Automated Software Engineering, 11, 289–321, 2004

Model Checking for Combined Logics with an Application to Mobile Systems*

MASSIMO FRANCESCHET m.franceschet@unich.it
Department of Sciences, University of Chieti-Pescara, Italy

ANGELO MONTANARI montana@dimi.uniud.it
Department of Mathematics and Computer Science, University of Udine, Italy

MAARTEN DE RIJKE mdr@science.uva.nl
Institute for Logic, Language, and Computation, University of Amsterdam, The Netherlands

Abstract. In this paper, we develop model checking procedures for three ways of combining (temporal) logics: temporalization, independent combination, and join. We prove that they are terminating, sound, and complete, we analyze their computational complexity, and we report on experiments with implementations. We take a close look at mobile systems and show how the proposed combined model checking framework can be successfully applied to the specification and verification of their properties.

Keywords: temporal logic, model checking, combined logics, mobile systems

1. Introduction

Logic combination is emerging as a relevant research topic at the intersection of mathematical logic and computer science. It essentially provides a formal account of classical notions, such as modularity and abstraction, which are at the basis of any software development and verification methodology. As any interesting real-world system is a complex entity, decomposing its descriptive and inferential requirements for design, verification, or maintenance purposes into simpler reasoning tasks is often the only plausible way forward (Gabbay and de Rijke, 2000). Assuming that we have methods and tools available to tackle restricted tasks, how do we combine them to solve complex tasks? In particular, how do we combine them in such a way that features of the components are inherited by the combination? This question is known as the *transfer problem* (Blackburn and de Rijke, 1996). Whether properties transfer from the components to the combination depends on the amount of interaction between the component logics; even in the presence of very weak forms of interaction (such as shared symbols), transfer may fail (Finger and Gabbay, 1996; Gabbay and Shehtman, 1998; Hemaspaandra, 1994). In the absence of interaction between the component logics, we often have transfer; such positive results are usually based on a *divide and conquer* strategy: split problems into sub-problems and delegate these to the components (Fine and

*This paper is an extended and revised version of Franceschet et al. (2000).

Schurz, 1996; Kracht and Wolter, 1991). From a *computational* point of view, the natural question in the setting of combining logics is: does it work? Can we *re-use* tools and procedures in a modular fashion? So far, most of the work towards answering this question has gone into putting together deductive engines. While there are no uniform solutions, there are many successful instances of combined proof procedures, especially for modal and modal-like logics; these are often based on calculi satisfying special criteria or on translating the component logics into a background logic. In this paper we study the combination of model checking procedures.

Our motivation for studying model checking algorithms for combined logics comes from a concrete example: *mobile reactive systems* (*mobile systems* for short) (Cardelli, 1999; Cardelli and Gordon, 2000a, 2000b). Mobile systems are strongly distributed reactive systems, whose behavior is subject to both temporal and spatial constraints. One of the novel and challenging features of mobile systems is the fact that their structure is subject to change as computations evolve. More precisely, the state of a mobile system depends on three orthogonal components that change over time: the configuration of the distributed system (its elements and their spatial localization), the distribution of the processes within the system, and the state of every single process. Accordingly, the behavior of a mobile system can be modeled in terms of the evolving combination of these three basic components. A model checking solution for mobile systems can thus be obtained (i) by modeling them as suitable combinations of spatial and temporal structures, possibly at different levels of abstraction, (ii) by using combinations of temporal (and spatial) logics to specify their properties, and (iii) by deploying combinations of model checkers for component logics.

In the following, we focus our attention on three well-known forms of logic combinations, namely, temporalization, independent combination, and join. For each of them, we provide syntax, semantics, and some examples. Furthermore, we analyze in some detail the behaviour of mobile systems, as well as their relevant computational properties, and we show that they can be captured in terms of the considered ways of combining structures and logics. Next, we show how model checking procedures for combined logics can be synthesized from suitable combinations of model checkers for simpler component logics. We analyze the computational complexity of the resulting procedures and we report on a number of experiments with implementations. It will turn out that, in contrast to combining deductive engines, combinations of model checking procedures are well behaved, even in the presence of interaction, thus supporting the general belief that modularity is easier to achieve in model checking than in theorem proving approaches (Halpern and Vardi, 1991). In particular, complexity upper bounds for model checking transfer from the components to the combination.

This paper is organized as follows. In Section 2 we introduce the three ways of combining logics: temporalization, independent combination, and join. In Section 3 we discuss the application of logic combination techniques to the specification and verification of mobile systems. In particular, we investigate the relationships between the ambient calculus (one of the major tools proposed in the literature to formalize mobile systems) and the proposed combined framework. In Section 4 we develop combined model checking procedures for the three modes of combination on which we focus, and we prove that they are terminating, sound, and complete. Section 5 contains results on computational complexity,

while Section 6 reports on our experiments with implementations. In Section 7 we discuss the strength and limitations of the combined approach, with a short comparison with related work. Section 8 provides an assessment of the work and it outlines further research directions.

2. Combining logics

Various forms of logic combination have been proposed in the literature. Temporalization, independent combination (or fusion), and join (or product) are probably the most popular ones as well as the ones that have been studied most extensively (Fine and Schurz, 1996; Finger and Gabbay, 1992, 1996; Gabbay et al., 2003; Gabbay and Shehtman, 1998; Kracht and Wolter, 1991; Spaan, 1993). They have been successfully applied in several areas, including databases (Finger, 1992, 1994; Finger and Reynolds, 2000) and artificial intelligence (Baader and Ohlbach, 1995; Engelfriet, 1996; Fagin et al., 1995; Halpern and Vardi, 1989; Meyer and van der Hoek, 1995; Wolter and Zakharyaschev, 1998). In this paper, we focus on the application of combined (temporal) logics to the specification and verification of (mobile) reactive systems.

In this section, we introduce syntax and semantics for temporalization, independent combination, and join. We will use the following general definition of temporal logic. The language of *temporal logic* is based on a set $\mathcal{P} = \{P, Q, \ldots\}$ of proposition letters and extends that of propositional logic with a set $OP = \{\mathbf{O}_1^{i_1}, \ldots, \mathbf{O}_n^{i_n}\}$ of *temporal operators* with arities $i_1, \ldots i_n$, respectively. The language of temporal logic is the smallest set of formulas generated by the following rules:

(P1) every proposition letter $P \in \mathcal{P}$ is a formula;
(P2) if ϕ, ψ are formulas, then $\phi \wedge \psi$ and $\neg\phi$ are formulas;
(P3) if $\mathbf{O}_j^{i_j} \in OP$ and $\phi_1, \ldots, \phi_{i_j}$ are formulas, then $\mathbf{O}_j^{i_j}(\phi_1, \ldots, \phi_{i_j})$ is a formula.

Boolean connectives $\vee$, $\rightarrow$, and $\leftrightarrow$ are defined as usual. Moreover, `true` abbreviates $P \vee \neg P$, for some fixed $P \in \mathcal{P}$, and `false` stands for $\neg$`true`. A *frame* for temporal logic is a pair $(W, \mathcal{R})$, where W is a nonempty set of *worlds*, or *states*, and $\mathcal{R}$ is a set of *accessibility relations* on W. We restrict ourselves to *binary* accessibility relations on W. A *model* for temporal logic is a Kripke structure $(W, \mathcal{R}, V)$, where $(W, \mathcal{R})$ is a frame and $V : W \rightarrow 2^{\mathcal{P}}$ is a *valuation function* mapping states into sets of proposition letters. The semantics of temporal logic extends that of propositional logic with clauses for the temporal operators in OP. For $n \geq 1$, formulas of *n-dimensional* temporal logics are evaluated at n-dimensional tuples of points (equivalently, formulas of n-dimensional temporal logics can be embedded in classical logics with n-ary predicates). Examples of one-dimensional temporal logics are Propositional Linear Temporal Logic (PLTL) and Computation Tree Logic (CTL and CTL*) (Emerson, 1990). Given an arbitrary logic $\mathbf{L}$, we use $\mathcal{L}_\mathbf{L}$ and $\mathsf{K}_\mathbf{L}$ to denote the language and the set of models of $\mathbf{L}$, respectively. We write $OP(\mathbf{L})$ to denote the set of operators of $\mathbf{L}$ different from Boolean ones.

Temporalization is the simplest of the three ways of combining logics that we consider; it only allows the two component languages to interact in a very restricted way (Finger

and Gabbay, 1992). Let $\mathbf{T}$ be a temporal logic and $\mathbf{L}$ be an arbitrary logic. For simplicity, we assume $\mathbf{L}$ to be an extension of propositional logic. We partition the set of $\mathbf{L}$-formulas into *Boolean combinations* $BC_{\mathbf{L}}$ and *monolithic formulas* $ML_{\mathbf{L}}$: α belongs to $BC_{\mathbf{L}}$ if its outermost operator is a Boolean connective; otherwise it belongs to $ML_{\mathbf{L}}$. We constrain the set $OP(\mathbf{T}) \cap OP(\mathbf{L})$ to be empty.

Definition 2.1 (Temporalization—Syntax). The language $\mathcal{L}_{\mathbf{T}(\mathbf{L})}$ of the *temporalization* $\mathbf{T}(\mathbf{L})$ of $\mathbf{L}$ by means of $\mathbf{T}$ over the set of proposition letters $\mathcal{P}$ is obtained by replacing the following atomic formation rule of $\mathcal{L}_{\mathbf{T}}$:

Every proposition letter $P \in \mathcal{P}$ is a formula,

by the following rule:

Every monolithic formula $\alpha \in \mathcal{L}_{\mathbf{L}}$ is a formula.

Notice that 'by construction' proposition letters occurring in a $\mathbf{T}(\mathbf{L})$ formula belong to $\mathcal{L}_{\mathbf{L}}$.

A *model* for $\mathbf{T}(\mathbf{L})$ is a triple $(W, \mathcal{R}, g)$, where $(W, \mathcal{R})$ is a frame for $\mathbf{T}$ and $g : W \to \mathsf{K}_{\mathbf{L}}$ a total function mapping worlds in W to models for $\mathbf{L}$.

Definition 2.2 (Temporalization—Semantics). Given a model $\mathcal{M} = (W, \mathcal{R}, g)$ and a state $w \in W$, the semantics of the temporalized logic $\mathbf{T}(\mathbf{L})$ is obtained by replacing the following semantic clause for proposition letters of $\mathbf{T}$:

$\mathcal{M}, w \models P$ iff $P \in V(w)$, whenever $P \in \mathcal{P}$,

by the following clause:

$\mathcal{M}, w \models \alpha$ iff $g(w) \models_{\mathbf{L}} \alpha$, whenever $\alpha \in ML_{\mathbf{L}}$.

Notice that, whenever the nested logic $\mathbf{L}$ is a modal or temporal logic, its formulas are obviously evaluated with respect to a given world.

As an example, consider the temporalization of PLTL_2 by means of PLTL_1, where PLTL_1 (resp. PLTL_2) is the propositional linear temporal logic PLTL over $\mathcal{P}$ with the temporal operators $\mathbf{X}$ (*next*) and $\mathbf{U}$ (*until*) renamed as $\mathbf{X}_1$ and $\mathbf{U}_1$ (resp. $\mathbf{X}_2$ and $\mathbf{U}_2$). The language of $\mathrm{PLTL}_1(\mathrm{PLTL}_2)$ is the smallest set of formulas generated by the following rules:

(P1) any monolithic formula in ML_{PLTL_2} is a formula;
(P2) if p, q are formulas, then $p \wedge q$ and $\neg p$ are formulas;
(P3) if p, q are formulas, then $\mathbf{X}_1 p$ and $p \mathbf{U}_1 q$ are formulas.

