\, \rho[U/X] \cap U = [\![\mathsf{T}]\!]\, \rho[U/X] \cap U = S \cap U = U =$$
$$[\![X]\!]\, \rho[U/X] = U = \emptyset \cup U = [\![\mathsf{F}]\!]\, \rho[U/X] \cup U = [\![f(X, X, \mu)]\!]\, \rho[U/X] \cup U$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using the induction hypothesis, we have:

$$[\![f(\varphi_1 \vee \varphi_2, X, \nu)]\!]\, \rho[U/X] \cap U = [\![f(\varphi_1, X, \nu) \vee f(\varphi_2, X, \nu)]\!]\, \rho[U/X] \cap U =$$
$$([\![f(\varphi_1, X, \nu)]\!]\, \rho[U/X] \cup [\![f(\varphi_2, X, \nu)]\!]\, \rho[U/X]) \cap U =$$
$$([\![f(\varphi_1, X, \nu)]\!]\, \rho[U/X] \cap U) \cup ([\![f(\varphi_2, X, \nu)]\!]\, \rho[U/X] \cap U) \subseteq$$
$$[\![\varphi_1]\!]\, \rho[U/X] \cup [\![\varphi_2]\!]\, \rho[U/X] \subseteq$$
$$([\![f(\varphi_1, X, \mu)]\!]\, \rho[U/X] \cup U) \cup ([\![f(\varphi_2, X, \mu)]\!]\, \rho[U/X] \cup U) =$$
$$([\![f(\varphi_1, X, \mu)]\!]\, \rho[U/X] \cup [\![f(\varphi_2, X, \mu)]\!]\, \rho[U/X]) \cup U =$$
$$[\![f(\varphi_1, X, \mu) \vee f(\varphi_2, X, \mu)]\!]\, \rho[U/X] \cup U = [\![f(\varphi_1 \vee \varphi_2, X, \mu)]\!]\, \rho[U/X] \cup U$$

Case $\varphi = \langle \alpha \rangle\, \varphi_1$ (similar proof for $\varphi = [\alpha]\, \varphi_1$). We have:

$$[\![f(\langle \alpha \rangle\, \varphi_1, X, \nu)]\!]\, \rho[U/X] \cap U = [\![\langle \alpha \rangle\, \varphi_1]\!]\, \rho[U/X] \cap U \subseteq [\![\langle \alpha \rangle\, \varphi_1]\!]\, \rho[U/X] \subseteq$$
$$[\![\langle \alpha \rangle\, \varphi_1]\!]\, \rho[U/X] \cup U = [\![f(\langle \alpha \rangle\, \varphi_1, X, \mu)]\!]\, \rho[U/X] \cup U$$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). We have:

$$[\![f(\mu Y.\varphi_1, X, \nu)]\!]\, \rho[U/X] \cap U = [\![\mu Y.\varphi_1]\!]\, \rho[U/X] \cap U \subseteq [\![\mu Y.\varphi_1]\!]\, \rho[U/X] \subseteq$$
$$[\![\mu Y.\varphi_1]\!]\, \rho[U/X] \cup U = [\![f(\mu Y.\varphi_1, X, \mu)]\!]\, \rho[U/X] \cup U$$

□

A.3 Proposition 3

Several lemmas are needed in order to prove Proposition 3. Lemma 4 establishes some basic properties of the predicates $g(\varphi, \mathcal{X})$ and $wg(\varphi, \mathcal{X})$. Lemma 5 estimates the set of free variables $fv(\varphi[\varphi'/X])$ in terms of the sets $fv(\varphi)$ and $fv(\varphi')$. Lemma 6 estimates the set of free variables $fv(f(\varphi, X, \sigma))$ in terms of the set $fv(\varphi)$. Lemma 7 states that the translations $t(\varphi)$ and $t'(\varphi)$ do not change the free variables of φ. Lemma 8 derives the (weakly) guardedness of $f(\varphi, X, \sigma)$ from the weakly guardedness of φ. Finally, Lemma 9 (resp. Lemma 10) derives the guardedness (resp. weakly guardedness) of $\varphi[\varphi'/X]$ from the guardedness (resp. weakly guardedness) of φ and φ'.

Lemma 4 *Let φ be a state formula. The following properties hold:*

1. $g(\varphi, \mathcal{X}) \Rightarrow wg(\varphi, \mathcal{X})$
2. $g(\varphi, \mathcal{X}_1 \cup \mathcal{X}_2) = g(\varphi, \mathcal{X}_1) \wedge g(\varphi, \mathcal{X}_2)$
3. $wg(\varphi, \mathcal{X}_1 \cup \mathcal{X}_2) = wg(\varphi, \mathcal{X}_1) \wedge wg(\varphi, \mathcal{X}_2)$
4. $g(\varphi, \emptyset) \Rightarrow g(\varphi, \mathcal{X} \setminus fv(\varphi))$
5. $wg(\varphi, fv(\varphi)) \Rightarrow wg(\varphi, \mathcal{X})$

for any sets of propositional variables $\mathcal{X}$, $\mathcal{X}_1$, $\mathcal{X}_2$.

Proof Straightforward, by structural induction on φ, using Definitions 1 and 2. Properties 3 and 4 are proven using Property 2, and Property 5 is proven using Properties 2, 3, and 4. □

Let φ be a state formula and $\mathcal{X}_1$, $\mathcal{X}_2$ be sets of propositional variables. Properties 2 and 3 of Lemma 4 imply the following statement, which will be also used in the sequel:

$$\mathcal{X}_1 \subseteq \mathcal{X}_2 \Rightarrow ((g(\varphi, \mathcal{X}_2) \Rightarrow g(\varphi, \mathcal{X}_1)) \wedge (wg(\varphi, \mathcal{X}_2) \Rightarrow wg(\varphi, \mathcal{X}_1)))$$

Lemma 5 *Let φ, φ' be state formulas such that $bv(\varphi) \cap fv(\varphi') = \emptyset$. Then:*

$$fv(\varphi) \setminus \{X\} \subseteq fv(\varphi[\varphi'/X]) \subseteq (fv(\varphi) \setminus \{X\}) \cup fv(\varphi')$$

for any propositional variable X.

