MATHEMATICS OF OPERATIONS RESEARCH
Vol. 27, No. 2, May 2002, pp. 272–293
Printed in U.S.A.

ON DECIDING STABILITY OF CONSTRAINED HOMOGENEOUS RANDOM WALKS AND QUEUEING SYSTEMS

DAVID GAMARNIK

We investigate stability of scheduling policies in queueing systems. To this day no algorithmic characterization exists for checking stability of a given policy in a given queueing system. In this paper we introduce a certain *generalized priority* policy and prove that the stability of this policy is algorithmically undecidable. We also prove that stability of a homogeneous random walk in $\mathcal{Z}_+^d$ is undecidable. Finally, we show that the problem of computing a fluid limit of a queueing system or of a constrained homogeneous random walk is undecidable. To the best of our knowledge these are the first undecidability results in the area of stability of queueing systems and random walks in $\mathcal{Z}_+^d$. We conjecture that stability of common policies like First-In-First-Out and priority policy is also an undecidable problem.

1. Introduction. We consider a queueing system that operates under a specific and fixed scheduling policy. The system consists of a single server and several buffers in which arriving jobs are stored. We assume that the arriving parts may require several stages of processing, in which case each stage corresponds to a different buffer. The jobs arrive in a deterministic fashion: The interarrival times are fixed and known. All the processing times are also deterministic. A scheduling policy is a rule that specifies how the arriving parts are processed in the queueing system. Common scheduling policies include First-In-First-Out (FIFO), Last-In-First-Out (LIFO), Longest-In-System (LIS), Shortest-In-System (SIS), priority policy, etc. The priority policy is an example of a state-dependent policy—The scheduling decision depends only on the current vector of queues of the system and is independent of the past configurations and the past decision rules. FIFO, LIS, and SIS, on the other hand, are not state dependent in that sense. One can make them state dependent if additional information, like order of arrivals, is incorporated into the state.

A scheduling policy is defined to be stable if there is a finite uniform upper bound on the total number of parts in the system at all times. A necessary condition for stability of any work-conserving policy is the load condition: The traffic intensity of the station is not bigger than one. Many results have demonstrated that this condition is not sufficient for stability. The results were obtained primarily in the context of stochastic networks (Bramson 1994, Dai 1995, Down and Meyn 1997, Lu and Kumar 1991, Rybko and Stolyar 1992), deterministic fluid networks (Bertismas et al. 1996, Dai 1995, Dai and Vande Vate 2000, Dai et al. 1999, Dai and Weiss 1996), and deterministic adversarial networks (Andrews et al. 2001, Borodin et al. 2001, Gamarnik 2000, Goel 1999). One of the earliest results in the area was obtained by Rybko and Stolyar (1992) and Lu and Kumar (1991). They showed that a simple priority policy can lead to instability in some queueing networks even if the load condition is met. Bramson (1994) and Seidman (1994) showed that even FIFO policy can be unstable in queueing networks. Instability of FIFO was later demonstrated in an adversarial queueing setting by Andrews et al. (2001). Dai (1995) and Stolyar (1995) established that stability of a deterministic fluid queueing network implies stability of a

Received May 22, 2000; revised November 9, 2001.
MSC 2000 subject classification. Primary: 60K25, 68M20, 90B22.
OR/MS subject classification. Primary: Queues/priority.
Key words. Queueing system, scheduling policy, stability.

0364-765X/02/2702/0272/$05.00
1526-5471 electronic ISSN,

stochastic queueing network. A similar result was established by Gamarnik (2000), which connects stability of fluid and adversarial queueing networks. A complete characterization of two-station fluid networks that are stable under any work-conserving policy was established by Bertsimas et al. (1996) and Dai and Vande Vate (2000). Goel (1999) constructed a complete characterization of adversarial queueing networks that are stable under the usual load condition. The result is extended by Gamarnik (1999).

Despite the progress, no explicit or algorithmic characterization is known for checking stability of a given policy in a given network. That is, no algorithm is available which, on an input "queueing system description + scheduling policy," would output "yes" if the policy is stable in the network and "no" otherwise. (Of course, the scheduling policy should be computable for this question to make sense.) Such an algorithm is not known even for specific policies like FIFO, LIFO, or priority policies.