The formula $\mathbf{X}_1\mathbf{X}_2 P$ is a $\mathrm{PLTL}_1(\mathrm{PLTL}_2)$-formula, while $\mathbf{X}_2\mathbf{X}_1 P$ is not. We interpret both PLTL_1 and PLTL_2 over $\mathcal{P}$-labeled infinite sequences, that is, Kripke structures $(\mathbb{N}, <, V)$,

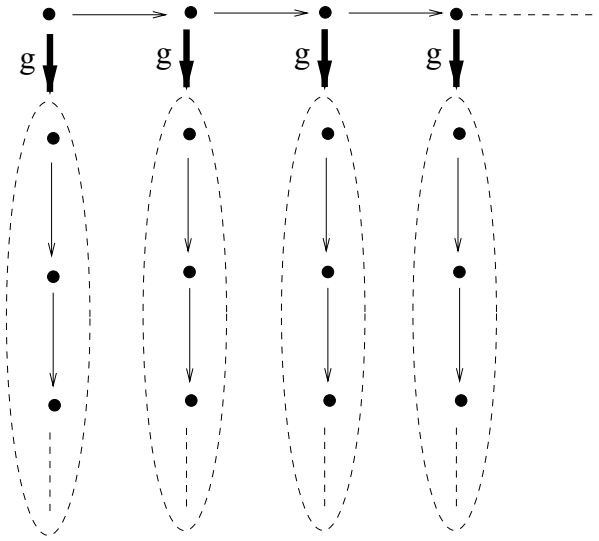

Figure 1. A temporalized model for PLTL(PLTL).

where $\mathbb{N}$ is the set of natural numbers, $<$ is the usual ordering relation over natural numbers, and $V : \mathbb{N} \rightarrow 2^{\mathcal{P}}$ is a valuation function. A temporalized model for $\text{PLTL}_1(\text{PLTL}_2)$ is a triple $(\mathbb{N}, <, g)$, where g maps natural numbers into $\mathcal{P}$-labeled infinite sequences. A (portion of an) unlabeled model for $\text{PLTL}_1(\text{PLTL}_2)$ is depicted in figure 1. For $i \in \{1, 2\}$, let $\models_i$ be the semantic relation of PLTL_i. Let $\mathcal{M} = (\mathbb{N}, <, g)$ be a temporalized model for $\text{PLTL}_1(\text{PLTL}_2)$ and $i \in \mathbb{N}$. The semantic relation of $\text{PLTL}_1(\text{PLTL}_2)$, denoted $\models_{1(2)}$, is defined as follows:

$$
\begin{array}{ll}
\mathcal{M}, i \models_{1(2)} \alpha & \text{iff } g(i), 0 \models_2 \alpha, \text{ whenever } \alpha \in ML_{\text{PLTL}_2} \\
\mathcal{M}, i \models_{1(2)} \phi \wedge \psi & \text{iff } \mathcal{M}, i \models_{1(2)} \phi \text{ and } \mathcal{M}, i \models_{1(2)} \psi \\
\mathcal{M}, i \models_{1(2)} \neg\phi & \text{iff it is not the case that } \mathcal{M}, i \models_{1(2)} \phi \\
\mathcal{M}, i \models_{1(2)} \phi \mathbf{U}_1 \psi & \text{iff } \mathcal{M}, j \models_{1(2)} \psi \text{ for some } j \geq i \text{ and} \\
 & \quad \mathcal{M}, k \models_{1(2)} \phi \text{ for every } i \leq k < j; \\
\mathcal{M}, i \models_{1(2)} \mathbf{X}_1 \psi & \text{iff } \mathcal{M}, i + 1 \models_{1(2)} \psi.
\end{array}
$$

The *independent combination* of two logics puts together the expressive power of the two component logics in an unrestricted way (Finger and Gabbay, 1996). Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two temporal logics defined over the same set of proposition letters $\mathcal{P}$, with $OP(\mathbf{T_1}) \cap OP(\mathbf{T_2}) = \emptyset$.

Definition 2.3 (Independent combination—Syntax). The language $\mathcal{L}_{\mathbf{T_1} \oplus \mathbf{T_2}}$ of the *independent combination* $\mathbf{T_1} \oplus \mathbf{T_2}$ of $\mathbf{T_1}$ and $\mathbf{T_2}$ over $\mathcal{P}$ is obtained by taking the union of the formation rules for $\mathbf{T_1}$ and $\mathbf{T_2}$.

To define the semantics of $\mathbf{T_1} \oplus \mathbf{T_2}$, some auxiliary notions are needed. Given a binary relation R, we denote by R^* its transitive closure and by R^{-1} its converse. Let $(W, \mathcal{R})$ be a frame. A *connected component* $(W', \mathcal{R}')$ of $(W, \mathcal{R})$ is a frame such that (i) $\emptyset \neq W' \subseteq W$ and $\mathcal{R}' = \{R|_{W'} \mid R \in \mathcal{R}\}$, (ii) $(W', \mathcal{R}')$ is *connected*, i.e., for every u and v in W', with $u \neq v$, we have $(u, v) \in (\bigcup\{(R \cup R^{-1}) \mid R \in \mathcal{R}\})^*$, and (iii) $(W', \mathcal{R}')$ is *maximal*, i.e., there is not a connected component $(W'', \mathcal{R}'')$ with $W' \subset W''$. Notice that an *isolated* point is a connected component. A *model* for the combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$ is a 4-tuple $(W, \mathcal{R}_1, \mathcal{R}_2, V)$, where the connected components of $(W, \mathcal{R}_1, V)$ are in $\mathsf{K}_{\mathbf{T_1}}$ (models for $\mathbf{T_1}$), the connected components of $(W, \mathcal{R}_2, V)$ are in $\mathsf{K}_{\mathbf{T_2}}$ (models for $\mathbf{T_2}$), and W is the (not necessarily disjoint) union of the sets of worlds that constitute each connected component. Finally, $V : W \rightarrow 2^{\mathcal{P}}$ is a valuation function.

Definition 2.4 (Independent combination—Semantics). The semantics of the independently combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$ is obtained by taking the union of the semantic clauses for $\mathbf{T_1}$ and $\mathbf{T_2}$.

As an example, we consider the independent combination of PLTL_1 and PLTL_2 over $\mathcal{P}$. The language of $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ is the smallest set of formulas generated by the following rules:

(P1) any proposition letter $P \in \mathcal{P}$ is a formula;
(P2) if p, q are formulas, then $p \wedge q$ and $\neg p$ are formulas;
(P3) if p, q are formulas, then $\mathbf{X}_1 p$, $\mathbf{X}_2 p$, $p\mathbf{U}_1 q$, and $p\mathbf{U}_2 q$ are formulas.

Unlike the case of $\mathrm{PLTL}_1(\mathrm{PLTL}_2)$, both $\mathbf{X}_1\mathbf{X}_2 P$ and $\mathbf{X}_2\mathbf{X}_1 P$ are $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$-formulas. An independently combined model for $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ is a quadruple $(W, <_1, <_2, V)$, where the connected components of $(W, <_1, V)$ and those of $(W, <_2, V)$ are $\mathcal{P}$-labelled infinite sequences. A (portion of an) unlabeled model for $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ is depicted in figure 2. We define two binary predicates $\mathtt{succ}_1$ and $\mathtt{succ}_2$ over W such that $\mathtt{succ}_1(w, v)$ if and only if $w <_1 v$ and there is no $z \in W$ such that $w <_1 z <_1 v$, and similarly for $\mathtt{succ}_2(w, v)$. Let $\mathcal{M} = (W, <_1, <_2, V)$ be a model for $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ and $w \in W$. The semantics of $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ is defined as follows:

$\mathcal{M}, w \models P$ iff $P \in V(w)$, for all $P \in \mathcal{P}$
$\mathcal{M}, w \models \phi \wedge \psi$ iff $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$
$\mathcal{M}, w \models \neg\phi$ iff it is not the case that $\mathcal{M}, w \models \phi$
$\mathcal{M}, w \models \phi\mathbf{U}_1\psi$ iff $\mathcal{M}, v \models \psi$ for some $v \geq_1 w$ and $\mathcal{M}, z \models \phi$ for every $w \leq_1 z <_1 v$;
$\mathcal{M}, w \models \mathbf{X}_1\psi$ iff $\mathcal{M}, v \models \psi$ and $\mathtt{succ}_1(w, v)$;
$\mathcal{M}, w \models \phi\mathbf{U}_2\psi$ iff $\mathcal{M}, v \models \psi$ for some $v \geq_2 w$ and $\mathcal{M}, z \models \phi$ for every $w \leq_2 z <_2 v$;
$\mathcal{M}, w \models \mathbf{X}_2\psi$ iff $\mathcal{M}, v \models \psi$ and $\mathtt{succ}_2(w, v)$.

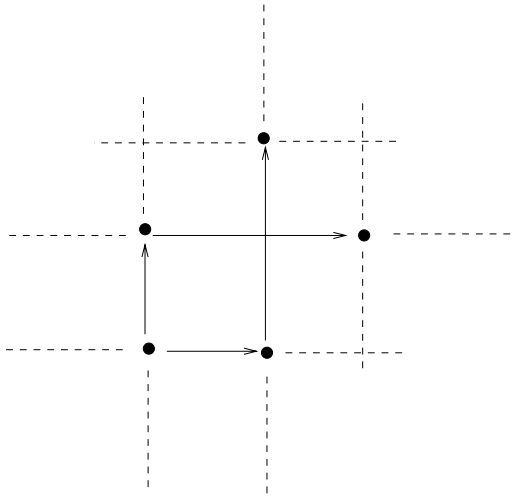

Figure 2. An independently combined model for PLTL $\oplus$ PLTL.

It is worth noting that the formula $\mathbf{X}_1\mathbf{X}_2 P \leftrightarrow \mathbf{X}_2\mathbf{X}_1 P$, that allows one to commute the two successor operators, is *not valid* in $\text{PLTL}_1 \oplus \text{PLTL}_2$, since, given a model $\mathcal{M} = (W, <_1, <_2, V)$ and $w \in W$, $\mathtt{succ}_2(\mathtt{succ}_1(w))$ is not necessarily equal to $\mathtt{succ}_1(\mathtt{succ}_2(w))$. This is the case, for instance, of the model described in figure 2.

The relationships between independent combination and temporalization deserve to be briefly analyzed. In particular, it is worth explaining why independent combination cannot be viewed as an arbitrary nesting of temporalizations. Let $L_0 = \mathbf{T_1}$, $\bar{L}_0 = \mathbf{T_2}$ and, for $i > 0$, let $L_i = \mathbf{T_1}(\bar{L}_{i-1})$ and $\bar{L}_i = \mathbf{T_2}(L_{i-1})$. Since any formula of the independent combination $\mathbf{T_1} \oplus \mathbf{T_2}$ is a formula of a temporalized logic L_i or $\bar{L}_i$, for some $i \geq 0$, one can be led to think of reducing the satisfiability/model checking problems for independent combination to the same problems for temporalization. However, such a reduction would be incorrect. Roughly speaking, the reason is that the semantics of independent combination and (nested) temporalization radically differ from each other. A model for independent combination is a *flat* structure, that is, all the (connected) components have the same nesting level. A formula may 'visit' a component, leave it for a while, and later come back. This is not the case with temporalization. A model for temporalization is a *nested* structure where different components have different nesting levels. Once a formula has 'entered' a component, it may only proceed forward, visiting the reached component or a nested component of it, but it can never come back. Consider the following example. Take PLTL over $\mathcal{P} = \{P\}$ interpreted over $\mathcal{P}$-labeled finite sequences, that is, Kripke structures $(\mathbb{I}, <, V)$, where $\mathbb{I}$ is an initial segment $\{0, 1, \ldots, n\}$ of the natural numbers, $<$ is the usual ordering relation over natural numbers, and $V : \mathbb{I} \to 2^{\mathcal{P}}$ is a valuation function. Consider the $\text{PLTL}_1(\text{PLTL}_2(\text{PLTL}_1))$-formula $\varphi = \mathbf{F_1}(\mathbf{G_2}\mathbf{G_1}\neg P \wedge \mathbf{F_1} P)$ (for any arbitrary nesting of temporalizations $\mathbf{T_1}(\mathbf{T_2}(\ldots(\mathbf{T_n})\ldots))$, we constrain $OP(\mathbf{T_i}) \cap OP(\mathbf{T_{i+1}})$ to be empty, for $i = 1, 2, \ldots, n-1$). Such a formula is satisfiable in $\text{PLTL}_1(\text{PLTL}_2(\text{PLTL}_1))$, but it

is unsatisfiable in $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$. We first build a $\mathrm{PLTL}_1(\mathrm{PLTL}_2(\mathrm{PLTL}_1))$-model which satisfies φ. Let $\mathcal{M}_P$ be a PLTL-model consisting of a single point labeled with letter P and $\mathcal{M}_{\neg P}$ be a PLTL-model consisting of a single point labeled with no letter. Furthermore, let $\mathcal{M}_1 = (\{0\}, <, g_1)$ be a PLTL(PLTL) model such that $g_1(0) = \mathcal{M}_{\neg P}$ and $\mathcal{M}_2 = (\{0\}, <, g_2)$ be a PLTL(PLTL) model such that $g_2(0) = \mathcal{M}_P$. Finally, let $\mathcal{M} = (\{0, 1\}, <, g)$ be a PLTL(PLTL(PLTL))-model such that $g(0) = \mathcal{M}_1$ and $g(1) = \mathcal{M}_2$. It is easy to see that $\mathcal{M}$ is a model of φ (proposition letters are interpreted over the innermost frame). However, there is not a model for the independent combination $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ which satisfies φ. Indeed, suppose that $\mathcal{M}, w \models \varphi$, with $\mathcal{M} = (W, <_1, <_2, V)$ and $w \in W$. It follows that there exists v such that $w \leq_1 v$ and $\mathcal{M}, v \models \mathbf{G_2G_1}\neg P \wedge \mathbf{F_1}P$, that is, $\mathcal{M}, v \models \mathbf{G_2G_1}\neg P$ and $\mathcal{M}, v \models \mathbf{F_1}P$. Since $\mathcal{M}, v \models \mathbf{G_2G_1}\neg P$, for every z such that $v \leq_2 z$, we have that $\mathcal{M}, z \models \mathbf{G_1}\neg P$. In particular, we have that $\mathcal{M}, v \models \mathbf{G_1}\neg P$, which contradicts $\mathcal{M}, v \models \mathbf{F_1}P$.

Both the temporalization and the independent combination of a pair of n-dimensional temporal logics are n-dimensional temporal logics. For instance, formulas of $\mathrm{PLTL}_1(\mathrm{PLTL}_2)$ and $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$ are evaluated at a single node of a model, as PLTL_1 and PLTL_2 formulas. The distinctive feature of the combining method of *join* is that it produces higher-dimensional temporal logics by combining lower-dimensional ones. For notational simplicity, we assume component logics to be one-dimensional. Given two (one-dimensional) temporal logics $\mathbf{T_1}$ and $\mathbf{T_2}$, defined over the same set of proposition letters $\mathcal{P}$, with $OP(\mathbf{T_1}) \cap OP(\mathbf{T_2}) = \emptyset$, their join is defined as follows.

