Composability of Infinite-State Activity Automata *

Zhe Dang[1], Oscar H. Ibarra[2],** and Jianwen Su[2]

[1]School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

[2]Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

Abstract. Let $\mathcal{M}$ be a class of (possibly nondeterministic) language acceptors with a one-way input tape. A system $(A; A_1, ..., A_r)$ of automata in $\mathcal{M}$, is *composable* if for every string $w = a_1...a_n$ of symbols accepted by A, there is an assignment of each symbol in w to one of the A_i's such that if w_i is the subsequence assigned to A_i, then w_i is accepted by A_i. For a nonnegative integer k, a k-lookahead delegator for $(A; A_1, ..., A_r)$ is a *deterministic* machine D in $\mathcal{M}$ which, knowing (a) the current states of $A, A_1, ..., A_r$ and the accessible "local" information of each machine (e.g., the top of the stack if each machine is a pushdown automaton, whether a counter is zero on nonzero if each machine is a multicounter automaton, etc.), and (b) the k lookahead symbols to the right of the current input symbol being processed, can uniquely determine the A_i to assign the current symbol. Moreover, every string w accepted by A is also accepted by D, i.e., the subsequence of string w delegated by D to each A_i is accepted by A_i. Thus, k-lookahead delegation is a stronger requirement than composability, since the delegator D must be deterministic. A system that is composable may not have a k-delegator for any k.

We look at the decidability of composability and existence of k-delegators for various classes of machines $\mathcal{M}$. Our results have applications to automated composition of e-services. E-services provide a general framework for discovery, flexible interoperation, and dynamic composition of distributed and heterogeneous processes on the Internet. Automated composition allows a specified composite e-service to be implemented by composing existing e-services. When e-services are modeled by automata whose alphabet represents a set of activities or tasks to be performed (namely, activity automata), automated design is the problem of "delegating" activities of the composite e-service to existing e-services so that each word accepted by the composite e-service can be accepted by those e-services collectively with each accepting a subsequence of the word, under possibly some Presburger constraints on the numbers and types of activities that can be delegated to the different e-services. Our results generalize earlier ones (and resolve some open questions) concerning composability of deterministic finite automata as e-services to finite automata that are augmented with unbounded storage (e.g., counters and pushdown stacks) and finite automata with discrete clocks (i.e., discrete timed automata).

1 Introduction

In traditional automata theory, an automaton is a language acceptor that is equipped with finite memory and possibly other unbounded storage devices such as a counter, a stack, a queue, etc. The

* The research of Oscar H. Ibarra and Jianwen Su was supported in part by NSF Grants IIS-0101134 and CCR02-08595.
** Corresponding author (ibarra@cs.ucsb.edu).

automaton "scans" a given input word in a one-way/two-way and nondeterministic/deterministic manner while performing state transitions. As one of the most fundamental concept in theoretical computer science, automata are also widely used in many other areas of computer science, in particular, in modeling and analyzing distributed and concurrent systems. For instance, one may view a symbol a in an input word that is read by the automaton as an input/output signal (event). This view naturally leads to automata-based formal models like I/O automata [20]. On the other hand, when one views symbol a as an (observable) activity that a system performs, the automaton can be used to specify the (observable) behavior model of the system; i.e., an activity automaton of the system. For instance, activity automata have been used in defining an event-based formal model of workflow [27]. Recently, activity (finite) automata are used in [5] to model e-services, which are an emerging paradigm for discovery, flexible interoperation, and dynamic composition of distributed and heterogeneous processes on the web or the Internet. An important goal as well as an unsolved challenging problem in service oriented computing [22] such as e-services is *automated composition*: how to construct an "implementation" of a desired e-service in terms of existing e-services.

To approach the automated composition problem, the technique adopted in [5] has two inputs. One input is a finite set of activity finite automata, each of which models an "atomic" e-service. The second is a desired global behavior, also specified as an activity finite automaton, that describes the possible sequences of activities of the e-service to be composed. The output of the technique is a *delegator* that will coordinate the activities of those atomic e-services through a form of delegation. Finding a delegator, if it exists, was shown to be in EXPTIME. The framework was extended in [13] by allowing "lookahead" of the delegator, i.e., to have the knowledge of future incoming activities. A procedure was given which computes a sufficient amount of lookahead needed to perform delegation; however, the procedure is not guaranteed to terminate.

We have noticed that the models studied in [5, 13] have significant limitations: only regular activities are considered since the underlying activity models are finite automata. In reality, more complex and non-regular activity sequences are possible. For instance, activity sequences describing a session of activities `releaseAs`, `allocateAs`, `releaseBs` and `allocateBs` satisfying the condition that the absolute difference between the number of `releaseAs` and the number of `allocateAs`, as well as the absolute difference between the number of `releaseBs` and the number of `allocateBs`, is bounded by 10 (the condition can be understood as some sort of fairness) are obviously non-regular (not even context-free). Therefore, in this paper, we will use the composition model of [13] but focus on, instead of finite automata, infinite-state (activity) automata. Additionally, automata-theoretic techniques are used in our presentation, which are different from the techniques used in [5, 13]. Notice that the problem is not limited only to e-services. In fact, similar automated design problems were also studied in the workflow context [28, 19] and verification communities (e.g., [6, 1, 24, 18]). In the future, we will also look at how our techniques and results can be applied to these latter problems.

In this paper, we use $A_1, ..., A_r$ to denote r number of activity automata (not necessary of finite-state), which specify the activity behaviors of some r existing e-services. We use A to denote an activity automaton (again, not necessary of finite-state), which specifies the desired activity behavior of the e-service to be composed from the existing e-services. Naturally, to address the automated composition problem, there are two issues to resolve. The first issue is how to automatically decide whether there is a delegator that can compose the $A_1, ..., A_r$ into the desired A. The second issue is how to automatically (from $A, A_1, \ldots, A_r$) create the delegator possibly under some resource bound.

The first issue concerns *composability*. The system $(A; A_1, ..., A_r)$ is *composable* if for every string (or sequence) $w = a_1...a_n$ of activities accepted by A, there is an assignment (or delegation) of each symbol in w to one of the A_i's such that if w_i is the subsequence assigned to A_i, then w_i is accepted by A_i. We start our discussion with $A, A_1, ..., A_r$ being restricted counter-machines (finite automata augmented with counters, each of which can be incremented/decremented and can be tested against 0). One of the restrictions we consider is when the counters are reversal-bounded [16]; i.e., for a reversal-bounded counter, the number of alternations between a nondecreasing mode and a nonincreasing mode is bounded by a given constant. As an example, the above mentioned release-allocate sequences can be accepted by a deterministic reversal-bounded counter-machine with 4 reversal-bounded counters. We use notations like DFAs or NFAs (deterministic or nondeterministic finite automata) and DCMs or NCMs (deterministic or nondeterministic reversal-bounded counter-machines). In [13], it was shown that composability is decidable for a system $(A; A_1, .., A_r)$ of DFAs. We generalize this result to the case when A is an NPCM (nondeterministic pushdown automaton with reversal-bounded counters) and the A_i's are DFAs. In contrast, we show that it is undecidable to determine, given DFAs A and A_1 and a DCM A_2 with *two* 1-reversal counters (i.e., once a counter decrements it can no longer increment), whether $(A; A_1, A_2)$ is composable. This also holds for the case when A_2 is a DFA augmented with a 1-reversal pushdown stack (thus, it accepts a deterministic linear context-free language). We also look at situations where composability is decidable. Further, we propose alternative definitions of composition (e.g., T-composability) and investigate decidability with respect to these new definitions.

