Reasoning about Finite-State Switched Systems

Dana Fisman[1,2⋆] and Orna Kupferman[1]

[1] School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel.
[2] IBM Haifa Research Lab, Haifa University Campus, Haifa 31905, Israel.

Abstract. A *switched system* is composed of components. The components do not interact with one another. Rather, they all interact with the same environment, which switches one of them on at each moment in time. In standard concurrency, a component restricts the environment of the other components, thus the concurrent system has fewer behaviors than its components. On the other hand, in a switched system, a component suggests an alternative to the other components, thus the switched system has richer behaviors than its components.
We study finite-state switched systems, where each of the underlying components is a finite-state transducer. While the main challenge, namely compositionality, is similar in standard concurrent systems and in switched systems, the problems and solutions are different. In the verification front, we suggest and study an assume-guarantee paradigm for switched systems, and study formalisms in which satisfaction of a specification in all components imply its satisfaction in the switched system. In the synthesis front, we show that while compositional synthesis and design are undecidable, the problem of synthesizing a switching rule with which a given switched system satisfies an LTL specification is decidable.

1 Introduction

Concurrent systems are composed of components. Traditional concurrency theory considers two types of concurrent composition operators: *synchronous parallel composition* and *asynchronous parallel composition* (a.k.a. *interleaving*). In the former the components proceed simultaneously and in the latter their behaviors are interleaved. In both, the components not only interact with the environment but also with one another. There are, however, many natural settings in which components do not interact with one another. Rather, at each moment in time one of the components determines the behavior of the system, while the other components are ignored. Such a "switching semantics" has been well-studied in the engineering community [12, 13]. In this paper, we study it for finite-state systems.

Given finite-state transducers $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$, all interacting with the same environment, we define the *switched system* $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ as a transducer that proceeds, in each moment in time, according to one of the underlying transducers.[3] There are two natural definitions for the $\oplus$ operator. In a *dormant* composition, components that are suspended are not active. That is, when a component is switched on again, it proceeds

⋆ The work of this author was done as part of the Valazzi-Pikovsky Fellowship Fund.

[3] A *transducer* is an input/output finite state machine, formally defined in Section 2. We use transducers to model concurrent systems.

from the state it has reached before its suspension. In an *active* setting, components that are suspended continue their dynamics and have full observability of the environment, but their output is ignored. [4]

As an example to a dormant composition, consider a window system; at each moment in time, several windows are open and the location of the mouse determines which window is active. The other windows are inactive. We would like the window system to have the property that if a keyboard input occurs while the active window is in an "insert password" subroutine, then the char * is displayed; and if the pressed key is "enter", then the last typed chars are matched against the correct password. Note that this property should hold even if the window system switches among different window while the user types. This example shows that, even in the dormant setting, the configuration of components that are switched off should be maintained.

As an example for a switched system with an active composition, consider a network of security cameras. The cameras are located in several locations, and each camera is equipped with a software analyzing the picture. If a suspicious behavior is detected by the software, the picture is frozen until another suspicious behavior is detected. At each moment in time the output of one of the cameras is displayed at the guard's control screen. We would like to reason about the switched system and the various possible switching rules for it. For example, under an arbitrary switching rule, the system does not satisfy the property "all suspicious behaviors are detected," and it does satisfy the property "if suspicious behaviors are detected simultaneously in all locations, then at least one of them is displayed on the control screen". Also, under certain assumptions, like the configuration of the building and the location of the cameras, it is possible to synthesize a switching rule with which at least one frozen picture of a sequence of suspicious behaviors is displayed. As another example to the active composition, consider a channel TV. Obviously, broadcasting continues (but is ignored) for channels that are switched off. Using the setting of switched systems, we can reason about properties of the entire system. For example, if we are an advertising agency, we would like to synthesize an advertisement scheduling so that a viewer may not be able to avoid advertisement no matter what his switching rule is.

Finite-state switched systems, as defined above, may also serve as an abstraction of other, not necessarily finite-state, switched systems. Examples to switched systems include software systems (c.f., internet communication protocols [7, 22]), mechanical systems (engines with gear transmission [8]), electrical circuits (power converters [6]), biological systems (gene regulating networks [3]), and embedded systems combining the above [2]. There has been extensive research in the control engineering community on analysis of continuous switched systems whose evolution is described by means of differential equations [12, 13]. The study there focuses on properties such as stability.

[4] Dormant switched systems may seem similar to co-routines. A *co-routine* specifies several points in the code, referred to as *yield points*. When the scheduler is invoked, it passes control to one of the co-routines that are at their yield point. Thus, as in dormant switched systems, when a component is reinvoked it continues from the state it has reached when last invoked (rather than from the initial state as in ordinary routines). Unlike switched systems, however, the components do have control on when the scheduler is invoked. Anyway, the theoretical aspects of co-routers have not been investigated.

The theory of verification considers other type of properties, those expressible in temporal logic. Thus, considering abstraction of continuous systems enables reasoning about other aspects of systems. For example, consider a cell phone that may move among different receiving zones. This is a popular example for continuous switched systems [14], yet many properties of the system can be specified in temporal logic. For example, we would like to check that whenever a *network available* signal appears, it stays valid as long as the cell phone does not change its location, and that if a call was issued, then eventually either the network is no longer available or the call gets to the target phone. These properties should hold even if the cell phone changes its location. Such a setting corresponds to the dormant composition – the operation of the cell phone in a particular zone is a component (note that the cell phone operates differently in different zones), and transiting among the zones correspond to switching.

The above examples highlight the *Gestalt principle*, which is accepted in the study of continuous switched systems. According to this principle "the sum of the whole is greater than its parts". For example, a continuous switched system may be stable even though its underlying components are not stable, and vice versa, a continuous switched system may be unstable even though its underlying components are stable [13]. This is in contrast with standard concurrency, where the concurrent system has fewer behaviors than its components. The fact that the composed system has fewer behavior than its components has played a central role in compositional reasoning. As shown in [1], both synchronous and asynchronous parallel compositions can be seen as intersection of the enhanced language of its components. Further classes of parallel compositions have been studied in [1]. They all, however, convey a notion of intersection between languages. As we shall show, our dormant and active compositions convey a notion of union rather than intersection. Thus, general ideas and patterns that are applicable in the study of standard concurrency cannot be applied in the setting of switched systems.[5]

As in standard concurrency, composing finite-state transducers via active or dormant compositions involves an exponential blowup. Thus, the main challenge in reasoning about finite-state switched systems is *compositional reasoning*, i.e., reducing reasoning about a concurrent system to reasoning about its individual components. While the main challenge, namely compositionality, is similar in switched systems and standard concurrent systems, the problems and solutions are different. We start by studying the compositional model-checking problem for switched systems. As noted above, an algorithm that constructs the switched system explicitly is possible. We show that the space complexity of LTL model checking is polynomial in the size of the underlying components, thus the exponential blow-up that the construction of an explicit switched system involves cannot in general be avoided.

