Towards an automata-theoretic counterpart of combined temporal logics

Massimo Franceschet and Angelo Montanari

Department of Mathematics and Computer Science, University of Udine, Italy
{*francesc*|*montana*}*@dimi.uniud.it*

Abstract. In this paper, we define a new class of combined automata, called *temporalized automata*, which can be viewed as the automata-theoretic counterpart of temporalized logics, and show that relevant properties, such as closure under Boolean operations, decidability, and expressive equivalence with respect to temporal logics, transfer from component automata to temporalized ones. Furthermore, we successfully apply temporalized automata to provide the full second-order theory of k-refinable downward unbounded layered structures with a temporal logic counterpart. Finally, we show how temporalized automata can be used to deal with relevant classes of reactive systems, such as granular reactive systems and mobile reactive systems.

1 Introduction

Logic combination is emerging as a relevant research topic at the intersection of mathematical logic with computer science [17]. It essentially provides a logical account of traditional computer science notions such as modularity and abstraction. When dealing with real-world systems, organizing their descriptive and inferential requirements in a structured way is often the only way to master the complexity of the design, verification, and maintenance tasks. Formulated in the setting of combined logics, the basic issue underlying such an approach is: how can we guarantee that the logical properties of the component logics, such as axiomatic completeness and decidability, are inherited by the combined one? This issue is known as the *transfer problem*. It has a natural analogue in terms of the associated methods and tools: can we reuse methods and tools developed for the component logics, such as deductive engines and model checkers, to obtain methods and tools for the combined one? In general, the answer depends on the amount of interaction among components: the transfer generally succeeds in the absence of interaction among the component logics [11, 14], but it can easily fail when they are allowed to interact. There are, however, some exceptions. As an example, as shown in [16], in contrast to combining deductive engines, combinations of model checking procedures are well behaved, even in the presence of interaction.

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 [11, 13, 14, 24]. They have been successfully applied in several areas, including databases [12, 26, 27], artificial intelligence [9, 19, 10], and system specification and verification [16]. We are mainly interested in this last application of combined logics. Our ultimate goal is to provide combined temporal logics with an automata-theoretic counterpart. Such an equivalent characterization of combined logics as combined automata presents several

advantages for automated system specification and verification. First, turning a formula of (combined) temporal logic into an automaton allows us to give a uniform representation of systems and specifications as automata. The problem of establishing whether a system P behaves according to a given specification ϕ (model checking problem) can then by reduced to the language containment problem $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, where $\mathcal{L}(\mathcal{A}_P)$ is the language recognized by the automaton $\mathcal{A}_P$, consisting of all and only the behaviors of P, and $\mathcal{L}(\mathcal{A}_\phi)$ is the language recognized by the automaton $\mathcal{A}_\phi$, consisting of all and only the models of ϕ [37]. Furthermore, if the considered class of automata is closed under Boolean operations, the language containment problem can be mapped into the emptiness one, that is, $\mathcal{L}(\mathcal{A}_P \cap \overline{\mathcal{A}_\phi}) = \emptyset$. Second, (combined) automata can be directly used as a specification formalism, provided with a natural graphical interpretation [28]. Moreover, to avoid the costly complementation of the specification automaton $\mathcal{A}_\phi$, some model checkers constrain the user to directly provide the automaton $\mathcal{A}_{\neg\phi}$ for $\neg\phi$, that is, they constrain the user to specify the unacceptable behaviors of the system instead of the good ones [21, 20]. As an alternative, one can replace specification automata with a costly complementation by other, equally expressive, automata for which the complementation is much easier [25]. For instance, if the specification language is (quantified) linear temporal logic, we can replace Büchi automata by Muller automata. Third, the specification automaton $\mathcal{A}_\phi$ can be used to cope with the state explosion problem by preventing the complete construction of the system automaton $\mathcal{A}_P$ from happening, whenever possible (on-the-fly model-checking) [5, 23, 36]. More precisely, some system states, which are incompatible with or irrelevant to the specification, may not be generated at all; furthermore, a counterexample for $\mathcal{A}_P \cap \overline{\mathcal{A}_\phi}$ can be detected before the completion of $\mathcal{A}_P$ construction, making such a completion no more necessary. Finally, the automata-theoretic characterization of (combined) temporal logics can help in finding the temporal logic counterpart of monadic theories. In many cases, this is a difficult task, involving a non-elementary blow up in the length of formulas. Ehrenfeucht games have been successfully exploited to deal with such a correspondence problem for first-order monadic theories [22] and well-behaved fragments of second-order ones, e.g. the path fragment of the monadic second-order theory of infinite binary trees [18]. Unfortunately, these techniques do not naturally lift to the full second-order case. The existence of a correspondence between (combined) temporal logics and (combined) automata, satisfying the usual closure properties, allows one to reduce the task of finding a temporal logic counterpart of a (second-order) monadic theory to the often easier one of finding an automata counterpart of it. The mapping of monadic formulas into automata (the difficult direction) can indeed greatly benefit from automata closure properties.

In this paper, we focus on the simplest case of temporalized logics. We define a new class of combined automata, called *temporalized automata*, which can be viewed as the automata-theoretic counterpart of temporalized logics, and show that relevant properties, such as closure under Boolean operations, decidability, and expressive equivalence with respect to temporal logics, transfer from component automata to temporalized ones. The way of combining automata we propose can be naturally viewed as a way of abstracting and concretizing over automata. We show how certain properties of an automaton can be checked on its abstracted versions, instead of on the automaton itself. This generally saves space and time. Furthermore, we successfully apply temporalized automata to provide the full second-order theory of k-refinable downward unbounded layered structures, which can be used to model infinitely refinable granular structures consisting of a coarsest domain and an infinite number of finer and finer domains, with an expressively complete

and elementarily decidable temporal logic counterpart (a problem that we left open in [15]). Finally, we show how temporalized logics and automata can be used to deal with relevant classes of reactive systems, such as granular reactive systems and mobile reactive systems. A *granular reactive system* is a reactive system whose behaviour can be naturally modeled with respect to a (possibly infinite) set of differently-grained temporal domains. This is the case, for instance, of reactive systems consisting of a set of processes which have dynamic behaviors regulated by very different—even by orders of magnitude—time constants [29]. A *mobile reactive system* is a reactive system whose processes may reside within a hierarchy of locations, called ambient structure, and modify it [3]. The operational behavior of a granular reactive system can be described as a suitable combination of temporal evolutions and temporal refinements [16]. Analogously, the operational behaviour of a mobile reactive system can be described as a suitable combination of two different components: the *spatial* distribution of processes within the ambient structure, and the *temporal* evolution of processes due to computations or movements.

2 Temporalized logics

In this section we give a general definition of temporal logic that will allow us to easily compare the expressive power of combined temporal logics and combined automata. Let Σ be a finite alphabet. The language of *temporal logic* is based on a set of propositional letters $\mathcal{P}_\Sigma = \{p_a \mid a \in \Sigma\}$ and extends that of *propositional logic* with a possibly infinite 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 $\mathbf{TL}[OP, \mathcal{P}_\Sigma]$ of *temporal logic* is the smallest set $\mathcal{L}$ of formulas generated by the following rules: every proposition letter $p_a \in \mathcal{P}_\Sigma$ is in $\mathcal{L}$; if ϕ, ψ are in $\mathcal{L}$, then $\phi \wedge \psi$ and $\neg\phi$ are in $\mathcal{L}$; if $\mathbf{O}_j^{i_j} \in OP$ and $\phi_1, \ldots, \phi_{i_j}$ are in $\mathcal{L}$, then $\mathbf{O}_j^{i_j}(\phi_1, \ldots, \phi_{i_j})$ is in $\mathcal{L}$.

Boolean connectives $\vee$, $\rightarrow$, and $\leftrightarrow$ are defined as usual. Moreover, given $p_a \in \mathcal{P}_\Sigma$, `true` abbreviates $p_a \vee \neg p_a$ and `false` stands for $\neg$`true`. A *frame* for the temporal logic $\mathbf{TL}[OP, \mathcal{P}_\Sigma]$ is a pair $(W, \mathcal{R})$, where W is a set of *worlds*, or *states*, and $\mathcal{R}$ is a set of *accessibility relations* on W. A *model* is a triple $(W, \mathcal{R}, V)$, where $(W, \mathcal{R})$ is a frame and $V : W \rightarrow \Sigma$ is a *labeling function* mapping every state into *one* symbol of Σ. The semantics of the propositional fragment of temporal logic $\mathbf{TL}[OP, \mathcal{P}_\Sigma]$ is as follows. Let $\mathcal{M} = (W, \mathcal{R}, V)$ be a model and $w \in W$:

(S1) $\mathcal{M}, w \models p_a$ iff $V(w) = a$, for every proposition letter $p_a \in \mathcal{P}_\Sigma$;
(S2) $\mathcal{M}, w \models \phi \wedge \psi$ iff $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$;
(S3) $\mathcal{M}, w \models \neg\phi$ iff it is not the case that $\mathcal{M}, w \models \phi$.

The full semantics of temporal logic $\mathbf{TL}[OP, \mathcal{P}_\Sigma]$ extends the one given above with clauses for the temporal operators in OP, depending on the particular choice for these operators. We give two notable examples:

Example 1. (Temporal logics)

1. The language of *Propositional Linear Temporal Logic* (PLTL) is $\mathbf{TL}[\{\mathbf{X}, \mathbf{U}\}, \mathcal{P}_\Sigma]$, where $\mathbf{X}$ (next) is a unary operator and $\mathbf{U}$ (until) is a binary operator. A model for PLTL is a triple $(W, \{R\}, V)$, also denoted (W, R, V), such that (W, R) is $(\mathbb{N}, <)$ (the natural numbers with the usual ordering). The semantic clauses for $\mathbf{X}$ and $\mathbf{U}$ are as follows. Let $\mathcal{M} = (W, R, V) \in \mathcal{S}(\Sigma)$ and $i \geq 0$:

$\mathcal{M}, i \models_{\text{PLTL}} \phi \mathbf{U} \psi$ iff $\mathcal{M}, j \models_{\text{PLTL}} \psi$ for some $j \geq i$, and
$\mathcal{M}, k \models_{\text{PLTL}} \phi$ for every $i \leq k < j$;
$\mathcal{M}, i \models_{\text{PLTL}} \mathbf{X} \phi$ iff $\mathcal{M}, i+1 \models_{\text{PLTL}} \phi$.