The second issue concerns the existence of a delegator within some resource bound. Here, we adopt the notion of k-lookahead delegator (or simply k-delegator) from [13] but for infinite-state automata. (We note that [5] only studied 0-lookahead delegators.) This special form of a delegator is assumed to be efficient, since in its implementation, the delegator does not need to look back to its delegation history to decide where the current activity shall be delegated. For a nonnegative integer k, a k-delegator for $(A; A_1, ..., A_r)$ is a DCM D which, knowing (1) the current states of $A, A_1, ..., A_r$ and the signs of their counters (i.e., zero or non-zero), and (2) the k lookahead symbols (i.e., the k "future" activities) to the right of the current input symbol being processed, can uniquely determine the A_i to assign the current symbol. Moreover, every string w accepted by A is also accepted by D, i.e., the subsequence of string w delegated by D to each A_i is accepted by A_i. Clearly, if a system $(A; A_1, ..., A_r)$ has a k-delegator for some k, then it must be composable. However, the converse is not true – a system may be composable but it may not have a k-delegator for any k.

In [5], the decidability of the existence of a 0-lookahead delegator (i.e., no lookahead) when the automata (i.e., $A, A_1, ..., A_r$) are DFAs was shown to be is in EXPTIME. The concept of lookahead was introduced in [13] where the focus was still on DFAs; algorithms for deciding composability and determining an approximate upper bound on k (if it exists) were obtained. A question left open in [13] is whether there is a decision procedure for determining for a given k, whether a system of DFAs has a k-lookahead delegator. We answer this question positively in this paper, even for the more general case when the automata are not necessarily finite-state (e.g., DCMs). Specifically, we show that it is decidable to determine, given a system $(A; A_1, ..., A_r)$ of DCMs and a nonnegative integer k, whether the system has a k-lookahead delegator.

Our results generalize to composition and lookahead delegation when we impose some linear constraints on the assignments/delegations of symbols. Doing this allows us to further specify some fairness linear constraint on a delegator. For instance, suppose that we impose a linear relationship, specified by a Presburger relation P, on the numbers and types of symbols that can be assigned to $A_1, ..., A_r$. We show that it is decidable to determine for a given k, whether a system $(A; A_1, ..., A_r)$ of DCMs has a k-delegator under constraint P. However, it is undecidable to determine, given a system $(A; A_1, A_2)$, whether it is composable under constraint P, even when A, A_1, A_2 are DFAs and P involves only the symbols assigned to A_2.

Composability and existence of k-lookahead delegators for systems consisting of other types of automata can also be defined and we study them as well. In particular, we show that composability is decidable for discrete timed automata [3] (these are NFAs augmented with discrete-valued clocks).

The paper has six sections, in addition to this section. Section 2 defines (actually generalizes) the notion of composability of activity automata and proves that it is undecidable for systems $(A; A_1, A_2)$, where A, A_1 are DFAs and A_2 is a DCM with two 1-reversal counters or a DFA with a 1-reversal pushdown stack. It is also undecidable when A, A_1, A_2 are DFAs when a Presburger constraint is imposed on the numbers and types of symbols that can be delegated to A_1 and A_2. In contrast, composability is decidable for systems $(A; A_1, ..., A_r)$ when $A_1, ..., A_r$ are DFAs (even NFAs) and A is an NPCM. Decidability holds for other restricted classes of automata as well. Section 3 introduces T-composability and shows that T-composability is decidable for various automata. Section 4 looks at the decidability of the existence for a given k of a k lookahead delegator and shows, in particular, that it is decidable to determine, given a system $(A; A_1, ..., A_r)$ of NCMs and a nonnegative integer k, whether the system has a k-delegator (even when A is an NPCM). The decidability holds, even if the delegation is under a Presburger constraint. Section 5 briefly studies the notion of "upper composability". Section 6 investigates composability of discrete timed automata. Section 7 is a brief conclusion.

All of the proofs can be found in the Appendix, which may be read by the PC members at their discretion.

2 Composability

Recall that, throughout this paper, we will use the following notations: a DFA (NFA) is a deterministic (nondeterministic) finite automaton; DCM (NCM) is a DFA (NFA) augmented with reversal-bounded counters; NPCM (DPCM) is a nondeterministic (deterministic) pushdown automaton augmented with reversal-bounded counters.

Machines with reversal-bounded counters have nice decidable properties (see, e.g., [16, 17, 11]), and the languages they accept have the so-called semilinear property. They have been useful in showing that various verification problems concerning infinite-state systems are decidable [8, 7, 9, 12, 10, 23].

Definition 1. *Let $(A; A_1, ..., A_r)$ be a system of activity automata that are DCMs over input (or activity) alphabet Σ. Assume that each DCM starts in its initial state with its counters initially zero. We say that a word (or a sequence of activities) $w = a_1 a_2 ... a_n$ is* composable *if there is an assignment of each symbol a_i to one of the $A_1, ..., A_r$ such that if the subsequence of symbols assigned to A_i is w_i, then w_i is accepted by A_i (for $1 \leq i \leq r$). We say that the system $(A; A_1, ..., A_r)$ is composable if every word w accepted by A is composable.*

Example 1. Consider an online club that offers its customers to first register (represented by *r*), and then pay for their accesses (*a*) with either cash (*s*) or credit cards (*c*). The e-Service is shown as A in Figure 1, which accepts the language $(r|(aa^*(s|c)))^*$. Assume that there are three existing e-Services, A_1, A_2, and A_3, where A_1 handles registration, cash payments for one or more accesses, A_3 is similar to A_1 except that some customers may use promotion for free accesses, and A_2 can also handle accesses and make credit card transactions. Clearly, the system $(A; A_1, A_2)$ is composable where processing of accesses will be done by whoever collects the payment, cash (A_1) or credit card (A_2).

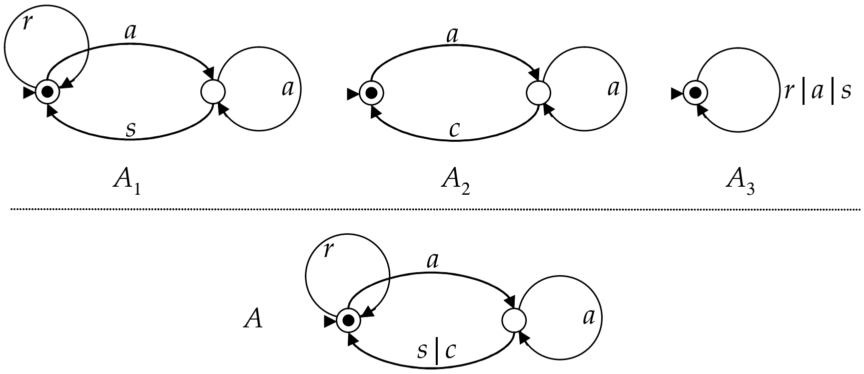

Fig. 1. Four e-Services

The system $(A; A_2, A_3)$ is also composable, but in this case, the delegator need only know if the customer will make a credit card payment in the next activity; if so A_2 will perform a, otherwise A_3 does it. Thus this system has a 1-lookahead delegator (to be defined more precisely later). ∎

It is known that it is decidable whether a system $(A; A_1, ..., A_r)$ of DFAs is composable [13]. The following result says that it becomes undecidable when one of the A_i's is augmented with two 1-reversal counters.

Theorem 1. *It is undecidable to determine, given a system $(A; A_1, A_2)$, where A and A_1 are DFAs and A_2 is a DCM with two 1-reversal counters, whether it is composable.*

Corollary 1. *It is undecidable to determine, given a system $(A; A_1, A_2)$, where A and A_1 are DFAs and A_2 is a DFA with a 1-reversal pushdown stack (thus A_2 accepts a deterministic linear context-free language), whether it is composable.*

Remark 1. Obviously, if the machines are NCMs, composability is undecidable. In fact, take A to be the trivial machine that accepts Σ^* (the universe). Take A_1 to be an an arbitrary NCM with one 1-reversal counter. Then the system $(A; A_1)$ is composable iff Σ^* is contained in $L(A_1)$. But the latter problem is known to be undecidable [4].