Definition 2.5 (Join—Syntax). The language $\mathcal{L}_{\mathbf{T_1} \otimes \mathbf{T_2}}$ of the *join* $\mathbf{T_1} \otimes \mathbf{T_2}$ of $\mathbf{T_1}$ and $\mathbf{T_2}$ over $\mathcal{P}$ is obtained by taking the union of the formation rules for $\mathbf{T_1}$ and $\mathbf{T_2}$.

Notice that the language of join coincides with that of independent combination. However, their semantics are quite different. A *model* for $\mathbf{T_1} \otimes \mathbf{T_2}$ is a 5-tuple $(W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$, where $(W_1, \mathcal{R}_1)$ is a $\mathbf{T_1}$-frame, $(W_2, \mathcal{R}_2)$ is a $\mathbf{T_2}$-frame, and $V : W_1 \times W_2 \rightarrow 2^{\mathcal{P}}$ is a valuation mapping *pairs* of worlds to sets of proposition letters. In order to simplify the definition of the truth of a $\mathbf{T_1} \otimes \mathbf{T_2}$-formula φ, we introduce some auxiliary notions. Given a $\mathbf{T_1} \otimes \mathbf{T_2}$-formula $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_n)$, with $\mathbf{O} \in OP(\mathbf{T_1})$ (resp. $\mathbf{O} \in OP(\mathbf{T_2})$), we denote by $\varphi^{\uparrow_1}$ (resp. $\varphi^{\uparrow_2}$) the $\mathbf{T_1}$-formula (resp. $\mathbf{T_2}$-formula) obtained from φ by replacing every maximal subformula $\alpha \in ML_{\mathbf{T_2}}$ (resp. $\alpha \in ML_{\mathbf{T_1}}$) by a proposition letter P_α.

Definition 2.6 (Join—Semantics). The truth of a formula φ in a model $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ at (s_1, s_2), with $s_1 \in W_1$ and $s_2 \in W_2$, is inductively defined as follows. If $\varphi = P$, with $P \in \mathcal{P}$, $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \neg\varphi_1$, then $\mathcal{M}, s_1, s_2 \models_{\mathbf{T_1} \otimes \mathbf{T_2}} \varphi$ is defined in the standard way. If $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_n)$, with $\mathbf{O} \in OP(\mathbf{T_1})$ (resp. $\mathbf{O} \in OP(\mathbf{T_2})$), the definition of $\mathcal{M}, s_1, s_2 \models_{\mathbf{T_1} \otimes \mathbf{T_2}} \varphi$ is obtained from that of $\mathcal{M}, s_1 \models_{\mathbf{T_1}} \varphi^{\uparrow_1}$ (resp. $\mathcal{M}, s_2 \models_{\mathbf{T_2}} \varphi^{\uparrow_2}$) by replacing each occurrence of $\mathcal{M}, x$ by $\mathcal{M}, x, s_2$ (resp. $\mathcal{M}, s_1, x$) and by replacing each occurrence of the proposition letter P_α by the formula α.

Our definition of the join of logics is close to that of Finger and Gabbay (1996). In Gabbay and Shehtman (1998), a slightly different, but equivalent, construction is proposed: the *product* of modal logics.

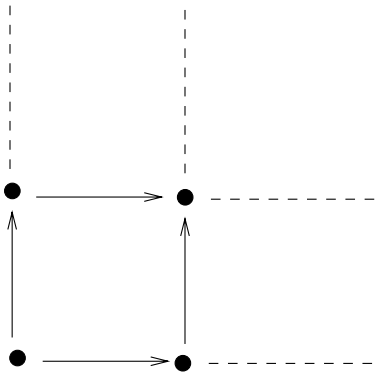

Figure 3. A joined model for PLTL ⊗ PLTL.

As an example, consider the join of PLTL_1 and PLTL_2 over $\mathcal{P}$. The language of $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$ is equal to the language of $\mathrm{PLTL}_1 \oplus \mathrm{PLTL}_2$. A joined model for $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$ is a 5-tuple $(\mathbb{N}, <, \mathbb{N}, <, V)$. A (portion of an) unlabeled model for $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$ is depicted in figure 3. Let $\mathcal{M} = (\mathbb{N}, <, \mathbb{N}, <, V)$ be a model for $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$, and i, j be two natural numbers. The semantics of $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$ is defined as follows.

$$
\begin{array}{ll}
\mathcal{M}, i, j \models P & \text{iff } P \in V(i, j), \text{ for all } P \in \mathcal{P} \\
\mathcal{M}, i, j \models \phi \wedge \psi & \text{iff } \mathcal{M}, i, j \models \phi \text{ and } \mathcal{M}, i, j \models \psi \\
\mathcal{M}, i, j \models \neg\phi & \text{iff it is not the case that } \mathcal{M}, i, j \models \phi \\
\mathcal{M}, i, j \models \phi \mathbf{U}_1 \psi & \text{iff } \mathcal{M}, r, j \models \psi \text{ for some } i \leq r \text{ and} \\
 & \quad \mathcal{M}, k, j \models \phi \text{ for every } i \leq k < r; \\
\mathcal{M}, i, j \models \mathbf{X}_1 \psi & \text{iff } \mathcal{M}, i+1, j \models \psi; \\
\mathcal{M}, i, j \models \phi \mathbf{U}_2 \psi & \text{iff } \mathcal{M}, i, r \models \psi \text{ for some } j \leq r \text{ and} \\
 & \quad \mathcal{M}, i, k \models \phi \text{ for every } j \leq k < r; \\
\mathcal{M}, i, j \models \mathbf{X}_2 \psi & \text{iff } \mathcal{M}, i, j+1 \models \psi;
\end{array}
$$

It is easy to prove that the formula $\mathbf{X}_1\mathbf{X}_2 P \leftrightarrow \mathbf{X}_2\mathbf{X}_1 P$, that allows one to commute the two successor operators, is *valid* in $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$. The grid structure of joined models is represented in figure 3. Figure 2 shows that this is not the case with independent models.

Logic combinations are also related to what is known as the *theory of institutions*, introduced by Burstall and Goguen in the late 1970s (Goguen and Burstall, 1992) to formally capture the informal notion of a logical system viewed from a model-theoretic perspective. As such, the theory is primarily concerned with the general principles underlying the development of different versions of the algebraic specification paradigm. Morphisms of institutions are translations among logical systems, and we believe that it should be possible to capture our three methods of combining logics within the abstract setting of the theory of institutions, although we are not aware of any publications on this. Important related work

on the interface of logic combinations and institutions is given in Sernadas et al. (1997), where the authors work out the category-theoretic counterpart of a method of combination that is similar to what we call *join*.

We conclude the section by summarizing the main transfer results given in the literature for the combining methods introduced above. First of all, the independent combination of two decidable normal polyadic polymodal modal logics is decidable (Wolter, 1998). The same result holds for modal logics with the *converse* operator interpreted over transitive frames (Wolter, 1995, 1996, 1997). Moreover, the properties of finite axiomatizability, soundness, and completeness transfer through the independent combination of monomodal logics (Fine and Schurz, 1996; Kracht and Wolter, 1991). As for one-dimensional temporal logics, it is known that PPLTL(PPLTL) and PPLTL $\oplus$ PPLTL are decidable, and that they admit a sound and complete finite axiomatization (we denote by PPLTL the extension of Propositional Linear Temporal Logic with Past Operators) (Finger and Gabbay, 1992, 1996). In the case of join, things get much harder. For instance, the modal logic $S5$ is NP-complete, $S5 \otimes S5$ ($S5^2$ for short) is NEXPTIME-complete (Marx, 1999), and $S5^3$ is undecidable (it does not even have the finite model property) (Kurucz, 2000). Moreover, PLTL $\otimes$ K_m is decidable, but PLTL $\otimes$ PLTL is not even recursively enumerable (Wolter, 2000).

Now that we have introduced our main mathematical apparatus, and before we examine further algorithmic aspects of combining methods, let us take a closer look at a domain where we can put it to use.

3. An application to mobile systems

In this section, we focus our attention on mobile systems. We first present the distinctive features of such systems and then we illustrate the main characteristics of the *ambient calculus*, one of the major formal tools for modeling mobile systems. Next, we show how the proposed combined approach can actually be used to deal with mobile systems. We present a combined framework, pairing temporal and hybrid logics, that allows us to model the temporal evolution of mobile system states as well as the spatial distribution of localities. The expressive power of the proposed framework is exemplified through the formalization of a number of meaningful properties of ambients. A simple, but paradigmatic, case study about banks, clients, and security doors follows.

3.1. Mobile systems and mobile ambients

Roughly speaking, a *mobile system* is a program that exploits a wide area computational infrastructure like the World-Wide-Web. A concrete example is the Universal Mobile Telecommunications System (UMTS), the third generation of mobile phones that is going to replace GSM. As discussed in Cardelli (1999), the Web violates many familiar assumptions about the behaviour of distributed systems. In particular, three phenomena that remain largely hidden in local area network architectures become readily observable on a wide area network:

(i) *Virtual locations*. Because of the presence of potential attackers, barriers are erected between mutually distrustful administrative domains. Therefore a program must be aware of where it is, and of how to move and to communicate between different domains;
(ii) *Physical locations*. On the planet-size structure, the speed of light becomes tangible and induces a notion of physical locations and physical distance between locations;
(iii) *Bandwidth fluctuations*. A global network is susceptible to unpredictable congestion and partitioning. Moreover, mobile devices may perceive bandwidth change as a consequence of physical movement.

In addition, on a wide area network like the Web there is no practical upper bound to communication delays. In particular, failures become indistinguishable from long delays, and thus undetectable. Failure recovery becomes indistinguishable from intermittent connectivity.

These phenomena influence the basic building blocks of computation for mobile systems, like the computational model, the programming constructs, and the kind of programs one can write, and the verification and validation processes must thus be adapted accordingly.

In the nontrivial task of modeling mobility, the first step is to isolate the peculiar features of mobile systems. Five features that characterize mobile systems may be identified:

- *Dynamic connectivity*. The communication channels are not fixed once and for all, but they can be established and released dynamically. As a consequence, the topology, or structure, of the system is not static and may change over time.
- *Distribution*. In a distributed system processes may occupy different locations and the interaction between processes depends on their relative positions. For example, the United States and the European Union are both enclosed by a political boundary that regulates the movement of people and software. Within a political boundary, private companies and public agencies and institutions may further regulate the flow of people and software across their doors. Special permissions (like passports or certificates) are required for people and for software in order to cross these boundaries.
- *Locality*. By locality we mean distribution-awareness: a process has some notion of the location it occupies, and of the existence of different locations, in an absolute or relative sense. If capable, it can exit its domain and enter different locations, or even dissolve a boundary enclosing a location.
- *Security*. Virtual boundaries are erected to protect special domains from possible attackers. These preclude the unfettered execution of possibly dangerous actions across domains. Examples are firewalls surrounding a major company and the encryption of a piece of data.
- *Mobility*. This is surely the most peculiar feature of mobile systems. In its general sense, there are two different notions of mobility. The first, *mobile computation*, has to do with virtual mobility, and concerns mobile code that moves between devices. An example of mobile software is the execution of an applet in Java. The second, *mobile computing*, has to do with physical mobility, and concerns computation that is carried out in mobile devices. Examples of mobile hardware are a wireless laptop entering a building and a mobile phone entering a phone cell. While mobile computation is possible over a static

(although possibly flaky) network, mobile computing implies a dynamic change of the network over time.

To see how combinations of logics can capture mobile systems, it is useful to move to a more abstract, mathematical description of mobile systems. To this end, we take advantage of terminology and concepts from the ambient calculus. The ambient calculus (Cardelli and Gordon, 2000b) is a process calculus which has been developed to deal with the above-described features of mobility in an effective way. It focuses on *mobile computational ambients* (or *locations*), that is, on places where computations happens and that are themselves mobile. An ambient has the following distinctive characteristics:

- An *ambient* is a bounded and named place where computation happens. If we want to move computations easily we must be able to determine what parts should move. A boundary determines what is inside and what is outside the ambient, and therefore determines what moves. Examples of ambients are: a web page, a single data object, a file system, a laptop, a university, a country, a union of countries.
- Ambients can be nested within other ambients. For instance, a university is placed in a certain country, and may contain several laptops connected to the network. The ambient structure may be organized as a graph in which nodes are ambients and edges represent the subambient relation. In the example of figure 4, the ambient a contains two ambients b and c, and the latter contains the ambient d.
- Each ambient has a collection of local running processes. A local process of an ambient is a process that is contained in an ambient, but not in any of its subambients. The top level local processes have direct control on the ambient and, in particular, they can instruct the ambient to move. In contrast, the local processes of a subambient have no direct control on the parent ambient. In figure 4, process P_1 is local to ambient a, P_2 is local to b, P_3 is local to c, and P_4 is local to d.
- Each ambient may move as a whole with all its subcomponents. An ambient, instructed by its local processes, may enter another ambient, exit the parent ambient, and open, that is, dissolve, an ambient. In the example depicted in figure 5, top part, ambient a, containing the local process P and the subfolder Q, enters ambient b, whose contents are represented by R, as a consequence of the execution of its local instruction *in* b. The result of the operation is that ambient a becomes a subambient of ambient b. Conversely, in the example depicted in figure 5, middle part, ambient a exits ambient b, as a consequence

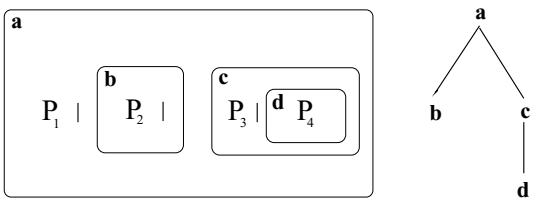

Figure 4. An ambient (left side) and its structure (right side).