Proof Straightforward, by structural induction on φ, using Definitions 1 and 7. □

Lemma 6 *Let φ be a state formula. Then:*

$$fv(\varphi) \setminus \{X\} \subseteq fv(f(\varphi, X, \sigma)) \subseteq fv(\varphi)$$

for any propositional variable X and any $\sigma \in \{\mu, \nu\}$.

Proof By structural induction on φ, using Definitions 1 and 3.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$fv(\mathsf{F}) \setminus \{X\} = \emptyset \setminus \{X\} = \emptyset = fv(\mathsf{F}) = fv(f(\mathsf{F}, X, \sigma))$$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:

$$fv(Y) \setminus \{X\} = \{Y\} \setminus \{X\} = \{Y\} = fv(Y) = fv(f(Y, X, \sigma))$$

b) $Y = X$. Let $\mathsf{C} =$ if $\sigma = \mu$ then F else T. We have:

$$fv(X) \setminus \{X\} = \{X\} \setminus \{X\} = \emptyset = fv(\mathsf{C}) = fv(f(X, X, \sigma)) \subseteq fv(X)$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using the induction hypothesis, we have:

$$\begin{array}{l} fv(\varphi_1 \vee \varphi_2) \setminus \{X\} = (fv(\varphi_1) \cup fv(\varphi_2)) \setminus \{X\} = \\ (fv(\varphi_1) \setminus \{X\}) \cup (fv(\varphi_2) \setminus \{X\}) \subseteq fv(f(\varphi_1, X, \sigma)) \cup fv(f(\varphi_2, X, \sigma)) = \\ fv(f(\varphi_1, X, \sigma) \vee f(\varphi_2, X, \sigma)) = fv(f(\varphi_1 \vee \varphi_2, X, \sigma)) = \\ fv(f(\varphi_1, X, \sigma)) \cup fv(f(\varphi_2, X, \sigma)) \subseteq fv(\varphi_1) \cup fv(\varphi_2) = fv(\varphi_1 \vee \varphi_2) \end{array}$$

Case $\varphi = \langle \alpha \rangle \varphi_1$ (similar proof for $\varphi = [\alpha] \varphi_1$). Using the induction hypothesis, we have:

$$\begin{array}{l} fv(\langle \alpha \rangle \varphi_1) \setminus \{X\} = fv(\varphi_1) \setminus \{X\} \subseteq fv(f(\varphi_1, X, \sigma)) \subseteq \\ fv(\varphi_1) = fv(\langle \alpha \rangle \varphi_1) = fv(f(\langle \alpha \rangle \varphi_1, X, \sigma)) \end{array}$$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). We have:

$$fv(\mu Y.\varphi_1) \setminus \{X\} = fv(f(\mu Y.\varphi_1, X, \sigma)) \setminus \{X\} \subseteq fv(f(\mu Y.\varphi_1, X, \sigma)) = fv(\mu Y.\varphi_1)$$

□

Lemma 7 *Let φ be a state formula. Then:*

$$fv(t(\varphi)) = fv(t'(\varphi)) = fv(\varphi).$$

Proof By structural induction on φ, using Definitions 1 and 4.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$fv(t(\mathsf{F})) = fv(\mathsf{F}) = fv(t'(\mathsf{F}))$$

Case $\varphi = X$. We have:

$$fv(t(X)) = fv(X) = fv(t'(X))$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using the induction hypothesis, we have:

$$fv(t(\varphi_1 \vee \varphi_2)) = fv(t(\varphi_1) \vee t(\varphi_2)) = fv(t(\varphi_1)) \cup fv(t(\varphi_2)) =$$
$$fv(\varphi_1) \cup fv(\varphi_2) = fv(\varphi_1 \vee \varphi_2) = fv(\varphi_1) \cup fv(\varphi_2) =$$
$$fv(t'(\varphi_1)) \cup fv(t'(\varphi_2)) = fv(t'(\varphi_1) \vee t'(\varphi_2)) = fv(t'(\varphi_1 \vee \varphi_2))$$

Case $\varphi = \langle\alpha\rangle\,\varphi_1$ (similar proof for $\varphi = [\alpha]\,\varphi_1$). Using the induction hypothesis, we have:

$$fv(t(\langle\alpha\rangle\,\varphi_1)) = fv(\langle\alpha\rangle\,t(\varphi_1)) = fv(t(\varphi_1)) = fv(\varphi_1) = fv(\langle\alpha\rangle\,\varphi_1) = fv(\varphi_1) =$$
$$fv(t(\varphi_1)) = fv(\langle\alpha\rangle\,t(\varphi_1)) = fv(t'(\langle\alpha\rangle\,\varphi_1))$$

Case $\varphi = \mu X.\varphi_1$ (similar proof for $\varphi = \nu X.\varphi_1$). Using Lemma 6 and the induction hypothesis, we have:

$$fv(t(\mu X.\varphi_1)) = fv(\mu X.f(t'(\varphi_1), X, \sigma)) = fv(f(t'(\varphi_1), X, \sigma)) \setminus \{X\} =$$
$$fv(t'(\varphi_1)) \setminus \{X\} = fv(\varphi_1) \setminus \{X\} = fv(\mu X.\varphi_1)$$

Using Lemmas 5, 6 and the induction hypothesis, we have:

$$fv(t'(\mu X.\varphi_1)) = fv(f(t'(\varphi_1), X, \sigma)[\mu X.f(t'(\varphi_1), X, \sigma)/X]) =$$
$$fv(f(t'(\varphi_1), X, \sigma)) \setminus \{X\} = fv(t'(\varphi_1)) \setminus \{X\} = fv(\varphi_1) \setminus \{X\} = fv(\mu X.\varphi_1)$$

□

Lemma 8 *Let φ be a state formula. The following properties hold:*

1. $wg(\varphi, \{X\}) \Rightarrow g(f(\varphi, X, \sigma), \{X\})$
2. $wg(\varphi, fv(\varphi)) \Rightarrow wg(f(\varphi, X, \sigma), fv(\varphi))$

for any propositional variable X and any $\sigma \in \{\mu, \nu\}$.