[5] By letting the variables of the different components be disjoint, it is possible to model the dormant and active compositions using known synchronous and asynchronous composition operators. Such a modeling, however, is less clean, and hides the switching mechanism. In [15], Mayer and Stockmeyer studied regular expressions extended with a *shuffle* operator on words, which interleaves its operands. As we show later, the shuffle operator corresponds to a dormant composition between closed systems. Our setting here is richer, as it considers open systems. We also study different problems than those studied in [15].

Note that since a component may be switched on forever, a required condition for a switched system to satisfy a property is that all the underlying components satisfy it. For some properties, this is also a sufficient condition, giving rise to a simple compositional model-checking procedure for them that avoids this blow-up. We characterize such properties for the dormant composition by means of *regular counting* properties, and conclude that, unfortunately, most interesting properties cannot enjoy this simple procedure. We then describe an assume-guarantee paradigm for switched systems [16], which enables us to reason about a switched system (with respect to all LTL specifications) by reasoning about its components, and often avoid this blow up. Formally, a transducer $\mathcal{T}$ satisfies the assume-guarantee specification $\langle \varphi, \psi \rangle$, for LTL specifications φ and ψ, and a composition operator $\oplus$, if for all transducers $\mathcal{T}'$, if $\mathcal{T} \oplus \mathcal{T}'$ satisfies φ, then $\mathcal{T} \oplus \mathcal{T}'$ also satisfies ψ. We study the problem of checking assume-guarantee specifications and show that it is PSPACE-complete. Unlike traditional concurrency, the problem cannot be reduced to checking whether $\mathcal{T}$ satisfies $\varphi \rightarrow \psi$ [16]. Indeed, the latter reduction depends on the fact that compositions that have $\mathcal{T}$ as a component have fewer behaviors than $\mathcal{T}$, which does not hold for switched systems. We show that for switched systems checking whether $\mathcal{T}$ satisfies the assume-guarantee specification $\langle \varphi, \psi \rangle$ has the flavor of checking validity of $\varphi \rightarrow \psi$. This is due to monotonicity that does hold for switched systems as well.

The model-checking problem checks whether a given switched system satisfies a specification under arbitrary switching. A more ambitious goal is synthesis – the automatic construction of systems from specifications. In the switched setting, given LTL specifications $\varphi_1, \varphi_2, \ldots, \varphi_n$, and ψ, and a composition operator $\oplus$, the *compositional-realizability* problem is to decide whether there are transducers $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ such that $\mathcal{T}_i$ satisfies φ_i for all $1 \leq i \leq n$, and $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ satisfies ψ. On the negative side, we show that, as with standard concurrency [18] compositional-realizability is undecidable. Sometimes, the details of the switching mechanism are known and may be controlled. On the positive side, we study the problem of synthesizing a *switching rule* according to which the switching system satisfies a specification. We show that the problem has the same flavor as the standard LTL control problem, and is 2EXPTIME-complete [17]. The solution to the problem, however, is different, as the synthesized switched rule does not disable transitions, as is the case in usual control. Rather, it chooses the component that is switched on.

2 Transducers and Switched Systems

Let I and O be finite sets of input and output signals. Let Σ_{I} and Σ_{O} denote the sets 2^I and 2^O, respectively. Let Σ_{IO} denote the set $\Sigma_{\mathrm{I}} \times \Sigma_{\mathrm{O}}$. A *transducer* is an automaton on finite words over the alphabet Σ_{I} in which each state is associated with a letter in the alphabet Σ_{O}. A transducer does not have an acceptance condition. The intuition is that the transducer models an *open system* that interacts with its environment. In each moment in time the system reads a set $i \in \Sigma_{\mathrm{I}}$ of input signals that are valid in this moment, changes its state according to i, and outputs a set $o \in \Sigma_{\mathrm{O}}$ of output signals that are valid in the new state.

Formally, a transducer is a tuple $\mathcal{T} = \langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S, \theta, \eta, L \rangle$, where S is a set of states, $\theta : \Sigma_{\mathrm{I}} \rightarrow S$ is an initialization function mapping the first input letter to an initial state, $\eta : S \times \Sigma_{\mathrm{I}} \rightarrow S$ is a transition function, and $L : S \rightarrow \Sigma_{\mathrm{O}}$ is a labeling function. The *run*

of $\mathcal{T}$ on an input sequence $i_0 \cdot i_1 \cdot i_2 \cdots \in {\Sigma_{\mathrm{I}}}^{\omega}$ is the sequence $s_0, s_1, s_2, \ldots$ of states for which $s_0 = \theta(i_0)$ and $s_{j+1} = \eta(s_j, i_{j+1})$ for all $j \geq 0$. A computation $w \in {\Sigma_{\mathrm{IO}}}^{\omega}$ is *generated* by $\mathcal{T}$ if $w = (i_0, o_0) \cdot (i_1, o_1) \cdot (i_2, o_2) \cdots$ is such that the run $s_0, s_1, s_2, \ldots$ of $\mathcal{T}$ on $i_0 \cdot i_1 \cdot i_2 \cdots$ satisfies $o_j = L(s_j)$ for all $j \geq 0$. We refer to the set of computations generated by $\mathcal{T}$ as the *language* of $\mathcal{T}$ and denote it $\mathcal{L}(\mathcal{T})$. Note that $\mathcal{T}$ is responsive and deterministic (that is, it suggests exactly one successor state for each input letter), and thus $\mathcal{T}$ has a single run, generating a single computation, on each input sequence.

A *switched system* is composed of several components. Each component is an open system that interacts with the environment. The components do not interact with each other. Rather, they all interact with the environment, but only one component, chosen by the environment, is *switched on* at a given moment. The other components are *suspended*. We define two types of compositions between transducers. In a *dormant* composition, components that are suspended are not active. That is, when a component is switched on again, it proceeds from the state it has reached in the last time it was switched on. In an *active* setting, components that are suspended continue their dynamics and have full observability of the environment, but their output is ignored.

We formalize the two types of compositions below. For simplicity we assume systems with two components. The generalization to any finite number of components is straightforward. Let $\mathcal{T}_1 = \langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S_1, \theta_1, \eta_1, L_1 \rangle$ and $\mathcal{T}_2 = \langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S_2, \theta_2, \eta_2, L_2 \rangle$ be two transducers. We define the *dormant* and *active* switched systems with components $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted $\mathcal{T}_1 \textcircled{\scriptsize D} \mathcal{T}_2$, and $\mathcal{T}_1 \textcircled{\scriptsize A} \mathcal{T}_2$, respectively, as the transducer $\langle \Sigma_{\mathrm{I}'}, \Sigma_{\mathrm{O}}, S, \theta, \eta, L \rangle$, defined as follows.