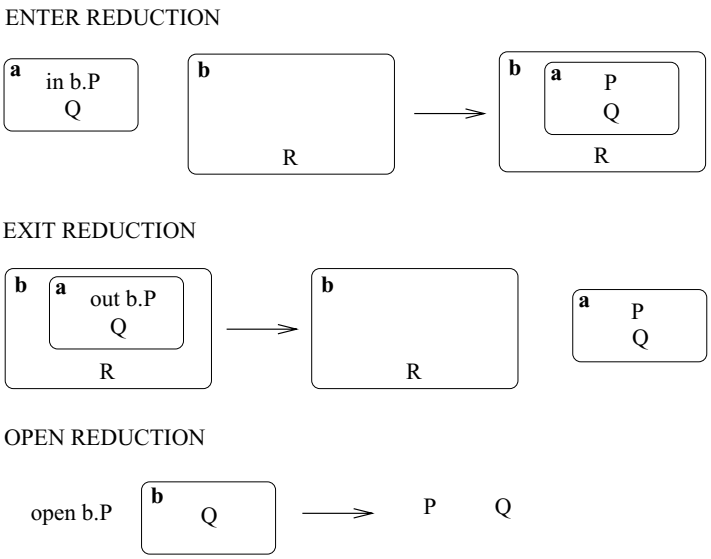

Figure 5. The movement and dissolution reductions.

of the execution of its local instruction *out b*. The result of the operation is that ambient *a* becomes a sibling of *b*. Finally, in the example depicted in figure 5, bottom part, the local process opens ambient *b*, as a consequence of the execution of its local instruction *open b*. The result is that the boundary of the ambient *b* is discarded, but its content is spilled in the place where the ambient *b* used to be.

- Processes communicate through the exchange of messages. A process may output a message, and another process may read the message, discard it, and proceed with the acquired knowledge. Typical contents of messages are ambient names and capabilities, that is, instructions to enter, exit, and open an ambient.

How can we use combined logics to reason about processes specified in the ambient calculus? A modal logic called *ambient logic*, based on the ambient calculus, has already been devised for specifying properties of mobile computations (Cardelli and Gordon, 2000a). A brief look at this logic will help us to identify the key ingredients of the combined logic we are looking for. Ambient logic is a polymodal logic which features two different sets of modalities: *temporal modalities*, to describe the temporal evolution of processes in the system, and *spatial modalities*, to capture the spatial distribution of processes among different locations. The *combination* of these modalities allows one to talk about time and space in the same formula and to express properties like: "eventually the agent will go away", and "there will always be an agent called *A* here".

The decidability and complexity of the model checking problem for formulas in the ambient logic against processes in the ambient calculus have been analyzed in Cardelli and Gordon (2000a), Charatonik et al. (2001) and Charatonik and Talbot (2001). In particular, Cardelli and Gordon (2000a) describe a simple algorithm for model checking the fragment

of the calculus devoid of replication and restriction against the fragment of the logic devoid of composition adjunct and revelation. Charatonik et al. (2001) present a PSPACE model checker for the above fragment and show the problem to be PSPACE-complete. Charatonik and Talbot (2001) show that either including replication in the calculus or composition adjunct in the logic leads to undecidability. Moreover, they extend the algorithm presented in Charatonik et al. (2001) to include restriction in the calculus and revelation in the logic, while preserving its PSPACE complexity.

Observe that the complexity results just mentioned are obtained by restricting the full logical language of ambient logic to fragments that can be managed algorithmically. As we will shortly see, with our method of combining logics we follow a different, bottom-up, strategy.

3.2. Mobile systems, hybrid logics, and combined logics

From a technical point of view, we can isolate two main ingredients of mobility: the notion of location and the change of spatial configurations of locations over time. In this section we show that the notion of location can be naturally expressed in hybrid logics, and that the evolution of spatial configurations of locations can be captured by means of a suitable combination of temporal and hybrid logics.

Modal and temporal logics have been successfully used as specification languages in the model checking task (Clarke and Schlingloff, 2001; Emerson, 1990; Pnueli, 1977). They are algorithmically well-behaved and mathematically natural fragments of classical logics. However, from a modeling point of view something crucial is missing in propositional modal and temporal logics: they lack mechanisms for *naming* individual locations and for dynamically creating new names for locations. *Hybrid logics* (Areces et al., 1999, 2000, 2001; Blackburn, 2000; HyLo: http://www.hylo.net) are extensions of modal logics that support reference to locations by introducing a new type of atomic formulas, called *nominals*. Syntactically, nominals behave like ordinary propositional variables, but they are names, which are true at exactly one location in any model. Hence, if i is a nominal, the formula i holds if and only if the current location is called i. But nominals are just the first ingredient of hybrid logics. Hybrid languages contain the *at operator* $@_i$ which gives random access to the location named by i: the formula $@_i\phi$ holds if and only if ϕ holds at the location named by i. They may also include the *downarrow binder* $\downarrow x.$ which creates a brand new name x and assigns it to the current location. The formula $\downarrow x.\phi$ holds if and only if ϕ holds whenever the current location has been named by x. The operator @ combines naturally with $\downarrow$: $\downarrow$ stores the current location, and @ retrieves the information stored. For instance, the formula

$$\downarrow x.\Diamond\downarrow y.\beta \wedge @_x\Box(\Diamond y \rightarrow \alpha),$$

interpreted over the domain of days, states that there exists a day in the future in which β is true and α is true from tomorrow until that future day. The model checking problem for hybrid logics has been investigated in Franceschet and de Rijke (2003) and Franceschet et al. (2003).

It is easy to capture the notion of location in hybrid logic: it corresponds to its fundamental notion of nominal. As far as model checking is concerned, nominals may not necessarily be unique, that is, there may be different locations with the same name. For instance, names for ambients are not assumed to be unique in the ambient calculus. In this case, the meaning of formula $@_i\phi$ is that ϕ is true at *some* location called i.

The evolution of spatial configurations of locations over time can easily be encoded in the combined framework we described in this paper. In fact, mobile systems can be embedded into a number of frameworks, which differ from each other in the expressive power and in the level of interaction between spatial and temporal components. In the following, we focus on a sample framework based on temporalization. A *mobile system state* describes the structure of ambients and, for each ambient, the program state of its local process. A mobile system state can be represented as a labeled rooted graph: a node represent an ambient and is labeled with (i) the ambient name and (ii) the program state of its local process. The edges represent the subambient relation. The root is the desktop ambient, that is the outermost location. This graph is not necessarily a tree; for instance, it may happen that the same ambient is contained in two distinct parent ambients. In such cases there is a node in the graph with two different parents. This labeled graph may be seen as a model for hybrid logic, where the ambient names are nominals and the local program states are sets of atomic propositions.

A mobile system state may evolve in different ways: a local process may execute an instruction, possibly modifying its local program state, and an ambient may decide to move, changing the topology of the ambient structure. Finally, a local process may open another ambient. In the latter case both the local program state and the topology of the structure are modified. We can model this evolution by temporalizing the hybrid model describing the mobile system state.

Formally, the operational behavior of a mobile system may be modeled by means of a temporalized model (W, R, g, w_0). The outer frame (W, R) models temporal evolution of mobile system states, and $w_0 \in W$ is the initial state. For every $w \in W$, the hybrid model $g(w)$ is a labeled rooted graph representing the mobile system state associated with w. The situation is depicted in figure 6, in which we label nodes with pairs like (a, s), where a is the ambient name and s denotes the local program state. The first column of the figure represents an enter reduction (ambient b enters into ambient c), the second column represents an exit reduction (ambient d exits from ambient c), and the third column represents an open reduction (a process local to ambient a opens ambient b, and the local program state of ambient c is updated).

A temporalized logic $\mathbf{T}_1(\mathbf{T}_2)$ can be used to specify mobile temporal requirements. Temporal requirements are captured by formulas of the temporal logic $\mathbf{T}_1$, while spatial statements are expressed by formulas of the hybrid logic $\mathbf{T}_2$. A model checker for $\mathbf{T}_1(\mathbf{T}_2)$ can be obtained by composing model checkers for the component logics $\mathbf{T}_1$ and $\mathbf{T}_2$. For instance, let $\mathbf{T}_1$ be Computation Tree Logic CTL and $\mathbf{T}_2$ be the full hybrid logic with nominals and the operators $@$ and $\downarrow$. In the latter, we use $\Diamond$ and $\Box$ for the existential and universal modalities, and $\Diamond^*$ and $\Box^*$ for their reflexive and transitive closures.

In the following, we show how some meaningful properties of ambients, mixing spatial and temporal requirements, can be encoded in the resulting temporalized logic:

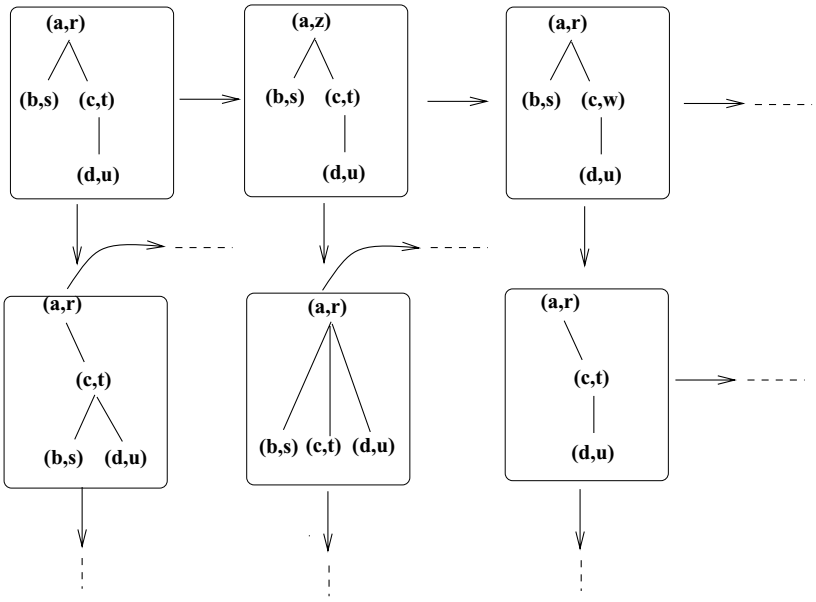

Figure 6. A temporalized model for a mobile system.

- The property "eventually ambient b will exit ambient a" is captured by the formula:

$$\mathbf{EF}@_a\Box^*\neg b$$

- The property "eventually ambient b will enter ambient a" is captured by the formula:

$$\mathbf{EF}@_a\Diamond^* b$$

- The property "eventually ambient a will be opened" is captured by the formula:

$$\mathbf{EF}\Box^*\neg a$$

- The property "if ambient b will enter ambient a, then eventually it will exit" is captured by the formula:

$$\mathbf{AG}(@_a\Diamond^* b \rightarrow \mathbf{AF}@_a\Box^*\neg b)$$

- The property "there will be always an ambient called a" is captured by the formula:

$$\mathbf{AG}\Diamond^* a$$

- The property "eventually the desktop will be clear (meaning, eventually the desktop will have no subambient)" is captured by the formula:

$$\mathbf{EF}\Box\mathtt{false}$$

- The property "ambient a will always have at most one subambient" captured by the formula:

$$\mathbf{AG}@_a(\Box\mathtt{false} \vee \downarrow x.\Diamond \downarrow y.@_x\Box y)$$

- The property "eventually there will be a shared ambient (meaning, eventually an ambient will have two parent ambients)" is captured by the formula:

$$\mathbf{EF} \downarrow r.\Diamond^* \downarrow x.@_r\Diamond^* \downarrow y.\neg@_x y \wedge @_x\Diamond u \wedge @_y\Diamond v \wedge @_u v$$

It is worth point out that the properties: "a is a black hole (meaning, everything that enters into a will not exit)" and "a is a white hole (meaning, everything that enters into a will eventually exit)" are beyond the scope of this framework.

3.3. A security protocol

We now make things more concrete by looking at a simple, but paradigmatic, example: a security protocol that a client has to follow to enter into a bank. The protocol is the following:

> To enter the *bank*, a *client* has to go first through a *security door*. When inside the door, the client has to ask for the permission to enter into the bank. The door may give or deny the permission. In the first case, the client enters the bank and processes the transaction. In the second case, the client may not enter the bank and is forced to exit the security door outside the bank. To exit the bank, the client has to go through the security door again and ask for the permission to exit. The door may give or deny the permission to exit. In the first case, the client exits the security door outside the bank, whereas in the second case the client remains inside the door until the permission to exit has been allowed.