Proof By structural induction on φ, using Definitions 1, 2, and 3.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

1. $wg(\mathsf{F}, \{X\}) = \mathsf{T} = g(\mathsf{F}, \{X\}) = g(f(\mathsf{F}, X, \sigma), \{X\})$
2. $wg(\mathsf{F}, fv(\mathsf{F})) = wg(f(\mathsf{F}, X, \sigma), fv(\mathsf{F}))$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:
   1. $wg(Y, \{X\}) = \mathsf{T} = Y \notin \{X\} = g(Y, \{X\}) = g(f(Y, X, \sigma), \{X\})$
   2. $wg(Y, fv(Y)) = wg(f(Y, X, \sigma), fv(Y))$

b) $Y = X$. Let C = if $\sigma = \mu$ then F else T. We have:
   1. $wg(X, \{X\}) = \mathsf{T} = g(\mathsf{C}, \{X\}) = g(f(X, X, \sigma), \{X\})$
   2. $wg(X, fv(X)) = \mathsf{T} = wg(\mathsf{C}, \{X\}) = wg(f(X, X, \sigma), fv(X))$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$).

1. Using the induction hypothesis, we have:

   $wg(\varphi_1 \vee \varphi_2, \{X\}) = wg(\varphi_1, \{X\}) \wedge wg(\varphi_2, \{X\}) \Rightarrow$
   $g(f(\varphi_1, X, \sigma), \{X\}) \wedge g(f(\varphi_2, X, \sigma), \{X\}) =$
   $g(f(\varphi_1, X, \sigma) \vee f(\varphi_2, X, \sigma), \{X\}) = g(f(\varphi_1 \vee \varphi_2, X, \sigma), \{X\})$

2. Using Lemma 4 (Properties 3, 5), Lemma 6 and the induction hypothesis, we have:

   $wg(\varphi_1 \vee \varphi_2, fv(\varphi_1 \vee \varphi_2)) =$
   $wg(\varphi_1, fv(\varphi_1) \cup fv(\varphi_2)) \wedge wg(\varphi_2, fv(\varphi_1) \cup fv(\varphi_2)) \Rightarrow$
   $wg(\varphi_1, fv(\varphi_1)) \wedge wg(\varphi_2, fv(\varphi_2)) \Rightarrow$
   $wg(f(\varphi_1, X, \sigma), fv(\varphi_1)) \wedge wg(f(\varphi_2, X, \sigma), fv(\varphi_2)) \Rightarrow$
   $wg(f(\varphi_1, X, \sigma), fv(f(\varphi_1, X, \sigma))) \wedge wg(f(\varphi_2, X, \sigma), fv(f(\varphi_2, X, \sigma))) \Rightarrow$
   $wg(f(\varphi_1, X, \sigma), fv(\varphi_1 \vee \varphi_2)) \wedge wg(f(\varphi_2, X, \sigma), fv(\varphi_1 \vee \varphi_2)) =$
   $wg(f(\varphi_1, X, \sigma) \vee f(\varphi_2, X, \sigma), fv(\varphi_1 \vee \varphi_2)) =$
   $wg(f(\varphi_1 \vee \varphi_2, X, \sigma), fv(\varphi_1 \vee \varphi_2))$

Case $\varphi = \langle \alpha \rangle \varphi_1$ (similar proof for $\varphi = [\alpha] \varphi_1$). We have:

1. $wg(\langle \alpha \rangle \varphi_1, \{X\}) = g(\varphi_1, \emptyset) = g(\langle \alpha \rangle \varphi_1, \{X\}) = g(f(\langle \alpha \rangle \varphi_1, X, \sigma), \{X\})$
2. $wg(\langle \alpha \rangle \varphi_1, fv(\langle \alpha \rangle \varphi_1)) = wg(f(\langle \alpha \rangle \varphi_1, X, \sigma), fv(\langle \alpha \rangle \varphi_1))$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). We have:

1. $wg(\mu Y.\varphi_1, \{X\}) = g(\varphi_1, \{X, Y\}) = g(\mu Y.\varphi_1, \{X\}) = g(f(\mu Y.\varphi_1, X, \sigma), \{X\})$
2. $wg(\mu Y.\varphi_1, fv(\mu Y.\varphi_1)) = wg(f(\mu Y.\varphi_1, X, \sigma), fv(\mu Y.\varphi_1))$

□

Lemma 9 *Let φ, φ' be state formulas and $\mathcal{X}$ be a set of propositional variables. Then:*

$$g(\varphi, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\varphi) \subseteq bv(\varphi') \Rightarrow g(\varphi[\varphi'/X], \mathcal{X})$$

for any propositional variable X.

Proof By structural induction on φ, using Definitions 1, 2, and 7.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$g(\mathsf{F}, \mathcal{X}) = g(\mathsf{F}[\varphi'/X], \mathcal{X})$$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:

$$g(Y, \mathcal{X}) = g(Y[\varphi'/X], \mathcal{X})$$

b) $Y = X$. Using Lemma 4 (Property 4), we have:

$$g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(X) \subseteq bv(\varphi') \Rightarrow g(\varphi', \emptyset) \wedge \mathcal{X} \subseteq bv(\varphi') \Rightarrow$$
$$g(\varphi', \mathcal{X} \setminus fv(\varphi')) \wedge \mathcal{X} \setminus fv(\varphi') = \mathcal{X} \Rightarrow g(\varphi', \mathcal{X}) = g(X[\varphi'/X], \mathcal{X})$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using the induction hypothesis, we have:

$$g(\varphi_1 \vee \varphi_2, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\varphi_1 \vee \varphi_2) \subseteq bv(\varphi') =$$
$$g(\varphi_1, \mathcal{X}) \wedge g(\varphi_2, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\varphi_1) \cup bv(\varphi_2) \subseteq bv(\varphi') =$$
$$(g(\varphi_1, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\varphi_1) \subseteq bv(\varphi')) \wedge$$
$$(g(\varphi_2, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\varphi_2) \subseteq bv(\varphi')) \Rightarrow$$
$$g(\varphi_1[\varphi'/X], \mathcal{X}) \wedge g(\varphi_2[\varphi'/X], \mathcal{X}) = g(\varphi_1[\varphi'/X] \vee \varphi_2[\varphi'/X], \mathcal{X}) =$$
$$g((\varphi_1 \vee \varphi_2)[\varphi'/X], \mathcal{X})$$