Theorem 2. *If A is an NPCM and $A_1, ..., A_r$ are DFAs (or even NFAs), then composability of $(A; A_1, ..., A_r)$ is decidable.*

It is of interest to determine the complexity of the composability problem. For example, a careful analysis of the proof of the above theorem and the use of Savitch's theorem that a nondeterministic $S(n)$ space-bounded TM can be converted to an equivalent deterministic $S^2(n)$ space-bounded TM [25], we can show the following:

Corollary 2. *Composability of a system $(A; A_1, ..., A_r)$ of NFAs can be decided in deterministic exponential space (in the sum of the sizes of the machines).*

There are other cases when composability becomes decidable, if we apply more restrictions to $A, A_1, \ldots, A_r$. A language L is bounded if $L \subseteq w_1^* ... w_k^*$ for some given k and strings $w_1, ..., w_k$ (which may not be distinct).

Theorem 3. *Composability is decidable for a system $(A; A_1, ..., A_r)$ of NCMs when A accepts a bounded language. The result holds even if A and one of the A_i's are NPCMs.*

Another restriction on the A_i's is the following. We assume that Σ_i is the input alphabet of A_i. An input symbol a is *shared* if $a \in \Sigma_i \cap \Sigma_j$ for some $i \neq j$. We say that $(A; A_1, \ldots, A_r)$ is n-composable if every word w accepted by A and containing at most n appearances of shared symbols is composable. Then we have:

Theorem 4. *The n-composability of $(A; A_1, ..., A_r)$ is decidable when A is an NPCM and each A_i is a DCM.*

For our next result, we recall the definitions of semilinear set and Presburger relation [14]. A set $R \subseteq \mathbf{N}^n$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbf{N}^n$ such that $R = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t,\ a_i \in \mathbf{N}\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set R. A set $R \subseteq \mathbf{N}^n$ is *semilinear* if it is a finite union of linear sets. It is known that R is a semilinear set if and only if it is a Presburger relation (i.e., can be specified by a Presburger formula).

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. For each string w in Σ^*, define the *Parikh map* of w to be $\psi(w) = (num_{a_1}(w), ..., num_{a_n}(w))$, where $num_{a_i}(x)$ is the number of occurrences of a_i in w. For a language $L \subseteq \Sigma^*$, the *Parikh map* of L is $\psi(L) = \{\psi(w) \mid w \in L\}$.

Let $A, A_1, ..., A_r$ is a system of DFAs over input alphabet Σ, and P be a Presburger relation (semilinear set). Suppose that we want to check whether the system is composable under constraint P on the numbers and types of symbols that are assigned/delegated to the A_i's. The constraint is useful in specifying a fairness constraint over the delegations (e.g., it is never true that the absolute value of the difference between the number of activities a assigned to A_1 and the number of activities a assigned to A_2 is larger than 10). Let $\Sigma = \{a_1, ..., a_n\}$ and P be a Presburger relation (formula) over $(r+1)n$ nonnegative integer variables (note that n is the cardinality of Σ and $r+1$ is the number of the DFAs, including A). The P-composability problem might take the the following form:

Presburger-constrained composability problem: Given a system $(A; A_1, ..., A_r)$ of DFAs, is the system composable subject to the constraint that for every string $w \in L(A)$, there is an assignment of the symbols in w such that if $w_1, ..., w_r$ are the subsequences assigned to $A_1, ..., A_r$, respectively,

(1) A_i accepts w_i ($1 \leq i \leq r$).
(2) $(\psi(w), \psi(w_1), ..., \psi(w_r))$ satisfies the Presburger relation P.

Unfortunately, because of Theorem 1, the above problem is undecidable:

Corollary 3. *The Presburger-constrained composability problem is undecidable for systems $(A; A_1, A_2)$ of DFAs and a Presburger formula P (even if the formula only involves symbols assigned to A_2).*

Later, we will see that a variation of the above problem is decidable.

3 T-Composability

From the above results, it seems difficult to obtain decidable composability for $(A; A_1, ..., A_r)$ when one or more of $A_1, \ldots, A_r$ are beyond DFAs. Below, we will apply more restrictions on how $A_1, \ldots, A_r$ are going to be composed such that a decidable composability can be obtained.

We define a mapping $T : \Sigma \to 2^{\{1,\ldots,r\}}$ such that each symbol $a \in \Sigma$ is associated with a type $T(a) \subseteq \{1, \ldots, r\}$. For $a \in \Sigma$ and $1 \leq i \leq r$, let $(a)_i = a$ if $i \in T(a)$ and $(a)_i = \epsilon$ (the null string) if $i \notin T(a)$. For a string $w = a_1 \ldots a_n$, we use $(w)_i$ to denote the result of $(a_1)_i \ldots (a_n)_i$. For each A_i, its input alphabet Σ_i consists of all a's with $i \in T(a)$. Therefore, $(w)_i$ is the result of projecting w under the alphabet of A_i. We now modify the definition of composability as follows. $(A; A_1, \ldots, A_r)$ is T-composable if, for every string w accepted by A, each $(w)_i$ is accepted by A_i. Notice that this definition is different from the original one in the sense that every symbol a in w is assigned to *each* A_i with $i \in T(a)$. Therefore, assignments of symbols in w is deterministic in the new definition (there is a unique way to assign every symbol). One can show:

Theorem 5. *The T-composability of $(A; A_1, \ldots, A_r)$ is decidable in the following cases:*

- *A is an NPCM and each A_i is a DCM;*
- *A is an NCM and each A_i is a DPCM.*

Theorem 5 does not generalize to the case when one of the A_i's is an NCM, for the same reason as we stated in Remark 1.

We may take another view of the composition of $A_1, \ldots, A_r$. As we have mentioned earlier, each activity automaton A_i is understood as the behavior specification of an e-service. Each sequence w_i of activities accepted by A_i is an allowable behavior of the service. In the original definition of composability, the activity automata $A_1, \ldots, A_r$ are composed through interleavings between the activities in the sequences $w_1, \ldots, w_r$. Clearly, if activities between two services are disjoint, the original definition of composability becomes T-composability with $T(a)$ being a singleton set for every symbol a (i.e., each activity a belongs to a unique activity automaton). When the activity automata share some common activities (e.g., a belongs to both A_1 and A_2; i.e., $T(a) = \{1, 2\}$), the T-composability definition implies that an a-activity in A_1 must be synchronized with an a-activity in A_2. This is why in T-composability, such a symbol a must be assigned to both A_1 and A_2. Notice that the assignments of each symbol (activity) is deterministic in T-composability. The determinism helps us generalize the above theorem as follows.

A *reset-NCM* M is an NCM that is equipped with a number of *reset states* and is further augmented with a number of *reset counters* (in addition to the reversal-bounded counters). The reset counters are all reset to 0 whenever M enters a reset state. (As usual, we assume that initially the counters start with 0, i.e., with a reset state) We further require that on any execution, the reset counters are reversal-bounded between any two resets. One may similarly define a *reset-NPCM*. Notice that an NCM (resp. NPCM) is a special case of a reset-NCM (resp. reset-NPCM) where there is no reset counter.

Theorem 6. *The emptiness problem for reset-NCMs is decidable.*

We use reset-NPM to denote a reset-NPCM that contains only reset counters and a stack. One can show that the emptiness of reset-NPMs is undecidable.