We model this security protocol exploiting by the combined framework proposed in Section 3.2. We identify three ambients: the client, the security door and the bank. We assume that everything happens in an outermost desktop ambient, but we do not take care of the desktop ambient. The door ambient always encloses the bank ambient, while the client may be outside the door ambient, inside the door ambient but outside the bank ambient, or inside the bank ambient. We use nominals `C` for the client, `D` for the security door, and `B` for the bank. Moreover, we use propositions `ENTER?` (respectively, `EXIT?`) to denote that the client asked for the permission to enter (respectively, exit) the bank, `ENTER!` (respectively, `EXIT!`) to denote that the client obtained the permission to enter (respectively, exit) the bank, `NOENTER` (respectively, `NOEXIT`) to denote that the client did not obtain

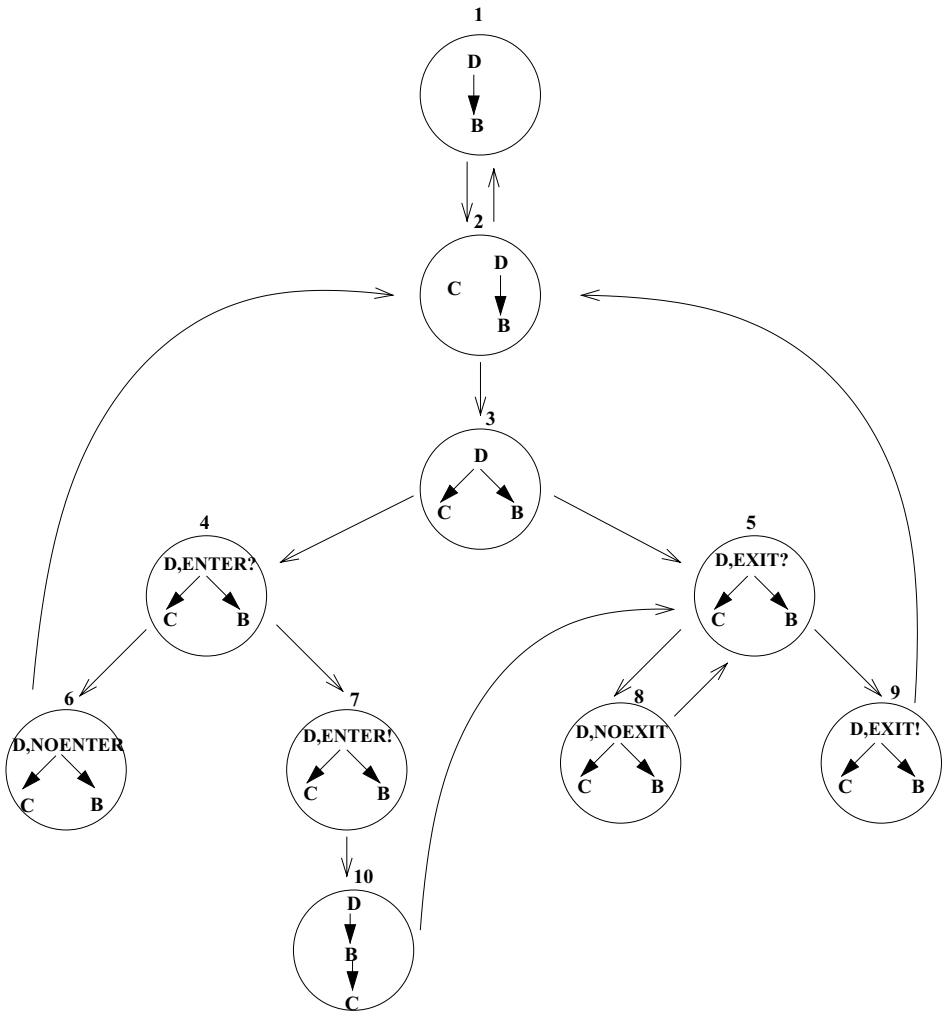

Figure 7. A model for the security protocol.

the permission to enter (respectively, exit) the bank. The temporalized model describing all possible scenarios of this security protocol is depicted in figure 7.

We specify the properties describing the security protocol in the temporalized logic CTL*(HL), where CTL* is Computation Tree Logic CTL* and HL is the basic Hybrid Logic containing nominals and the @ operator only (as before, we use $\Diamond$ and $\Box$ for the existential and universal modalities, and $\Diamond^*$ and $\Box^*$ for their reflexive and transitive closures, in the latter logic). First, we must specify that the client cannot enter the bank without receiving the permission to enter. This is written as follows:

$$\neg \mathbf{E}(@_D \neg \mathtt{ENTER!}\, \mathbf{U} @_B \Diamond C)$$

Moreover, the client cannot leave the security without receiving the permission to exit. This is encoded as follows:

$$\mathbf{AG}(@_B\Diamond C \rightarrow \neg\mathbf{E}(@_D\neg\mathtt{EXIT!}\,\mathbf{U}@_D\Box\neg C))$$

Furthermore, if the entrance is denied, the client has to leave the bank immediately. This is specified as follows:

$$\mathbf{AG}(@_D\mathtt{NOENTER} \rightarrow \mathbf{AX}@_D\Box\neg C)$$

Finally, if the exit is denied, the client has to remain inside the door until the permission to exit has been given. This is implemented as follows:

$$\mathbf{AG}(@_D\mathtt{NOEXIT} \rightarrow \mathbf{A}(\mathbf{G}@_D\Diamond C \vee @_D\Diamond C\mathbf{U}@_D\mathtt{EXIT!}))$$

Notice that the first three formulas are in CTL, while the last formula is in CTL*, but not in CTL. By taking the conjunction of the above four properties one obtains a specification that forces the client to follow the security protocol informally described above. One may verify whether a model satisfies this specification by combining a model checker for CTL* and a model checker for HL and by automatically checking the specification against the model. In particular, one may check that the model described in figure 7 actually satisfies the above four properties.

It is worth noticing that the client, the security door, and the bank may be metaphors for different scenarios. For instance, the client may be a program that has to interact with a protected system (the bank), and the security door may be a firewall that allows the interaction only to secure programs. Alternatively, the client may be a web user and the bank may be an encrypted file. The encryption protecting the file is played by the security door which may give or deny the encryption key to the web user.

4. Combined model checkers

In this section we define model checking procedures for temporalized, independently combined, and joined logics.

We first focus on the case of temporalization. The global model checking problem for a temporalized logic $\mathbf{T}(\mathbf{L})$ is defined as follows. Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a $\mathbf{T}(\mathbf{L})$-model. We say that $\mathcal{M}$ is *finite* if W and $\mathcal{R}$ are finite and, for every $w \in W$, $g(w)$ is finite. Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a finite $\mathbf{T}(\mathbf{L})$-model and ψ be a formula in $\mathcal{L}_{\mathbf{T}(\mathbf{L})}$. The *global* model checking problem for $\mathbf{T}(\mathbf{L})$ consists in checking whether there exists $w \in W$ such that $\mathcal{M}, w \models_{\mathbf{T}(\mathbf{L})} \psi$. We call model checker a program that solves the global model checking problem. Let ψ be a $\mathbf{T}(\mathbf{L})$-formula and $\mathrm{MML}_{\mathbf{L}}(\psi)$ be the set of *maximal* monolithic subformulas of ψ belonging to $\mathcal{L}_L$. We denote by $\psi^{\uparrow}$ the $\mathbf{T}$-formula obtained from ψ by replacing every formula $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$ by a new proposition letter P_α.

Let $\mathtt{MC}_{\mathbf{T}}$ and $\mathtt{MC}_{\mathbf{L}}$ be model checkers for $\mathbf{T}$ and $\mathbf{L}$, respectively. Given an appropriate model checking instance, these programs return `true` if the corresponding instance is

Function $\mathtt{MC}_{\mathbf{T(L)}}$
Input: a $\mathbf{T(L)}$-model $\mathcal{M} = (W, \mathcal{R}, g)$ and a formula $\psi \in \mathcal{L}_{\mathbf{T(L)}}$

compute $\mathrm{MML}_{\mathbf{L}}(\psi)$ and $\psi^{\uparrow}$
for every $w \in W$
  $V(w) = \emptyset$
for every $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$
  for every $w \in W$
    if $\mathtt{MC}_{\mathbf{L}}(g(w), \alpha) = \mathtt{true}$ **then**
      $V(w) = V(w) \cup \{P_\alpha\}$
return $\mathtt{MC}_{\mathbf{T}}((W, \mathcal{R}, V), \psi^{\uparrow})$

Figure 8. A model checking procedure for temporalized logics.

a "yes" instance, `false` otherwise. In figure 8, we present the pseudo-code of a model checker $\mathtt{MC}_{\mathbf{T(L)}}$ for $\mathbf{T(L)}$ that exploits $\mathtt{MC}_{\mathbf{T}}$ and $\mathtt{MC}_{\mathbf{L}}$. Given a model $\mathcal{M} = (W, \mathcal{R}, g)$ and a formula ψ, $\mathtt{MC}_{\mathbf{T(L)}}$ first computes the set $\mathrm{MML}_{\mathbf{L}}(\psi)$ and the formula $\psi^{\uparrow}$. Furthermore, for every world $w \in W$, it inizializes a suitable valuation function V to the empty set. Then, for every maximal monolithic formula $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$ and every world $w \in W$, it invokes the model checker $\mathtt{MC}_{\mathbf{L}}$ on the input consisting of the $\mathbf{L}$-model $g(w)$ and the $\mathbf{L}$-formula α. If the execution of $\mathtt{MC}_{\mathbf{L}}$ on w and α returns true, then the proposition letter P_α is added to $V(w)$. Finally, it calls the model checker $\mathtt{MC}_{\mathbf{T}}$ on the input consisting of the $\mathbf{T}$-model $(W, \mathcal{R}, V)$ and the $\mathbf{T}$-formula $\psi^{\uparrow}$, and it returns the output of this invocation. The following theorem can be easily proved.

Theorem 4.1 (*Termination, soundness, and completeness*)**.** *Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a finite model for $\mathbf{T(L)}$ and $\psi \in \mathcal{L}_{\mathbf{T(L)}}$. If $\mathtt{MC}_{\mathbf{L}}$ and $\mathtt{MC}_{\mathbf{T}}$ are terminating, sound and complete, then*:

1. Termination: $\mathtt{MC}_{\mathbf{T(L)}}$, *with input $\mathcal{M}$ and ψ, terminates, returning either* `true` *or* `false`;
2. Soundness: *if* $\mathtt{MC}_{\mathbf{T(L)}}$ *returns* `true` *on input $\mathcal{M}$ and ψ, then there exists $w \in W$ such that $\mathcal{M}, w \models_{\mathbf{T(L)}} \psi$;*
3. Completeness: *if* $\mathtt{MC}_{\mathbf{T(L)}}$ *returns* `false` *on input $\mathcal{M}$ and ψ, then, for every $w \in W$, $\mathcal{M}, w \not\models_{\mathbf{T(L)}} \psi$.*

We now give a general algorithm to solve the global model checking problem for the independently combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two temporal logics and $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ be a model for $\mathbf{T_1} \oplus \mathbf{T_2}$. We say that $\mathcal{M}$ is *finite* if $W, \mathcal{R}_1$, and $\mathcal{R}_2$ are finite, and, for every $w \in W$, $V(w)$ is finite. The global model checking problem for $\mathbf{T_1} \oplus \mathbf{T_2}$ is defined exactly as for $\mathbf{T(L)}$. Let $\mathcal{C}^1_{\mathcal{M}}$ and $\mathcal{C}^2_{\mathcal{M}}$ be the sets of connected components of $(W, \mathcal{R}_1)$ and $(W, \mathcal{R}_2)$, respectively. Since $\mathcal{M}$ is a model for $\mathbf{T_1} \oplus \mathbf{T_2}$, every connected component in $\mathcal{C}^1_{\mathcal{M}}$ (resp. $\mathcal{C}^2_{\mathcal{M}}$) is a model for $\mathbf{T_1}$ (resp. $\mathbf{T_2}$). Let $Sub(\varphi)$ be the set of subformulas of φ.

Procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$
Input: a $\mathbf{T_1} \oplus \mathbf{T_2}$-model $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ and a formula $\psi \in \mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$

compute $\mathcal{C}^1_{\mathcal{M}}$, $\mathcal{C}^2_{\mathcal{M}}$, and $Sub(\psi)$
for every $w \in W$ **let** $\overline{V}(w) = V(w)$
for every $i = 1, \ldots, |\psi|$
  for every $\varphi \in Sub(\psi) \cap \mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$ such that $|\varphi| = i$
    case on the form of φ
      $\varphi = P$, $P \in \mathcal{P}$: **skip**
      $\varphi = \varphi_1 \wedge \varphi_2$: **for every** $w \in W$
        if $(\varphi_1 \in V(w)$ **and** $\varphi_2 \in V(w))$ **then**
          $V(w) = V(w) \cup \{\varphi\}$; $\overline{V}(w) = \overline{V}(w) \cup \{P_\varphi\}$
      $\varphi = \neg\varphi_1$: **for every** $w \in W$
        if (**not** $\varphi_1 \in V(w)$) **then**
          $V(w) = V(w) \cup \{\varphi\}$; $\overline{V}(w) = \overline{V}(w) \cup \{P_\varphi\}$
      $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, $\mathbf{O} \in OP(\mathcal{L}_{T_i})$, $i \in \{1, 2\}$
      let $\varphi' = \varphi$
      for every $j \in \{1, \ldots c\}$ **replace** φ_j in φ' with P_{φ_j}
      for every $(U, \mathcal{S}) \in \mathcal{C}^i_{\mathcal{M}}$
        for every $u \in U$ **let** $V'(u) = \overline{V}(u)$
        $\mathtt{MC}_{\mathbf{T}_i}((U, \mathcal{S}, V'), \varphi')$
        for every $u \in U$
          if $\varphi' \in V'(u)$ **then**
            $V(u) = V(u) \cup \{\varphi\}$; $\overline{V}(u) = \overline{V}(u) \cup \{P_\varphi\}$

Figure 9. A model checking procedure for independently combined logics.