2. Let $k \geq 1$. The language of *Directed Computational Tree Logic* (CTL^*_{k} for short) is $\mathbf{TL}[\{\mathbf{E}, \mathbf{U}, \mathbf{X_0}, \ldots, \mathbf{X}_{k-1}\}, \mathcal{P}_\Sigma]$, where $\mathbf{E}$ and $\mathbf{X_0}, \ldots, \mathbf{X}_{k-1}$ are unary operators and $\mathbf{U}$ is a binary operator.
A model $\mathcal{M}$ for CTL^*_{k} is a triple (W, R, V), where (W, R) isomorphic to $(\{0, \ldots, k-1\}^*, <)$ (where $<$ is the prefix ordering). The semantic clauses for temporal operators are as follows. A *path* X in $\mathcal{M}$ is an infinite sequence $X(0), X(1) \ldots$ such that, for every $i \geq 0$, $R(X(i), X(i+1))$; $X(0)$ is the *root* of X. Let $\mathcal{M} = (W, R, V)$ be a model for CTL^*_{k}, X a path in $\mathcal{M}$ and $i \geq 0$ a position on X:
$\mathcal{M}, X, i \models_{\text{CTL}^*_{\text{k}}} \mathbf{E}\phi$ iff there is a path Y in $\mathcal{M}$ rooted at $X(i)$ such that
$\mathcal{M}, Y, 0 \models_{\text{CTL}^*_{\text{k}}} \phi$;
$\mathcal{M}, X, i \models_{\text{CTL}^*_{\text{k}}} \phi \, \mathbf{U} \, \psi$ iff $\mathcal{M}, X, j \models_{\text{CTL}^*_{\text{k}}} \psi$ for some $j \geq i$, and
$\mathcal{M}, X, k \models_{\text{CTL}^*_{\text{k}}} \phi$ for every $i \leq k < j$;
$\mathcal{M}, X, i \models_{\text{CTL}^*_{\text{k}}} \mathbf{X_j} \phi$ iff $X(i+1) = X(i)j$ and $\mathcal{M}, X, i+1 \models_{\text{CTL}^*_{\text{k}}} \phi$.

Given a temporal logic $\mathbf{T}$, we use $\mathcal{L}_\mathbf{T}$ and $\mathsf{K}_\mathbf{T}$ to denote the language and the set of models of $\mathbf{T}$, respectively. We write $OP(\mathbf{T})$ to denote the set of temporal operators of $\mathbf{T}$.

Temporalization is a simple mode of combining logics in which the two component languages are allowed to interact in a very restricted way [13]. More specifically, let $\mathbf{T_1}$ and $\mathbf{T_2}$ be temporal logics. We partition the set of $\mathbf{T_2}$-formulas into *Boolean combinations* $BC_{\mathbf{T_2}}$ and *monolithic formulas* $ML_{\mathbf{T_2}}$: α belongs to $BC_{\mathbf{T_2}}$ if its outermost operator is a Boolean connective; otherwise, it belongs to $ML_{\mathbf{T_2}}$. We assume that $OP(\mathbf{T_1}) \cap OP(\mathbf{T_2}) = \emptyset$.

Definition 2. (Temporalization – Syntax)

The *combined language* $\mathcal{L}_{\mathbf{T_1}(\mathbf{T_2})}$ of the *temporalization* $\mathbf{T_1}(\mathbf{T_2})$ of $\mathbf{T_2}$ by means of $\mathbf{T_1}$ over the set of proposition letters $\mathcal{P}_\Sigma$ is obtained by replacing the atomic formation rule (P1) of $\mathcal{L}_{\mathbf{T_1}}$ by the following rule: (P1') every monolithic formula $\alpha \in ML_{\mathbf{T_2}}$ is a $\mathcal{L}_{\mathbf{T_1}(\mathbf{T_2})}$-formula. □

A *model* for $\mathbf{T_1}(\mathbf{T_2})$ is a triple $(W, \mathcal{R}, g)$, where $(W, \mathcal{R})$ is a frame for $\mathbf{T_1}$ and $g : W \rightarrow \mathsf{K}_{\mathbf{T_2}}$ a total function mapping worlds in W into models for $\mathbf{T_2}$.

Definition 3. (Temporalization – Semantics)

Given a model $\mathcal{M} = (W, \mathcal{R}, g)$ and a state $w \in W$, the semantics of the temporalized logic $\mathbf{T_1}(\mathbf{T_2})$ is obtained by replacing the semantic clause (S1) for proposition letters of $\mathbf{T_1}$ by the following clause: (S1') $\mathcal{M}, w \models_{\mathbf{T_1}(\mathbf{T_2})} \alpha$ iff $g(w) \models_{\mathbf{T_2}} \alpha$, for every monolithic formula $\alpha \in ML_{\mathbf{T_2}}$. □

Finger and Gabbay [13] study the problem of transferring logical properties, such as axiomatic completeness, decidability, and separability, from two component logics $\mathbf{T_1}$ and $\mathbf{T_2}$ to the temporalized logic $\mathbf{T_1}(\mathbf{T_2})$. Franceschet et al. [16] address the problem of model checking for temporalized logics, proving that complexity upper bounds transfer through temporalization.

In the following, we restrict ourselves to consider temporal logics over frames of the form $(W, \{R\})$, also denoted (W, R), where R is a *binary* accessibility relation on W (the same restriction adopted in [13]).

3 Temporalized automata

In this section, we provide temporalization with an operational counterpart in terms of automata (cf. Definition 6 below). We begin with some preliminary definitions. An *infinite sequence* is a pair (W, R), where W is the set $\mathbb{N}$ of natural numbers and R is the usual ordering $<$ over $\mathbb{N}$. We denote by $\mathcal{S}(\Sigma)$ the set of *Σ-labeled sequences* (W, R, V) such that $(W, R) = (\mathbb{N}, <)$ and $V : W \to \Sigma$. A Σ-labeled sequence (W, R, V) will be also denoted by $x_0, x_1, \ldots$, where $x_i = V(i)$ for every $i \geq 0$.

Definition 4. (String automata)

A *string automaton* consists of (i) a labeled transition system (LTS for short) $(Q, q_0, R, M, \Sigma, \Omega)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Σ and Ω are finite alphabets, $R \subseteq Q \times \Sigma \times Q$ is a transition relation, and $M \subseteq Q \times \Omega$ is a labeling of states, and (ii) an accepting condition AC.

A string automaton accepts Σ-labeled sequences taken form $\mathcal{S}(\Sigma)$. Given an automaton A and a Σ-labeled sequence $x = x_0, x_1, \ldots$, a *run* of A on x is a Q-labeled sequence $q_0, q_1, \ldots$ such that $(q_i, x_i, q_{i+1}) \in R$, for every $i \geq 0$. The automaton A accepts x if there is a run y of A on x such that $AC(y)$, i.e., the accepting condition holds on y. We denote by $\mathcal{L}(A) \subseteq \mathcal{S}(\Sigma)$ the language of A, i.e., the set of Σ-labeled sequences accepted by A. □

Example 5. (Büchi string automata)

A Büchi automaton is a string automaton $A = (Q, q_0, R, M, \Sigma, \Omega)$ such that $\Omega = \{\texttt{final}\}$. We call final a state q such that $(q, \texttt{final}) \in M$. The Büchi acceptance conditions states that A accepts x iff there is a run y of A over x in which at least one final state occurs infinitely often. □

A *class of string automata* $\mathcal{A}$ over an alphabet Σ is a set of automata sharing the alphabet Σ and the accepting condition AC. We say that a class $\mathcal{A}$ of automata is *closed* under an n-ary operation O over $\mathcal{A}$ if $O(A_1, \ldots, A_n) \in \mathcal{A}$ whenever $A_1, \ldots, A_n \in \mathcal{A}$. We say that $\mathcal{A}$ is *effectively closed* under O if $\mathcal{A}$ is closed under O and there is an algorithm that computes $O(A_1, \ldots, A_n)$. Moreover, we say that a class $\mathcal{A}$ of automata is *decidable* if, for every $A \in \mathcal{A}$, the problem $\mathcal{L}(A) = \emptyset$ is decidable.

The link between logics and automata is the following. Let $\mathcal{A}$ be a class of string automata over Σ and $\mathbf{T}$ be a logic interpreted over $\mathcal{S}(\Sigma)$. Given a formula φ of $\mathbf{T}$, we denote by $\mathcal{M}(\varphi)$ the set of models $\mathcal{M} = (W, R, V) \in \mathcal{S}(\Sigma)$ such that $\mathcal{M}, 0 \models \varphi$. A Σ-labeled sequence $(W, R, V) \in \mathcal{S}(\Sigma)$ can be regarded either as a string accepted by some automaton in $\mathcal{A}$ or as a model for some temporal formula in $\mathbf{T}$. A couple of questions naturally arises: given a string automaton $A \in \mathcal{A}$, is there a formula φ_A in $\mathbf{T}$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$? Moreover, given a formula φ in $\mathbf{T}$, is there an automaton $A_\varphi \in \mathcal{A}$ such that $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$? We say that $\mathcal{A}$ is *expressively complete* with respect to $\mathbf{T}$, denoted $\mathbf{T} \to \mathcal{A}$, if, given $\varphi \in \mathcal{L}_\mathbf{T}$, there is $A_\varphi \in \mathcal{A}$ such that $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$; we say that $\mathbf{T}$ is expressively complete with respect to $\mathcal{A}$, denoted $\mathcal{A} \to \mathbf{T}$, if, given $A \in \mathcal{A}$, there is $\varphi_A \in \mathcal{L}_\mathbf{T}$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. Finally, we say that $\mathcal{A}$ is *expressively equivalent* with respect to $\mathbf{T}$, denoted $\mathcal{A} \leftrightarrow \mathbf{T}$, if both $\mathbf{T} \to \mathcal{A}$ and $\mathcal{A} \to \mathbf{T}$. In the following, unless otherwise specified, we will use the word automata to refer to a string automata.

A *sequence of Σ-labeled sequences* is a triple (W, R, g), where $(W, R) = (\mathbb{N}, <)$ and $g : W \to \mathcal{S}(\Sigma)$ is a total function mapping worlds in W into Σ-labeled sequences in $\mathcal{S}(\Sigma)$. We denote by $\mathcal{S}(\mathcal{S}(\Sigma))$ the set of all sequence of Σ-labeled sequences. We are now ready to define the notion of temporalized automaton. In Theorem 11 we will show that

this notion of temporalized automata is indeed the automata-theoretic counterpart of temporalized logics.

Definition 6. (Temporalized automata)

Let Σ be a finite alphabet and let $\mathcal{A}_2$ be a class of automata over Σ. Let $\Gamma(\Sigma)$ be a finite subset of $\mathcal{A}_2$. We can regard any element of $\Gamma(\Sigma)$ as a symbol of an alphabet. Consider a class $\mathcal{A}_1$ of automata over the alphabet $\Gamma(\Sigma)$. Given an automaton A in $\mathcal{A}_1$, a *temporalized automaton* $A^{\downarrow}$ over $\Gamma(\Sigma)$ accepts elements of $\mathcal{S}(\mathcal{S}(\Sigma))$. It has the same LTS than A, and differs because of the accepting condition. The *combined accepting condition* for $A^{\downarrow}$ is the following. Let $x = (W, R, g) \in \mathcal{S}(\mathcal{S}(\Sigma))$. We say that $A^{\downarrow}$ accepts x if and only if there is $y = (W, R, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that $y \in \mathcal{L}(A)$ and, for every $w \in W$, $g(w) \in \mathcal{L}(V(w))$. The class $\mathcal{A}_1(\mathcal{A}_2)$ contains all the temporalized automata $A^{\downarrow}$ over $\Gamma(\Sigma)$ such that A is in $\mathcal{A}_1$. □

In the following, we will assume that $\mathcal{A}_2$ is a class of automata over an alphabet Σ and $\mathcal{A}_1$ is a class of automata over an alphabet $\Gamma(\Sigma) \subseteq \mathcal{A}_2$. Given a temporalized automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$, we denote by $A^{\uparrow}$ the automaton in $\mathcal{A}_1$ with the same LTS than A and with the accepting condition of $\mathcal{A}_1$. Moreover, given an automaton $A \in \mathcal{A}_1$, we denote by $A^{\downarrow}$ the automaton in $\mathcal{A}_1(\mathcal{A}_2)$ with the same LTS than A and with the combined accepting condition of $\mathcal{A}_1(\mathcal{A}_2)$.