Motivated by a queueing network model, stability of homogeneous random walks in a nonnegative orthant $\mathbb{Z}_+^d$ ($\mathbb{Z}_+$ is the set of nonnegative integers) was considered in several papers: Malyshev (1972), Menshikov (1974), Fayolle (1989), Ignatyuk and Malyshev (1993), and Malyshev (1993). The transition vectors Δ have deterministically bounded length in max norm, and the transition probability $p(\Lambda, \Delta)$ along the direction Δ depends only on the face Λ that the random walk is currently on (the transition probabilities depend only on which coordinates of the current state are positive and which are zero). Such a random walk is defined to be stable if it is positive recurrent. We will also consider deterministic walks for which $p(\Lambda, \Delta)$ is always zero or one (the transition deterministically depends on the face that the walk is currently on).

A complete characterization of stable homogeneous random walks in $\mathbb{Z}_+^d$ for $d \leq 4$ was obtained in Malyshev (1972), Menshikov (1974), and Ignatyuk and Malyshev (1993), respectively, but no extension of this classification to higher dimensions has been obtained. A very interesting and deep connection between the homogeneous random walks and continuous dynamical systems on compact manifolds is constructed by Malyshev (1993). This paper shows that the difficulty of classifying stable random walks is of the same nature as the difficulty of understanding the dynamics of these dynamical systems. Specifically, the complicated dynamics precludes obtaining classification of stable random walks for $d = 5$, although no formal proof of the impossibility of the classification is provided.

In this paper we demonstrate the fundamental reason for the absence of stability characterizations for the models above. We prove that even for simplified *deterministic* homogeneous walks in $\mathbb{Z}_+^d$, stability is not an algorithmically decidable property for general d. No conceivable computational procedure can exactly characterize stability conditions for this model. This settles in a somewhat unexpected way an open problem on characterizing stable random walks in $\mathbb{Z}_+^d$ for $d \geq 4$. We then propose a certain class of *generalized priority* scheduling policies. For this class of policies the scheduler makes a decision depending only on which buffers have parts present and which buffers are empty. In other words, the scheduling decision is a function of the vector $b \in \{0, 1\}^n$, describing the presence/absence status of each buffer, where n is the number of buffers. As such, the policy is completely state dependent. The scheduling policy allows idling—For some binary vectors the corresponding decision could be "Do not serve until the state has changed." We prove that the stability of a generalized priority policy is algorithmically undecidable. The result holds under fairly conservative assumption on the queueing system: All of the interarrival times are deterministic, and all of the processing times are equal to one time unit. As a result, our model is more restrictive than the stochastic or adversarial models mentioned above. We conjecture that the stability of more common scheduling policies like FIFO or priority policies is undecidable as well.

Our undecidability results are, to the best of our knowledge, the first undecidability results in the area of stability of queueing systems and random walks. It has been largely motivated by similar results in the area of control theory. It is known that certain dynamical systems,

such as hybrid systems or piecewise affine systems, can simulate a Turing machine; see Henzinger et al. (1995) and Koiran et al. (1994). The stability of these systems is then reduced to the halting problem of the Turing machine, which is a classical example of an undecidable problem: No algorithm exists that, given "Turing machine + input word," will tell whether the Turing machine halts on the input word. For a definition of a Turing machine and discussion of a Turing halting problem, see Sipser (1997).

Recently, Blondel et al. (2001) demonstrated that global stability of a piecewise affine dynamical system is undecidable. This was established via reduction from a so-called *counter machine* (see Hooper 1966, Hopcroft and Ullman 1969), which is a variation of a Turing machine. We will also use a counter machine as a reduction tool. We will show how a counter machine can be embedded into a single-station queueing system and how a halting problem can be reduced to the question of stability. The rest of the paper is structured as follows. In the following section we describe homogeneous random walks in $\mathcal{Z}_+^d$ and a queueing model. In §3 we introduce a counter machine and state the *halting undecidability* result for a counter machine. In §4 we establish the undecidability of stability of a homogeneous random walk in a nonnegative orthant $\mathcal{Z}_+^d$ by a simple reduction from a counter machine. In §5 we prove undecidability of stability of a generalized priority policy in a single-station queueing system. Some extensions are discussed in §6. Specifically, we show that the problem of computing a fluid limit of a queueing system or of a constrained homogeneous random walk is undecidable. Concluding thoughts and some open problems are presented in §7.