Theorem 7. *The emptiness problem for reset-NPMs and hence reset-NPCMs is undecidable.*

Now, we generalize Theorem 5 as follows.

Theorem 8. *T-composability of $(A; A_1, \ldots, A_r)$ is decidable when A is an NCM and each A_i is a reset-DCM.*

Let NPDA (DPDA) denote a nondeterministic (deterministic) pushdown automaton. Thus, an NPDA is a special case of a reset-NPM, one that does not have reset counters. Using Theorem 7, one can show,

Theorem 9. *T-composability of $(A; A_1, ..., A_r)$ is undecidable when A is a DPDA and each A_i is a reset-DCM, even for the case when $r = 1$.*

4 Lookahead Delegator

Given k, a k-lookahead delegator (or simply k-delegator) for the system of NCMs $(A; A_1, ..., A_r)$ is a DCM D which, knowing the current states of $A, A_1, ..., A_r$ and the statuses (i.e., signs) of their counters (i.e., zero or non-zero), and the k lookahead symbols to the right of the current input symbol being processed, D can uniquely determine the transition of A, the assignment of the current symbol to one of $A_1, ..., A_r$, and the transition of the assigned machine. Moreover, for every string x accepted by A, D also accepts, i.e., the subsequence of string x delegated by D to each A_i is accepted by A_i. Clearly, if a system has a k-delegator (for some k), then it must be composable. However, the converse is not true, in general. For example, the system in Figure 1(a) is composable, but it does not have a k-delegator for any k.

Example 2. Consider again Example 1 and in particular the system $(A; A_1, A_2)$. It is easy to see that all a activities immediately preceding an s or c has to be delegated to A_1 or A_2, respectively. Without knowing which letter, s or c, will be coming, the delegator cannot correctly determine whether A_1 or A_2 should perform the activities a. Thus, the system has no k-delegator for any k. On the other hand, the system $(A; A_2, A_3)$ has a 1-delegator. It is straightforward to generalize this example (by adding additional states) to show that for every k, there exists a system that has a $(k+1)$-delegator but not a k-delegator. ∎

So that we can always have k lookahead, let \$ be a new symbol and f be a new state. Extend the transition function of A by defining the transition from any state, including f, on symbol \$ to f. Then make f the only (unique) accepting state. Thus the new NCM accepts the language $L(A)\$^+$ and it has only one accepting state f. We can do the same thing for $A_1, ..., A_r$ with $f_1, ..., f_r$ their unique accepting states. For convenience, call the new machines also $A, A_1, ..., A_r$.

For ease in exposition, in what follows, we assume that $r = 2$, and each of A, A_1, A_2 has only one reversal-bounded counter. Generalizations to any $r \geq 2$ and machines having multiple reversal-bounded counters is straightforward. Note that the transition of A has the form: $\delta_A(q, a, s) = \{..., (p, d), ...\}$, which means that if A in in state q and the input is a and the sign of its counter is s (zero or non-zero), then A can change state to p and increment the counter by d where $d = 0, +1, -1$, with the constraint that if $s = 0$, then $d = 0, +1$. The same holds for transitions δ_1 and δ_2 of A_1 and A_2. We assume that the counters are initially zero.

Let k be a nonnegative integer. We can construct a candidate k-delegator DCM D as follows: each state of D is a tuple (q, p_1, p_2, u), where q is a state of A, p_i is a state of A_i, and u is a string of length k. However, in the case (q^0, p_1^0, p_2^0, u), where q_0 is the initial state of A and p_i^0 the initial state of A_i, the length of u can be less that k, including zero length, in which case $u = \epsilon$. Then the initial state of D is $(q^0, p_1^0, p_2^0, \epsilon)$. The transition δ of D is defined as follows:

1. $\delta((q^0, p_1^0, p_2^0, \epsilon), 0, 0, 0, a) = ((q^0, p_1^0, p_2^0, a), 0, 0, 0)$ for all symbol a.
2. $\delta((q^0, p_1^0, p_2^0, v), 0, 0, 0, a) = ((q^0, p_1^0, p_2^0, va), 0, 0, 0)$ for all string v such that $|v| < k$ and symbol a.
3. $\delta((q, p_1, p_2, av), s, s_1, s_2, b) = ((q', p_1', p_2', vb), d, d_1, d_2)$ for all q, p_1, p_2, s, s_1, s_2, all string v such that $|v| = k$ and symbols a, b, where:
   (a) $(q', d) \in \delta_A(q, a, s)$;
   (b) either $p_1' = p_1$, $d_1 = 0$, and $(p_2', d_2) \in \delta_2(p_2, a, s_2)$
   or $p_2' = p_2$, $d_2 = 0$, and $(p_1', d_1) \in \delta_1(p_1, a, s_1)$.

   Moreover, the choice $((q', p_1', p_2'), d, d_1, d_2)$ once made is unique for the parameters $((q, p_1, p_2, av), s, s_1, s_2)$. (Note that, in general, there are many choices that can be made for the given parameters.)

4. Note that in (q, p_1, p_2, u), any suffix of u may be a string of \$'s.
5. Then $(f, f_1, f_2, \$^k)$ is the accepting state of D, where f, f_1, f_2 are the unique accepting states of A, A_1, A_2.

Now D is a DCM. Since the class of languages accepted by DCMs is effectively closed under complementation, we can construct a DCM E accepting the complement of $L(D)$. Then D is a k-delegator of $(A; A_1, A_2)$ iff $L(A) \cap L(E) = \emptyset$. We can construct from NCM A and DCM E an NCM F accepting $L(A) \cap L(E)$. We can then check the emptiness of $L(F)$ since the emptiness problem for NCMs is decidable. Now D is just one candidate for a k-delegator. There are finitely many such candidates. Every choice that can be made in item 3) above corresponds to one such candidate. By exhaustively checking all candidates, we either find a desired k-delegator or determine that no such k-delegator exists. Thus, we have shown the following:

Theorem 10. *It is decidable to determine, given a system of NCMs $(A; A_1, ..., A_r)$ and a nonnegative integer k, whether the system has a k-delegator.*

Since the emptiness problem for NPCMs is also decidable, we can generalize the above result to:

Corollary 4. *It is decidable to determine, given a system $(A; A_1, ..., A_r)$, where A is an NPCM and $A_1, ..., A_r$ are NCMs, and a nonnegative integer k, whether the system has a k-delegator.*

Corollary 5. *If we impose some Presburger constraint P on the delegation of symbols by the k-delegator (e.g., some linear relationships on the number of symbols delegated to $A_1, ..., A_r$), then the existence of such a P-constrained k-delegator is also decidable.*

Open Question: Is it decidable to determine, given a system of DCMs $(A, A_1, ..., A_r)$, whether it has a k-delegator for some k?