Note that a structure $(W, R, g) \in \mathcal{S}(\mathcal{S}(\Sigma))$ may be regarded either as a string accepted by a temporalized automaton or as a model for a temporalized formula. Hence, the notions of expressive completeness and expressive equivalence may be naturally lifted to temporalized logics and temporalized automata. We define a *transfer problem* for temporalized automata as follows: assuming that automata classes $\mathcal{A}_1$ and $\mathcal{A}_2$ enjoy some property, does $\mathcal{A}_1(\mathcal{A}_2)$ enjoy the same property? We investigate the transfer problem with respect to the following properties of automata: (i) (Effective) *closure* under Boolean operations (union, intersection, and complementation): if $\mathcal{A}_1$ and $\mathcal{A}_2$ are (effectively) closed under Boolean operations, is $\mathcal{A}_1(\mathcal{A}_2)$ (effectively) closed under Boolean operations? (ii) *Decidability*: if $\mathcal{A}_1$ and $\mathcal{A}_2$ are decidable, is $\mathcal{A}_1(\mathcal{A}_2)$ decidable? (iii) *Expressive equivalence* with respect to temporal logic: if $\mathcal{A}_1 \leftrightarrow \mathbf{T_1}$ and $\mathcal{A}_2 \leftrightarrow \mathbf{T_2}$, does $\mathcal{A}_1(\mathcal{A}_2) \leftrightarrow \mathbf{T_1}(\mathbf{T_2})$?

The following lemma is crucial in the rest of this section. It shows that every temporalized automaton is equivalent to another temporalized automaton whose transitions are labeled with automata that form a partition of the set of Σ-labeled sequences. Hence, *different* labels in the 'partitioned automaton' correspond to (automata accepting) *disjoint* sets of Σ-labeled sequences. Moreover, the partitioned automaton can be effectively constructed from the original one. A similar partition lemma holds for temporalized logics too (cf. Lemma 10 below).

Lemma 7. (*Partition Lemma for temporalized automata*)

Let A be a temporalized automaton in $\mathcal{A}_1(\mathcal{A}_2)$. If $\mathcal{A}_2$ is closed under Boolean operations (union, intersection, and complementation), then there exists a finite alphabet $\Gamma'(\Sigma) \subseteq \mathcal{A}_2$ and a temporalized automaton A' over $\Gamma'(\Sigma)$ such that $\mathcal{L}(A) = \mathcal{L}(A')$ and $\{\mathcal{L}(X) \mid X \in \Gamma'(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. Moreover, if $\mathcal{A}_2$ is effectively closed under Boolean operations, and $\mathcal{A}_2$ is decidable, then A' can be effectively computed from A.

Proof. To construct $\Gamma'(\Sigma)$ and $\mathcal{A}'$ we proceed as follows. Suppose $\Gamma(\Sigma) = \{X_1, \ldots X_n\} \subseteq \mathcal{A}_2$ and $A = (Q, q_0, R, M, \Gamma(\Sigma), \Omega)$. For every $1 \leq i \leq n$ and $j \in \{0, 1\}$, let $X_i^j = X_i$

if $j = 0$, and $X_i^j = \mathcal{S}(\Sigma) \setminus X_i$ if $j = 1$. Given $(j_1, \ldots, j_n) \in \{0,1\}^n$, let $\mathtt{Cap}_{(j_1,\ldots,j_n)} = \bigcap_{i=1}^{n} X_i^{j_i}$. We define $\Gamma_1(\Sigma)$ as the set of all and only $\mathtt{Cap}_{(j_1,\ldots,j_n)}$ such that $(j_1, \ldots, j_n) \in \{0,1\}^n$. Since $\mathcal{A}_2$ is closed under Boolean operations, $\Gamma_1(\Sigma) \subseteq \mathcal{A}_2$. Moreover, let $\Gamma_2(\Sigma) = \{X \in \Gamma_1(\Sigma) \mid \mathcal{L}(X) \neq \emptyset\}$. We set $\Gamma'(\Sigma) = \Gamma_2(\Sigma)$, and, for $1 \leq i \leq n$, $\Gamma'(\Sigma)(i) = \{X \in \Gamma'(\Sigma) \mid X \cap X_i \neq \emptyset\}$. Note that $\{\mathcal{L}(X) \mid X \in \Gamma'(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. Moreover, for every $1 \leq i \leq n$, $\{\mathcal{L}(X) \mid X \in \Gamma'(\Sigma)(i)\}$ forms a partition of $\mathcal{L}(X_i)$. We define $A' = (Q, q_0, R', M, \Gamma'(\Sigma), \Omega)$, where R' contains all and only the triple $(q_1, X, q_2) \in Q \times \Gamma'(\Sigma) \times Q$ such that there is i with $1 \leq i \leq n$, $X \in \Gamma'(\Sigma)(i)$, and $(q_1, X_i, q_2) \in R$. It is easy to see that $\mathcal{L}(A) = \mathcal{L}(A')$. □

We now prove the first transfer theorem: closure under Boolean operations transfers through temporalized automata.

Theorem 8. (*Transfer of closure under boolean operations*)

Closure under boolean operations (union, intersection, and complementation) transfers through temporalized automata: given two classes $\mathcal{A}_1$ and $\mathcal{A}_2$ of automata which are (effectively) closed under Boolean operations, the class $\mathcal{A}_1(\mathcal{A}_2)$ of temporalized automata is (effectively) closed under Boolean operations.

Proof. Let $X, Y \in \mathcal{A}_1(\mathcal{A}_2)$.

Union We must provide an automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$ that recognizes the language $\mathcal{L}(X) \cup \mathcal{L}(Y)$. Define $A = (X^\uparrow \cup Y^\uparrow)^\downarrow$. We show that $\mathcal{L}(A) = \mathcal{L}(X) \cup \mathcal{L}(Y)$. Let $x = (W, R, g) \in \mathcal{L}(A)$. Hence, there is there is $y = (W, R, V) \in \mathcal{L}(A^\uparrow) = \mathcal{L}(X^\uparrow) \cup \mathcal{L}(Y^\uparrow)$ such that, for every $w \in W$, $g(w) \in \mathcal{L}(V(w))$. Suppose $y \in \mathcal{L}(X^\uparrow)$. It follows that $x \in \mathcal{L}(X)$. Hence $x \in \mathcal{L}(X) \cup \mathcal{L}(Y)$. Similarly if $y \in \mathcal{L}(Y)$.

We now show the opposite direction. Suppose that $x = (W, R, g) \in \mathcal{L}(X) \cup \mathcal{L}(Y)$. If $x \in \mathcal{L}(X)$, then there is $y = (W, R, V) \in \mathcal{L}(X^\uparrow)$ such that, for every $w \in W$, $g(w) \in \mathcal{L}(V(w))$. Hence, $y \in \mathcal{L}(X^\uparrow) \cup \mathcal{L}(Y^\uparrow) = \mathcal{L}(X^\uparrow \cup Y^\uparrow) = \mathcal{L}(A^\uparrow)$. It follows that $x \in \mathcal{L}(A)$. Similarly if $x \in \mathcal{L}(Y)$. Note that, in this case, we exploit only closure under union of $\mathcal{A}_1$-automata.

Complementation We must provide an automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$ that recognizes the language $\mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. Given the Partition Lemma, we may assume that $\{\mathcal{L}(Z) \mid Z \in \Gamma(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. We define $A = (\mathcal{S}(\Gamma(\Sigma)) \setminus X^\uparrow)^\downarrow$. We show that $\mathcal{L}(A) = \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. Let $x = (W, R, g) \in \mathcal{L}(A)$. Hence, there exists $y = (W, R, V) \in \mathcal{S}(\Gamma(\Sigma)) \setminus X^\uparrow$ such that, for every $w \in W$, $g(w) \in \mathcal{L}(V(w))$. Suppose, by contradiction, that $x \in \mathcal{L}(X)$. It follows that there exists $z = (W, R, V') \in \mathcal{L}(X^\uparrow)$ such that, for every $w \in W$, $g(w) \in \mathcal{L}(V'(w))$. Hence, for every $w \in W$, $g(w) \in \mathcal{L}(V(w)) \cap \mathcal{L}(V'(w))$. Since, for every $w \in W$, $\mathcal{L}(V(w)) \cap \mathcal{L}(V'(w)) = \emptyset$ whenever $V(w) \neq V'(w)$, we conclude that $V(w) = V'(w)$. Hence $V = V'$ and thus $y = z$. This is a contradiction since y and z belong to disjoint sets. It follows that $x \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$.

We now show the opposite direction. Let $x = (W, R, g) \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. It follows that, for every $y = (W, R, V) \in \mathcal{L}(X^\uparrow)$, there exists $w \in W$ such that $g(w) \notin \mathcal{L}(V(w))$. Suppose, by contradiction, that $x \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(A)$. It follows that, for every $z = (W, R, V) \in \mathcal{L}(A^\uparrow) = \mathcal{S}(\Gamma(\Sigma)) \setminus \mathcal{L}(X^\uparrow)$, there exists $w \in W$ such that $g(w) \notin \mathcal{L}(V(w))$. We can conclude that, for every $v = (W, R, V) \in \mathcal{S}(\Gamma(\Sigma))$, there exists $w \in W$ such that $g(w) \notin \mathcal{L}(V(w))$. This is a contradiction: since $\{\mathcal{L}(Z) \mid Z \in \Gamma(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$, for every $w \in W$, there is $Y_w \in \Gamma(\Sigma)$ such that $g(w) \in \mathcal{L}(Y_w)$. We have that

(W, R, V'), with $V'(w) = Y_w$, is an element of $\mathcal{S}(\Sigma)$ and, for every $w \in W$, $g(w) \in \mathcal{L}(V'(w))$. We conclude that $x \in \mathcal{L}(A)$.