- $\Sigma_{\mathrm{I}'} = \Sigma_{\mathrm{I}} \times \{1, 2\}$. The $\{1, 2\}$ component of an input letter indicates which component will be switched on in the next cycle. We use $\langle i, who \rangle$ to refer to a letter in $\Sigma_{\mathrm{I}'}$ where $i \in \Sigma_{\mathrm{I}}$ and $who \in \{1, 2\}$. We can think of who as a fresh input signal defined over the domain $\{1, 2\}$.
- $S = S_1 \times S_2 \times \{1, 2\}$. That is, a state in the switched system is composed of the states of $\mathcal{T}_1$ and $\mathcal{T}_2$, and a flag indicating the component that is currently switched on. This component generates the current output.
  In the dormant composition, it is technically convenient to add to S_1 and S_2 a special state s_{init}, for components that have never been activated.
- The initialization function θ is defined as follows.
  - $\textcircled{\scriptsize D}$ In the dormant composition, the component that has never been switched on waits in the special state s_{init} until it is switched on for the first time. Accordingly, $\theta(\langle i, 1 \rangle) = \langle \theta_1(i), s_{init}, 1 \rangle$ and $\theta(\langle i, 2 \rangle) = \langle s_{init}, \theta_2(i), 2 \rangle$.
  - $\textcircled{\scriptsize A}$ In an active composition, the component that is not switched on proceeds as if it was active. Thus, $\theta(\langle i, who \rangle) = \langle \theta_1(i), \theta_2(i), who \rangle$.
- The transition function η is defined according to the type of composition as follows. Consider a state $\langle s_1, s_2, k \rangle \in S$ and an input letter $\langle i, who \rangle \in \Sigma_{\mathrm{I}'}$.
  - $\textcircled{\scriptsize D}$ In a dormant composition, the component that is suspended stays in its current state until it is switched on again. Thus,

$$\eta(\langle s_1, s_2, k \rangle, \langle i, who \rangle) = \begin{cases} \langle \eta_1(s_1, i), s_2, who \rangle & \text{if } who = 1 \\ \langle s_1, \eta_2(s_2, i), who \rangle & \text{if } who = 2 \end{cases}$$

  In addition, for $who \in \{1, 2\}$, we have $\eta_{who}(s_{init}, i) = \theta_{who}(i)$.

- ◐ In an active composition, the component that is suspended proceeds as if it was active. Thus,

$$\eta(\langle s_1, s_2, k\rangle, \langle i, who\rangle) = \langle \eta_1(s_1, i), \eta_2(s_2, i), who\rangle.$$

– For all states $\langle s_1, s_2, k\rangle \in S$, we have $L(\langle s_1, s_2, k\rangle) = L_k(s_k)$. That is, the output of the current state is determined by the component that is switched on.

Note that the underlying transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ do not have who in their set of input signals. Thus, a component does not know whether it is switched on or not, and its behavior does not depend on this information.

A specification to the switched-system is over the set $I \cup O$ of signals. By allowing specifications to refer also to the signal who, we can easily restrict attention to compositions in which assumptions on the switching can be made. Formally, since our specification formalism is linear, we can replace a specification ψ over $I \cup O$ by the specification $\psi_{fair} \rightarrow \psi$, where ψ_{fair} is a formula over who describing assumptions on the switching. We will elaborate on the extended setting for problems studied in the following sections.

2.1 The input-output language of a switched system

Recall that the language $\mathcal{L}(\mathcal{T})$ of a transducer $\mathcal{T}$ is defined over the alphabet Σ_{IO}. Accordingly, $\mathcal{L}(\mathcal{T}_1 \oplus \mathcal{T}_2)$ refers also to the input signal who, which we often want to abstract. For a switched system, we also define the *IO-language* of $\mathcal{T}_1 \oplus \mathcal{T}_2$, denoted $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus \mathcal{T}_2)$, which is obtained by projecting $\mathcal{L}(\mathcal{T}_1 \oplus \mathcal{T}_2)$ on Σ_{IO} (that is, ignoring the $\{1, 2\}$ component).

In Lemma 1 below we show that natural properties of the interleaving operator used in standard concurrent composition apply also to switched systems. On the other hand, it is not hard to see that unlike the case of interleaving, it is not necessarily the case that $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus \mathcal{T}_2) \subseteq \mathcal{L}(\mathcal{T}_1)$ or $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus \mathcal{T}_2) \subseteq \mathcal{L}(\mathcal{T}_2)$.

Lemma 1. *Let $\oplus \in \{◫, ◐\}$ be a composition operator. For all transducers $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{T}_3$, the following hold.*

– *Commutativity: $\mathcal{L}(\mathcal{T}_1 \oplus \mathcal{T}_2) = \mathcal{L}(\mathcal{T}_2 \oplus \mathcal{T}_1)$.*
– *Associativity: $\mathcal{L}_{\text{IO}}((\mathcal{T}_1 \oplus \mathcal{T}_2) \oplus \mathcal{T}_3) = \mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus (\mathcal{T}_2 \oplus \mathcal{T}_3))$.*
– *Monotonicity: If $\mathcal{L}(\mathcal{T}_1) \subseteq \mathcal{L}(\mathcal{T}_2)$ then $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus \mathcal{T}_3) \subseteq \mathcal{L}_{\text{IO}}(\mathcal{T}_2 \oplus \mathcal{T}_3)$ for all $\mathcal{T}_3$.*

It is not hard to see that, when restricted to their IO-languages, the dormant and active compositions corresponds to the *shuffle* and *merge* of languages. For two words $u, v \in \Sigma^\omega$, let

– $u ◫ v = \{u_1 v_1 u_2 v_2 u_3 v_3 \cdots \mid u_i, v_i \in \Sigma^*, u = u_1 u_2 u_3 \cdots \text{ and } v = v_1 v_2 v_3 \cdots\}$
– $u ◐ v = \{u_1 v_2 u_3 v_4 \cdots \mid u_i, v_i \in \Sigma^*, |u_i| = |v_i|, u = u_1 u_2 u_3 \cdots \text{ and } v = v_1 v_2 v_3 \cdots\}$

Thus, $u ◫ v$ shuffles the letters of u and v by interleaving subwords of u and v, whereas $u ◐ v$ merges u and v by locating in each position i the i-th letter of either u or v. Note that since the subwords v_i and u_i may be empty, we have that u and v are members of $u ◫ v$, and similarly for $u ◐ v$. The standard concurrency operator, a.k.a *interleaving*, is

often confused with shuffle, though its operation is different. Indeed, as shown in [1], since interleaving is applied to components that own variables, it corresponds to conjunction of the enhanced language of its components [1]. This is not valid for the shuffle operation. The shuffle and merge operators naturally extends to languages. In Figure 1, we demonstrate the application of the shuffle and merge operators on some languages.

L_1	L_2	$L_1 \shuffle L_2$	$L_1 \merge L_2$
0^ω	1^ω	$(0+1)^\omega$	$(0+1)^\omega$
$0^\omega + 1^\omega$	$0^\omega + 1^\omega$	$(0+1)^\omega$	$(0+1)^\omega$
$(01)^\omega$	$(10)^\omega$	$((01)+(10))^\omega$	$(0+1)^\omega$
$(01)^\omega$	$(01)^\omega$	$0((01)+(10))^\omega$	$(01)^\omega$
0^ω	0^*10^ω	$0^\omega + 0^*10^\omega$	$0^\omega + 0^*10^\omega$
0^*10^ω	0^*10^ω	$0^*10^\omega + 0^*10^*10^\omega$	$0^\omega + 0^*10^\omega + 0^*10^*10^\omega$
$0^*1(0+1)^\omega$	$0^*1(0+1)^\omega$	$0^*1(0+1)^\omega$	$(0+1)^\omega$
$0^+1(0+1)^\omega$	$0^+1(0+1)^\omega$	$0^+1(0+1)^\omega$	$0(0+1)^\omega$
$(0+1)^*0^\omega$	$(0+1)^*0^\omega$	$(0+1)^*0^\omega$	$(0+1)^*0^\omega$
$(0^*1)^\omega$	$(0^*1)^\omega$	$(0^*1)^\omega$	$(0+1)^\omega$

Fig. 1. Shuffle and merge of languages.