2. Model description.

2.1. A homogeneous random walk in a nonnegative orthant. A discrete time homogeneous random walk (also called constrained random walk) $Q(t), t = 1, 2, \ldots,$ has a nonnegative orthant $\mathcal{Z}_+^d$ as its state space. For each $\Lambda \subset \{1, 2, \ldots, d\}$ let $\mathcal{Z}_\Lambda$ be the corresponding face:

$$\mathcal{Z}_\Lambda = \{(z_1, z_2, \ldots, z_d) \in \mathcal{Z}_+^d \colon z_i > 0 \text{ for } i \in \Lambda, z_i = 0 \text{ for } i \notin \Lambda\}.$$

The transition probabilities depend entirely on the face, which the random walk currently belongs to, and the transition vectors have at most unit length in max norm. In other words, for each $\Lambda \subset \{1, 2, \ldots, d\}$ and each $\Delta \in \{-1, 0, 1\}^d$ a certain value $p(\Lambda, \Delta)$ (the transition probability) is defined. These values satisfy

$$\sum_{\Delta \in \{-1, 0, 1\}^d} p(\Lambda, \Delta) = 1,$$

for each Λ and $p(\Lambda, \Delta) = 0$ if for some $i \notin \Lambda, \Delta_i = -1$. The latter condition is simply a consistency condition that prevents transitions into states with negative components. Given a current state $Q(t) \in \mathcal{Z}_+^d$ of the random walk, the next state is chosen to be $Q(t) + \Delta$ with probability $p(\Lambda, \Delta)$, if the state $Q(t)$ belongs to the face $\mathcal{Z}_\Lambda$. Note that a homogeneous random walk is defined in finitely many (although exponentially in d) terms.

We now consider a deterministic variant of our random walk, for which $p(\Lambda, \Delta) \in \{0, 1\}$. In other words, the transition vector $\Delta = \Delta(\Lambda)$ deterministically depends on the face. We call it a *deterministic homogeneous walk in a nonnegative orthant*. A homogeneous random (or deterministic) walk with initial state $Q(0)$ is defined to be stable if there exists some $C > 0$ (which may depend on the initial state $Q(0)$) such that the random walk starting from $Q(0)$ visits the set $\{z \in \mathcal{Z}_+^d : \sum_{i=1}^d z_i \le C\}$ infinitely often with probability one. Of course, stability may depend on the initial state $Q(0)$. We say that that the random walk is globally stable if it is stable for all the initial states. It is not hard to see that a deterministic

homogeneous walk is stable if and only if it is periodic: $Q(t+r) = Q(t)$ for some r and all large enough t. A random walk is defined to be ergodic if it is stable and irreducible. As a result it possesses a unique stationary distribution.

2.2. A single-station queueing system. A single-station queueing system $\mathcal{Q}$ consists of a single server and I types of parts arriving externally. The parts corresponding to type $i = 1, 2, \ldots, I$ visit the station J_i times. On each visit the part must receive a service before proceeding to the next visit. Only one part among all of the types can receive service at a time. While a part waits to receive the service on its jth visit, it is stored in buffer B_{ij}. We denote by n the total number of buffers $n = \sum_{i=1}^{I} J_i$. The service time for each part is a certain (deterministic) integral value $p_{i,j}$ that depends on the type i and the visit number $j \le J_i$. The type i parts arrive into the station in regular deterministic and integral intervals of length $1/\lambda_i$. In other words, λ_i is the arrival rate of the type i parts. For concreteness assume that the first arrival occurs at time $1/\lambda_i$ for all types. All of the arrivals and service completions occur at integer times $t = 0, 1, 2, \ldots$. The integrality of interarrival and service times is assumed for convenience but is not restrictive.

A scheduling policy π is defined to be a *generalized priority* policy if it operates in the following manner. A map π: $\{0,1\}^n \to \{0, 1, 2, \ldots, n\}$ is given. At each time $t = 0, 1, 2, \ldots$, the scheduler looks at the system and computes the binary vector $b = (b_1, b_2, \ldots, b_n) \in \{0,1\}^n$, where $b_i = 1$ if there are parts in the ith buffer and $b_i = 0$ otherwise. Then the value $k = \pi(b), 0 \le k \le n$ is computed. If $k > 0$, then the station processes a part in the kth buffer. If $k = 0$, the server idles. The map π is assumed to satisfy the consistency condition: if $\pi(b) = k > 0$, then $b_k = 1$. That is, processing can be done in buffer k only when there are jobs in buffer k. Note that the generalized priority scheduling policy is defined in finitely many terms and is completely state dependent—The scheduling decision at time t does not depend on the state of the queueing system at times $t' < t$. A usual priority policy corresponds to the case when there is some permutation θ of the buffers $\{1, 2, \ldots, n\}$ and $\pi(b) = k$ if and only if $b_k = 1$ and $b_i = 0$ for all i such that $\theta(i) < \theta(k)$. In words, priority scheduling policy processes parts in buffers with lowest value (highest priority) θ, which still has parts.