Intersection Follows from closure under union and complementation using De Morgan's laws. □

Note that, if $A = (X^\uparrow \cap Y^\uparrow)^\downarrow$, then $\mathcal{L}(A) \subseteq \mathcal{L}(X) \cap \mathcal{L}(Y)$, but equivalence does not hold in general. Here is a counterexample of $\mathcal{L}(A) \supseteq \mathcal{L}(X) \cap \mathcal{L}(Y)$. Let $\Gamma(\Sigma) = \{a, b\}$, $X^\uparrow$ be the automata accepting strings starting with an a and $Y^\uparrow$ be the automata accepting strings starting with a b. Then, $X^\uparrow \cap Y^\uparrow = \emptyset$ and hence $\mathcal{L}(A) = \emptyset$. Given $\Sigma = \{0, 1\}$, let a be an automaton accepting strings with an odd number of 1's and b be an automaton recognizing strings with a prime number of 1's. Thus $\mathcal{L}(X) \cap \mathcal{L}(Y)$ contains a combined structure starting with a string with exactly 13 1's, and hence it is not empty.

We now investigate the transfer problem for automata with respect to the decidability property. Given $A \in \mathcal{A}_1(\mathcal{A}_2)$, a sufficient condition for $\mathcal{L}(A) = \emptyset$ is that $\mathcal{L}(A^\uparrow) = \emptyset$. However, this condition is not necessary, since A may be labeled with some $\mathcal{A}_2$-automaton accepting the empty language. Nevertheless, if we know that A is labeled with $\mathcal{A}_2$-automata recognizing non-empty languages, then the condition $\mathcal{L}(A^\uparrow) = \emptyset$ is both necessary and sufficient for $\mathcal{L}(A) = \emptyset$. In the following theorem, we apply these considerations to devise an algorithm that checks emptiness of a temporalized automaton.

Theorem 9. (*Transfer of decidability*)

Decidability transfers through temporalized automata: given two decidable classes $\mathcal{A}_1$ and $\mathcal{A}_2$ of automata, the class $\mathcal{A}_1(\mathcal{A}_2)$ of temporalized automata is decidable.

Proof. Let A be a temporalized automaton in $\mathcal{A}_1(\mathcal{A}_2)$. We describe an algorithm that returns 1 if $\mathcal{L}(A) = \emptyset$ and returns 0 otherwise.

Step 1 Check whether $\mathcal{L}(A^\uparrow) = \emptyset$ using $\mathcal{A}_1$-emptiness algorithm. If $\mathcal{L}(A^\uparrow) = \emptyset$, then returns 1.

Step 2 For every $X \in \Gamma(\Sigma)$, if $\mathcal{L}(X) = \emptyset$ (this check is performed exploiting $\mathcal{A}_2$-emptiness algorithm), then remove symbol X from the LTS of A, i.e., remove every transition of the form (q_1, X, q_2) from the transition relation of A.

Step 3 Let B be the temporalized automaton resulting from A after Step 2. Check, using $\mathcal{A}_1$-emptiness algorithm, whether $\mathcal{L}(B^\uparrow) = \emptyset$. If $\mathcal{L}(B^\uparrow) = \emptyset$, then returns 1, else returns 0.

Note that the above algorithm always terminates returning either 1 or 0. We prove that the algorithm returns 1 whenever $\mathcal{L}(A) = \emptyset$, and returns 0 otherwise. Suppose the algorithm returns 1. If $\mathcal{L}(A^\uparrow) = \emptyset$, then, by definition of combined accepting condition (Definition 6), we have that $\mathcal{L}(A) = \emptyset$. Suppose now that $\mathcal{L}(A^\uparrow) \neq \emptyset$ and $\mathcal{L}(B^\uparrow) = \emptyset$. Note that $\mathcal{L}(A) = \mathcal{L}(B)$, since B is obtained from A cutting off empty automata. Assume, by contradiction, that there is $x \in \mathcal{L}(A)$. Since $\mathcal{L}(A) = \mathcal{L}(B)$, we have that $x \in \mathcal{L}(B)$. Hence $\mathcal{L}(B)$ in not empty, contradicting the fact that $\mathcal{L}(B^\uparrow) = \emptyset$.

We now show that the algorithm returns 0 whenever $\mathcal{L}(A) \neq \emptyset$. Suppose the algorithm returns 0. Then $\mathcal{L}(B^\uparrow)$ contains at least one element, say $x = (W, R, V)$. Since B uses only non-empty $\mathcal{A}_2$-automata as alphabet symbols, we have that, for every $w \in W$, $\mathcal{L}(V(w)) \neq \emptyset$. Hence $y = (W, R, g)$, with g such that, for every $w \in W$, $g(w)$ equals to some element of $\mathcal{L}(V(w))$, is an element of $\mathcal{L}(A)$. □

Finally, we investigate the transfer of expressive equivalence with respect to temporal logics through temporalized automata. We first state a partition lemma for temporalized logics. The proof is similar to the one of the corresponding lemma for temporalized automata.

Lemma 10. (*Partition Lemma for temporalized logics*)

Let φ be a temporalized formula of $\mathbf{T_1}(\mathbf{T_2})$ and $\alpha_1, \ldots, \alpha_n$ be the maximal $\mathbf{T_2}$-formulas of φ. Then, there is a finite set Λ of $\mathbf{T_2}$-formulas such that:

1. *the set $\{\mathcal{M}(\alpha) \mid \alpha \in \Lambda\}$ forms a partition of $\bigcup_{i=1}^{n} \mathcal{M}(\alpha_i)$, and*
2. *the formula φ' obtained by replacing every $\mathbf{T_2}$-formula α_i in φ with $\bigvee\{\alpha \mid \alpha \in \Lambda \wedge \mathcal{M}(\alpha) \cap \mathcal{M}(\alpha_i) \neq \emptyset\}$ is equivalent to φ, i.e., $\mathcal{M}(\varphi) = \mathcal{M}(\varphi')$.*

We now show that expressive equivalence transfers through temporalized automata.

Theorem 11. (*Transfer of expressive equivalence w.r.t. temporal logic*)

Expressive equivalence w.r.t. temporal logic transfers through temporalized automata: if $\mathcal{A}_1 \leftrightarrows \mathbf{T_1}$ and $\mathcal{A}_2 \leftrightarrows \mathbf{T_2}$, then $\mathcal{A}_1(\mathcal{A}_2) \leftrightarrows \mathbf{T_1}(\mathbf{T_2})$.

Proof. We first prove that $\mathcal{A}_1(\mathcal{A}_2) \rightarrow \mathbf{T_1}(\mathbf{T_2})$. Let $A \in \mathcal{A}_1(\mathcal{A}_2)$ be a temporalized automaton. We have to find a temporalized formula $\varphi_A \in \mathbf{T_1}(\mathbf{T_2})$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. Let $\Gamma(\Sigma) = \{X_1, \ldots, X_n\} \subseteq \mathcal{A}_2$ be the combined alphabet of A. Because of the Partition Lemma for temporalized automata, we may assume that $\{\mathcal{L}(X_1), \ldots, \mathcal{L}(X_n)\}$ partitions $\mathcal{S}(\Sigma)$. Since $\mathcal{A}_1 \rightarrow \mathbf{T_1}$, we have a translation τ_1 from $\mathcal{A}_1$-automata to $\mathbf{T_1}$-formulas such that, for every $X \in \mathcal{A}_1$, $\mathcal{L}(X) = \mathcal{M}(\tau_1(X))$. Let $\varphi_{A^\uparrow} = \tau_1(A^\uparrow)$. The formula $\varphi_{A^\uparrow}$ uses proposition letters in $\{p_{X_1}, \ldots, p_{X_n}\}$. Moreover, since $\mathcal{A}_2 \rightarrow \mathbf{T_2}$, we have a translation σ_1 from $\mathcal{A}_2$-automata to $\mathbf{T_2}$-formulas such that, for every $X \in \mathcal{A}_2$, $\mathcal{L}(X) = \mathcal{M}(\sigma_1(X))$. For every $1 \leq i \leq n$, let $\varphi_{X_i} = \sigma_1(X_i)$. For every proposition letter p_{X_i} appearing in $\varphi_{A^\uparrow}$, replace p_{X_i} with φ_{X_i} in $\varphi_{A^\uparrow}$ and let φ_A be the resulting formula. Note that $\varphi_A \in \mathbf{T_1}(\mathbf{T_2})$. We claim that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$.

($\subseteq$) Let $x = (W, R, g) \in \mathcal{L}(A)$. By definition of combined acceptance condition, there is $x^\uparrow = (W, R, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that $x^\uparrow \in \mathcal{L}(A^\uparrow)$ and, for every $w \in W$, $g(w) \in \mathcal{L}(V(w))$. Since $\mathcal{L}(A^\uparrow) = \mathcal{M}(\varphi_{A^\uparrow})$, we have that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow})$. Moreover, for every $w \in W$ and $j = 1, \ldots, n$, we have that $x^\uparrow, w \models p_{X_j}$ iff $V(w) = X_j$. We prove that $V(w) = X_j$ iff $g(w) \in \mathcal{L}(X_j)$. The left to right direction of the previous claim follows since $g(w) \in \mathcal{L}(V(w))$. We prove the right to left direction by contradiction. Suppose $g(w) \in \mathcal{L}(X_j)$ and $V(w) = X_k \neq X_j$. Hence $g(w) \in \mathcal{L}(X_k)$ and thus $g(w) \in \mathcal{L}(X_j) \cap \mathcal{L}(X_k)$. A contradiction, since $\mathcal{L}(X_j) \cap \mathcal{L}(X_k) = \emptyset$. Finally, $g(w) \in \mathcal{L}(X_j)$ iff $g(w) \in \mathcal{M}(\varphi_{X_j})$ iff $x, w \models \varphi_{X_j}$. Summing up, we have that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow})$ and, for every $w \in W$ and $j = 1, \ldots, n$, $x^\uparrow, w \models p_{X_j}$ iff $x, w \models \varphi_{X_j}$. It follows that $x \in \mathcal{M}(\varphi_A)$.

($\supseteq$) Let $x = (W, R, g) \in \mathcal{M}(\varphi_A)$. We define $x^\uparrow = (W, R, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that, for every $w \in W$, $V(w) = X_j$ if and only if $g(w) \in \mathcal{M}(\varphi_{X_j}) = \mathcal{L}(X_j)$. Note that $V(w)$ is always defined, since $\{\mathcal{L}(X_1), \ldots, \mathcal{L}(X_n)\}$ partitions $\mathcal{S}(\Sigma)$. For every $w \in W$ and $j = 1, \ldots, n$, we have that $x^\uparrow, w \models p_{X_j}$ iff $V(w) = X_j$. We prove that $V(w) = X_j$ iff $g(w) \in \mathcal{L}(X_j)$. The left to right direction of the previous claim follows by definition of $x^\uparrow$.The right to left direction follows since $\mathcal{L}(X_j) \cap \mathcal{L}(X_k) = \emptyset$ whenever $k \neq j$. Finally, $g(w) \in \mathcal{L}(X_j)$ iff $g(w) \in \mathcal{M}(\varphi_{X_j})$ iff $x, w \models \varphi_{X_j}$. It follows that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow}) = \mathcal{L}(A^\uparrow)$. Moreover, for every $w \in W$, $g(w) \in \mathcal{M}(\varphi_{V(w)}) = \mathcal{L}(V(w))$. Therefore, $x \in \mathcal{L}(A)$.