The definition of the dormant and active composition immediately implies their correspondence to the shuffle and merge operators. Formally, we have the following.

Lemma 2. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transducers. Then, $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \shuffle \mathcal{T}_2) = \mathcal{L}_{\text{IO}}(\mathcal{T}_1) \shuffle \mathcal{L}_{\text{IO}}(\mathcal{T}_2)$ and $\mathcal{L}_{\text{IO}}(\mathcal{T}_1 \merge \mathcal{T}_2) = \mathcal{L}_{\text{IO}}(\mathcal{T}_1) \merge \mathcal{L}_{\text{IO}}(\mathcal{T}_2)$.*

In [15] it was shown that the shuffle operator provides succinctness in the sense that there exist languages that can be described exponentially more succinctly by using shuffle.[6] We show that the results extends for dormant composition and holds for active composition as well.

Theorem 1. *Let $n \in \mathbb{N}$. There are transducers $\mathcal{T}_1, \ldots, \mathcal{T}_n$ such that the size of $\mathcal{T}_i$ is $O(1)$, and there is no transducer $\mathcal{T}$ with less than 2^{n-1} states such that $\mathcal{L}(\mathcal{T}) = \mathcal{L}_{\text{IO}}(\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n)$.*

The idea of the proof is to show that for any $n \in \mathbb{N}$ the set of all words over the alphabet $\Gamma_n = \{1, \ldots, n, \#\}$ in which each letter from $\{1, 2, \ldots, n\}$ appears at most once can be expressed as an active or dormant compositions of n transducers. By [15], this language cannot be generated by a transducer with less than 2^{n-1} states. The full proof is given in the full version of the paper.

3 Compositional Model Checking

The model-checking problem for a switched system is to decide, given transducers $\mathcal{T}_1, \ldots, \mathcal{T}_n$, a composition operator $\oplus \in \{\shuffle, \merge\}$, and an LTL formula ψ, whether

[6] [15] refers to shuffle also as interleaving. Their definition, however, corresponds to shuffle as defined above.

the switched system $\mathcal{T}_1 \oplus \cdots \oplus \mathcal{T}_n$ satisfies ψ. Note that the formulation of the problem has an implicit universal quantification and the switched system has to satisfy the specification under arbitrary switching. As with the interleaving operator, it is possible to construct $\mathcal{T}_1 \oplus \cdots \oplus \mathcal{T}_n$ and model check it. As shown in Theorem 1, however, such a construction may involve an exponential blow up. Assume-guarantee reasoning avoids the blowup by inferring satisfaction of specifications in the composed system from satisfaction of specifications in the underlying components [16].

Note that since a component may be switched on forever, a required condition for a switched system to satisfy a property is that all the underlying components satisfy it. For some properties, this is also a sufficient condition, giving rise to a simple compositional model-checking procedure for them. In Section 5, we characterize such properties for the active composition. Since most interesting properties do not satisfy the characterization, we describe, in this section, an assume-guarantee paradigm for switched systems for arbitrary properties. We first show that, as with interleaving, the blow-up that the construction of $\mathcal{T}_1 \oplus \cdots \oplus \mathcal{T}_n$ involves cannot be avoided. We do so by analyzing the system-complexity of the model-checking problem, namely the complexity of the system in terms of the size of the underlying components, assuming the specification is fixed.

Theorem 2. *The system complexity of the LTL model-checking problem of switched systems is PSPACE-complete.*

Remark 1. The key to the PSPACE-hardness result is the fact that even though the components interact with the environment one at a time, they resume their interaction from a state that has to be maintained (either the state they have reached in the last time they were switched on, in a dormant composition, or the state they have reached in their silent interaction, in an active composition). A substantially different type of composition is one in which interaction is resumed from a fixed state. Then, it is possible to define the state space of the switched systems as a union of the underlying state spaces, and the system complexity of the LTL model-checking problem is NLOGSPACE complete. Fixing a state from which dynamics is resumed is even more crucial in the infinite-state setting. For example, reachability in o-minimal hybrid systems is decidable only when each discrete control state has a single initial value for the continues elements [11]. Obviously, however, resuming the interaction from a fixed state is a much weaker composition mechanism.

Remark 2. In [15], Mayer and Stockmeyer studied the complexity of membership and inequality for regular expressions extended with the shuffle operator, which as we discussed previously provides the dormant composition operator in the setting of closed system. They showed that membership is NP-complete and inequality is EXPSPACE-complete. Since equivalence is two-sided inclusion and since model checking amounts to inclusion (the language of the system should be contained in the language of the formula), their results imply that model checking of closed system restricted to finite words can be done in EXPSPACE. As Theorem 2 shows, the special case of the $A \subseteq B$ problem in which only B uses shuffle is easier, and is in PSPACE, even for the case of open systems and infinite words. Indeed, the lower bound proof in [15] uses shuffle in both sides.

We are now ready to describe an assume-guarantee paradigm for switched systems.

Definition 1. *Let $\mathcal{T}$ be a transducer. Let φ_1 and φ_2 be temporal logic formulas. Let $\oplus \in \{\oplus, \blacklozenge\}$ be a composition operator. We say that $\langle\varphi_1\rangle\mathcal{T}\oplus\langle\varphi_2\rangle$ if for every $\mathcal{T}'$, we have that $\mathcal{T} \oplus \mathcal{T}' \models \varphi_1$ implies $\mathcal{T} \oplus \mathcal{T}' \models \varphi_2$. When $\oplus$ is clear from the context, we simply write $\langle\varphi_1\rangle\mathcal{T}\langle\varphi_2\rangle$.*

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transducers, and let φ_1, φ_2, and φ_3 be LTL formulas. Below are two typical assume-guarantee rules, for a composition operator $\oplus \in \{\oplus, \blacklozenge\}$ (as with the known composition semantics, many more rules exist [16]).

$$\frac{\langle\varphi_1\rangle\mathcal{T}_1\langle\varphi_2\rangle \quad \langle\varphi_2\rangle\mathcal{T}_2\langle\varphi_3\rangle}{\langle\varphi_1\rangle\mathcal{T}_1 \oplus \mathcal{T}_2\langle\varphi_3\rangle} \qquad \frac{\langle \mathit{true}\rangle\mathcal{T}_1\langle\varphi_1\rangle \quad \langle \mathit{true}\rangle\mathcal{T}_2\langle\varphi_2\rangle}{\langle \mathit{true}\rangle\mathcal{T}_1 \oplus \mathcal{T}_2\langle\varphi_1 \wedge \varphi_2\rangle}$$