Corollary 6. *It is decidable to determine, given a system $(A; A_1, ..., A_r)$ and a nonnegative integer k, where A is a DPDA (deterministic pushdown automaton), A_1 is a PDA (nondeterministic pushdown automaton) and $A_2, ..., A_r$ are NFAs, whether the system has a DPDA k-delegator. (Here, the delegation depends also on the top of the stack of A_1.)*

For the special case when the machines are NFAs, we can prove the following (from the proof of Theorem 10 and Savitch's theorem):

Corollary 7. *We can decide, given a system $(A; A_1, ..., A_r)$ of NFAs and a nonnegative integer k, whether the system has a k-delegator in nondeterministic exponential time (in k and the sum of the sizes of the machines) and hence, also, in deterministic exponential space.*

5 Upper Composability

The definition of composability of $(A; A_, \ldots, A_r)$ in Section 2 implies that every behavior accepted by activity automaton A is an interleaving of behaviors accepted by $A_1, \ldots, A_r$. Therefore, A could be considered as a lower-approximation of the composition $(A_1, \ldots, A_r)$ from $A_1, \ldots, A_r$. Naturally, one would also consider an upper-approximation of $(A_1, \ldots, A_r)$. This suggests the following definition of upper composability. We say that the system $(A_1, \ldots, A_r; A)$ is *upper-composable* if, for any word $w = a_1a_2...a_n$ and for any assignment of each symbol a_j to one of the $A_1, ..., A_r$ such that w_i (the subsequence of symbols assigned to A_i) is accepted by A_i, w is accepted by A. That is, all the interleavings of any r words accepted by $A_1, \ldots, A_r$ respectively are also accepted by A. It is not hard to show that

Theorem 11. *The upper-composability of* $(A_1, \ldots, A_r; A)$ *is decidable in the following cases:*

- *Each* A_i *is an NCM and* A *is a DPCM.*
- $r = 1$ *and* A_1 *is an NPCM and* A *is a DCM.*

Clearly, due to Remark 1, The upper-composability of $(A_1; A)$ is undecidable when A_1 is a DFA and A is an NCM.

6 Composability of Timed Automata

A timed automaton [3] can be considered as a finite automaton augmented with a finite number of clocks. The clocks can reset to zero or progress at the same rate, and can be tested against clock constraints in the form of clock regions (i.e., comparisons of a clock or the difference of two clocks against an integer constant, e.g., $x - y > 6$, where x and y are clocks.). Timed automata are widely regarded as a standard model for real-time systems, because of their ability to express quantitative time requirements. In particular, by using the standard region technique, it has been shown that region reachability for timed automata is decidable [3]. This fundamental result and the technique are useful, both theoretically and practically, in formulating various timed temporal logics and developing verification tools (see [2] for a survey).

In this section, we study composability of discrete timed automata (DTA) A, where clocks take integer values. Formally, A *clock constraint* is a Boolean combination of *atomic clock constraints* in the following form: $x \sim c, x - y \sim c$, where $\sim$ denotes $\leq, \geq, <, >$, or $=$, c is an integer, x, y are integer-valued clocks. Let $\mathbf{N}$ be the set of integers with $\mathbf{N}^+$ for nonnegative integers. Let $\mathcal{L}_X$ be the set of all clock constraints on clocks X. A *discrete timed automaton* (DTA) A is a tuple $\langle Q, \Sigma, X, T \rangle$ where Q is a finite set of *(control) states*, Σ is the input alphabet, $X = \{x_1, \ldots, x_k\}$ is a finite set of integer-valued clocks, $T \subseteq Q \times 2^X \times \mathcal{L}_X \times (\Sigma \cup \{\epsilon\}) \times Q$ is a finite set of *transitions*. Each transition

$$\langle q, \lambda, l, a, q' \rangle \tag{1}$$

denotes a transition from state q to state q' with input a, *enabling condition* $l \in \mathcal{L}_X$ and a set of clock resets $\lambda \subseteq X$. Note that λ may be empty. Also note that since a state may be connected to more than one state through multiple edges with the same enabling condition, A is, in general, nondeterministic.

The semantics of A is defined as follows. A is equipped with a one-way input tape. Initially, A starts in a designated initial state with all the clocks being 0. An execution of A consists of firing a sequence of transitions in the form of (1). Firing a transition t in (1) takes A from the current state q to the next state q' while consuming (reading) input $a \in \Sigma \cup \{\epsilon\}$. This is possible only if the current clock values satisfies the enabling condition on the transition. Clocks values are updated as a result of the transition. That is, if there are no clock resets on the transition t (t is a *progress transition*, i.e., $\lambda = \emptyset$), then all the clocks progress by one time unit. If, however, $\lambda \neq \emptyset$ (i.e., t is a *reset transition*), then every clock in λ resets to 0 while every clock not in λ does not change. For simplicity, we may assume that on a progress transition, the input a read by the transition is always ϵ. otherwise, t can be simulated by a reset transition that resets a dummy clock and reads the symbol $a \in \Sigma$ followed by a progress transition that reads ϵ. The global clock is a clock that never resets (i.e., a clock indicating the current time). Without loss of generality, we assume that A contains a global clock.

We say that a word w is *accepted* by A when w is provided on the input tape, if A is able to enter a designated accepting state. We use $L(A)$ to denote the set of words accepted by A. For DTAs, one may develop a similar definition of composability as in Section 2. However, the definition does not justify the intended meaning of composability. For instance, let A_1 and A_2 be two DTAs, and suppose ac (resp. bd) are accepted by A_1 (resp. A_2). Observe that an interleaving like $abcd$ of the two words is not necessarily accepted by the DTA composed from A_1 and A_2. This is because, when composing, A_1 and A_2 share the same global clock. To devise a proper definition of composability for DTAs, we first introduce timed words [3].

A timed word is a sequence of pairs

$$(a_1, t_1) \dots (a_n, t_n) \tag{2}$$

such that each $a_i \in \Sigma, t_i \in \mathbf{N}^+$, and $t_1 \leq \dots \leq t_n$. We say that the timed word is *accepted* by A if $w = a_1 \dots a_n$ is accepted by A and this fact is witnessed by some accepting run of A such that each t_i is the timestamp (the value of the global clock) when symbol a_i is read in the run. Thus, the timed word not only records the sequence of symbols $a_1 \dots a_n$ accepted by A but also remembers the timestamp when each symbol is read. Let $A, A_1, \dots, A_r$ be DTAs. A timed word in the form of (2) is *timed composable* if there is an assignment of each pair (a_j, t_j) to one of the $A_1, ..., A_r$ such that, for $1 \leq i \leq r$, the subsequence (also a timed word) of pairs assigned to A_i is accepted by A_i. We say that $(A; A_1, \dots, A_r)$ is timed composable if every timed word accepted by A is timed composable. The main result of this section is the following:

Theorem 12. *The timed composability of discrete timed automata* $(A; A_1, \dots, A_r)$ *is decidable.*

7 Conclusion

In this paper, we looked at the problems of composability and k-delegatability in systems of infinite-state automata (e.g. machines with reversal-bounded counters and pushdown stacks). Our investigation was motivated by automated design problems in the area of e-services/web-services. We derived decidable and undecidable results for various types of machines. In particular, we generalized earlier results on composability and k-delegatability and resolved some interesting problems in the literature. In the future, we plan to look at how our techniques and results can be applied to other areas, e.g., in workflows and verification.

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. 16th Int. Colloq. on Automata, Languages and Programming*, 1989.
2. R. Alur. Timed automata. In *CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999.
3. R. Alur and D. Dill. Automata for modeling real-time systems. *Theoretical Computer Science*, 126(2):183–236, 1994.
4. B. Baker and R. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8:315–332, 1974.
5. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC)*, volume 2910 of *LNCS*, pages 43–58, 2003.
6. J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
7. Z. Dang. Pushdown time automata: a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302:93–121, 2003.