We now prove that $\mathbf{T_1}(\mathbf{T_2}) \rightarrow \mathcal{A}_1(\mathcal{A}_2)$. Let $\varphi \in \mathbf{T_1}(\mathbf{T_2})$ be a temporalized formula. We have to find a temporalized automaton $A_\varphi \in \mathcal{A}_1(\mathcal{A}_2)$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi)$. Let $\alpha_1, \ldots, \alpha_n$ be the maximal $\mathbf{T_2}$-formulas of φ. Because of the Partition Lemma for temporalized logics, we may assume that there is a finite set Λ of $\mathbf{T_2}$-formulas such that:

1. the set $\{\mathcal{M}(\alpha) \mid \alpha \in \Lambda\}$ forms a partition of $\bigcup_{i=1}^{n} \mathcal{M}(\alpha_i)$, and
2. every maximal $\mathbf{T_2}$-formula α_i in φ has the form $\bigvee\{\alpha \mid \alpha \in \Lambda \wedge \mathcal{M}(\alpha) \cap \mathcal{M}(\alpha_i) \neq \emptyset\}$.

Let $\varphi^\uparrow$ be the formula obtained from φ by replacing every $\mathbf{T_2}$-formula $\alpha \in \Lambda$ appearing in φ with proposition letter p_α and adding to the resulting formula the conjunct $\beta \vee \neg\beta$, where $\beta = \neg \bigvee_{i=1}^{n} \alpha_i$.

Since $\mathbf{T_1} \rightarrow \mathcal{A}_1$, we have a translation τ_2 from $\mathbf{T_1}$-formulas to $\mathcal{A}_1$-automata such that, for every $\psi \in \mathbf{T_1}$, $\mathcal{M}(\psi) = \mathcal{L}(\tau_2(\psi))$. Let $A_{\varphi^\uparrow} = \tau_2(\varphi^\uparrow)$. The automaton $A_{\varphi^\uparrow}$ labels its transitions with symbols in $\{p_\alpha \mid \alpha \in \Lambda \cup \{\beta\}\}$. Moreover, since $\mathbf{T_2} \rightarrow \mathcal{A}_2$, we have a translation σ_2 from $\mathbf{T_2}$-formulas to $\mathcal{A}_2$-automata such that, for every $\psi \in \mathbf{T_2}$, $\mathcal{M}(\psi) = \mathcal{L}(\sigma_2(\psi))$. For every $\alpha \in \Lambda \cup \{\beta\}$, let $A_\alpha = \sigma_2(\alpha)$. Finally, let A_φ be the automaton obtained by replacing every label p_α on a transition of $A_{\varphi^\uparrow}$ with the automaton A_α. Note that $A_\varphi \in \mathcal{A}_1(\mathcal{A}_2)$. We claim that $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$. The proof is similar to the case $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. □

As a corollary, we have the following:

Corollary 12. *If $\mathcal{A}_1 \leftrightarrows \mathbf{T_1}$, $\mathcal{A}_2 \leftrightarrows \mathbf{T_2}$, $\mathcal{A}_1$ (resp. $\mathbf{T_1}$) and $\mathcal{A}_2$ (resp. $\mathbf{T_2}$) are decidable, then $\mathbf{T_1}(\mathbf{T_2})$ (resp. $\mathcal{A}_1(\mathcal{A}_2)$) is decidable.*

We invite the interested reader to compare the above result with Theorem 3.1 in [13]. Moreover, the following proposition holds:

Proposition 13. *Let $\mathbf{T_i}$, for $i = 1, \ldots, 4$, be temporal logics. If $\mathbf{T_1} \leftrightarrows \mathbf{T_3}$ and $\mathbf{T_2} \leftrightarrows \mathbf{T_4}$, then $\mathbf{T_1}(\mathbf{T_2}) \leftrightarrows \mathbf{T_3}(\mathbf{T_4})$.*

We conclude this section by giving a more general notion of automata: tree automata. Let $k \geq 1$. A *k-ary tree* is a pair (W, R) such that $W = \{0, \ldots, k-1\}^*$ and $<_P$ is the usual prefix ordering over $\{0, \ldots, k-1\}^*$. We denote by $\mathcal{T}_k(\Sigma)$ the set of *Σ-labeled k-ary trees* (W, R, V) such that $(W, R) = (\{0, \ldots, k-1\}^*, <_P)$ and $V : W \rightarrow \Sigma$. A Σ-labeled k-ary tree will be also denoted as a function $x : \{0, \ldots, k-1\}^* \rightarrow \Sigma$, where $x(i) = V(i)$ for every $i \in \{0, \ldots, k-1\}^*$.

Definition 14. (Tree automata)

A *k-ary tree automaton* consists of (i) a labeled transition system (LTS for short) $(Q, q_0, R, M, \Sigma, \Omega)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Σ and Ω are finite alphabets, $R \subseteq Q \times \Sigma \times Q^k$ is a transition relation, and $M \subseteq Q \times \Omega$ is a labeling of states, and (ii) an accepting condition AC.

A k-ary tree automaton accepts Σ-labeled k-ary trees taken form $\mathcal{T}_k(\Sigma)$. Given a k-ary tree automaton A and a Σ-labeled k-ary tree x, a run of A on x is a Q-labeled k-ary tree y such that $(y(i), x(i), y(i0), \ldots, y(i(k-1))) \in R$, for every $i \in \{0, \ldots, k-1\}^*$. The automaton A accepts x if there is a run y of A on x such that $AC(y)$, i.e., the accepting condition holds on y. We denote by $\mathcal{L}(A) \subseteq \mathcal{S}(\Sigma)$ the language of A, i.e., the set of Σ-labeled sequences accepted by A. □

Example 15. (Rabin tree automata)

A k-ary Rabin tree automaton is a k-ary tree automaton $A = (Q, q_0, R, M, \Sigma, \Omega)$ such that $\Omega = \{\texttt{finite}_i, \texttt{infinite}_i \mid 1 \leq i \leq n\}$ for some finite n. The Rabin acceptance conditions states that A accepts x iff there is a run y of A over x such that, for all infinite paths of y, there is $1 \leq i \leq n$ such that all the states labeled with $\texttt{finite}_i$ occurs finitely often and at least one state labeled with $\texttt{infinite}_i$ occurs infinitely often. □

Theorems 8, 9 and 11 immediately generalize to k-ary tree automata. They also hold for finite sequences and finite k-ary trees. Moreover, it is possible to combine string and tree automata. Let $\mathcal{A}_1$ be a class of string automata and $\mathcal{A}_2$ be a class of k-ary tree automata. Automata in the class $\mathcal{A}_1(\mathcal{A}_2)$ accept in the set of sequences of Σ-labeled k-ary trees, denoted $\mathcal{S}(\mathcal{T}_k(\Sigma))$, and automata in the class $\mathcal{A}_2(\mathcal{A}_1)$ accept in the set of k-ary trees of Σ-labeled sequences, denoted $\mathcal{T}_k(\mathcal{S}(\Sigma))$. In the next section, we will show an example of temporalized automata accepting infinite sequences of trees.

4 Combining Büchi and Rabin automata

In this section, we apply temporalized automata to solve a correspondence problem related to temporal logics for time granularity. In [30], Montanari et al. defined the monadic second-order theory of k-refinable downward unbounded layered structures and showed that it can be used to model infinitely refinable granular structures consisting of a coarsest domain and an infinite number of finer and finer domains. In [15], we provided an alternative view of these structures as infinite sequences of infinite k-ary trees. Then, by a suitable application of the Ehrenfeucht game, we defined an expressively complete and elementarily decidable combined temporal logic counterpart of the path fragment of this theory. We left open the problem of finding a (combined) temporal logic able to capture the full power of the monadic second-order theory of infinite sequences of infinite k-ary trees, possibly preserving elementary decidability. In the following, we solve this problem by exploiting a suitable combination of Büchi and Rabin automata.

Let $\mathcal{B}$ be the class of Büchi automata (cf. Example 5) and $\mathcal{R}_k$ be the class of k-ary Rabin automata, with $k \geq 1$ (cf. Example 15 above). We define the class of $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ *automata* as $\mathcal{B}(\mathcal{R}_k)$. A $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automaton accepts tree sequences in $\mathcal{S}(\mathcal{T}_k(\Sigma))$. In this section, we compare the expressiveness of $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automata with that of monadic second-order logic interpreted over tree sequences.

Definition 16. (MSO over tree sequences)

Let $\text{MSO}_\Sigma[<_1, <_2, (\texttt{succ}_i)_{i=0}^{k-1}, +1]$ be the second-order language with equality built up as follows: (i) *atomic formulas* are of the forms $x = y$, $x <_1 y$, $x <_2 y$, $\texttt{succ}_i(x) = y$, $x + 1 = y$, $x \in X$ and $x \in P_a$, where $0 \leq i \leq k-1$, x, y are individual variables, X is a set variable, and $a \in \Sigma$; (ii) *formulas* are built up from atomic formulas by means of the Boolean connectives $\neg$ and $\wedge$, and the quantifier $\exists$ ranging over both individual and set variables.

Let $\mathcal{T}_k = \{0, \ldots k-1\}^*$ and $\mathcal{ST}_k = \mathcal{T}_k \times \mathbb{N}$. Let $\mathcal{ST}_k(\Sigma)$ be the set of all maps $ts : \mathcal{ST}_k \rightarrow \Sigma$. We define two partial orders $<_1$ and $<_2$ over $\mathcal{ST}_k$ as follows: $(x, i) <_1 (y, j)$ iff $x = \epsilon$, $y = \epsilon$, and $i < j$ (where $<$ is the usual ordering over natural numbers), and $(x, i) <_2 (y, j)$ iff $i = j$ and $x <_P y$ (where $<_P$ is the prefix ordering over $\mathcal{T}_k$). Moreover, let $+1$ and, for $0 \leq n \leq k-1$, let $\texttt{succ}_n$ be binary predicates over $\mathcal{ST}_k$ defined as

follows: $(x,i)+1=(y,j)$ iff $x=\epsilon$, $y=\epsilon$ and $j=i+1$ (where $+$ is the sum over natural numbers) and, for $0 \leq n \leq k-1$, $\mathtt{succ}_n((x,i)) = (y,j)$ iff $i=j$ and $y=xn$. Any given tree sequence $ts \in \mathcal{ST}_k(\Sigma)$ can be represented as a relational structure as follows $\hat{ts} = (\mathcal{ST}_k, <_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1, (P_a)_{a\in\Sigma})$, where $P_a = \{(x,i) \in \mathcal{ST}_k \mid ts((x,i)) = a\}$, for every $a \in \Sigma$. Given $ts : \mathcal{ST}_k \rightarrow \Sigma$, let $t_i^{ts} \in \mathcal{T}_k(\Sigma)$ be the i-th tree of ts, i.e., the tree such that, for every $x \in \mathcal{T}_k$, $t_i^{ts}(x) = ts((x,i))$. Is should be clear that $\mathcal{ST}_k(\Sigma)$ is isomorphic to $\mathcal{S}(\mathcal{T}_k(\Sigma))$: given $ts \in \mathcal{ST}_k(\Sigma)$, ts corresponds to a tree sequence $ts' = (W,R,g) \in \mathcal{S}(\mathcal{T}_k(\Sigma))$ such that, for every $i \geq 0$, $g(i) = t_i^{ts}$. Hence, in the following, we will not distinguish between structures ts and ts'.