Case $\varphi = \langle\alpha\rangle\, \varphi_1$ (similar proof for $\varphi = [\alpha]\, \varphi_1$). Using the induction hypothesis, we have:

$$g(\langle\alpha\rangle\, \varphi_1, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\langle\alpha\rangle\, \varphi_1) \subseteq bv(\varphi') \Rightarrow$$
$$g(\varphi_1, \emptyset) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \subseteq bv(\varphi') \Rightarrow$$
$$g(\varphi_1[\varphi'/X], \emptyset) = g(\langle\alpha\rangle\, (\varphi_1[\varphi'/X]), \mathcal{X}) = g((\langle\alpha\rangle\, \varphi_1)[\varphi'/X], \mathcal{X})$$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). Two cases are possible.

a) $Y \neq X$. Using the induction hypothesis, we have:

$$g(\mu Y.\varphi_1, \mathcal{X}) \wedge g(\varphi', \emptyset) \wedge \mathcal{X} \cup bv(\mu Y.\varphi_1) \subseteq bv(\varphi') =$$
$$g(\varphi_1, \mathcal{X} \cup \{Y\}) \wedge g(\varphi', \emptyset) \wedge (\mathcal{X} \cup \{Y\}) \cup bv(\varphi_1) \subseteq bv(\varphi') \Rightarrow$$
$$g(\varphi_1[\varphi'/X], \mathcal{X} \cup \{Y\}) = g(\mu Y.(\varphi_1[\varphi'/X]), \mathcal{X}) = g((\mu Y.\varphi_1)[\varphi'/X], \mathcal{X})$$

b) $Y = X$. We have:

$$g(\mu X.\varphi_1, \mathcal{X}) = g((\mu X.\varphi_1)[\varphi'/X], \mathcal{X})$$

□

Lemma 10 *Let φ, φ' be state formulas. Then:*

$$wg(\varphi, fv(\varphi)) \wedge g(\varphi, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi) \subseteq bv(\varphi') \Rightarrow wg(\varphi[\varphi'/X], fv(\varphi) \cup fv(\varphi'))$$

for any propositional variable X.

Proof By structural induction on φ, using Definitions 1, 2, and 7.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$wg(\mathsf{F}, fv(\mathsf{F})) \wedge g(\mathsf{F}, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\mathsf{F}) \subseteq bv(\varphi') = g(\varphi', \emptyset) \Rightarrow$$
$$\mathsf{T} = wg(\mathsf{F}, fv(\mathsf{F}) \cup fv(\varphi')) = wg(\mathsf{F}[\varphi'/X], fv(\mathsf{F}) \cup fv(\varphi'))$$

Case $\varphi = Y$. Two cases are possible.

a) $Y \neq X$. We have:

$$wg(Y, fv(Y)) \wedge g(Y, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(Y) \subseteq bv(\varphi') = g(\varphi', \emptyset) \Rightarrow$$
$$\mathsf{T} = wg(Y, fv(Y) \cup fv(\varphi')) = wg(Y[\varphi'/X], fv(Y) \cup fv(\varphi'))$$

b) $Y = X$. We have:

$$wg(X, fv(X)) \wedge g(X, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(X) \subseteq bv(\varphi') =$$
$$\mathsf{F} \Rightarrow wg(X[\varphi'/X], fv(X) \cup fv(\varphi'))$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using Lemma 4 (Properties 3 and 5), Lemma 5 and the induction hypothesis, we have:

$$wg(\varphi_1 \vee \varphi_2, fv(\varphi_1 \vee \varphi_2)) \wedge g(\varphi_1 \vee \varphi_2, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1 \vee \varphi_2) \subseteq bv(\varphi') =$$
$$wg(\varphi_1, fv(\varphi_1) \cup fv(\varphi_2)) \wedge wg(\varphi_2, fv(\varphi_1) \cup fv(\varphi_2)) \wedge g(\varphi_1, \{X\}) \wedge g(\varphi_2, \{X\}) \wedge$$
$$g(\varphi', \emptyset) \wedge bv(\varphi_1) \cup bv(\varphi_2) \subseteq bv(\varphi') \Rightarrow$$
$$(wg(\varphi_1, fv(\varphi_1)) \wedge g(\varphi_1, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \subseteq bv(\varphi')) \wedge$$
$$(wg(\varphi_2, fv(\varphi_2)) \wedge g(\varphi_2, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_2) \subseteq bv(\varphi')) \Rightarrow$$
$$wg(\varphi_1[\varphi'/X], fv(\varphi_1) \cup fv(\varphi')) \wedge wg(\varphi_2[\varphi'/X], fv(\varphi_2) \cup fv(\varphi')) \Rightarrow$$
$$wg(\varphi_1[\varphi'/X], fv(\varphi_1[\varphi'/X])) \wedge wg(\varphi_2[\varphi'/X], fv(\varphi_2[\varphi'/X])) \Rightarrow$$
$$wg(\varphi_1[\varphi'/X], fv(\varphi_1 \vee \varphi_2) \cup fv(\varphi')) \wedge wg(\varphi_2[\varphi'/X], fv(\varphi_1 \vee \varphi_2) \cup fv(\varphi')) =$$
$$wg(\varphi_1[\varphi'/X] \vee \varphi_2[\varphi'/X], fv(\varphi_1 \vee \varphi_2) \cup fv(\varphi')) =$$
$$wg((\varphi_1 \vee \varphi_2)[\varphi'/X], fv(\varphi_1 \vee \varphi_2) \cup fv(\varphi'))$$