In order to develop a model checker for $\mathbf{T_1} \oplus \mathbf{T_2}$, it is convenient to view model checkers as *procedures* that receive a model $(W, \mathcal{R}, V)$ and a formula ψ as input, and that extend the valuation V (which maps a state to a set of proposition letters) to a valuation V' mapping states to sets of *subformulas* of ψ in the following way: for every subformula φ of ψ and every node w, $V'(w)$ contains φ iff φ is true at w in $(W, \mathcal{R}, V)$. Let $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ be model checkers for $\mathbf{T_1}$ and $\mathbf{T_2}$, respectively. In figure 9, we present the pseudo-code of a model checker for $\mathbf{T_1} \oplus \mathbf{T_2}$ that exploits the procedures $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$. Given a model $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ and a formula ψ, the procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$ first computes the sets of connected components $\mathcal{C}^1_{\mathcal{M}}$ and $\mathcal{C}^2_{\mathcal{M}}$ and the set of formulas $Sub(\psi)$. Then, it model checks formulas in $Sub(\psi)$ in increasing order with respect to their lengths, and accordingly it extends the valuation V. In particular, propositional cases are easily solved, while cases of formulas $\varphi \in Sub(\psi)$, with main operator in the language of T_i, $i \in \{1, 2\}$, are resolved by taking advantage of the corresponding model checker for $\mathbf{T}_i$. The following can be easily proved.

Theorem 4.2 (*Termination, soundness, and completeness*). *Let* $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ *be a finite model for* $\mathbf{T_1} \oplus \mathbf{T_2}$ *and* $\psi \in \mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$. *If* $\mathtt{MC}_{\mathbf{T_1}}$ *and* $\mathtt{MC}_{\mathbf{T_2}}$ *are terminating, sound,*

and complete, then:
1. Termination: *the procedure* $\mathtt{MC}_{\mathbf{T_1} \oplus \mathbf{T_2}}$, *with input* $\mathcal{M}$ *and* ψ, *terminates*;
2. Soundness *and* Completeness: *let* V *be the* (*extended*) *valuation function after termination of procedure* $\mathtt{MC}_{\mathbf{T_1} \oplus \mathbf{T_2}}$, *with input* $\mathcal{M}$ *and* ψ. *Then, for every subformula* φ *of* ψ *and every world* $w \in W$, $\varphi \in V(w)$ *if and only if* $\mathcal{M}, w \models_{\mathbf{T_1} \oplus \mathbf{T_2}} \varphi$.

Finally, we give a general algorithm that solves the global model checking problem for $\mathbf{T_1} \otimes \mathbf{T_2}$. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be temporal logics and $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a model for $\mathbf{T_1} \otimes \mathbf{T_2}$. We say that $\mathcal{M}$ is *finite* if W_1, W_2, $\mathcal{R}_1$ and $\mathcal{R}_2$ are finite, and, for every $(w_1, w_2) \in W_1 \times W_2$, $V((w_1, w_2))$ is finite. Let $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \otimes \mathbf{T_2}$-model and $\psi \in \mathcal{L}_{\mathbf{T_1} \otimes \mathbf{T_2}}$. The *global* model checking problem for $\mathbf{T_1} \otimes \mathbf{T_2}$ is to check whether there exist $w_1 \in W_1$ and $w_2 \in W_2$ such that $\mathcal{M}, w_1, w_2 \models_{\mathbf{T_1} \otimes \mathbf{T_2}} \psi$.

Given a $\mathbf{T_1} \otimes \mathbf{T_2}$-model $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$, a binary relation $R \in \mathcal{R}_1$ on W_1 (resp. $R \in \mathcal{R}_2$ on W_2), and a world $w \in W_2$ (resp. $w \in W_1$), we use $\hat{R}^1_w$ (resp. $\hat{R}^2_w$) to denote the binary relation on $W_1 \times W_2$ (resp. $W_2 \times W_1$) such that $\hat{R}^1_w((x_1, y_1), (x_2, y_2))$ if and only if $R(x_1, x_2)$ and $y_1 = y_2 = w$ (resp. $\hat{R}^2_w((x_1, y_1), (x_2, y_2))$ if and only if $R(y_1, y_2)$ and $x_1 = x_2 = w$). Moreover, let $\hat{\mathcal{R}}^1_w = \{\hat{R}^1_w \mid R \in \mathcal{R}_1\}$ and $\hat{\mathcal{R}}^2_w = \{\hat{R}^2_w \mid R \in \mathcal{R}_2\}$. Finally, let $\overline{i} = 2$ if $i = 1$, and $\overline{i} = 1$ if $i = 2$.

In figure 10, we present the pseudo-code for a model checker for $\mathbf{T_1} \otimes \mathbf{T_2}$ that exploits model checkers $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ for the component logics $\mathbf{T_1}$ and $\mathbf{T_2}$, respectively. The implementation is similar to that of the model checker for $\mathbf{T_1} \oplus \mathbf{T_2}$.

Theorem 4.3 (*Termination, soundness, and completeness*). *Let* $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ *be a finite model for* $\mathbf{T_1} \otimes \mathbf{T_2}$ *and* $\psi \in \mathcal{L}_{\mathbf{T_1} \otimes \mathbf{T_2}}$. *If* $\mathtt{MC}_{\mathbf{T_1}}$ *and* $\mathtt{MC}_{\mathbf{T_2}}$ *are terminating, sound, and complete, then*:
1. Termination: *the procedure* $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$, *with input* $\mathcal{M}$ *and* ψ, *terminates*;
2. Soundness *and* Completeness: *let* V *be the* (*extended*) *valuation function after termination of procedure* $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$, *with input* $\mathcal{M}$ *and* ψ. *Then, for every subformula* φ *of* ψ *and every pair* $w_1, w_2 \in W_1 \times W_2$, $\varphi \in V((w_1, w_2))$ *if and only if* $\mathcal{M}, w_1, w_2 \models_{\mathbf{T_1} \otimes \mathbf{T_2}} \varphi$.

5. Computational complexity

We now turn to an analysis of the computational complexity of the model checkers proposed in the previous section. An instance for the model checking problem has two components: a model $(W, \mathcal{R}, V)$ and a formula ψ. In our analysis, we will consider three main complexity parameters: the cardinality n of W, the sum m of the cardinalities of the relations in $\mathcal{R}$, and the length k of ψ, i.e., the number of operators and proposition letters in ψ. Given $w \in W$, we will heavily use the following operations on the (extended) valuation $V(w)$: checking whether a formula φ belongs to $V(w)$, and adding a formula φ to $V(w)$. Both operations can be efficiently implemented in constant time by representing V as a 2-dimensional bit array of size $n \times k$ (Clarke et al., 1986).

We will express the complexity of the combined model checker in terms of that of the component model checkers. It will turn out that the complexity of the combined model checker is the sum of two factors: the communication overhead and the model checking cost.

Procedure $\mathtt{MC}_{\mathbf{T}_1 \otimes \mathbf{T}_2}$
Input: a $\mathbf{T}_1 \otimes \mathbf{T}_2$-model $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ and a formula $\psi \in \mathcal{L}_{\mathbf{T}_1 \otimes \mathbf{T}_2}$
compute $Sub(\psi)$
for every $w \in W_2$ **compute** $\widehat{\mathcal{R}}^1_w$; **for every** $w \in W_1$ **compute** $\widehat{\mathcal{R}}^2_w$
for every $(w_1, w_2) \in W_1 \times W_2$ **let** $\overline{V}((w_1, w_2)) = V((w_1, w_2))$
for every $i = 1, \ldots, |\psi|$
  for every $\varphi \in Sub(\psi) \cap \mathcal{L}_{\mathbf{T_1} \otimes \mathbf{T_2}}$ such that $|\varphi| = i$
    case on the form of φ
      $\varphi = P$, $P \in \mathcal{P}$: **skip**
      $\varphi = \varphi_1 \wedge \varphi_2$: **for every** $(w_1, w_2) \in W_1 \times W_2$
        if $\varphi_1 \in V((w_1, w_2))$ **and** $\varphi_2 \in V((w_1, w_2))$ **then**
          $V((w_1, w_2)) = V((w_1, w_2)) \cup \{\varphi\}$
          $\overline{V}((w_1, w_2)) = \overline{V}((w_1, w_2)) \cup \{P_\varphi\}$
      $\varphi = \neg\varphi_1$: **for every** $(w_1, w_2) \in W_1 \times W_2$
        if not $\varphi_1 \in V((w_1, w_2))$ **then**
          $V((w_1, w_2)) = V((w_1, w_2)) \cup \{\varphi\}$
          $\overline{V}((w_1, w_2)) = \overline{V}((w_1, w_2)) \cup \{P_\varphi\}$
      $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, $\mathbf{O} \in OP(\mathcal{L}_{T_i})$, $i \in \{1, 2\}$
      let $\varphi' = \varphi$
      for every $j \in \{1, \ldots c\}$ **replace** φ_j in φ' with P_{φ_j}
      for every $w \in W_{\bar{i}}$
      if $i = 1$ **then** $D = W_1 \times \{w\}$ **else** $D = \{w\} \times W_2$
        for every $(u, v) \in D$ **let** $V'((u, v)) = \overline{V}((u, v))$
        $\mathtt{MC}_{\mathrm{T}_i}((D, \widehat{\mathcal{R}}^i_w, V'), \varphi')$
        for every $(u, v) \in D$
          if $\varphi' \in V'((u, v))$ **then** $V((u, v)) = V((u, v)) \cup \{\varphi\}$;
            $\overline{V}((u, v)) = \overline{V}((u, v)) \cup \{P_\varphi\}$

Figure 10. A model checking procedure for joined logics.

The *communication overhead* is the time spent for "packing" the inputs for the components and for "unpacking" their outputs; this represents the cost of the interaction between the components. The *model checking cost* represents the cost of performing the actual model checking of the component logics.

We first consider the case of temporalization. Let $\mathbf{L}$ be a logic and $\mathbf{T}$ a temporal logic. We write $C_{\mathbf{T(L)}}(\cdot, \cdot, \cdot)$ (resp. $C_{\mathbf{L}}(\cdot, \cdot)$, $C_{\mathbf{T}}(\cdot, \cdot, \cdot)$) for the complexity function of the model checker $\mathtt{MC}_{\mathbf{T(L)}}$ (resp. $\mathtt{MC}_{\mathbf{L}}$, $\mathtt{MC}_{\mathbf{T}}$). Note that $C_{\mathbf{L}}(\cdot, \cdot)$ has two parameters (the size of the model and the length of the formula).

Theorem 5.1. *Let* $(W, \mathcal{R}, g)$ *be a finite* $\mathbf{T(L)}$*-model and* ψ *a* $\mathbf{T(L)}$*-formula. The complexity of* $\mathtt{MC}_{\mathbf{T(L)}}$ *on input* $\mathcal{M}$ *and* ψ *is*

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) \cdot C_{\mathbf{L}}(N, O(k))] + C_{\mathbf{T}}(n, m, O(k)),$$

where $n = |W|$, $m = \sum_{R \in \mathcal{R}} |R|$, $k = |\psi|$ *and* $N = \max_{w \in W} |g(w)|$.

Proof: The set of formulas $\mathrm{MML}_{\mathrm{L}}(\psi)$ and the formula $\psi^{\uparrow}$ can be computed in one pass through ψ, hence in $\mathcal{O}(k)$.

The subsequent nested **for** loop costs

$$\sum_{\alpha \in \mathrm{MML}_{\mathrm{L}}(\psi)} \sum_{w \in W} C_{\mathbf{L}}(|g(w)|, |\alpha|) \leq n \cdot \sum_{\alpha \in \mathrm{MML}_{\mathrm{L}}(\psi)} C_{\mathbf{L}}(N, |\alpha|).$$

To bound the last sum, notice that the set $\mathrm{MML}_{\mathrm{L}}(\psi)$ contains only subformulas of ψ and its cardinality is $O(k)$. Since a set of cardinality n can be partitioned either into $\Theta(n)$ sets of cardinality $\Theta(1)$ or into $\Theta(1)$ sets of cardinality $\Theta(n)$, the above sum is

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) \cdot C_{\mathbf{L}}(N, O(k))].$$

Finally, as the length of $\psi^{\uparrow}$ is $O(k)$, the unique call to $\mathtt{MC}_{\mathrm{T}}$ costs $C_{\mathbf{T}}(n, m, O(k))$. Summing up, the overall cost is

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) C_{\mathbf{L}}(N, O(k))] + C_{\mathbf{T}}(n, m, O(k)).$$ □

The communication overhead is the cost of computing the set $\mathrm{MML}_{\mathbf{L}}(\psi)$ and the formula $\psi^{\uparrow}$. It equals $O(k)$ and is dominated by the model checking cost. For instance, if $\mathbf{T}$ is CTL (hence $C_{\mathbf{T}}(n, m, k) = O((n + m) \cdot k)$ (Emerson, 1990)), and $\mathbf{L}$ is a logic such that $C_{\mathbf{L}}(n, k) = O(n \cdot k)$, then the model checking cost is $O(k \cdot (n \cdot N + m))$, hence still linear in the size of the model and in the length of the formula.