Given a generalized priority policy π and a vector of queue lengths $Q(0) = (Q_1(0), \ldots, Q_n(0))$ at time $t = 0$, a triplet $(\mathcal{Q}, \pi, Q(0))$ is defined to be stable if there exists a finite number $C > 0$ such that the total number of parts in the queueing system $\mathcal{Q}$ never exceeds C, when the system starts with $Q_i(0)$ parts in buffer i and operates using policy π. A pair $(\mathcal{Q}, \pi)$ is defined to be globally stable if it is stable for all the initial states $Q(0)$. Our goal is to show that the properties "$(\mathcal{Q}, \pi, Q(0))$ is stable" and "$(\mathcal{Q}, \pi)$ is globally stable" are undecidable. Note that the necessary condition for stability is the following load condition:

$$\rho \equiv \sum_{i=1}^{I} \sum_{j=1}^{J_i} \lambda_i p_{i,j} \le 1. \tag{1}$$

This condition is also sufficient for stability if the policy is work conserving, which does not apply here, since we allow idling $\pi(b) = 0$. Throughout the paper we assume that the load condition above holds.

3. Counter machines. A counter machine (see Blondel et al. 2001, Hopcroft and Ullman 1969) is a deterministic computing machine that is described by two counters R_1, R_2 and a finite number of states S. Each counter contains some nonnegative integer in its register. Depending on the current state $s \in S$ and depending on whether the content of the registers is positive or zero, the counter machine is updated as follows: The current state

s is updated to a new state $s' \in S$, and one of the counters has its number in the register incremented by one, decremented by one, or no change in the counters occurs.

Formally, a counter machine is a pair (S, Γ). $S = \{s_1, s_2, \ldots, s_m\}$ is a finite set of states, and Γ is the configuration update function Γ: $S \times \{0,1\}^2 \to S \times \{-2,-1,0,1,2\}$. A configuration of a counter machine is an arbitrary triplet $(s, z_1, z_2) \in S \times \mathcal{Z}_+^2$. A configuration (s, z_1, z_2) is updated to a configuration (s', z_1', z_2') as follows. First, a binary vector $b = (b_1, b_2)$ is computed where $b_i = 1$ if $z_i > 0$ and $b_i = 0$ if $z_i = 0$, $i = 1, 2$. If $\Gamma(s, b) = (s', 1)$, then the current state is changed from s to s', the content of the first counter is incremented by one, and the second counter does not change: $z_1' = z_1 + 1, z_2' = z_2$. We will also write Γ: $(s, z_1, z_2) \to (s', z_1 + 1, z_2)$ and Γ: $s \to s'$, Γ: $z_1 \to z_1 + 1$, Γ: $z_2 \to z_2$. If $\Gamma(s, b) = (s', -1)$, then the current state becomes s', $z_1' = z_1 - 1, z_2' = z_2$. Similarly, if $\Gamma(s, b) = (s', 2)$ or $\Gamma(s, b) = (s', -2)$, the new configuration becomes $(s', z_1, z_2 + 1)$ or $(s', z_1, z_2 - 1)$, respectively. If $\Gamma(s, b) = (s', 0)$ then the state is updated to s', but the contents of the counters do not change. This definition can be extended to the one that incorporates more than two counters but, in most cases, such extension is not necessary for our purposes.

Given an initial configuration (s^0, z_1^0, z_2^0) the counter machine uniquely determines subsequent configurations $(s^1, z_1^1, z_2^1), (s^2, z_1^2, z_2^2), \ldots, (s^t, z_1^t, z_2^t), \ldots$. We fix a certain configuration (s^*, z_1^*, z_2^*) and call it a *halting* configuration. If this configuration is reached, then the process halts and no additional updates are executed. The following theorem establishes the undecidability of a halting property.

THEOREM 1. (a) *Given a counter machine* (S, Γ), *initial configuration* (s^0, z_1^0, z_2^0), *and the halting configuration* (s^*, z_1^*, z_2^*), *the problem of determining whether the halting configuration is reached in finite time is undecidable. It remains undecidable even if* z_1^*, z_2^* *are restricted to be a zero.*

(b) *Given a counter machine* (S, Γ) *and the halting configuration* (s^*, z_1^*, z_2^*), *the problem of determining whether the halting configuration is reached in finite time for every initial configuration is undecidable.*

The part (a) of this theorem is a classical result and can be found in Hooper (1966). Part (b) was proven in Blondel et al. (2001). This theorem is the key result for the analysis in this paper.