Consider for example the left rule. To see that this rule is sound, note that, by definition, for every $\mathcal{T}'$ we have (1) if $\mathcal{T}_1 \oplus \mathcal{T}' \models \varphi_1$ then $\mathcal{T}_1 \oplus \mathcal{T}' \models \varphi_2$ and (2) if $\mathcal{T}_2 \oplus \mathcal{T}' \models \varphi_2$ then $\mathcal{T}_2 \oplus \mathcal{T}' \models \varphi_3$. In particular, for every $\mathcal{T}''$ we have that (1) holds when $\mathcal{T}'$ is $\mathcal{T}_2 \oplus \mathcal{T}''$ and (2) holds when $\mathcal{T}'$ is $\mathcal{T}_1 \oplus \mathcal{T}''$. Hence, for every $\mathcal{T}''$, if $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \mathcal{T}'' \models \varphi_1$ then $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \mathcal{T}'' \models \varphi_2$ and if $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \mathcal{T}'' \models \varphi_2$ then $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \mathcal{T}'' \models \varphi_3$. Hence, $\langle\varphi_1\rangle\mathcal{T}_1 \oplus \mathcal{T}_2\langle\varphi_3\rangle$. Thus, the rule is sound. Similar reasoning applies for the right rule.

For the standard concurrent composition operator, interleaving, we have that $\langle\varphi_1\rangle\mathcal{T}\langle\varphi_2\rangle$ iff $\mathcal{T} \models \varphi_1 \rightarrow \varphi_2$. Thus, it is possible to reduce checking of an assume-guarantee specification to LTL model checking. This simple reduction relies on the fact that the language of a concurrent system is contained in the languages of its underlying components. This fact is not valid for switched systems. Instead, we should check the $\varphi_1 \rightarrow \varphi_2$ implication in a richer context:

Lemma 3. *Let φ and ψ be LTL formulas, $\oplus \in \{\oplus, \blacklozenge\}$, and $\mathcal{T}$ a transducer. Then, $\langle\varphi_1\rangle\mathcal{T}\oplus\langle\varphi_2\rangle$ iff for every transducer $\mathcal{T}'$, we have $\mathcal{T} \oplus \mathcal{T}' \models \varphi_1 \rightarrow \varphi_2$.*

In Lemma 1, we have shown that the operators $\oplus$ and $\blacklozenge$ are monotone. Thus, checking $\mathcal{T} \oplus \mathcal{T}' \models \varphi_1 \rightarrow \varphi_2$ for every $\mathcal{T}'$, can be reduced to checking $\varphi_1 \rightarrow \varphi_2$ in the composition of $\mathcal{T}$ with the most challenging $\mathcal{T}'$, namely one whose language is ${\Sigma_{\mathrm{IO}}}^{\omega}$. Note that the monotonicity property also implies that if $\mathcal{L}(\mathcal{T}_1') = \mathcal{L}(\mathcal{T}_2')$, then $\mathcal{L}_{\mathrm{IO}}(\mathcal{T}_1' \oplus \mathcal{T}) = \mathcal{L}_{\mathrm{IO}}(\mathcal{T}_2' \oplus \mathcal{T})$. Thus, any transducer whose language is ${\Sigma_{\mathrm{IO}}}^{\omega}$ will do. Since a deterministic transducer generates a single computation for each input sequence, a transducer whose language is ${\Sigma_{\mathrm{IO}}}^{\omega}$ has to be nondeterministic. Let $\mathcal{U}$ be the nondeterministic transducer that has $|\Sigma_{\mathrm{O}}|$ states, all of them are initial, and for which each state has transitions, on all input letter in Σ_{I}, to all other states. It is easy to see that $\mathcal{L}(\mathcal{U}) = {\Sigma_{\mathrm{IO}}}^{\omega}$, and that the definitions of the composition operators in Section 2 extends to a composition with a nondeterministic transducer in a straightforward way.

Lemma 4. *Let φ be an LTL formula, $\oplus \in \{\oplus, \blacklozenge\}$, $\mathcal{T}$ be a transducer, and $\mathcal{U}$ a transducer such that $\mathcal{L}(\mathcal{U}) = {\Sigma_{\mathrm{IO}}}^{\omega}$. Then $\mathcal{T} \oplus \mathcal{U} \models \varphi$ iff for every $\mathcal{T}'$ we have $\mathcal{T} \oplus \mathcal{T}' \models \varphi$.*

Corollary 1. *Let φ, ψ be LTL formulas, $\oplus \in \{\oplus, \blacklozenge\}$, $\mathcal{T}$ be a transducer, and $\mathcal{U}$ a transducer such that $\mathcal{L}(\mathcal{U}) = {\Sigma_{\mathrm{IO}}}^{\omega}$. Then $\langle\varphi\rangle\mathcal{T}\oplus\langle\psi\rangle$ iff $\mathcal{T} \oplus \mathcal{U} \models \varphi \rightarrow \psi$.*

Theorem 3. *Model checking assume-guarantee specifications of switched systems is PSPACE-complete.*

Proof. As discussed above, for every transducer $\mathcal{T}$, LTL formulas φ_1 and φ_2, and a composition operator $\oplus$, we have that $\langle\varphi_1\rangle\mathcal{T}\langle\varphi_2\rangle$ iff $\mathcal{T} \oplus \mathcal{U} \models \varphi_1 \rightarrow \varphi_2$. Membership in PSPACE then follows from the fact that checking the latter requires space that is polynomial in φ_1 and φ_2 and logarithmic in $|\mathcal{T}| \cdot |\Sigma_{\mathrm{o}}|$. The lower bound follows from the PSPACE hardness of the validity problem for LTL. Indeed, φ is valid iff $\langle \mathit{true}\rangle\mathcal{U}\langle\varphi\rangle$. Note that validity of LTL is PSPACE-hard already for a fixed number of propositions, thus we can consider $\mathcal{U}$ to be of a fixed size, and by classifying all the propositions as input signals, $\mathcal{U}$ is also deterministic. Thus, PSPACE-hardness holds already for deterministic transducers. □

For an arbitrary switching rule, the IO-language of the composition $\mathcal{T} \oplus \mathcal{U}$ is Σ_{IO}, thus $\mathcal{T} \oplus \mathcal{U} \models \varphi_1 \rightarrow \varphi_2$ iff the implication $\varphi_1 \rightarrow \varphi_2$ is valid. Things become more interesting when assumptions on the switching are made. If, for example, $\mathcal{T} \models \mathsf{GF}\mathit{grant}_1 \rightarrow \mathsf{GF}\mathit{grant}_2$, then $\langle \mathsf{GF}\mathit{grant}_1\rangle\mathcal{T}\langle \mathsf{GF}\mathit{grant}2\rangle$ in a fair switching in which all components are switched on infinitely often even though $\mathsf{GF}\mathit{grant}_1 \rightarrow \mathsf{GF}\mathit{grant}_2$ may not be valid. As discussed in Section 2, such assumptions are easy to make by augmenting the specification by a precondition over who.