Case $\varphi = \langle \alpha \rangle \varphi_1$ (similar proof for $\varphi = [\alpha] \varphi_1$). Using Lemma 9, we have:

$$wg(\langle \alpha \rangle \varphi_1, fv(\langle \alpha \rangle \varphi_1)) \wedge g(\langle \alpha \rangle \varphi_1, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\langle \alpha \rangle \varphi_1) \subseteq bv(\varphi') =$$
$$g(\varphi_1, \emptyset) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \subseteq bv(\varphi') \Rightarrow g(\varphi_1[\varphi'/X], \emptyset) =$$
$$wg(\langle \alpha \rangle (\varphi_1[\varphi'/X]), fv(\langle \alpha \rangle \varphi_1) \cup fv(\varphi')) =$$
$$wg((\langle \alpha \rangle \varphi_1)[\varphi'/X], fv(\langle \alpha \rangle \varphi_1) \cup fv(\varphi'))$$

Case $\varphi = \mu Y.\varphi_1$ (similar proof for $\varphi = \nu Y.\varphi_1$). Two cases are possible.

a) $Y \neq X$. Using Lemma 4 (Properties 1 and 2), Lemma 9 and the induction hypothesis, we have:

$$
\begin{array}{l}
wg(\mu Y.\varphi_1, fv(\mu Y.\varphi_1)) \wedge g(\mu Y.\varphi_1, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\mu Y.\varphi_1) \subseteq bv(\varphi') = \\
g(\varphi_1, fv(\mu Y.\varphi_1) \cup \{Y\}) \wedge g(\varphi_1, \{X,Y\}) \wedge g(\varphi', \emptyset) \wedge bv(\mu Y.\varphi_1) \subseteq bv(\varphi') = \\
g(\varphi_1, (fv(\varphi_1) \setminus \{Y\}) \cup \{X,Y\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \cup \{Y\} \subseteq bv(\varphi') = \\
g(\varphi_1, fv(\varphi_1) \cup \{X,Y\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \cup \{Y\} \subseteq bv(\varphi') \Rightarrow \\
(wg(\varphi_1, fv(\varphi_1)) \wedge g(\varphi_1, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\varphi_1) \subseteq bv(\varphi')) \wedge \\
(g(\varphi_1, \{Y\}) \wedge g(\varphi', \emptyset) \wedge \{Y\} \cup bv(\varphi_1) \subseteq bv(\varphi')) \Rightarrow \\
g(\varphi_1[\varphi'/X], fv(\varphi_1) \cup fv(\varphi')) \wedge g(\varphi_1[\varphi'/X], \{Y\}) = \\
g(\varphi_1[\varphi'/X], fv(\varphi_1) \cup \{Y\} \cup fv(\varphi')) = \\
g(\varphi_1[\varphi'/X], ((fv(\varphi_1) \setminus \{Y\}) \cup fv(\varphi')) \cup \{Y\}) = \\
wg(\mu Y.(\varphi_1[\varphi'/X]), (fv(\varphi_1) \setminus \{Y\}) \cup fv(\varphi')) = \\
wg((\mu Y.\varphi_1)[\varphi'/X], fv(\mu Y.\varphi_1) \cup fv(\varphi'))
\end{array}
$$

b) $Y = X$. Using Lemma 4 (Properties 2 and 4), we have:

$$
\begin{array}{l}
wg(\mu X.\varphi_1, fv(\mu X.\varphi_1)) \wedge g(\mu X.\varphi_1, \{X\}) \wedge g(\varphi', \emptyset) \wedge bv(\mu X.\varphi_1) \subseteq bv(\varphi') \Rightarrow \\
g(\varphi_1, fv(\mu X.\varphi_1) \cup \{X\}) = g(\varphi_1, (fv(\varphi_1) \setminus \{X\}) \cup \{X\}) = \\
g(\varphi_1, fv(\varphi_1) \cup \{X\}) = g(\varphi_1, \emptyset) \wedge g(\varphi_1, fv(\varphi_1) \cup \{X\}) \Rightarrow \\
g(\varphi_1, fv(\varphi') \setminus fv(\varphi_1)) \wedge g(\varphi_1, fv(\varphi_1) \cup \{X\}) = \\
g(\varphi_1, fv(\varphi') \cup fv(\varphi_1) \cup \{X\}) = g(\varphi_1, ((fv(\varphi_1) \setminus \{X\}) \cup fv(\varphi')) \cup \{X\}) = \\
wg(\mu X.\varphi_1, fv(\mu X.\varphi_1) \cup fv(\varphi')) = wg((\mu X.\varphi_1)[\varphi'/X], fv(\mu X.\varphi_1) \cup fv(\varphi'))
\end{array}
$$

□

Proof (*Proposition 3*) Let φ be a state formula. We must show two properties:

1. $g(t(\varphi), \emptyset)$
2. $wg(t'(\varphi), fv(\varphi))$

We prove these two properties simultaneously, by structural induction on φ, using Definitions 1, 2, and 4.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

1. $g(t(\mathsf{F}), \emptyset) = g(\mathsf{F}, \emptyset) = \mathsf{T}$
2. $wg(t'(\mathsf{F}), fv(\mathsf{F})) = wg(\mathsf{F}, \emptyset) = \mathsf{T}$

Case $\varphi = X$. We have:

1. $g(t(X), \emptyset) = g(X, \emptyset) = X \notin \emptyset = \mathsf{T}$
2. $wg(t'(X), fv(X)) = wg(X, \{X\}) = \mathsf{T}$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$).