8. Z. Dang, O. Ibarra, T. Bultan, R. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *Proc. Int. Conf. on Computer-Aided Verification (CAV)*, pages 69–84, 2000.
9. Z. Dang, O. H. Ibarra, and R. A. Kemmerer. Generalized discrete timed automata: decidable approximations for safety verification. *Theoretical Computer Science*, 296:59–74, 2003.
10. Z. Dang, O. H. Ibarra, and P. San Pietro. Liveness Verification of Reversal-bounded Multicounter Machines with a Free Counter. In *FSTTCS'01*, volume 2245 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2001.
11. Z. Dang, O. H. Ibarra, and Z. Sun. On the emptiness problems for two-way nondeterministic finite automata with one reversal-bounded counter. In *ISAAC'02*, volume 2518 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2002.
12. Z. Dang, P. San Pietro, and R. A. Kemmerer. Presburger liveness verification for discrete timed automata. *Theoretical Computer Science*, 299:413–438, 2003.
13. C. E. Gerede, R. Hull, and J. Su. Automated composition of e-services with lookahead. Technical report, UCSB, 2004.
14. S. Ginsburg and E. Spanier. Semigroups, presburger formulas, and languages. *Pacific J. of Mathematics*, 16:285–296, 1966.
15. T. Harju, O. Ibarra, J. Karhumaki, and A. Salomaa. Some decision problems concerning semilinearity and computation. *Journal of Computer and System Sciences*, 65:278–294, 2002.
16. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.
17. O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines. *SIAM J. Comput.*, 24:123–137, 1995.
18. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. IEEE Symposium on Logic In Computer Science*, 2001.
19. S. Lu. *Semantic Correctness of Transactions and Workflows*. PhD thesis, SUNY at Stony Brook, 2002.
20. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. Principles of Distributed Computing*, pages 137–151, 1987.
21. M. Minsky. Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.
22. M. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44(4):71–77, 2001.
23. P. San Pietro and Z. Dang. Automatic verification of multi-queue discrete timed automata. In *COCOON'03*, volume 2697 of *Lecture Notes in Computer Science*, pages 159–171. Springer, 2003.
24. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. IEEE Symp. on Foundations of Computer Science*, 1990.
25. W. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
26. G Senizergues. The equivalence problem for deterministic pushdown automata is decidable. volume 1256 of *Lecture Notes in Computer Science*, pages 671–681. Springer, 1997.
27. M. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Proc. Workshop on Database Programming Languages (DBPL)*, 1995.
28. W. M. P. van der Aalst. On the automatic generation of workflow processes based on product structures. *Computer in Industry*, 39(2):97–111, 1999.

Appendix: Proofs not presented in the paper

Proof of Theorem 1.

Proof. The proof uses the fact that it is undecidable to determine, given a (deterministic) two-counter machine, whether it halts when started with the two counters being zero [21]. Without loss of generality, we assume that when the machine halts, it does so after a positive even number of steps.

Let $\Sigma = \{\$, \underline{\$}, \theta, \lambda, a, b, c, \underline{a}, \underline{b}, \underline{c}\}$ be an alphabet. Let $m > 0$. An *m-block word* is a string in the following form:

$$B_1 \ldots B_m \$\underline{\$}\theta. \tag{3}$$

Each *block* B_i is in the following form:

$$\$\underline{\$}C\lambda\$\underline{\$}C'\lambda$$

where C is in the form of $a^i b^j c^k$ (for some numbers i, j, k) and C' is in the form of $\underline{a}^{i'}\underline{b}^{j'}\underline{c}^{k'}$ (for some numbers i', j', k'). C is intended to encode a configuration of a two-counter machine where the state is i, the two counter values are j and k respectively. C' is also intended to encode a configuration where the state is i', the two counter values are j' and k' respectively. However, since C is a word on alphabet $\{a, b, c\}$ and C' is a word on alphabet $\{\underline{a}, \underline{b}, \underline{c}\}$, we call C as a *plain configuration* and C' as a *dot configuration*. The string $\$\underline{\$}$ is called a separator. Hence, each word in (3) can be described as a concatenation of m blocks, followed by an additional separator along with an end marker θ. Each block is a concatenation of a separator, a plain configuration, a λ, a separator, a dot configuration, and a λ. A *block word* is an m-block word for some $m > 0$.

Let M be a deterministic two-counter machine. A block word is *valid* if it is in (3) for some $m > 0$, and in the word,

- the state encoded in the plain/dot configuration in every block is in the range of $1..n$ (where n is the number of states of M),
- the first plain configuration encodes the initial configuration of M and the last dot configuration encodes a halting configuration of M.

Let L be the set of all valid block words. Clearly, L is a regular language and can be accepted by a DFA A.

Before we proceed further, some more definitions are needed. Let C and C' be a plain configuration and a dot configuration, respectively. A *semi-block* is a word that is either $\$\underline{\$}C\lambda\$\underline{\$}C'\lambda\$$ or $\$\underline{\$}C'\lambda\$\underline{\$}C\lambda\$$, for some C and C'.

Notice that for each block word in the form of (3), there are many substrings that are semi-blocks. Each such substring is called an *embedded semi-block*. Now, we construct two languages L_1 and L_2. L_1 is the set of all words w such that w is a result of dropping one embedded semi-block from some block word. Clearly, after dropping an embedded semi-block, a block word is no longer a block word. Therefore, $L \cap L_1 = \emptyset$. L_2 is the union of two languages L_2^1 and L_2^2. L_2^1 is the set of all semi-blocks in the form of $\$\underline{\$}C\lambda\$\underline{\$}C'\lambda\$$ such that C can not reach C' by one move in M. Similarly, L_2^2 is the set of all semi-blocks in the form of $\$\underline{\$}C'\lambda\$\underline{\$}C\lambda\$$ such that C' can not reach C by one move in M. Since a semi-block itself is not a block word, we have $L \cap L_2 = \emptyset$. Clearly, L_1 is still a regular language and can be accepted by a DFA A_1. L_2 can be accepted by a DCM A_2 with two 1-reversal counters.

We claim that $(A; A_1, A_2)$ is composable iff M does not halt right after $2m-1$ moves for some $m > 0$.

($\Rightarrow$) Assume that $(A; A_1, A_2)$ is composable. Let w be a valid block word in (3). Hence, $w \in L$. Therefore, there is a way to assign each symbol in w either to A_1 or to A_2, such that $w_1 \in L_1$ and $w_2 \in L_2$, where w_1 (resp. w_2) is the subsequence of symbols assigned to A_1 (resp. A_2). Since L is disjoint with L_1 and with L_2, both w_1 and w_2 are not empty words. Notice that A_2 only accepts words w_2 in the form of a semi-block, i.e., w_2 is

$$\$\underline{\$}C\lambda\$\underline{\$}C'\lambda\$ \tag{4}$$

or

$$\$\underline{\$}C'\lambda\$\underline{\$}C\lambda\$ \tag{5}$$

for some C and C'. We use w_2' to denote the result of dropping the first symbol and the last symbol from w_2. That is, $w_2 = \$w_2'\$$. A key observation here is that the substring w_2' in the word w_2 delegated to A_2 must be a substring of w. To see this, we write $w = w_p w_s$, where the w_p is delegated entirely to A_1 and the first symbol of w_s is delegated to A_2. Since $w_2 = \$w_2'\$$, the first symbol of w_s is also the first symbol \$ in w_2. Now, where is the second symbol (that must be $\underline{\$}$) in w_s delegated ? We have two cases to consider.

Case 1. The second symbol $\underline{\$}$ in w_s is delegated to A_2. In this case, all the immediately following non-\$ symbols must also be delegated to A_2. Otherwise, the resulting w_1 can not be in L_1. Suppose that w_2 is in (4) (the case when w_2 is in (5) is similar). In this case, these non-\$ symbols are exactly $C\lambda$. Then, we write w into $w_p\$\underline{\$}C\lambda w_s'$, where w_p is delegated to A_1 and $\$\underline{\$}C\lambda$ is delegated to A_2. Clearly, w_s' starts with \$. This \$ must also be delegated to A_2. Otherwise, the prefix of w_s' that contains $\$\underline{\$}$ followed by a dot configuration must be entirely delegated to A_1. This implies that w_1 either starts with $\$\underline{\$}$ followed by a dot configuration or contains a substring that is a dot configuration (the last configuration encoded in w_p) followed (delimited with $\lambda\$\underline{\$}$) by another dot configuration (the first configuration encoded in w_s'). This is not possible according to the definition of L_1. Therefore, the first symbol \$ in w_s' must be delegated to A_2. Similar reasonings will show that the second symbol $\underline{\$}$ in w_s' as well as all the immediately following non-\$ symbols must also be delegated to A_2. These latter non-\$ symbols are exactly $C'\lambda$. Therefore, w_2' is a substring of w.