$\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$-formulas are interpreted over (relational structures corresponding to) tree sequences in $\mathcal{ST}_k(\Sigma)$. A *sentence* of $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ is a formula devoid of free first-order variables and with free second-order variables in $\{P_a \mid a \in \Sigma\}$. Given a sentence φ in $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$, a model for φ is a tree sequence $ts \in \mathcal{ST}_k(\Sigma)$ such that $ts \models \varphi$. We denote by $\mathcal{M}(\varphi)$ the set of models of φ.

Proposition 17. (*Embedding of* $\mathtt{Büchi(Rabin_k)}$ *automata into* MSO)

Let A be a $\mathtt{Büchi(Rabin_k)}$ *automaton. Then, there exists a sentence φ_A in* $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ *such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$.*

Proof. We prove the proposition for $k=2$. The generalization to $k \neq 2$ is straightforward. Let $A = (Q, q_0, R, M, \Gamma(\Sigma), \Omega)$ be the $\mathtt{Büchi(Rabin_k)}$ automaton over $\Gamma(\Sigma)$. We assume $Q = \{0, \ldots m\}$ and $q_0 = 0$. For every $Z \in \Gamma(\Sigma)$, let $Z = (Q_Z, q_Z^0, R_Z, M_Z, \Sigma, \Omega_Z)$, with $Q_Z = \{0, \ldots m_Z\}$, $q_Z^0 = 0$, and $\Omega_Z = \{\mathtt{finite}_i, \mathtt{infinite}_i \mid 1 \leq i \leq r_Z\}$. Let $L_i^Z = \{q \in Q_Z \mid \Omega(q) = \mathtt{finite}_i\}$, and $U_i^Z = \{q \in Q_Z \mid \Omega(q) = \mathtt{infinite}_i\}$, for $1 \leq i \leq r_Z$.

Let $\mathtt{T}^0(x)$ be an abbreviation for $\neg\exists y \bigvee_{i=0}^{k-1} \mathtt{succ}_i(y) = x$ and let $(\epsilon, 0) \in X$ be a shorthand for $\exists x(x \in X \wedge \mathtt{T}^0(x) \wedge \forall y(\mathtt{T}^0(y) \rightarrow ((x=y) \vee x <_1 y)))$. Moreover, let $\mathtt{Path}(W,x)$ be a shorthand for the formula that says "W is a path rooted at x". The formula φ_A of $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ is the following:

$$
\begin{array}{l}
(\exists Q_Z)_{Z\in\Gamma(\Sigma)} (\exists X_i)_{i=0}^m \\
(\bigwedge_{i=0}^m \forall x(x \in X_i \rightarrow \mathtt{T}^0(x)) \wedge \bigwedge_{Z\in\Gamma(\Sigma)} \forall x(x \in Q_Z \rightarrow \mathtt{T}^0(x)) \wedge \\
(\epsilon, 0) \in X_0 \wedge \bigwedge_{i\neq j} \neg\exists y(y \in X_i \wedge y \in X_j) \wedge \\
\forall x(\mathtt{T}^0(x) \rightarrow \bigvee_{(i,Z,j)\in R} (x \in X_i \wedge x \in Q_Z \wedge x+1 \in X_j)) \wedge \\
\bigvee_{i\in F} \forall x(\mathtt{T}^0(x) \rightarrow \exists y(\mathtt{T}^0(y) \wedge x <_2 y \wedge y \in X_i)) \wedge \\
\bigwedge_{Z\in\Gamma(\Sigma)} \forall x(x \in Q_Z \rightarrow \mathtt{RAC}(x,Z))
\end{array}
$$

where $\mathtt{RAC}(x,Z)$ stands for:

$$
\begin{array}{l}
(\exists Y_i)_{i=0}^{m_Z} (\bigwedge_{i=0}^{m_Z} \forall y(y \in Y_i \rightarrow x \leq_1 y) \wedge \\
x \in Y_0 \wedge \bigwedge_{i\neq j} \neg\exists y(y \in Y_i \wedge y \in Y_j) \wedge \forall y(x \leq_1 y \rightarrow \\
\bigvee_{(i,a,j_0,j_1)\in R_Z} (y \in Y_i \wedge y \in P_a \wedge \mathtt{succ}_0(y) \in Y_{j_0} \wedge \mathtt{succ}_1(y) \in Y_{j_1})) \wedge \\
\forall W(\mathtt{Path}(W,x) \rightarrow \\
\bigvee_{i=0}^{r_Z} (\bigwedge_{j\in L_i^Z} \exists u(u \in W \wedge \forall v(v \in W \wedge u <_1 v \rightarrow v \notin Y_j)) \wedge \\
\bigvee_{j\in U_i^Z} \forall u(u \in W \rightarrow \exists v(v \in W \wedge u <_1 v \wedge v \in Y_j)))))
\end{array}
$$

The above sentence encodes the acceptance condition for $\mathtt{Büchi(Rabin_k)}$ automata. The outermost part of the sentence codifies the Büchi acceptance condition for Büchi automaton $A^{\uparrow}$ over the first layer of the tree sequence. It uses letter Q_Z as a landmark for

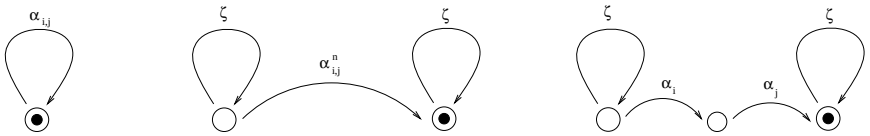


Fig. 1. Combined automata for atomic formulas.

Rabin automaton Z. The innermost part $\mathtt{RAC}(x, Z)$ captures Rabin acceptance condition for Rabin automaton Z over the tree rooted at x. □

Proposition 18. (*Expressive completeness of* $\mathtt{Büchi(Rabin_k)}$ *automata*)

For any sentence φ *of* $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$*, there exists a* $\mathtt{Büchi(Rabin_k)}$ *automaton* A_φ *such that* $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$.

Proof. This proof follows the same pattern than the proof for Büchi Theorem (see [35]). We first remove the ordering relations $<_1$ and $<_2$ as follows:

$$x <_1 y \text{ iff } \forall X(\textstyle\bigwedge_{i=0}^{k-1} \mathtt{succ}_i(x) \in X \wedge \forall z(z \in X \rightarrow \bigwedge_{i=0}^{k-1} \mathtt{succ}_i(z) \in X) \rightarrow y \in X),$$

$$x <_2 y \text{ iff } \mathtt{T}^0(x) \wedge \mathtt{T}^0(y) \wedge \forall X(x+1 \in X \wedge \forall z(z \in X \rightarrow z+1 \in X) \rightarrow y \in X)).$$

Hence $\mathrm{MSO}_\Sigma[<_1, <_2, (\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ is as expressive as $\mathrm{MSO}_\Sigma[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$. In the following, we introduce an equivalent variant of $\mathrm{MSO}_\Sigma[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$, which will be denoted by $\mathrm{MSO}[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$, which uses free set variables X_i in place of predicate symbols P_a. $\mathrm{MSO}[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ considers formulas $\varphi(X_1, \ldots, X_n)$ without predicate symbols P_a and interpret them over tree sequences over the special alphabet $\{0,1\}^n$. With respect to a tree sequence $ts \in \mathcal{ST}_k(\{0,1\}^n)$, the formula $x \in X_i$ says that the letter at node x of ts has 1 in its ith position. By embedding a given alphabet Σ into a set $\{0,1\}^n$ for suitable n, every $\mathrm{MSO}_\Sigma[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$-formula can be easily encoded into an equivalent $\mathrm{MSO}[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$-formula $\varphi(X_1, \ldots, X_n)$.

We now reduce $\mathrm{MSO}[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$ to a simpler formalism $\mathrm{MSO}_0[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$, where only second order variables X_i occur and atomic formulas are of the forms $X_i \subseteq X_j$ (X_i is a subset of X_j), $\mathtt{Succ}_m(X_i, X_j)$, with $m = 0, \ldots, k-1$ (X_i and X_j are the singletons $\{x\}$ and $\{y\}$, respectively, and $\mathtt{succ}_m(x) = y$), and $\mathtt{Succ}(X_i, X_j)$ (X_i and X_j are the singletons $\{x\}$ and $\{y\}$, respectively, and $x + 1 = y$). This step is performed as in the proof of Büchi Theorem.

Finally, given a $\mathrm{MSO}_0[(\mathtt{succ}_i)_{i=0}^{k-1}, +1]$-formula $\varphi(X_1, \ldots, X_n)$, we prove, by induction on the complexity of φ, that there exists a combined automaton $\mathcal{A}_\varphi$ over $\{0,1\}^n$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. As for atomic formulas, let $\alpha_{i,j}$ be the Rabin tree automaton over $\{0,1\}^n$ for $X_i \subseteq X_j$. The combined automaton for $X_i \subseteq X_j$ is depicted in Figure 1 (left). Moreover, let ζ be the Rabin tree automaton over $\{0,1\}^n$ accepting the singleton set containing a tree labeled with 0^n everywhere, and let $\alpha_{i,j}^m$ be the Rabin tree automaton over $\{0,1\}^n$ for $\mathtt{Succ}_m(X_i, X_j)$. The combined automaton for $\mathtt{Succ}_m(X_i, X_j)$ is depicted in Figure 1 (middle). Finally, let α_i be the Rabin tree automaton over $\{0,1\}^n$ accepting the singleton set containing a tree labeled with $0^{i-1}10^{n-i}$ at the root, and labeled with 0^n elsewhere. The combined automaton for $\mathtt{Succ}(X_i, X_j)$ is depicted in Figure 1 (right).