1. Using the induction hypothesis, we have:

   $\mathsf{T} = g(t(\varphi_1), \emptyset) \wedge g(t(\varphi_2), \emptyset) = g(t(\varphi_1) \vee t(\varphi_2), \emptyset) = g(t(\varphi_1 \vee \varphi_2), \emptyset)$

2. Using Lemma 4 (Property 5), Lemma 7 and the induction hypothesis, we have:

   $\mathsf{T} = wg(t'(\varphi_1), fv(\varphi_1)) \wedge wg(t'(\varphi_2), fv(\varphi_2)) \Rightarrow$
   $wg(t'(\varphi_1), fv(t'(\varphi_1))) \wedge wg(t'(\varphi_2), fv(t'(\varphi_2))) \Rightarrow$
   $wg(t'(\varphi_1), fv(\varphi_1 \vee \varphi_2)) \wedge wg(t'(\varphi_2), fv(\varphi_1 \vee \varphi_2)) =$
   $wg(t'(\varphi_1) \vee t'(\varphi_2), fv(\varphi_1 \vee \varphi_2)) = wg(t'(\varphi_1 \vee \varphi_2), fv(\varphi_1 \vee \varphi_2))$

Case $\varphi = \langle\alpha\rangle\,\varphi_1$ (similar proof for $\varphi = [\alpha]\,\varphi_1$). Using the induction hypothesis, we have:

1. $\mathsf{T} = g(t(\varphi_1), \emptyset) = g(\langle\alpha\rangle\, t(\varphi_1), \emptyset) = g(t(\langle\alpha\rangle\,\varphi_1), \emptyset)$
2. $\mathsf{T} = g(t(\varphi_1), \emptyset) = wg(\langle\alpha\rangle\, t(\varphi_1), fv(\langle\alpha\rangle\,\varphi_1)) = wg(t'(\langle\alpha\rangle\,\varphi_1), fv(\langle\alpha\rangle\,\varphi_1))$

Case $\varphi = \mu X.\varphi_1$ (similar proof for $\varphi = \nu X.\varphi_1$).

1. Using Lemma 4 (Property 5), Lemma 7, Lemma 8 (Property 1) and the induction hypothesis, we have:

   $\mathsf{T} = wg(t'(\varphi_1), fv(\varphi_1)) = wg(t'(\varphi_1), fv(t'(\varphi_1))) \Rightarrow wg(t'(\varphi_1), \{X\}) \Rightarrow$
   $g(f(t'(\varphi_1), X, \mu), \{X\}) = g(\mu X.f(t'(\varphi_1), X, \mu), \emptyset) = g(t(\mu X.\varphi_1), \emptyset)$

2. Using Lemma 4 (Properties 3 and 5), Lemmas 6, 7, 8 and the induction hypothesis, we have:

   $\mathsf{T} = wg(t'(\varphi_1), fv(\varphi_1)) = wg(t'(\varphi_1), fv(t'(\varphi_1))) \Rightarrow$
   $wg(t'(\varphi_1), fv(t'(\varphi_1))) \wedge wg(t'(\varphi_1), \{X\}) \Rightarrow$
   $wg(f(t'(\varphi_1), X, \mu), fv(t'(\varphi_1))) \wedge g(f(t'(\varphi_1), X, \mu), \{X\}) \Rightarrow$
   $wg(f(t'(\varphi_1), X, \mu), fv(f(t'(\varphi_1), X, \mu))) \wedge g(f(t'(\varphi_1), X, \mu), \{X\}) \wedge$
   $g(\mu X.f(t'(\varphi_1), X, \mu), \emptyset)$

   Since $bv(f(t'(\varphi_1), X, \mu)) \subseteq bv(\mu X.f(t'(\varphi_1), X, \mu))$, by applying Lemma 10 (for $\varphi = f(t'(\varphi_1), X, \mu)$ and $\varphi' = \mu X.f(t'(\varphi_1), X, \mu)$) we obtain:

   $wg(f(t'(\varphi_1), X, \mu)[\mu X.f(t'(\varphi_1), X, \mu)/X],$
   $\quad fv(f(t'(\varphi_1), X, \mu)) \cup fv(\mu X.f(t'(\varphi_1), X, \mu)))$

   Using Lemma 4 (Property 3), Lemmas 6 and 7, this further implies:

   $wg(f(t'(\varphi_1), X, \mu)[\mu X.f(t'(\varphi_1), X, \mu)/X], fv(f(t'(\varphi_1), X, \mu))) \Rightarrow$
   $wg(f(t'(\varphi_1), X, \mu)[\mu X.f(t'(\varphi_1), X, \mu)/X], fv(t'(\varphi_1)) \setminus \{X\}) =$
   $wg(f(t'(\varphi_1), X, \mu)[\mu X.f(t'(\varphi_1), X, \mu)/X], fv(\varphi_1) \setminus \{X\}) =$
   $wg(t'(\mu X.\varphi_1), fv(\varphi_1) \setminus \{X\}) = wg(t'(\mu X.\varphi_1), fv(\mu X.\varphi_1))$

□

A.4 Proposition 5

Proof Let φ be a state formula, ρ be a propositional context, and $X_1, ..., X_n$ be propositional variables. We must show that:

$$S \setminus [\![\varphi]\!]\,\rho = [\![d(\varphi, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n]$$

We proceed by structural induction on φ, using Definition 5 and the interpretation of state formulas defined in Table 1.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$\begin{array}{l} S \setminus [\![\mathsf{F}]\!]\,\rho = S \setminus \emptyset = S = [\![\mathsf{T}]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(\mathsf{F}, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \end{array}$$

Case $\varphi = X$. Two cases are possible.

a) $X \notin \{X_1, ..., X_n\}$. We have:

$$\begin{array}{l} S \setminus [\![X]\!]\,\rho = S \setminus \rho(X) = S \setminus (\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n])(X) = \\ S \setminus [\![X]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![\neg X]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(X, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \end{array}$$

b) $\exists i \in [1, n].X = X_i$. We have:

$$\begin{array}{l} S \setminus [\![X_i]\!]\,\rho = S \setminus \rho(X_i) = (\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n])(X_i) = \\ [\![X_i]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(X_i, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \end{array}$$

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). Using the induction hypothesis, we have:

$$\begin{array}{l} S \setminus [\![\varphi_1 \vee \varphi_2]\!]\,\rho = S \setminus ([\![\varphi_1]\!]\,\rho \cup [\![\varphi_2]\!]\,\rho) = (S \setminus [\![\varphi_1]\!]\,\rho) \cap (S \setminus [\![\varphi_2]\!]\,\rho) = \\ [\![d(\varphi_1, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \cap \\ [\![d(\varphi_2, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(\varphi_1, \{X_1, ..., X_n\}) \wedge d(\varphi_2, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(\varphi_1 \vee \varphi_2, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \end{array}$$