We now treat the independent combination of two temporal logics $\mathbf{T_1}$ and $\mathbf{T_2}$.

Theorem 5.2. *Let $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \oplus \mathbf{T_2}$-model and ψ a $\mathbf{T_1} \oplus \mathbf{T_2}$-formula. The complexity of $\mathtt{MC}_{\mathbf{T_1} \oplus \mathbf{T_2}}$ on input $\mathcal{M}$ and ψ is*:

$$O(m_1 + m_2 + n \cdot k) + \sum_{i=1}^{2} \big(O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k)) \big)$$

where $n = |W|$, $m_i = \sum_{R \in \mathcal{R}_i} |R|$, for $i = 1, 2$, and $k = |\psi|$.

Proof: The set of connected components $\mathcal{C}^1_{\mathcal{M}}$ (resp. $\mathcal{C}^2_{\mathcal{M}}$) can be computed in time $O(n + m_1)$ (resp. $O(n + m_2)$) by means of a depth-first visit of the graph $(W, \mathcal{R}_1)$ (resp. $(W, \mathcal{R}_2)$). The cost of computing $Sub(\psi)$ is linear in the size of ψ, and hence it is $O(k)$.

The cost of the second part of the computation is: $\sum_{\varphi \in Sub(\psi)} C(\varphi)$, where the cost factor $C(\varphi)$ depends on the form of φ. In particular, if φ is a proposition letter, then $C(\varphi) = O(1)$. If $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \neg\varphi_1$, then $C(\varphi) = O(n)$. If $i \in \{1, 2\}$ and $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{T_i})$, then the cost $C(\varphi)$ is computed as follows. Let $\mathcal{C}^i_{\mathcal{M}} = \{(U^i_j, S^i_j) \mid j = 1, \ldots, c_i\}$, n^i_j and m^i_j be the cardinalities of U^i_j and S^i_j, respectively, for $j = 1, \ldots, c_i$. The replacement of subformulas in φ' with proposition letters costs $O(c) = O(|\varphi|)$. Moreover, for every connected component (U^i_j, S^i_j) in $\mathcal{C}^i_{\mathcal{M}}$, the following steps are performed:

- the valuation V' is computed in $O(n_j^i)$;
- the formula φ' is model checked in $C_{\mathbf{T}_i}(n_j^i, m_j^i, O(|\varphi|))$;
- the valuation V is updated in $O(n_j^i)$.

It follows that, in this case, $C(\varphi)$ amounts to

$$O(|\varphi|) + \sum_{j=1}^{c_i} \left(O\left(n_j^i\right) + C_{\mathbf{T}_i}\left(n_j^i, m_j^i, O(|\varphi|)\right)\right).$$

Since $\mathcal{C}_{\mathcal{M}}^i$ contains either $\Theta(n)$ connected components with $\Theta(1)$ nodes or $\Theta(1)$ connected components with $\Theta(n)$ nodes, $C(\varphi)$ is as follows:

$$O(|\varphi|) + O(n) + O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(|\varphi|)) + O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(|\varphi|)).$$

Moreover, since the set $Sub(\psi)$ contains either $\Theta(k)$ formulas of length $\Theta(1)$ or $\Theta(1)$ formulas of length $\Theta(k)$, the cost of the second part of the computation is:

$$O(n \cdot k) + \sum_{i=1}^{2} \left(O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k)) \right)$$

and, hence, the overall complexity is:

$$O(m_1 + m_2 + n \cdot k) + \sum_{i=1}^{2} \left(O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k)) \right).$$

□

The communication overhead is the cost of computing the connected components, of preparing the valuation as input to the model checking procedure, and of updating the valuations when the procedure returns. It adds up to $\mathcal{O}(m_1 + m_2 + n \cdot k)$, which is more significant than in the case of temporalization. By way of example, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL, and $m = m_1 = m_2$, then the communication overhead is $O(m + n \cdot k)$, which is proportional to the model checking cost of $O((n + m) \cdot k)$. So, the overall cost of the model checker for CTL $\oplus$ CTL is $O((n + m) \cdot k)$, which is linear in the size of the model and in the length of the formula. If both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL*, then the model checking cost is exponential in the length k of the formula, and hence it dominates the communication overhead.

Finally, we consider the join of temporal logics $\mathbf{T_1}$ and $\mathbf{T_2}$.

Theorem 5.3. *Let $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \otimes \mathbf{T_2}$-model and ψ a $\mathbf{T_1} \otimes \mathbf{T_2}$-formula. Let $\bar{1} = 2$ and $\bar{2} = 1$. The complexity of $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$ on input $\mathcal{M}$ and ψ is*:

$$O(n_1 \cdot m_2 + n_2 \cdot m_1 + n_1 \cdot n_2 \cdot k) + \sum_{i=1}^{2} O(n_{\bar{i}}) \cdot \big[O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))\big],$$

where $n_i = |W_i|$, $m_i = \sum_{R \in \mathcal{R}_i} |R|$, for $i = 1, 2$, and $k = |\psi|$.

Proof: The sets $\hat{\mathcal{R}}^1_w$ and $\hat{\mathcal{R}}^2_w$ can be computed in $O(n_2 \cdot m_1)$ and $O(n_1 \cdot m_2)$, respectively. The cost of computing $Sub(\psi)$ is linear in the size of ψ, hence it is $O(k)$. The cost of the second part of the computation is: $\sum_{\varphi \in Sub(\psi)} C(\varphi)$, where the cost factor $C(\varphi)$ depends on the form of φ. In particular, if φ is a proposition letter, then $C(\varphi) = O(1)$. If $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \neg\varphi_1$, then $C(\varphi) = O(n_1 \cdot n_2)$. If $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{\mathbf{T_1}})$, the cost $C(\varphi)$ amounts to $O(c) = O(|\varphi|)$ to replace subformulas in φ' with proposition letters, plus n_2 times the sum of the following factors:

- n_1 to compute V';
- $C_{\mathbf{T_1}}(n_1, m_1, O(|\varphi|))$ to model check φ;
- n_1 to update V.

That is, $C(\varphi)$ is

$$O(|\varphi|) + O(n_2) \cdot \big[O(n_1) + C_{\mathbf{T_1}}(n_1, m_1, O(|\varphi|))\big].$$

Similarly, if $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{T_2})$, the cost $C(\varphi)$ amounts to

$$O(|\varphi|) + O(n_1) \cdot \big[O(n_2) + C_{\mathbf{T_2}}(n_2, m_2, O(|\varphi|))\big].$$

It follows that the cost of the second part of the computation is

$$O(n_1 \cdot n_2 \cdot k) + \sum_{i=1}^{2} O(n_{\bar{i}}) \cdot \big[O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))\big],$$

and, hence, the overall complexity is:

$$O(n_1 \cdot m_2 + n_2 \cdot m_1 + n_1 \cdot n_2 \cdot k) + \sum_{i=1}^{2} O(n_{\bar{i}}) \cdot \big[O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))\big].$$

□

The communication overhead is the cost of computing the sets $\hat{\mathcal{R}}^1_w$ and $\hat{\mathcal{R}}^2_w$ plus the cost of preparing and updating the valuation functions before and after the invocation of the model checking procedure, respectively. It amounts to $\mathcal{O}(n_2 \cdot m_1 + n_1 \cdot m_2 + n_1 \cdot n_2 \cdot k)$. For instance, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL, $n = n_1 = n_2$, and $m = m_1 = m_2$, then the communication cost is $O(n \cdot (m + n \cdot k))$, which is proportional to the model checking cost of $O(n \cdot (n + m) \cdot k)$. Hence, the overall cost of the model checker for CTL $\otimes$ CTL is $O(n \cdot (n+m) \cdot k)$. If $m = \Theta(n)$, the cost is $O(n^2 \cdot k)$, hence it is linear in the size of the model and the length of the formula, else if $m = \Theta(n^2)$, then the complexity is $O(n^3 \cdot k)$. As in the case of the independent combination, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL*, the model checking dominates the communication overhead.

6. Experimental results

In this section, we report on experimental results based on implementations of combined model checkers for CTL(CTL) and CTL $\oplus$ CTL. Both of them have been built on top of a model checker for CTL, implemented in C, and are available from `http://www.science.uva.nl/~mdr/ACLG/Software/`. Tests were carried on a Sun ULTRA II (300 MHz) with 1 Gb RAM, under Solaris 5.2.5.

First, we treat the case of temporalization of CTL by means of CTL. We tested the model checker on "linear" and "dense" models, varying the size of the model. Let $\mathcal{M}$ be a model for CTL(CTL) and φ a formula in the language of CTL(CTL).

In the first test, we fixed φ to be $\mathbf{A_1G_1A_2}(P\ \mathbf{U_2}\ Q)$, and we adopted as model $\mathcal{M}_1 = (W, R, g)$, where (W, R) is a complete binary tree of height h_1 and, for every $w \in W$, $g(w)$ is a labeled complete binary tree of height h_2. Hence, this model contains $n = n_1 \cdot (n_2 + 1)$ nodes, and $m = n_1 \cdot n_2 - 1$ edges, where $n_1 = 2^{h_1+1} - 1$ and $n_2 = 2^{h_2+1} - 1$. Moreover, the trees $g(w)$ are labeled so that every node and every edge is processed during the checking of $\mathbf{A_2}(P\ \mathbf{U_2}\ Q)$. The experimental outcomes we have obtained on this instance are summarized in Table 1, where t_{ms} represents the CPU time in milliseconds. Note that the time needed to perform the model checking grows linearly in the size of the model.

In the second test, we checked the formula $\varphi = \mathbf{A_1G_1E_2}(P\ \mathbf{U_2}\ Q)$, and we adopted as model $\mathcal{M}_2 = (W, R, g)$, where (W, R) is a complete graph of n_1 nodes and, for every $w \in W$, $g(w)$ is a complete graph of n_2 nodes. Hence, this model contains $n = n_1 \cdot (n_2 + 1)$

Table 1. Trees and $\mathbf{A_1G_1A_2}(P\ \mathbf{U_2}\ Q)$.

h_1	h_2	No. of nodes	No. of edges	t_{ms}
4	4	992	960	10
5	5	4032	3968	30
6	6	16256	16128	110
7	7	65280	65024	380
8	8	261632	261120	1490
9	9	1047552	1046528	5850

Table 2. Complete graphs and $\mathbf{A_1G_1E_2}(P\ \mathbf{U_2}\ Q)$.

n_1	n_2	No. of nodes	No. of edges	t_{ms}
32	32	1056	33792	20
64	64	4160	266240	110
128	128	16512	2113536	820
256	256	65792	16842752	6010
512	512	262656	134479872	47970
1024	1024	1049600	1074790400	386760

nodes, and $m = n_1 \cdot (n_2^2 + n_1)$ edges. The models $g(w)$ are labeled in an appropriate way in order to process every node and every edge during the checking of $\mathbf{E_2}(P\ \mathbf{U_2}\ Q)$. The outcomes are summarized in Table 2. Once again, the time needed to perform the model checking grows linearly in the size of the model. Moreover, as expected, the complexity of the model checker depends on the number of edges too. Indeed, if we compare the costs of the above two instances for the same number of nodes, we note that checking the second instance in harder. This is because $\mathcal{M}_2$ contains *dense* graphs, i.e., graphs in which the number of edges is quadratic in the number of nodes, while $\mathcal{M}_1$ is based on *linear* graphs, i.e., graphs in which the number of edges is linear in the number of nodes.

Next, we treat the case of the independent combination of CTL and CTL. We tested the model checker on "square grid" models, varying the size of the model (and fixing the formula) or varying the "degree of interaction" of the formula (and fixing the model). Let $\mathcal{M}$ be a model for CTL $\oplus$ CTL and φ be a formula in the language of CTL $\oplus$ CTL.

In the first test, we fixed φ to be $\mathbf{A}_1\mathbf{G}_1 Q \wedge \mathbf{A}_2\mathbf{G}_2 Q$, and we adopted as our model $\mathcal{M} = (W, R_1, R_2, V)$ a square grid in which the rows are the connected components of (W, R_1) and the columns are the connected components of (W, R_2). We tested the model checker on square grids with a side of size l, hence with number of nodes $n = l^2$ and a number of edges $m = 2 \cdot l \cdot (l-1)$. The outcomes are summarized in Table 3. Note that the time needed to perform the model checking grows linearly in the size of the model.

In our second test, we fixed the model $\mathcal{M}$ to be a square grid with a side of size 256, and hence with 65396 nodes and 130560 edges, and we changed the degree of interaction

Table 3. Square grids and $\mathbf{A}_1\mathbf{G}_1 Q \wedge \mathbf{A}_2\mathbf{G}_2\ Q$.

l	No. of nodes	No. of edges	t_{ms}
32	1024	1984	90
64	4096	8024	340
128	16384	32512	1400
256	65396	130560	5760
512	262144	532264	23480
1024	1048576	2095104	118980

Table 4. Fixed square grids and alternating formulas.

r	0	3	7	11	15	19
t_{ms}	1870	2540	2970	3860	4720	5590

of the formula. Define $f_0 = Q$, and $f_{k+1} = \mathbf{E}_i\mathbf{X}_i f_k$, for $k \geq 0$ and $i \in \{1, 2\}$. We call the token $\mathbf{E}_i\mathbf{X}_i$ in f_k a *quantifier* of f_k and the token $\mathbf{E}_i\mathbf{X}_i\mathbf{E}_j\mathbf{X}_j$, with $i \neq j$, an *alternation of quantifiers* of f_k. We tested the model checker on f_{20}, varying the number r of alternations of quantifiers of f_{20} from 0 (no interaction at all) to 19 (maximal interaction). The outcomes are summarized in Table 4. As expected, the greater the degree of interaction of the formula is, the longer the response time of the model checker becomes. Indeed, the communication overhead is higher when checking formulas with strong interaction, due to the time spent on packing the input and unpacking the output during the switches between the main model checker and the component's ones.