4 Synthesis of a Switching Rule

In this section we show how to synthesize a switching rule with which the composition of a given transducers satisfies a desired LTL property. Before we do so, we show that the harder problems of compositional realizability and compositional design are undecidable.

4.1 Undecidable Problems

Given LTL specifications $\varphi_1, \varphi_2, \ldots, \varphi_n$, and ψ, and a composition operator $\oplus$, the *compositional-realizability* problem is to decide whether there are transducers $\mathcal{T}_1, \mathcal{T}_2$, $\ldots, \mathcal{T}_n$ such that $\mathcal{T}_i$ satisfies φ_i for all $1 \leq i \leq n$, and $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ satisfies ψ. In [18] it was shown that compositional realizability is undecidable where $\oplus$ is the synchronous parallel composition. It was further shown that if, however, the processes admit a piplelined architecture the problem is decidable. In this section we show that for switched systems, though the architecture is extremely simple, compositional realizability is undecidable for both dormant and active compositions.

The *compositional-design* problem is to decide whether every switched system $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ such that $\mathcal{T}_i$ satisfies φ_i for all $1 \leq i \leq n$, also satisfies ψ. The problems of compositional-realizability and compositional design are strongly connected. Indeed, in a setting in which the formulas φ_i are realizable, the answer to the compositional-realizability problem with input $\varphi_1, \ldots, \varphi_n, \psi$ is 'yes' iff there exist transducers $\mathcal{T}_1, \mathcal{T}_2$, $\ldots, \mathcal{T}_n$ such that $\mathcal{T}_i$ satisfies φ_i for all $1 \leq i \leq n$, and $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ satisfies ψ. The latter holds iff the answer to the compositional-design problem with input $\varphi_1, \ldots, \varphi_n, \neg\psi$ is 'no'.

Theorem 4. *The compositional realizability and design problems are undecidable.*

By the above, it suffices to show that compositional-realizability problem is undecidable. The problem of compositional-realizability for standard concurrency was shown to be undecidable by Pnueli and Rosner in [18]. The key to their undecidability proof is an architecture of two processes that do *not* communicate with one another. Such lack of communication exists also in our setting, and enables an adoption of their proof with some minor adjustments.

4.2 Synthesis of a Switching Rule

Recall that the model-checking problem checks whether a switched system satisfies a specification under arbitrary switching or a switching that satisfies some assumption. Sometimes the details of the switching mechanism are known and may be controlled. In this section we study the problem of deciding, given transducers $\mathcal{T}_1, \ldots, \mathcal{T}_n$ and a specification φ, whether there is a *switching rule* according to which the switched system $\mathcal{T}_1 \oplus \cdots \oplus \mathcal{T}_n$ satisfies φ, and the problem of synthesizing such a rule in case the answer is positive.[7]

We model a switching rule by a transducer $\mathcal{S}$ with input alphabet Σ_{I} and output alphabet $\{1, 2\}$. Consider transducers $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$, a composition operator $\oplus \in \{\CIRCLE, \LEFTcircle\}$, and a switching rule $\mathcal{S}$. The switched system $\mathcal{T}_1 \oplus \mathcal{T}_2 \oplus \cdots \oplus \mathcal{T}_n$ with switching rule $\mathcal{S}$ has input in Σ_{I} (rather than in $\Sigma_{\mathrm{I}} \times \{1, 2\}$) and the component that is switched on after reading an input sequence $w \in {\Sigma_{\mathrm{I}}}^*$ is determined by the output of the state of $\mathcal{S}$ after reading w. Formally, let $\mathcal{T}_1 = \langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S_1, \theta_1, \eta_1, L_1 \rangle$, $\mathcal{T}_2 = \langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S_2, \theta_2, \eta_2, L_2 \rangle$, and $\mathcal{S} = \langle \Sigma_{\mathrm{I}}, \{1, 2\}, S, \theta, \eta, L \rangle$. Then, the switched system with components $\mathcal{T}_1$ and $\mathcal{T}_2$, and switching rule $\mathcal{S}$, is the transducer $\langle \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, S', \theta', \eta', L' \rangle$, defined as follows:

- $S' = S_1 \times S_2 \times \{1, 2\} \times S$. Intuitively, the switched system is identical to the one without the switching rule, only that the *who* element is determined by the switching rule rather than by the environment.
- $\theta'(i) = \langle \theta_1(i), \theta_2(i), L(\theta(i)), \theta(i) \rangle$. That is, the initialization function maps each state component according to the respective initialization function, and determines the next state to be the output of the switching rule on the first input.
- The transition function η is defined according to the type of composition as follows. Consider a state $\langle s_1, s_2, k, s \rangle \in S'$ and a letter $i \in \Sigma_{\mathrm{I}}$.
  - $\LEFTcircle$ $\eta(\langle s_1, s_2, k, s \rangle, i) = \begin{cases} \langle \eta_1(s_1, i), s_2, L(s), \eta(s, i) \rangle & \text{if } L(s) = 1 \\ \langle s_1, \eta_2(s_2, i), L(s), \eta(s, i) \rangle & \text{if } L(s) = 2. \end{cases}$
  - $\CIRCLE$ $\eta(\langle s_1, s_2, k, s \rangle, i) = \langle \eta_1(s_1, i), \eta_2(s_2, i), L(s), \eta(s, i) \rangle$.
- For all $\langle s_1, s_2, k, s \rangle \in S'$, we have $L(\langle s_1, s_2, k, s \rangle) = L_k(s_k)$.

The solution to the switching-rule synthesis problem involves automata on infinite trees (see [17] or the full version of the paper).

When we synthesize a switching rule, we are given the transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, and the transducer we are after only has to generate an infinite sequence over $\{1, 2\}$. The

[7] A recent work [24] advocates the use of ω-regular languages over the alphabet of subcomponents identifiers for describing switching constraints even for continuous switched systems.

setting is then similar to the *control* problem for LTL [17]. Unlike the solution there, however, here the controller does not disable transitions. Rather, it determines which component should be active at each moment in time.

Theorem 5. *The switching-rule synthesis problem for LTL is 2EXPTIME-complete.*

Proof. Consider an LTL formula ψ. Let $\mathcal{A}_\psi = \langle \Sigma_{\mathrm{o}}, Q, q_0, \delta, \alpha \rangle$ be a deterministic parity word automaton (DPW) recognizing ψ. We define a deterministic parity tree automaton (DPT) $\mathcal{A}_{\forall\psi}^{\mathcal{T}_1,\mathcal{T}_2}$ that accepts switching rules with which $\mathcal{T}_1 \oplus \mathcal{T}_2$ satisfies ψ. Formally, $\mathcal{A}_{\forall\psi}^{\mathcal{T}_1,\mathcal{T}_2} = \langle \{1,2\}, \Sigma_{\mathrm{I}}, S_1 \times S_2 \times \{1,2\} \times Q, s_0, \delta', S_1 \times S_2 \times \{1,2\} \times \alpha \rangle$, where s_0 is a new state and for $who \in \{1,2\}$ we have $\delta(s_0, who) = \langle \theta_1(who), \theta_2(who), who, q_0 \rangle$ and for all $\langle s_1, s_2, k, q \rangle \in S_1 \times S_2 \times \{1,2\} \times Q$ we have

- $\delta'(\langle s_1, s_2, k, q \rangle, who) = \begin{cases} \bigwedge_{i \in \Sigma_{\mathrm{I}}} (i, \langle \eta_1(s_1, i), s_2, who, \delta(q, \langle i, L_k(s_k) \rangle) \rangle) & \text{if } L(q) = 1 \\ \bigwedge_{i \in \Sigma_{\mathrm{I}}} (i, \langle s_1, \eta_2(s_2, i), who, \delta(q, \langle i, L_k(s_k) \rangle) \rangle) & \text{if } L(q) = 2 \end{cases}$

- $\delta'(\langle s_1, s_2, k, q \rangle, who) = \bigwedge_{i \in \Sigma_{\mathrm{I}}} (i, \langle \eta_1(s_1, i), \eta_2(s_2, i), who, \delta(q, \langle i, L_k(s_k) \rangle) \rangle)$.