Case $\varphi = \langle\alpha\rangle\,\varphi_1$ (similar proof for $\varphi = [\alpha]\,\varphi_1$). Using the induction hypothesis, we have:

$$\begin{array}{l} S \setminus [\![\langle\alpha\rangle\,\varphi_1]\!]\,\rho = S \setminus \{s \in S \mid \exists s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\,\rho\} = \\ \{s \in S \mid \forall s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \Rightarrow s' \in S \setminus [\![\varphi_1]\!]\,\rho\} = \\ \{s \in S \mid \forall s \xrightarrow{a} s' \in T.a \in [\![\alpha]\!] \Rightarrow \\ \qquad s' \in [\![d(\varphi_1, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n]\} = \\ [\![[\alpha]\,d(\varphi_1, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\ [\![d(\langle\alpha\rangle\,\varphi_1, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] \end{array}$$

Case $\varphi = \mu X.\varphi_1$ (similar proof for $\varphi = \nu X.\varphi_1$). Using the induction hypothesis, we have:

$$
\begin{array}{l}
S \setminus [\![\mu X.\varphi_1]\!]\,\rho = S \setminus \bigcap\{U \subseteq S \mid [\![\varphi_1]\!]\,\rho[U/X] \subseteq U\} = \\
\bigcup\{S \setminus U \mid S \setminus U \subseteq S \setminus [\![\varphi_1]\!]\,\rho[U/X]\} = \\
\bigcup\{S \setminus U \mid S \setminus U \subseteq \\
\qquad [\![d(\varphi_1, \{X_1, ..., X_n, X\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n, S \setminus U/X]\} = \\
\bigcup\{V \subseteq S \mid V \subseteq \\
\qquad [\![d(\varphi_1, \{X_1, ..., X_n, X\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n][V/X]\} = \\
[\![\nu X.d(\varphi_1, \{X_1, ..., X_n, X\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n] = \\
[\![d(\mu X.\varphi_1, \{X_1, ..., X_n\})]\!]\,\rho[S \setminus \rho(X_1)/X_1, ..., S \setminus \rho(X_n)/X_n]
\end{array}
$$

□

A.5 Lemma 3

Proof Let $M = (S, A, T, s_0)$ be an ALTS, φ be a state formula, $X_1, ..., X_n$ be propositional variables, $A_1, B_1, ..., A_n, B_n \subseteq S$ be state sets, ρ be a propositional context, and $\mathcal{Y} \subseteq \{X_1, ..., X_n\}$ be a set of variables such that $g(\varphi, \mathcal{Y})$. We must show that:

$$
\begin{array}{l}
\forall s \in [\![\varphi]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![d(\varphi, \{X_1, ..., X_n\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n]\ . \\
\exists i \in [1, n]\ .\ \exists s' \in A_i \cap B_i\ .\ s \xrightarrow{*} s' \wedge (X_i \in \mathcal{Y} \Rightarrow s' \neq s)
\end{array}
$$

We proceed by structural induction on φ, using Definitions 2, 5 and the interpretation of state formulas defined in Table 1.

Case $\varphi = \mathsf{F}$ (similar proof for $\varphi = \mathsf{T}$). We have:

$$
\begin{array}{l}
[\![\mathsf{F}]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![d(\mathsf{F}, \{X_1, ..., X_n\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n] = \\
[\![\mathsf{F}]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![\mathsf{T}]\!]\,\rho[B_1/X_1, ..., B_n/X_n] = \emptyset \cap S = \emptyset
\end{array}
$$

and the statement holds vacuously.

Case $\varphi = X$. Two cases are possible.

a) $X \notin \{X_1, ..., X_n\}$. We have:

$$
\begin{array}{l}
[\![X]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![d(X, \{X_1, ..., X_n\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n] = \\
[\![X]\!]\,\rho[A_1/X_1, ..., A_n/X_n] \cap [\![\neg X]\!]\,\rho[B_1/X_1, ..., B_n/X_n] = \\
\rho(X) \cap (S \setminus \rho(X)) = \emptyset
\end{array}
$$

and the statement holds vacuously.

b) $\exists i \in [1,n].X = X_i$. We have:

$$[\![X_i]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(X_i,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n] =$$
$$[\![X_i]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![X_i]\!]\,\rho[B_1/X_1,...,B_n/X_n] = A_i \cap B_i$$

Let $s \in A_i \cap B_i$. Since by hypothesis we have $g(X_i,\mathcal{Y})$, which is equivalent to $X_i \notin \mathcal{Y}$, we can take $s' = s$ ($s \xrightarrow{*} s'$ in 0 steps).

Case $\varphi = \varphi_1 \vee \varphi_2$ (similar proof for $\varphi = \varphi_1 \wedge \varphi_2$). We have:

$$[\![\varphi_1 \vee \varphi_2]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap$$
$$[\![d(\varphi_1 \vee \varphi_2,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n] =$$
$$([\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cup [\![\varphi_2]\!]\,\rho[A_1/X_1,...,A_n/X_n]) \cap$$
$$([\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n] \cap$$
$$[\![d(\varphi_2,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]) \subseteq$$
$$([\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]) \cup$$
$$([\![\varphi_2]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\varphi_2,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n])$$

Two cases are possible.

a) $s \in [\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]$. By hypothesis we have $g(\varphi_1 \vee \varphi_2,\mathcal{Y})$, which by Definition 2 implies $g(\varphi_1,\mathcal{Y})$. By induction hypothesis applied to s, φ_1, $A_1, B_1, ..., A_n, B_n$, and $\mathcal{Y}$, we obtain the desired statement: there exists $i \in [1,n]$ and $s' \in A_i \cap B_i$ such that $s \xrightarrow{*} s'$ and $X_i \in \mathcal{Y} \Rightarrow s' \neq s$.

b) $s \in [\![\varphi_2]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\varphi_2,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]$. Similar to the previous case.