The induction step is clear from the closure of $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automata under Boolean operations and projection. Closure under boolean operations follows from Theorem 8. It is easy to see that $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automata are closed under projection: given a $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automaton A, the corresponding projected $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automaton is obtained by projecting every Rabin automaton that labels some transition of A. □

Propositions 17 and 18 allow us to conclude that, modulo the obvious isomorphism between corresponding structures, $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)} \leftrightarrows \mathrm{MSO}_\Sigma[<_1, <_2, (\texttt{succ}_i)_{i=0}^{k-1}, +1]$. Let QLTL be *Quantified* Linear Temporal Logic and $\mathrm{QCTL}^*_\mathrm{k}$ be *Quantified* Directed Computation Tree Logic [8]. Quantified versions of propositional temporal logics add to the language quantified formulas of the form $\exists Q\varphi$, where Q is a proposition letter and φ is a formula. It is known that $\texttt{Büchi} \leftrightarrows \mathrm{QLTL}$ [1] and $\texttt{Rabin}_\texttt{k} \leftrightarrows \mathrm{QCTL}^*_\mathrm{k}$ [7]. By applying Theorem 11, we have that $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)} \leftrightarrows \mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$. It follows that $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$ is expressively equivalent to $\mathrm{MSO}_\Sigma[<_1, <_2, (\texttt{succ}_i)_{i=0}^{k-1}, +1]$. Since QLTL and $\mathrm{QCTL}^*_\mathrm{k}$ are nonelementarily decidable [1, 33], by Corollary 12, we have that $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$ is nonelementarily decidable. Let EQLTL be the fragment of QLTL consisting of formulas of the form $\exists Q_1 \ldots \exists Q_n \varphi$, where φ is a PLTL-formula. Similarly, we define $\mathrm{EQCTL}^*_\mathrm{k}$. It is easy to show that $\texttt{Büchi} \leftrightarrows \mathrm{EQLTL}$, and thus $\mathrm{EQLTL} \leftrightarrows \mathrm{QLTL}$. Similarly, $\mathrm{EQCTL}^*_\mathrm{k} \leftrightarrows \mathrm{QCTL}^*_\mathrm{k}$. By Proposition 13, we have that $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k}) \leftrightarrows \mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$. Moreover, it is known that both EQLTL and $\mathrm{EQCTL}^*_\mathrm{k}$ are elementarily decidable [34]. Hence, again by Corollary 12, we have that $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ is elementarily decidable. This allows us to conclude that $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ is a temporal logic which is elementarily decidable and expressively equivalent to $\mathrm{MSO}_\Sigma[<_1, <_2, (\texttt{succ}_i)_{i=0}^{k-1}, +1]$, which is the result we were looking for.

5 Applications

In this section we show how to encode (the state-transition representation of) granular reactive systems and mobile reactive systems into combined models (in particular, models for temporalization). Accordingly, combined temporal logics or combined automata may be adopted for the specification of requirements for such systems. This makes it possible to apply combined model checking/satisfiablity procedures to verify granular reactive systems and mobile reactive systems. The main advantage of the combining strategy is that different components are treated as different entities during both the modeling and specifying tasks. Other advantages of using combined tools to verify a system include *modularity* and *flexibility*. Well-known modules, consisting of modeling and specification languages, as well as verification techniques, are combined in such a way that the resulting framework inherits nice features of the components. This improves the reuse of tools and software. Moreover, the result of the combination forms a bag of solutions, instead of a specific tailored one, where each possible solution is obtained by 'plugging' the preferred modules and 'playing' the combined solution. This is particularly useful for mobile systems, which are heterogeneous systems composed of differently specified components.

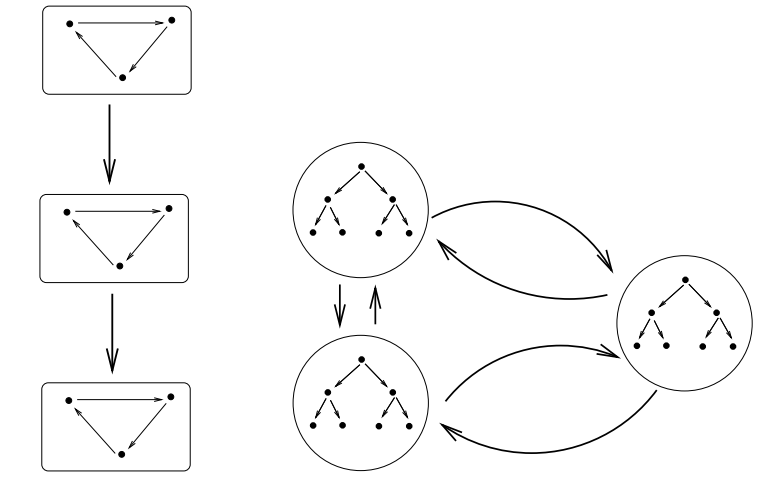
5.1 An Application to Time Granularity

A *granular reactive system* (*GRS*) is a reactive system whose components have dynamic behaviors regulated by very different—even by orders of magnitude—time constants [29].

As an example, consider a pondage power station consisting of a reservoir, with filling and emptying times of days or weeks, generator units, possibly changing state in a few seconds, and electronic control devices, evolving in milliseconds or even less [4]. A complete specification of the power station must include the description of these components and of their interactions. A natural description of the temporal evolution of the reservoir state will probably use days: "During rainy weeks, the level of the reservoir increases 1 meter a day", while the description of the control devices behavior may use milliseconds: "When an alarm comes from the level sensors, send an acknowledge signal in 50 milliseconds". It is somewhat unnatural, and sometimes impossible, to force the specifier of these systems to use a unique time granularity when describing the behavior of all the components. Indeed, a good specification language must allow the specifier to easily describe all simple and intuitively clear facts. Therefore, a specification language for granular reactive systems must support *different time granularities*. Granular logics have been extensively studied in [29–32], where a many-level view of temporal structures, with matching decidability results (based on automata-theoretic techniques), has been proposed.

A GRS can be viewed as a layered set of system components, each one evolving at a different time granularity. Each *system component* can be regarded as a (flat) reactive system: *component states* are represented as sets of propositions that hold at the state; *component computations* are modeled as infinite sequences of component states. The *component behavior* is a set of component computations, and it is usually represented as a Kripke structure. The set of time granularities of the system (and thus the set of system components) is totally ordered on the basis of the degree of fineness (coarseness) of its elements. As an example, consider the set of time granularities including `weeks`, `days`, `seconds`, and `milliseconds`. Time granularities are ordered as follows: `weeks` $\prec$ `days` $\prec$ `seconds` $\prec$ `milliseconds`, where $\mathtt{g}_1 \prec \mathtt{g}_2$ means that $\mathtt{g}_1$ is coarser than $\mathtt{g}_2$. If $\mathtt{g}_1 \prec \mathtt{g}_2$, then there exists a *conversion factor* between the $\mathtt{g}_1$ and $\mathtt{g}_2$ that allows one to map each time point of $\mathtt{g}_1$ into a set of time points of $\mathtt{g}_2$. For instance, the conversion factor between `weeks` and `days` is 7. The behavior of the whole GRS can be described in terms of the synchronized behavior of its components. A *system state* corresponds to the set of states of all system components at/during an instant of the coarsest time granularity; it can be represented as a tree such that the elements of its i-th layer are the states of the i-th system component. Accordingly, a *system computation* can be defined as an infinite sequence of system states (this is the state-based view). Equivalently, a system computation can be expressed by a layered set of component computations, one computation for each component (this is the component-based view). Finally, the *system behavior* is a set of system computations.

The operational behavior of a GRS can be naturally described as a suitable combination of temporal *evolutions* (sequences of component states) and temporal refinements (mapping of a component state into a finite sequence of states belonging to a finer component). According to such a point of view, it can be represented by means of a combined structure. In the following, we propose two simple verification frameworks based on temporalization. A temporalized pointed model $\mathcal{M} = (W, R, g, w_0)$ can be used in both the component-based and a state-based fashion in order to represent the operational behaviour of a GRS. In the *component-based framework*, the 'top' frame (W, R, w_0) models the granular relationships among the different components: wRv means that the component associated with v is a refinement of the one associated with w. The initial state w_0 represents the coarsest component. Moreover, for $w \in W$, the model $g(w)$ captures the in-



(a) component-based (b) state-based

Fig. 2. Frameworks for a GRS

ternal behavior of a single component (Figure 2(a)). Accordingly, the combined temporal logic $\mathbf{T}_1(\mathbf{T}_2)$, or the combined automata class $\mathcal{A}_1(\mathcal{A}_2)$, may be adopted as specification language, where $\mathbf{T}_1$ (resp. $\mathcal{A}_1$) models temporal refinement among different components, and $\mathbf{T}_2$ (resp. $\mathcal{A}_2$) expresses temporal evolution of the single components. *Component-based properties*, that is, formulas that predicate over temporal evolutions, are easily expressible in this framework. A typical component-based property is the following one: "there exists a computation of a system component during which p always holds". This condition can be expressed in PLTL(CTL) by means of the formula: $\mathbf{F}_1\mathbf{E}_2\mathbf{G}_2 p$ (we distinguish PLTL and CTL operators by using different indexes, namely, 1 for PLTL and 2 for CTL).

In the *state-based framework*, the interpretation of the model is different: the 'top' frame (W, R, w_0) models the evolution of the system states (w_0 is the initial system state), while, for $w \in W$, the model $g(w)$ captures the behavior of all temporal refinements of w (Figure 2(b)). The combined temporal logic $\mathbf{T}_1(\mathbf{T}_2)$, or the combined automata class $\mathcal{A}_1(\mathcal{A}_2)$, may be adopted as specification language. As for expressiveness of such a framework, *state-based properties*, which refer to temporal refinements, are immediately expressible. A typical state-based requirement is the following one: "there exists a system state s such that every component state belonging to s satisfies p". This condition can be captured in PLTL(CTL) by means of the formula: $\mathbf{F_1}\mathbf{A}_2\mathbf{G}_2 p$.

5.2 An Application to Mobile Ambients

The *ambient calculus* is a process calculus devised to model *mobile computation* (movement of processes) and *mobile computing* (movement of devices) through administrative domains [3]. The use of modal logic to describe properties of mobile computations has been proposed in [2]. Following [3], we define a *mobile reactive system* (*MRS*) as a reactive system whose processes (may) reside within a hierarchy of locations (*ambient structure*) and modify it (Figure 3). The system may evolve in time by computing (a process in

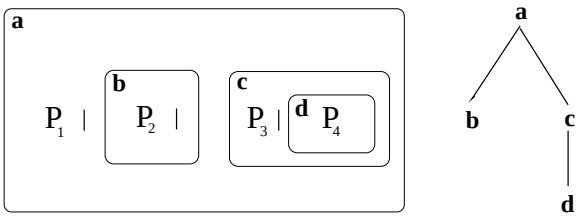


Fig. 3. A Mobile Reactive System and its ambient structure

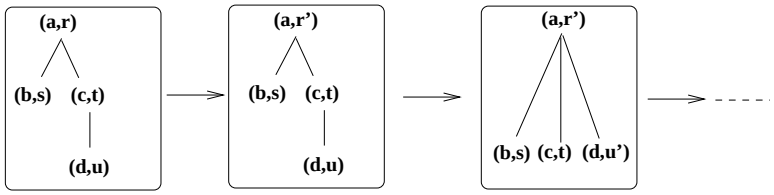


Fig. 4. A temporalized model for an MRS

the system executes one step of computation) or moving (an ambient moves with its subambients inside or outside another ambient). Moreover, a process may dissolve the ambient of another process. The execution of a computation step modifies the program state of the process that executes it, whereas the movement or dissolution of an ambient modifies the ambient structures.