Case 2. The second symbol $\underline{\$}$ in w_s is delegated to A_1. Recall that the first symbol \$ in w_s is already delegated to A_2 (i.e., is the first symbol in w_2). We write $w_s = \$\underline{\$}w_p'w_s'$, where all symbols (except the first symbol \$) in $\$\underline{\$}w_p'$ are entirely delegated to A_1, and $\underline{\$}$ is the first symbol of w_s' and delegated to A_2. Notice also that w_p' must contain a substring encoding a plain/dot configuration. Now, we are using arguments similar to the ones made in Case 1 to show that w_2' is a prefix of w_s'. To see this, we assume that w_2 is in (4) (the case when w_2 is in (5) is similar). That is, $w_2' = \underline{\$}C\lambda\$\underline{\$}C'\lambda$. First, notice that after the first symbol $\underline{\$}$ in w_s' is delegated to A_2 (this $\underline{\$}$ corresponds to the first symbol in w_2'), all the immediately following non-\$ symbols in w_s' must also be delegated to A_2. Otherwise, the resulting $w_1 \notin L_1$. Hence, w_s' can be written into $\underline{\$}C\lambda w_s''$ for some w_s''. Clearly, the w_s'' must start with \$. Then, where is this \$ delegated? It must be also delegated to A_2. Otherwise, the resulting w_1 would contain a dot configuration (the last configuration encoded in w_p') immediately followed (delimitted by $\lambda\$\underline{\$}$) by another dot configuration (the first configuration encoded in w_s'') and clearly such w_1 can not be in L_1 by definition. Hence, the first symbol \$ in w_s'' must be delegated to A_2. The second symbol $\underline{\$}$ in w_s'' must also be delegated to A_2. Otherwise, the resulting w_1 contains at least two occurrences of $\underline{\$}$ such that in each occurrence, the symbol immediately before the $\underline{\$}$ is not \$ (The first occurrence is the second symbol $\underline{\$}$ in w_s which is delegated to A_1; the second occurrence is the second symbol $\underline{\$}$ in w_s'', which would be delegated to A_1). These two occurrences of $\underline{\$}$'s make $w_1 \notin L_1$. From here, one can show that the first two symbols in w_s'' as well as all the immediately following non-\$ symbols must be delegated to A_2; i.e., $\$\underline{\$}C'\lambda$ is a prefix of w_s''. Therefore, w_2' is a substring of w.

Since w'_2 is a substring of w, w does not encode an execution of M; i.e., M does not halt right after $2m-1$ moves for some $m>0$, since each w in (3) contains exactly $2m$ configurations.

($\Leftarrow$) Assume that $(A; A_1, A_2)$ is not composable. Therefore, there is a valid block word w in L that witnesses the assumption. Let w_2 be any embedded semi-block in w. Clearly, when we assign this w_2 in w to A_2, the remaining word w_1 is in L_1. The assumption forces that $w_2 \notin L_2$. Hence, w_2 encodes a valid move in M. The result holds for every embedded semi-block w_2. Therefore, w itself encodes a halting execution of M; i.e., M halts right after $2m-1$ moves for some $m>0$.

Since the halting problem for two-counter machines (starting with both counters zero) is undecidable, it follows it is undecidable whether $(A; A_1, A_2)$ is composable when A and A_1 are DFAs and A_2 is a DCM with two 1-reversal counters. ∎

Proof of Corollary 1.

Proof. Consider a single-tape Turing machine Z over the alphabet $\{s_1, s_2, s_3\}$ (one symbol represents blank). Let $q_1, q_2, \ldots$ be the states (distinct from s_1, s_2, s_3), where q_1 is the initial state, and q_2 is the unique halting state. A configuration of the TM on the tape is a string (word) $C = uq_iv$, where u and v are strings in $\{s_1, s_2, s_3\}^*$. This represents the configuration where the tape content is uv, the read/write head is on the first symbol of v, and the state is q_i. As in Theorem 1, we can introduce dot symbols to encode dot configurations and use markers to separate plain configurations and dot configurations. Now a block B_i is in the following form: $\$\underline{\$}\underline{C}\lambda\$\underline{\$}\underline{C'^r}\lambda$ where C'^r is a dot configuration written in reverse. In the construction of A_2 to accept $L_2 = L_2^1 \cup L_2^2$, A_2 checks that given any two successive configurations (note that one is written in reverse relative to the other), the second is not a valid successor of the first. Clearly, A_2 needs only a 1-reversal pushdown stack. The result follows from the undecidability of the halting problem for TMs. ∎

Proof of Theorem 2.

Proof. First we construct from $A_1, ..., A_r$ an NFA B which accepts a string $w \in \Sigma^*$ iff there is an assignment of the symbols in w to the A_i's such that if w_i is the subsequence assigned to A_i, then w_i is accepted by A_i.

B is constructed as follows. Let δ_i be the transition function of A_i. The initial state of C is $(q_1^0, ..., q_r^0)$, where q_i^0 is the initial state of A_i. The transition δ of B is defined by: $\delta((q_1, ..., q_r), a) = \{(p_1, ..., p_r) \mid \text{ for some } 1 \le i \le r, p_i \in \delta_i(q_i, a) \text{ and } p_j = q_j \text{ for all } j \neq i\}$. The accepting states of B are all states $(q_1, ..., q_r)$ such that each q_i is an accepting state of A_i.

Then we construct from A and B an NPCM C that accepts a string x if it is accepted by A but is not accepted by B. C operates by guessing the symbols of x bit-by-bit and simulating A and B in parallel on x. Simulation of A is straightforward. To simulate B, C uses the "subset construction" technique (for converting an NFA to DFA without actually doing the conversion) and builds/updates the subset as it processes the symbols of x. At the end of the input, C checks that there is no accepting state in the reachable subset. C accepts if A accepts and B rejects. Clearly, the system is not composable iff C accepts a nonempty language. The result follows since the emptiness problem for NPCMs is decidable [16]. ∎

Proof of Theorem 3.

Proof. We prove the result for the general case when A is an NPCM accepting a language which is a subset of $w_1^* ... w_k^*$, and one of the A_i's is an NPCM. As in the proof of Theorem 2, we construct an NPCM B which accepts a string $w \in \Sigma^*$ iff there is an assignment of the symbols in w to the

A_i's such that if w_i is the subsequence assigned to A_i, then w_i is accepted by A_i. Then we modify B to an NPCM B' that accepts the language $L(B) \cap w_1^* ... w_k^*$. Clearly, the system is composable iff $L(A) \subseteq L(B')$. The result follows since the containment problem for NPCMs accepting bounded languages is decidable [16]. ∎

Proof of Theorem 4.

Proof. Notice that n is fixed. Therefore, there are only finitely many choices to assign the (at most) n shared symbols to $A_1, \ldots, A_r$. Let τ be one such choice. Clearly, under the choice τ, each symbol in a word w can be uniquely assigned to one of $A_1, \ldots, A_r$; let w_i denote the subsequence assigned to A_i from w. One can construct a DCM B^τ from $A_1, \ldots, A_r$ such that w is accepted by B^τ iff each $w_i \neq \epsilon$ is accepted by A_i. The result follows, since the n-composability is equivalent to checking, for each τ, $L(A) \subseteq L(B^\tau)$, which is decidable. This is because we can construct, from A and B^τ, an NPCM C that accepts a string x if it is accepted by A but is not accepted by B^τ. As in the proof of Theorem 2, C operates by guessing the symbols of x bit-by-bit and simulating A and B in parallel on x. Simulation of A is straightforward, as is the simulation of B^τ, since it is deterministic. Then the system is not n-composable (for the given τ) iff C accepts a nonempty language, which is decidable. ∎

Proof of Corollary 3.