Case $\varphi = \langle\alpha\rangle\,\varphi_1$ (similar proof for $\varphi = [\alpha]\,\varphi_1$). We have:

$$[\![\langle\alpha\rangle\,\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\langle\alpha\rangle\,\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n] =$$
$$[\![\langle\alpha\rangle\,\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![[\alpha]\,d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n] =$$
$$\{s \in S \mid \exists s \xrightarrow{a} s'.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n]\} \cap$$
$$\{s \in S \mid \forall s \xrightarrow{a} s'.a \in [\![\alpha]\!] \Rightarrow s' \in [\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]\} \subseteq$$
$$\{s \in S \mid \exists s \xrightarrow{a} s'.a \in [\![\alpha]\!] \wedge s' \in [\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap$$
$$[\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]\}$$

Thus, there exists a state $s' \in [\![\varphi_1]\!]\,\rho[A_1/X_1,...,A_n/X_n] \cap [\![d(\varphi_1,\{X_1,...,X_n\})]\!]\,\rho[B_1/X_1,...,B_n/X_n]$ such that $s \rightarrow s'$. By hypothesis $g(\langle\alpha\rangle\,\varphi_1,\mathcal{Y})$, which by Definition 2 implies $g(\varphi_1,\emptyset)$. By induction hypothesis applied to s', φ_1, $A_1, B_1, ..., A_n, B_n$, and $\emptyset$, there exists $i \in [1,n]$ and $s'' \in A_i \cap B_i$ such that $s' \xrightarrow{*} s''$. Note that $s'' \neq s$, because otherwise there would be a non empty cycle $s \rightarrow s' \xrightarrow{*} s'' = s$ (contradiction with M acyclic). Hence, we have shown the desired statement: there exists an appropriate transition sequence $s \rightarrow s' \xrightarrow{*} s''$ with $s'' \neq s$.

Case $\varphi = \mu X.\varphi_1$ (similar proof for $\varphi = \nu X.\varphi_1$). We have:

$$[\![\mu X.\varphi_1]\!]\rho[A_1/X_1, ..., A_n/X_n] \cap [\![d(\mu X.\varphi_1,\{X_1, ..., X_n\})]\!]\rho[B_1/X_1, ..., B_n/X_n] = [\![\mu X.\varphi_1]\!]\rho[A_1/X_1, ..., A_n/X_n] \cap [\![\nu X.d(\varphi_1,\{X_1, ..., X_n, X\})]\!]\rho[B_1/X_1, ..., B_n/X_n]$$

We note $A_{n+1} = [\![\mu X.\varphi_1]\!]\,\rho[A_1/X_1, ..., A_n/X_n]$ and $B_{n+1} = [\![\nu X.d(\varphi_1, \{X_1, ..., X_n, X\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n]$. By unfolding the fixed point formulas $\mu X.\varphi_1$ and $\nu X.d(\varphi_1, \{X_1, ..., X_n, X\})$ and applying Proposition 1, the above expression becomes $[\![\varphi_1]\!]\,\rho[A_1/X_1, ..., A_n/X_n, A_{n+1}/X] \cap [\![d(\varphi_1, \{X_1, ..., X_n, X\})]\!]\,\rho[B_1/X_1, ..., B_n/X_n, B_{n+1}/X] = A_{n+1} \cap B_{n+1}$. Let $s \in A_{n+1} \cap B_{n+1}$. By hypothesis, we have $g(\mu X.\varphi_1, \mathcal{Y})$, which by Definition 2 is equivalent to $g(\varphi_1, \mathcal{Y} \cup \{X\})$. By induction hypothesis applied to s, φ_1, $A_1, B_1, ..., A_{n+1}, B_{n+1}$, and $\mathcal{Y} \cup \{X\}$, there exists $i \in [1, n+1]$ and $s' \in A_i \cap B_i$ such that $s \xrightarrow{*} s'$ and $X_i \in \mathcal{Y} \cup \{X\} \Rightarrow s' \neq s$. Two cases are possible.

a) $i \neq n+1$. This means $X_i \neq X$, which implies $X_i \in \mathcal{Y} \cup \{X\} \Leftrightarrow X_i \in \mathcal{Y}$. In this case we are done, because we have found a state $s' \in A_i \cap B_i$ such that $s \xrightarrow{*} s'$ and $X_i \in \mathcal{Y} \Rightarrow s' \neq s$.

b) $i = n+1$. Since $X \in \mathcal{Y} \cup \{X\}$, this means we have found a non empty transition sequence from state $s \in A_{n+1} \cap B_{n+1}$ to another state $s' \in A_{n+1} \cap B_{n+1}$. We can apply again the induction hypothesis to s', φ_1, $A_1, B_1, ..., A_{n+1}, B_{n+1}$, and $\mathcal{Y} \cup \{X\}$, and find $j \in [1, n+1]$ and $s'' \in A_j \cap B_j$ such that $s' \xrightarrow{*} s''$ and $X_j \in \mathcal{Y} \cup \{X\} \Rightarrow s'' \neq s'$. Note that $s'' \neq s$, because otherwise there would be a non empty cycle $s \xrightarrow{*} s' \xrightarrow{*} s'' = s$ (contradiction with M acyclic). If $j \neq n+1$, by repeating the reasoning of case a), we are done, since we have found a state s'' such that $s \xrightarrow{*} s' \xrightarrow{*} s''$ and $s'' \neq s$. If $j = n+1$, we can again apply the induction hypothesis to s'' and find another transition sequence starting at s''. This process will eventually lead to a state $s_{final} \notin A_{n+1} \cap B_{n+1}$, because otherwise there would be a non empty cycle between states in $A_{n+1} \cap B_{n+1}$ (contradiction with M acyclic). By induction hypothesis $s_{final} \in A_k \cap B_k$ for some $k \in [1, n]$, and moreover $s_{final} \neq s$ because M is acyclic. In this case we are done, because we have found an appropriate transition sequence $s \xrightarrow{*} s' \xrightarrow{*} \cdots \xrightarrow{*} s_{final}$ with $s_{final} \neq s$.

□

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopˆole de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

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