The operational behaviour of an MRS can be naturally described as a suitable combination of two different components: the *spatial* distribution of processes within the ambient structure, and the *temporal* evolution of processes due to computation or movement. According to such a point of view, it can be represented by means of a combined structure. MRSs can be embedded into a number of verification frameworks based on combined structures and languages, which differ from each other in the expressive power and in the level of interaction between components. In the following, we focus on a simple framework based on temporalization. An MRS *state* describes the ambient structure and the program states of the processes residing in each ambient. We model an MRS state as a finite tree, labeled with pairs belonging to $\mathcal{A} \times 2^{\mathcal{P}}$, where $\mathcal{A}$ is a set of ambient names and $\mathcal{P}$ is a set of proposition letters. The label (n, s) represents the fact that the process in ambient n has (program) state s; an edge from (n, s) to (m, t) expresses the fact that m is a subambient of n. The operational behavior of an MRS is modeled by means of a temporalized pointed model (W, R, g, w_0). The frame (W, R) models temporal process evolution (both computation of processes and movement of ambients), and, for every $w \in W$, $g(w)$ models the system state associated with w. Finally, w_0 is associated with the initial system state (Figure 4). A combined temporal logic $\mathbf{T}_1(\mathbf{T}_2)$, or the combined automata class $\mathcal{A}_1(\mathcal{A}_2)$, can be used to specify mobile temporal requirements. Requirements about temporal evolutions of processes may be captured using formulas of $\mathbf{T}_1$ or automata in $\mathcal{A}_1$, and the structure of system states may be specified by means of $\mathbf{T}_2$-formulas or $\mathcal{A}_2$-automata. For instance, the property "there will be a future in which there will be no more viruses everywhere" can be encoded in PLTL(CTL) by means of the formula $\mathbf{F}_1\mathbf{G}_1\mathbf{A}_2\mathbf{G}_2\neg p_{\texttt{virus}}$.

6 Related and future work

In this paper, we proposed a new class of automata, called temporalized automata, which can be obtained by combining simpler automata in a suitable way. We proved that relevant properties, such as closure under Boolean operations, transfer from component automata to temporalized ones. We also showed that a specific class of temporalized automata, namely, $\texttt{Büchi(Rabin}_\texttt{k}\texttt{)}$ automata, is expressively equivalent to the full second-order monadic theory of k-refinable downward unbounded layered structures. Finally, we showed how they can be used to deal with relevant classes of reactive systems, such as granular reactive systems and mobile reactive systems.

The proposed way of combining automata can be naturally viewed as a way of abstracting and concretizing objects. More precisely, we have defined two operations over temporalized automata: the operation $^\uparrow$, that, given a temporalized automaton in the class $\mathcal{A}_1(\mathcal{A}_2)$, returns an automaton in the outermost class $\mathcal{A}_1$, and the operation $^\downarrow$, that, given an automaton in the class $\mathcal{A}_1$, returns an automaton in the combined class $\mathcal{A}_1(\mathcal{A}_2)$. These operations can be interpreted as *abstraction* and *concretization* operations, respectively: the automaton $A^\uparrow$ may be viewed as an abstraction of A, that is, it is a simpler object encoding only part of the information of A, while the automaton $A^\downarrow$ may be viewed as a concretization of A, that is, it is a more complex object encoding more information than A. Thanks to its simple form, the abstraction $A^\uparrow$ can be used to efficiently answer particular queries about A. For instance, to check the emptiness of A one can check the emptiness of $A^\uparrow$. Checking the emptiness of $A^\uparrow$ is in general easier than checking the emptiness of A. If the language of $A^\uparrow$ is empty, then we know for sure that the language of A is empty too. These operations of abstraction and concretization can be generalized to more structured temporalizations. Let $\mathcal{A}$ be a class of automata. We define $\mathcal{A}^1 = \mathcal{A}$, and, for $n \geq 2$, we define $\mathcal{A}^n = \mathcal{A}(\mathcal{A}^{n-1})$. For $A \in \mathcal{A}^n$, we define the i-th abstraction of A as follows: $A^{\uparrow^1} = A^\uparrow$, and, for $i = 2, \ldots, n-1$, $A^{\uparrow^i} = (A^{\uparrow^{i-1}})^\uparrow$. Similarly, we define the i-th concretization of $A \in \mathcal{A}$. Given $A \in \mathcal{A}^n$, it is sometime possible to analyze A by studying its abstractions $A^{\uparrow^1}, \ldots, A^{\uparrow^{n-1}}$, which belong, respectively, to the classes $\mathcal{A}^{n-1}, \ldots, \mathcal{A}^1$. For instance, if $A^{\uparrow^{n-1}}$ accepts the empty language, then every $A^{\uparrow^i}$, for $i = 1, \ldots, n-1$, as well as A, accepts the empty language.

It is worth remarking that some forms of automata combination have been already considered in the literature to increase the expressive power of temporal logics. As an example, extensions of PLTL with connectives defined by means of finite automata over ω-strings are investigated in [37]. To gain the expressive power of the full second-order monadic theory of natural numbers with the usual order, Vardi and Wolper's Extended Temporal Logic (ETL) replaces the until operator of PLTL by an infinite bunch of automata connectives, that is, ETL allows formulas to occur as arguments of an automaton connective (as many formulas as the symbols of the automaton alphabet are). Given the well-known correspondence between formulas and automata, the application of automata connectives to formulas can be viewed as a form of automata combination. Unlike the case of temporalized automata, however, the switch from PLTL to ETL does not involve any change in the domain of interpretation (ω-structures). An extension of CTL* that substitutes ETL operators for PLTL ones, is given in [6]. Also in this case, however, the underlying temporal structure does not change (binary trees). On the contrary, in the case of temporalized logics, component temporal logics refer to different temporal structures, and thus the combination of logics is paired with a combination of their temporal structures.

We are currently exploring the possibility of extending our correspondence results to other forms of logic combination, such as independent combination and join. Furthermore, we are investigating the relationships between temporalized and classical automata. On the one hand, the languages recognized by temporalized automata are structurally different from those recognized by classical automata, e.g., `Büchi(Büchi)` automata recognize infinite strings of infinite strings. On the other hand, this fact does not imply that language problems for temporalized automata cannot be reduced to the corresponding problems for classical automata. As an example, the emptiness problem for `Büchi(Büchi)` automata can be reduced to the emptiness problem for `Büchi` automata.

References

1. A. P. Sistla. *Theoretical Issues in the Design of Distributed and Concurrent Systems*. PhD thesis, Harvard University, Cambridge, MA, 1983.
2. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 365–377, Boston, Massachusetts, January 19–21, 2000.
3. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, June 2000.
4. E. Ciapessoni, E. Corsetti, A. Montanari, and P. San Pietro. Embedding time granularity in a logical specification language for synchronous real-time systems. *Science of Computer Programming*, 20:141–171, 1993.
5. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In Edmund M. Clarke and Robert P. Kurshan, editors, *Proceedings of Computer-Aided Verification (CAV '90)*, volume 531 of *LNCS*, pages 233–242, Berlin, Germany, 1991. Springer.
6. M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126, 1994.
7. E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, June 1984.
8. E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 995–1072. Elsevier Science Publishers B.V., 1990.
9. J. Engelfriet. Minimal temporal epistemic logic. *Notre Dame Journal of Formal Logic*, 37:233–259, 1996.
10. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.
11. K. Fine and G. Schurz. Transfer theorems for multimodal logics. In J. Copeland, editor, *Logic and Reality: Essays on the Legacy of Arthur Prior*, pages 169–213. Oxford University Press, Oxford, 1996.
12. M. Finger. Handling database updates in two-dimensional temporal logic. *Journal of Applied Non-Classical Logics*, 2(2), 1992.
13. M. Finger and D.M. Gabbay. Adding a temporal dimension to a logic system. *Journal of Logic Language and Information*, 1:203–233, 1992.
14. M. Finger and D.M. Gabbay. Combining temporal logic systems. *Notre Dame Journal of Formal Logic*, 37:204–232, 1996.
15. M. Franceschet and A. Montanari. Branching within time: an expressively complete and elementarily decidable temporal logic for time granularity. *Journal of Language and Computation*, 2001. To appear.
16. M. Franceschet, A. Montanari, and M. de Rijke. Model checking for combined logics. In *Proceedings of the 3rd International Conference on Temporal Logic (ICTL)*, 2000. Submitted to the Journal of Logic and Computation.

17. D. M. Gabbay and M. de Rijke, editors. *Frontiers of Combining Systems 2*, volume 7 of *Studies in Logic and Computation*. Research Studies Press/Wiley, 2000.
18. T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In T. Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium*, volume 267 of *LNCS*, pages 269–279. Springer, 1987.
19. Joseph Y. Halpern and Moshe Y. Vardi. The complexity of reasoning about knowledge and time I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.
20. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, Englewood Cliffs, NJ, 1991.
21. G. J. Holzmann and D. Peled. The state of spin. In *8th International Conference on Computer Aided Verification*, number 1102 in LNCS, pages 385–389, New Brunswick, NJ, USA, 1996. Springer Verlag.
22. N. Immerman and D. Kozen. Definability with bounded number of *Information and Computation*, 83(2):121–139, 1989.
23. C. Jard and T. Jeron. On-line model checking for finite linear temporal logic specifications. In *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Lecture Notes on Computer Science, pages 189–196, Grenoble, France, 1989. Springer Verlag.
24. M. Kracht and F. Wolter. Properties of independently axiomatizable bimodal logics. *Journal of Symbolic Logic*, 56(4):1469–1485, 1991.
25. R. Kurshan. *Computer-aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.
26. M. Finger. *Changing the past: database applications of two-dimensional executable temporal logics*. PhD thesis, Imperial College, Department of Computing, 1994.
27. M. Reynolds M. Finger. Two-dimensional executable temporal logic for bitemporal databases. In *Proceedings of Advances in Temporal Logic*, pages 393–411. Kluver Academic, 2000.
28. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$ automata. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages (POPL)*, pages 1–12, 1987.
29. A. Montanari. *Metric and Layered Temporal Logic for Time Granularity*. ILLC Dissertation Series 1996-02, Institute for Logic, Language and Computation, University of Amsterdam, 1996.
30. A. Montanari, A. Peron, and A. Policriti. Decidable theories of ω-layered metric temporal structures. *Logic Journal of the IGPL*, 7(1):79–102, 1999.
31. A. Montanari, A. Peron, and A. Policriti. The taming (timing) of the states. *Logic Journal of the IGPL*, 8(5):681–699, 2000.
32. A. Montanari and A. Policriti. Decidability results for metric and layered temporal logics. *Notre Dame Journal of Formal Logic*, 37:260–282, 1996.
33. P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Standford University, Palo Alto, CA, 1982.
34. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Buchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.
35. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 133–191. Elsevier Science Publishers, 1990.
36. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS '86)*, pages 332–345, Washington, D.C., USA, 1986. IEEE Computer Society Press.
37. M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.