Intuitively, a state $\langle s_1, s_2, k, q \rangle$ stands for the transducer $\mathcal{T}_1$ being in s_1, the transducer $\mathcal{T}_2$ being in s_2, the transducer that is switched on is $\mathcal{T}_k$, and the automaton $\mathcal{A}_\psi$ is in state q. In the dormant composition, only $\mathcal{T}_k$ changes its state. In both compositions, the O-element of the letter that $\mathcal{A}_\psi$ reads in q is the output of $\mathcal{T}_k$. It is not hard to prove that $\mathcal{A}_{\forall\psi}^{\mathcal{T}_1,\mathcal{T}_2}$ accepts a full tree with directions from Σ_{I} generated by a transducer $\mathcal{S}$ iff the composition of $\mathcal{T}_1$ and $\mathcal{T}_2$ according to $\mathcal{S}$ satisfies ψ.

We reduced the switching-rule synthesis problem to the nonemptiness problem for $\mathcal{A}_{\forall\psi}^{\mathcal{T}_1,\mathcal{T}_2}$. The number of states of the DPW $\mathcal{A}_\psi$ is doubly-exponential in $|\psi|$, and its index is exponential in $|\psi|$ [20, 23]. Therefore, the number of states of the DPT $\mathcal{A}_{\forall\psi}^{\mathcal{T}_1,\mathcal{T}_2}$ is linear in $|\mathcal{T}_1|$ and $|\mathcal{T}_2|$ and doubly-exponential in $|\psi|$, and its index is exponential in $|\psi|$. Since the nonemptiness problem for DPT can be solved in time polynomial in the state space and exponential in the index [5], the upper bound follows. Note that the doubly-exponential complexity is only in terms of $|\psi|$, and the algorithm is polynomial in $|\mathcal{T}_1|$ and $|\mathcal{T}_2|$.

For the lower bound, note that the synthesis problem for LTL is 2EXPTIME-hard already for a formula ψ with $O = \{p\}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be single-state transducers that satisfy p and $\neg p$, respectively. A switching rule for $\mathcal{T}_1$ and $\mathcal{T}_2$ then corresponds to a transducer with $O = \{p\}$, and the synthesis problem for ψ can be reduced to the switching-rule synthesis problem for $\mathcal{T}_1, \mathcal{T}_2$, and ψ. □

Note that since the switching rule $\mathcal{S}$ reads the inputs to all components, nothing prevents it from naively recomputing the output of the components. The essence of a switching rule, however, is to avoid this computation. For example, in the security-camera network discussed in Section 1, a scheduler that implements the software that detects suspicious behaviors is not of much interest. One way to prevent the switching rule from recomputing the output of the components is to restrict its input. In practice, however, optimizing the switching rule obtained in the construction in Theorem 5 would project out the parts that are not essential for the switching rule.

Assumptions on the switching, and hence restrictions on the synthesized switching rule, can be made by replacing ψ by $\psi_{fair} \rightarrow \psi$. The automaton $\mathcal{A}^{\mathcal{T}_1,\mathcal{T}_2}_{\forall\psi}$ then continues to read the I-component of the alphabet from the directions of the tree, the O-component from the active transducer, and reads who from the input tree.

5 Language Characterization

Recall that a component may be switched on forever. Thus, a required condition for a switched system to satisfy a property is that all the underlying components satisfy it. For some properties, this is also a sufficient condition, giving rise to a simple compositional model-checking procedure for them. In this section we seek a characterization of such properties. We solve the problem for the active composition and leave it open for the dormant composition.

In Section 2.1 we showed that the dormant and active compositions correspond to *shuffle* and *merge* of languages. Let $\oplus \in \{ \CIRCLE , \circledcirc \}$ be a composition operator. We say that a language L is *closed under* $\oplus$ iff $L \oplus L \subseteq L$. That is, for every $u, v \in L$, we have that $u \oplus v \in L$. For example (recall the table in Figure 1), the language $(0+1)^*0^\omega$ is closed under both $\CIRCLE$ and $\circledcirc$, the language $(01)^\omega$ is closed under $\CIRCLE$ but not under $\circledcirc$, the language $(0^*1)^\omega$ is closed under $\circledcirc$ but not under $\CIRCLE$, and the language 0^*10^ω is closed under neither $\CIRCLE$ nor $\circledcirc$.

As the examples above demonstrate, a language that is closed under shuffle or merge need not be a safety or a co-safety language. It turns out that an exact characterization of the languages that are closed under shuffle or merge is a challenging combinatorial problem. As described below, we have succeeded to obtain an exact characterization for merge. The problem of an exact characterization for shuffle remains open.

Recall that a language L is closed under merge if for every two words $u, v \in L$, all words obtained by locating in position i the i-th letter in either u or v are in L. This means that each of the requirements imposed by L refers to a precise location (e.g., the 4-th letter is 0), or is an eventuality, in which case the requirement in the scope of the eventuality is a safety property (e.g., eventually always 0). Formally, we characterize closure under merge by means of *regular counting*, defined below.

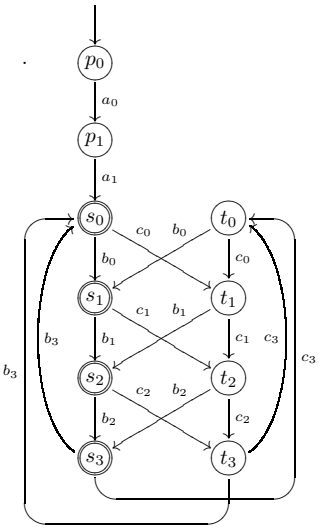
Definition 2. *A language L is* regular counting *if there are $n, k \in \mathbb{N}$ and functions $f_0 : \{0, \ldots, n-1\} \mapsto 2^\Sigma$ and $f_1, f_2 : \{0, \ldots, k-1\} \mapsto 2^\Sigma$ such that for all $0 \leq j \leq k-1$, we have $f_2(j) \subseteq f_1(j)$ and $w \in L$ **iff** for all $0 \leq j \leq n-1$ we have $w[j] \in f_0(j)$ and there is $i \geq n$ such that for all $j \geq n$, if $j < i$, then $w[j] \in f_1(j \bmod k)$, and if $j \geq i$, then $w[j] \in f_2(j \bmod k)$.*

Intuitively, the function f_0 describes how the prefix of length n of all the words in L behaves – each location j in this prefix can take letters from the subset $f_0(j)$ of Σ. After the prefix of length n the words in L behaves in some cyclic manner, for a cycle of length k. For some bounded number of locations, this cyclic behavior is described by f_1 – each location j in this infix can take letters from the subset $f_1(j \bmod k)$ of Σ. Eventually, however, the cyclic behavior is described by f_2, which is more restricted than f_1. It is not hard to see that a language L is safety iff it is regular counting with $f_1 = f_2$.