4. Stability of a deterministic homogeneous walk in $\mathcal{Z}_+^d$. In this section we prove that it is generally impossible to obtain stability condition for a deterministic walk in a nonnegative orthant $\mathcal{Z}_+^d$, given the finite set of transition rules.

Recall from §2.1 that a deterministic homogeneous walk $Q(t)$ in $\mathcal{Z}_+^d$ is described by a collection of deterministic transition vectors $\{\Delta(\Lambda)\}_\Lambda$, such that each d-dimensional vector $\Delta(\Lambda)$ has all the components equal to $-1, 0, 1$ and if $Q(t) \in \mathcal{Z}_\Lambda$, then $Q(t+1) = Q(t) + \Delta(\Lambda)$. Then, given the initial position of the walk $Q(0)$ the trajectory of the walk $Q(1), Q(2), \ldots, Q(t), \ldots$ is uniquely determined. We defined the walk $Q(t)$ to be stable if there exist $C > 0$ such that $|Q(t)| \le C$ for infinitely many t, where $|Q(t)| = \sum_{i=1}^d Q_i(t)$. Observe that since the walk is deterministic, then the stability implies the existence of $C > 0$ for which $|Q(t)| \le C$ for all t.

THEOREM 2. *There does not exist an algorithm that, on an input* $(Q(0), \{\Delta(\Lambda)\}_\Lambda)$, *outputs "yes" if the deterministic walk* $Q(t)$ *with initial state* $Q(0)$ *and transition rules* $\{\Delta(\Lambda)\}_\Lambda$ *is stable, and outputs "no" otherwise. Thus, stability of a deterministic homogeneous walk in* $\mathcal{Z}_+^d$ *is not decidable.*

PROOF. We prove the theorem by a simple reduction from a counter machine. Given an arbitrary counter machine (S, Γ) with initial configuration (s^0, z_1^0, z_2^0) and halting configuration $(s^*, 0, 0)$, we will construct a deterministic walk that has dynamics very similar to

the dynamics of the counter machine. We then argue that if we had an algorithm for checking stability of a deterministic walk we could use this algorithm for checking whether the counter machine halts.

Let $S = \{s_1, s_2, \ldots, s_m\}$ and let $i^* \in \{1, 2, \ldots, m\}$ be the index of the halting state s^*. That is $s^* = s_{i^*}$. Our deterministic walk has a state space $\mathcal{X}_+^{m+2}$. The first m coordinates will be used to encode the states of the counter machine. We encode the state $s_i \in S$ by an m-dimensional unit vector e_i, where e_i has ith component equal to one and all other components equal to zero. The last two coordinates contain the values of the counters of the counter machine. Thus, the configuration (s_i, z_1, z_2) corresponds to the following state Q of the walk: $Q_i = 1, Q_{m+1} = z_1, Q_{m+2} = z_2$, and $Q_j = 0$ for all other coordinates j. We now describe the transition vectors Δ for each face $\mathcal{X}_\Lambda$ of the state space $\mathcal{X}_+^{m+2}$. Suppose $\Lambda = \{i, m+1, m+2\}$ for some $i \in \{1, 2, \ldots, m\}$. We compute $\Gamma(s_i, (1,1))$. If $\Gamma(s_i, (1,1)) = (s_j, 1)$ and $i \neq j$, then we set $\Delta_i(\Lambda) = -1, \Delta_j(\Lambda) = 1, \Delta_{m+1}(\Lambda) = 1$, and $\Delta_k(\Lambda) = 0$ for all other k. This simply means that if the current state $Q(t)$ of the walk encodes a state s_i and both coordinates $m+1, m+2$ correspond to positive contents, then $Q(t+1) = Q(t) + \Delta(\Lambda)$ will encode the state s_j, and the coordinate $m+1$ (the first counter) increases its value by one. If $i = j$, then we set $\Delta_{m+1}(\Lambda) = 1, \Delta_i(\Lambda) = 0$ for all other i. This corresponds to the case when the state s_i does not change. If $\Gamma(s_i, (1,1)) = (s_j, 2)$ then we set the vector $\Delta(\Lambda)$ similarly, except $\Delta_{m+2}(\Lambda) = 1$ (the second counter should increase its value by one). If $\Lambda = \{i, m+1\}$ or $\Lambda = \{i, m+2\}$ or $\Lambda = \{i\}$, we construct $\Delta(\Lambda)$ similarly, by applying the rule Γ to $(s_i, (1,0))$, $(s_i, (0,1))$, and $(s_i, (0,0))$, respectively. Specifically for $\Lambda = \{i^*\}$ we put $\Delta(\Lambda) = 0$ as $(s^*, 0, 0)$ corresponds to a halting configuration. For all the remaining Λ we set $\Delta(\Lambda) = 0$. It is not hard to see that with this set of transition vectors $\Delta(\Lambda)$, if $Q(t) \in \mathcal{X}_\Lambda$ and $Q(t)$ encodes the current configuration (s^t, z_1^t, z_2^t) of the counter machine, then $Q(t+1) = Q(t) + \Delta(\Lambda)$ encodes the updated configuration $(s^{t+1}, z_1^{t+1}, z_2^{t+1})$.