Proof. This follows directly from the proof of Theorem 1. We just treat L_2 as the intersection of a regular language (hence, accepted by a DFA and a language definable by some Presburger formula P. ∎

Proof of Theorem 5.

Proof. Clearly, $(A; A_1, \ldots, A_r)$ is not T-composable if and only if there is some w and $1 \leq t \leq r$ such that $w \in L(A)$ but $(w)_t \notin L(A_t)$. Hence (for both parts), we can construct an NPCM B which, when given a string w, guesses t and simulates A (guessing the transitions of A, since it is nondeterministic) and A_t in parallel and accepts if A accepts w and $(w)_t \neq \epsilon$ and is not accepted by A_t . The result follows since the emptiness problem for NPCMs is decidable. ∎

Proof of Theorem 6.

Proof. Let A be a reset-NCM. Without loss of generality, we assume that, when A accepts an input word w, it does so on a reset state and at the end of the input and with all the counters 0. Since an r-reversal-bounded counter can be simulated by $2r$ 1-reversal-bounded counters, we further assume, WLOG, each reversal-bounded counter in A makes exactly one reversal. An accepting execution of A on w can therefore be split into one or more *segments*, where each segment starts and ends with a reset state (i.e., the reset counters are all 0 at the beginning and at the end of the segment) and in between, A does not enter a reset state. We say that a segment is *monotonic* if each reversal-bounded counter is either nondecreasing or nonincreasing on the segment (i.e., the reversal-bounded counters do not make any reversals on the segment). We identify the segment with q (the starting reset state), q' (the ending reset state), and τ (a mode vector that tells the mode (nondecreasing/nonincreasing) of each reversal-bounded counter within the segment). Clearly, there are only a bounded number B of segments that are not monotonic segments on any accepting executions on all w. On a monotonic segment with identification (q, q', τ), we use vector $\mathbf{v}$ to denote the net increment/decrement for each reversal-bounded counter. Since, on the segment, A can be simulated by an NCM, all such $\mathbf{v}$ for all possible segments with the same identification clearly forms a semilinear set (or, equivalently, a Presburger relation). Thus, the set can be "generated" by $M_{(q,q',\tau)}$, a counter machine with nondecreasing counters [15]. (By convention, we assume that the machine crashes prematurely if the set is empty.) Now, we are ready to construct an NCM M to simulate A on w. M starts with the initial state of A. Whenever A runs on a monotonic segment identified with some (q, q', τ), M runs $M_{(q,q',\tau)}$ to update (increment/decrement according to τ) its own reversal-bounded counters. Whenever A runs on a nonmonotonic segment, M simulates A faithfully. Notice that the reversal-bounded counters in M are still reversal-bounded even though we use $M_{(q,q',\tau)}$ to perform updates. When A accepts, M makes sure that each reversal-bounded counter returns to 0. Clearly, A accepts a nonempty language iff M does. The result follows, since M only uses a finite number of reversal-bounded counters: the original reversal-bounded counters in A, monotonic counters in each $M_{(q,q',\tau)}$, reset counters (only resets at most B times, e.g., can be made reversal-bounded). ∎

Proof of Theorem 7.

Proof. Let M be a deterministic two-counter machine. Similar to the proof of Theorem 1, we use $a^i b^j c^k$, called a plain configuration, to encode a configuration of a two-counter machine where the state is i and the two counter values are j and k respectively. For the same configuration, we may also use $a^i c^k b^j$, called a reverse configuration, to encode it. Consider a sequence w

$$\$C_0\$C_1\$\ldots\$C_m\$$$

of configurations, where C_0, C_2, C_4,... are plain configurations, and C_1, C_3, C_5,... are reverse configurations. w is *accepting* if it encodes a halting execution of M; i.e, C_0 is the initial configuration of M, C_m is an accepting configuration of M, and for each $1 \leq t \leq m$, C_{t-1} reaches C_t in a move of M. One can construct a reset-NPM to accept the set of all valid w's. The result follows. ∎

Proof of Theorem 8.

Proof. Similar to the proof of Theorem 5 but using Theorem 6 instead. ∎

Proof of Theorem 9.

Proof. Let M be a reset-NPM. We modify M into another reset-NPM M' as follows. M' simulates M on input word w. Whenever M fires a transition t, M' reads an input symbol t. Of course, if the transition of M also reads an input symbol a, M' reads the same input symbol a along with the input symbol t. The stack and counter operations remain in M'. Hence, M' accepts an accepting execution of M. Note that M' is deterministic since the state transitions are provided on its input tape. Clearly, $L(M) = \emptyset$ iff $L(M') = \emptyset$. Now, let A be obtained from M' by ignoring the operations for reset counters, and A'_1 be obtained from M' by ignoring the stack operations. Notice that $L(A) \cap L(A'_1) = L(M')$. A is a DPDA, and A'_1 is a DCM. Take A_1 to be the DCM that accepts the complement of $L(A'_1)$. Observe that $(A; A_1)$ is not T-composable iff $L(A) \cap L(A'_1) = L(M') = \emptyset$. The result follows from Theorem 7. ∎

Proof of Corollary 5.

Proof. It is known that every Presburger set P (or, equivalently, semilinear set) can be accepted by a DCM [16]. Thus, we can augment the DCM D in the proof of Theorem 10 and Corollary 4 with a DCM accepting P. ∎

Proof of Corollary 6.

Proof. The proof is similar to that of Theorem 10 using the fact that equivalence of DPDAs is decidable [26]. ∎

Proof of Theorem 12.

Proof. We need to represent a timed word w in (2) as a *relative word* $\hat{w}$, in the following form:

$$\$^{t_1} a_1 \$^{t_2 - t_1} a_2 \ldots \$^{t_n - t_{n-1}} a_n \tag{6}$$

where \$ is a new symbol. We now construct two DTAs A' and A'', both of which run on relative words. On a relative word $\hat{w}$ in (6), A' simulates A as follows. A' is exactly the same as A except that, whenever the global clock in A progresses by one time unit, so does the global clock in A' together with a read of symbol \$ from $\hat{w}$. On the other hand, A'' simulates a composition of $A_1, \ldots, A_r$. On a relative word $\hat{w}$ in (6), A'' runs all of $A_1, \ldots, A_r$ in parallel and synchronously (they all share the same global clock) as follows. A'' keeps reading symbols from $\hat{w}$. When A'' reads a symbol \$ from $\hat{w}$, every A_i executes a progress transition. When A'' reads a symbol a_i from $\hat{w}$, A'' guesses one of $A_1, \ldots, A_r$ and let the guessed automaton execute a reset transition that reads the symbol a_i. At any moment before and after A'' reads a symbol (\$ or an a_i), A'' may also let each of $A_1, \ldots, A_r$ execute a sequence of reset transitions that reads only ϵ's (the length of the sequence for each A_i is nondeterministically determined). Initially, all of $A_1, \ldots, A_r$ start with their initial states. A'' accepts $\hat{w}$ if when the entire word is read by A'', each of $A_1, \ldots, A_r$ enters an accepting state. Notice that both A' and A'' are DTAs. Clearly, $(A; A_1, \ldots, A_r)$ is timed composable iff $L(A') \subseteq L(A'')$. The result follows, since languages (sets of words, instead of sets of timed words) accepted by DTAs are regular using the region technique [3]. ∎