To understand the notion of regular counting better, we now describe an automata-theoretic characterization of it.

Definition 3. *An automaton* $\mathcal{A} = \langle \Sigma, Q, q^0, \delta, \alpha \rangle$ *is a* counting automaton *if* $\mathcal{A}$ *is a deterministic co-Büchi automaton (DCW) and* Q *can be partitioned into three disjoints sets* $P = \{p_0, \ldots, p_{n-1}\}$, $S = \{s_0, \ldots, s_{k-1}\}$, *and* $S' = \{s'_0, \ldots, s'_{k-1}\}$ *such that:*

1. *For every* $0 \leq i \leq n-1$, *there is* $\emptyset \neq \Omega_i \subseteq \Sigma$ *such that for all* $\sigma \in \Omega_i$, *we have* $\delta(p_i, \sigma) = p_{i+1}$ *(with* p_n *standing for* s_0*) and for all* $\sigma \notin \Omega_i$, *we have* $\delta(p_i, \sigma) = \emptyset$.
2. *For every* $0 \leq i < k-1$, *there are* $\Omega_i, \Omega'_i \subseteq \Sigma$ *such that* $\Omega_i \cap \Omega'_i = \emptyset$ *and* $\Omega'_i \neq \emptyset$, *such that*
   − *for all* $\sigma \in \Omega_i$, *we have* $\delta(s_i, \sigma) = \delta(s'_i, \sigma) = s_{i+1}$,
   − *for all* $\sigma \in \Omega'_i$, *we have* $\delta(s_i, \sigma) = \delta(s'_i, \sigma) = s'_{i+1}$,
   − *and for all* $\sigma \in \Sigma \setminus (\Omega_i \cup \Omega'_i)$, *we have* $\delta(s_i, \sigma) = \delta(s'_i, \sigma) = \emptyset$.
3. $\alpha = S$.



Example 1. The automaton described in the above figure is a counting automaton accepting the language $a_0 a_1((b_0 \vee c_0)(b_1 \vee c_1)(b_2 \vee c_2)(b_3 \vee c_3))^*(c_0 c_1 c_2 c_3)^\omega$.

Proposition 1. *Let* $L \subseteq \Sigma^\omega$. *There exists a counting automaton* $\mathcal{A}$ *such that* $\mathcal{L}(\mathcal{A}) = L$ *if and only if* L *is regular counting.*

We are now ready to state our main theorem for this section.

Theorem 6. $L \subseteq \Sigma^\omega$ *is regular and preserved under merge iff* L *is regular counting.*

The difficult direction is proving that if L is regular and preserved under merge, then L is regular counting. As detailed in the full version, we do this by first proving that if $L \subseteq \Sigma^\omega$ is preserved under merge and is regular, then L is accepted by a deterministic co-Büchi automaton. Essentially, in [10], Landweber proves that a deterministic Rabin automaton has an equivalent deterministic Büchi automaton iff its accepting strongly connected components are upward closed (that is, if S is accepting, so are all components $S' \supseteq S$). We prove that the rejecting strongly connected components of a deterministic Streett automaton for a language L that is preserved under merge are downward closed, and conclude that L can be accepted by a deterministic co-Büchi automaton. We then prove that the states of the deterministic co-Büchi automaton can be partitioned as required in the definition of a counting automaton.

Acknowledgements

We thank Yoad Lustig, Michael Margaliot, Kedar Namjoshi, Dana Porrat, and Gera Weiss for helpful discussions and references.

References

1. N. Amla, E.A. Emerson, K.S. Namjoshi, and R.J. Trefler. Abstract Patterns of Compositional Reasoning. In *Proc 14th CONCUR*, pages 423-438, LNCS 2761, 2003.
2. P. Antsaklis. Proceedings of the IEEE, special issue on hybrid systems: theory and applications. 88(7), 2000.
3. H. de Jong, J.-L. Gouze, C. Hernandez, M. Page, T. Sari, and J. Geiselmann. Qualitative simulation of genetic regulatory networks using piecewise-linear models. *Bulletin of Mathematical Biology*, 66:301–340, 2004.
4. E. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, LNCS 193, pages 79–87, 1985.
5. E. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st LICS*, pages 267–278, 1986.
6. R. Erickson and D. Maksimovic. *Fundamentals of power electronics*. Kluwer, 2001.
7. J. Hespanha, S. Bohacek, K. Obraczka, and J. Lee. Hybrid modeling of TCP congestion control. In *Hybrid systems: Computation and control*, LNCS 2034, pages 291–304, 2001.
8. K. Johansson, J. Lygeros, and S. Sastry. Modeling of hybrid systems. Encyclopedia of life support systems, 2004.
9. D. Kozen. Lower bounds for natural proof systems. In *18th FOCS*, pages 254–266, 1977.
10. L. H. Landweber. Decision Problems for omega-Automata. *Mathematical Systems Theory*, 3(4):376–384, 1969.
11. G. Lafferriere, G. Pappas, and S. Sastry. O-minimal hybrid systems. *Math. Control Signal Systems*, 13:1–21, 2000.
12. D. Liberzon. *Switching in Systems and Control*. Birkhauser, 2003.
13. M. Margaliot. Stability analysis of switched systems using variational principles: an introduction. *Automatica*, 42(12):2059–2077, 2006.
14. A. F. Molisch, J. R. Foerster, and M. Pendergrass. Channel models for ultrawideband personal area networks. *Wireless Communications, IEEE*, 10(6):14–21, 2003.
15. A. J. Mayer and L. J. Stockmeyer. The complexity of word problems - this time with interleaving. *Information and Computation* 115:293–311, 1994.
16. A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 123–144. Springer, 1985.
17. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
18. A. Pnueli and R. Rosner. Distributed Reactive Systems are Hard to Synthesize. In 31st FOCS, pages 746–757, 1990.
19. M. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. of the AMS*, 141:1–35, 1969.
20. S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319–327, 1988.
21. S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th STOC*, 1992.
22. R. Shorten, D. Leith, J. Foy, and R. Kilduff. Analysis and design of AIMD congestion control algorithms in communication networks. *Automatica*, 41:725–730, 2005.
23. M. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, 1994.
24. G. Weiss and R. Alur. Automata Based Interfaces for Control and Scheduling. *Hybrid Systems*, 4416:601-613, 2007.
25. P. Winkler. Mathematical Puzzles: A Connoisseur's Collection. *A K Peters*, pages 109-111, 2004.