7. Related work and discussion

Let us briefly review the methodology we have been proposing. In order to verify a requirement against a (mobile) system, we first have to encode the behavior of the system into a suitable combined model. Then, we have to express the requirement in a sufficiently expressive combined language. Finally, we can perform combined model checking as described in Section 4. The response time of the combined model checker depends on those of the component model checkers, and in many concrete situations it may be quite appealing (e.g., if we specify our requirements in a combined language whose components admit polynomial-time model checkers, the combined model checker runs in polynomial time).

One of the main reasons for the relative ease with which we can make combinations work is that the ways of combining that we consider require no synchronization between the components. Although there may be *interaction* between the components—as we have seen with the join—, this is only a very loose kind of interaction. This is in contrast with *modular model checking*, which has been proposed as a way to address the so-called state-explosion problem. In modular verification, the specification of a module consists of two parts. One part describes the guaranteed behavior of the module. The other part describes the assumed behavior of the environment with which the module is interacting; this is called the assume-guarantee paradigm (Jones, 1983; Lamport, 1983). The level of interaction between the module and its environment is far more intricate than the kinds of interaction we have been discussing, and, hence, in modular model checking the computational overhead for the combination is much more significant than in our setting (Kupferman and Vardi, 1998, 2000).

It is also worth remarking that we do not consider in this paper the nontrivial problem of decomposing a complex system into simpler components. We simply assume that the target system is already expressible as a combination of components according to some combining method. We showed that in some cases, for instance in the case of mobile reactive systems, this assumption is satisfied. However, we are aware that there are models that cannot be

obtained as a composition of simpler components. For instance, not every $S5_2$-model is a join of two models for $S5$. For systems whose behaviour is captured by such models, our modular approach cannot be adopted.

We conclude this section with a short comparison between our approach to model check mobile systems and the one developed by Cardelli et al. based on ambient calculus and ambient logic. As for the system modeling task, we use a relational structure, while the approach by Cardelli et al. exploits algebraic expressions belonging to the ambient calculus. To specify system properties, we use a combined logic, which embeds a spatial component into a temporal one, while Cardelli et al. use ambient logic. As for the model checking task, we only program the interface, which must be supplied with the model checkers for the component logics to obtain a model checker for the combined logic, while Cardelli et al. developed various model checkers tailored to different versions of the ambient calculus and ambient logic. One advantage of the ambient-based approach is its high-level modeling language, which is somehow more natural than our low-level language of labeled graphs. However, one has to regard the graph representation as a 'machine language' on which reasoning is particularly efficient. Automatic translations from high-level model representations to low-level state transition ones can be devised (Clarke et al., 1999; Manna and Pnueli, 1992). The main advantage of our combined approach is flexibility: the final model checker is the composition of existing ones. Only the interface is programmed; the component blocks may be selected according to the needs of expressiveness and complexity. For instance, a model checker for CTL(HL), where HL is the basic hybrid logic with nominals and the @ operator runs in polynomial time. If higher expressiveness is required at the temporal level, one may substitute CTL with CTL*, or, whenever a higher expressive power is needed at the spatial level, one may add the hybrid binders to the basic hybrid logic. In such cases, the resulting model checker runs in polynomial space and exponential time, with respect to the length of the formula. Finally, in our approach one has to take care of the so-called *state-explosion problem*. Whenever the system is the concurrent composition of several processes, the size of the representation graph grows exponentially in the number of processes of the system. However, many techniques have been proposed to successfully cope with the state explosion problem, including modular, on-the-fly, symbolic, symmetric, and abstract model checking (a survey of these techniques can be found in Clarke et al. (1999).

8. Conclusions

We have addressed the problem of model checking for combined (temporal) logics and structures. In contrast to combined deductive engines, combinations of model checking procedures are very well behaved, even in the presence of strong forms of interaction. In particular, complexity upper bounds transfer from the components to the combination, and in most cases the communication overhead is insignificant when compared to the actual model checking cost. One of the motivations for this work was the need to develop a model checking framework for mobile systems and corresponding logics. We have shown that this is indeed possible, using a *divide and conquer* strategy: we first isolate the orthogonal 'simple' entities in which a system can be decomposed. Then, we apply well-known structures and logics to

the component entities. Finally, we compose the results in order to infer general properties of the entire system. We feel that this *divide and conquer* approach can be useful to model and analyze many other complex systems, which, inherently, are the composition of simpler entities and where the composition itself is 'loose' in the sense that synchronization between the components is not called for.

Acknowledgment

We would like to thank the reviewers for their valuable comments. Massimo Franceschet and Angelo Montanari were supported by the MURST Project 'Saladin'. Maarten de Rijke was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 220-80-001, 365-20-005, 612.000.106, 612.000.207, 612-13-001, and 612.069.006.

References

Areces, C., Blackburn, P., and Marx, M. 1999. A road-map on complexity for hybrid logics. In J. Flum and M. Rodriguez-Artalejo, editors, *Proc. of the Annual Conference of the European Association for Computer Science Logic*, vol. 1683 of LNCS, Springer, pp. 307–321.

Areces, C., Blackburn, P., and Marx, M. 2000. The computational complexity of hybrid temporal logics. *Logic Journal of the IGPL*, 8(5):653–679.

Areces, C., Blackburn, P., and Marx, M. 2001. Hybrid logics: Characterization, interpolation, and complexity. *Journal of Symbolic Logic*, 66(3):977–1010.

Baader, F. and Ohlbach, H. 1995. A multidimensional terminological knowledge representation language. *Applied NonClassical Logic*, 5:153–197.

Blackburn, P. 2000. Representation, reasoning, and relational structures: A hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–365.

Blackburn, P. and de Rijke, M. (eds.) 1996. Special Issue on Combining Structures, Logics, and Theories. *Notre Dame Journal of Formal Logic*, 37:161–380.

Cardelli, L. 1999. Abstractions for mobile computations. In J. Vitek and C. Jensen editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, vol. 1603 of LNCS, Springer, pp. 51–94.

Cardelli, L. and Gordon, A.D. 2000a. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, 19–21, pp. 365–377.

Cardelli, L. and Gordon, A.D. 2000b. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213.

Charatonik, W., Dal Zilio, S., Gordon, A.D., Mukhopadhyay, S., and Talbot, J.-M. 2001. The complexity of model checking mobile ambients. In F. Honsell and M. Miculan, editors, *Proc. of the International Conference on Foundations of Software Science and Computation Structures*, vol. 2030 of LNCS, Springer, pp. 52–167.

Charatonik, W. and Talbot, J.-M. 2001. The decidability of model checking mobile ambients. In *Proc. of the 15th Annual Conference of the European Association for Computer Science Logic*, Springer, pp. 339–354.

Clarke, E., Emerson, E.A., and Sistla, A.P. 1986. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263.

Clarke, E.M., Grumberg, O., and Peled, D.A. 1999. *Model Checking*. Cambridge MA: The MIT Press.

Clarke, E.M. and Schlingloff, H. 2001. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, vol. II, Elsevier Science, chap. 24, pp. 1635–1790.

Emerson, E.A. 1990. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, Elsevier Science Publishers B.V., pp. 995–1072.

Engelfriet, J. 1996. Minimal temporal epistemic logic. *Notre Dame Journal of Formal Logic*, 37:233–259.

Fagin, R., Halpern, J.Y., Moses, Y., and Vardi, M.Y. 1995. *Reasoning about Knowledge*. Cambridge, MA: MIT Press.

Fine, K. and Schurz, G. 1996. Transfer theorems for multimodal logics. In J. Copeland, editor, *Logic and Reality: Essays on the Legacy of Arthur Prior*, Oxford, Oxford University Press, pp. 169–213.

Finger, M. 1992. Handling database updates in two-dimensional temporal logic. *Journal of Applied Non-Classical Logics*, 2(2):201–224.

Finger, M. 1994. *Changing the Past: Database Applications of Two-Dimensional Executable Temporal Logics*. PhD thesis, Imperial College, Department of Computing.

Finger, M. and Gabbay, D.M. 1992. Adding a temporal dimension to a logic system. *Journal of Logic Language and Information*, 1:203–233.

Finger, M. and Gabbay, D.M. 1996. Combining temporal logic systems. *Notre Dame Journal of Formal Logic*, 37:204–232.

Finger, M. and Reynolds, M. 2000. Two-dimensional executable temporal logic for bitemporal databases. In *Advances in Temporal Logic*, Kluver Academic Publishers, pp. 393–411.

Franceschet, M. and de Rijke, M. 2003. Model checking for hybrid logics. In *Proceedings of the 3rd International Workshop on Methods for Modalities (M4M)*, pp. 109–123.

Franceschet, M., de Rijke, M., and Schlingloff, H. 2003. Hybrid logics on linear structures: Expressivity and complexity. In *Proc. of the 10th International Symposium on Temporal Representation and Reasoning and of the 4th International Conference on Temporal Logic (TIME-ICTL)*. IEEE Computer Society Press.

Franceschet, M., Montanari, A., and de Rijke, M. 2000. Model checking for combined logics. In *Proc. of the 3rd International Conference on Temporal Logic*, pp. 65–73.

Gabbay, D., Kurucz, A., Wolter, F., and Zakharyaschev, M. 2003. *Many-Dimensional Modal Logics: Theory and Applications*. Elsevier.

Gabbay, D.M. and de Rijke M. (eds.) 2000. *Frontiers of Combining Systems 2*, vol. 7 of *Studies in Logic and Computation*. Research Studies Press/Wiley.

Gabbay, D.M. and Shehtman, V. 1998. Products of modal logics, part I. *Logic Journal of the IGPL*, 6:73–146.

Goguen, J.A. and Burstall, R.M. 1992. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–147.

Halpern, J.Y. and Vardi, M.Y. 1989. The complexity of reasoning about knowledge and time I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237.

Halpern, J.H. and Vardi, M.Y. 1991. Model checking vs. theorem proving: A manifesto. In *Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, pp. 325–334.

Hemaspaandra, E. 1994. Complexity transfer for modal logic. In *Proc. of the 9th Symposium on Logic in Computer Science*, Los Alamitos, CA., USA: pp. 164–175. IEEE Computer Society Press.

HyLo: The Hybrid Logic home page. URL: http://www.hylo.net.

Jones, C.B. 1983. Specification and design of (parallel) programs. In *IFIP World Computer Congress*, pp. 321–332.

Kracht, M. and Wolter, F. 1991. Properties of independently axiomatizable bimodal logics. *Journal of Symbolic Logic*, 56(4):1469–1485.

Kupferman, O. and Vardi, M.Y. 1998. Modular model checking. In *Compositionality: The Significant Difference. International Symposium, COMPOS97*, volume 1536 of LNCS, Springer, pp. 381–401.

Kupferman, O. and Vardi, M.Y. 2000. An automata-theoretic approach to modular model checking. *ACM Transactions on Programming Languages and Systems*, 22:87–128.

Kurucz, A. 2000. $S5^3$ lacks the finite model property. In *Proc. of the 3rd International Conference on Temporal Logic (ICTL)*.

Lamport, L. 1983. Specifying concurrent program modules. *ACM Transaction on Programming Language and Systems*, 5:190–222.

Manna, Z. and Pnueli, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer.

Marx, M. 1999. Complexity of products of modal logics. *Journal of Logic and Computation*, 9:221–238.

Meyer, J. and van der Hoek, W. 1995. *Epistemic Logic for AI and Computer Science*. Cambridge University Press.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings IEEE Symposium of Foundations of Computer Science*, pp. 46–77.

Sernadas, A., Sernadas, C., and Caleiro, C. 1997. Synchronization of logics with mixed rules: Completeness preservation. In M. Johnson, editor, *Algebraic Methodology and Software Technology—AMAST97*, vol. 1349 of LNCS, Springer, pp. 465–478.

Spaan, E. 1993. *Complexity of Modal Logics*. PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam.

Wolter, F. 1995. The finite model property in tense logic. *The Journal of Symbolic Logic*, 60(3):757–774.

Wolter, F. 1996. A counterexample in tense logic. *Notre Dame Journal of Formal Logic*, 37(2):167–173.

Wolter, F. 1997. Completeness and decidability of tense logics closely related to logics above K4. *The Journal of Symbolic Logic*, 62(1):131–158.

Wolter, F. 1998. Fusions of modal logics revisited. In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic*. CSLI, Stanford, CA.

Wolter, F. 2000. The product of converse PDL and polymodal K. *Journal of Logic and Computation*, 10(2):223–251.

Wolter, F. and Zakharyaschev, M. 1998. Satisfiability problem in description logics with modal operators. In *Proc. of the 6th Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, pp. 512–523.