To complete the proof of the theorem we show that if we had an algorithm $\mathcal{C}$ for checking stability of a deterministic homogeneous walk in $\mathcal{X}_+^{m+2}$, we would have an algorithm for checking whether a counter machine halts on a given initial configuration (s^0, z_1^0, z_2^0). Given a counter machine construct a deterministic walk with a specific initial state and transition rules as described above. Check this walk for stability using algorithm $\mathcal{C}$. If the walk is unstable, we declare the counter machine nonhalting. This is accurate, because if it were halting, then the walk would end up in a "trapping" face $\mathcal{X}_\Lambda$ with $\Lambda = \{i^*\}$ and would stay in the same state forever (in particular, it would be stable). If, however, the walk is determined to be stable, then we simply follow the dynamics of our counter machine until either it halts or a certain configuration is repeated, that is, for some $t_1 < t_2$ $(s^{t_1}, z_1^{t_1}, z_2^{t_1}) = (s^{t_2}, z_1^{t_2}, z_2^{t_2})$. In fact, by stability, the content of the counters in the counter machine is bounded, and if the counter machine does not halt, then it should repeat a configuration and enter a cycle. That, of course, corresponds to a nonhalting case and we declare the counter machine nonhalting. This completes the proof of the theorem. □

As Theorem 1 part (b) states, determining whether a counter machine halts starting from every initial configuration is also undecidable. Using this result, we now show that the ergodicity of a constrained homogeneous random walk is not decidable. Recall that a random walk is ergodic if it is stable and irreducible. For the case of a deterministic walk, this simply means that there exists a unique cycle that is entered by the walk starting from an arbitrary initial point.

COROLLARY 1. *Ergodicity of a homogeneous deterministic walk is not decidable.*

PROOF. Consider an arbitrary counter machine (S, Γ) and its embedding into a deterministic walk $Q(t)$ in $\mathcal{X}_+^{|S|+2}$, as described in the proof of the previous theorem. Specifically, as in that construction, we stipulate that when the face Λ represents the halting configuration $(s_{i^*}, 0, 0)$, the corresponding transition $\Delta(\Lambda) = 0$. Then this walk has at least one cycle consisting of one point e_{i^*}, where e_{i^*} has i^*th coordinate 1 and all the other coordinates 0.

For every face $\Lambda \neq \varnothing$ that does not correspond to an encoding of some counter machine configuration, we define the corresponding transition $\Delta(\Lambda)$ as follows: $\Delta_i(\Lambda) = -1$ for $i \in \Lambda$ and $\Delta_i(\Lambda) = 0$ for $i \notin \Lambda$. In words, every nonzero component of the state $Q(t)$ in such a face is decreased by one. Finally, if $\Lambda = \varnothing$, then $\Delta_{i^*}(\Lambda) = 1$ and $\Delta_i(\Lambda) = 0$ for $i \neq i^*$. That is, if $Q(t) = 0$ then $Q(t+1)$ represents the halting configuration $(s_{i^*}, 0, 0)$. By the construction above, if $Q(0)$ does not correspond to an encoding of some configuration of a counter machine, then either for some time t, $Q(t)$ represents the halting configuration and the walk enters a single state cycle, or at some time t, $Q(t)$ does represent some configuration of the counter machine. Suppose now there exists an algorithm for checking ergodicity of a deterministic walk. We apply the algorithm to the walk constructed above. If the output of the algorithm is "ergodic," then there exists only one cycle that is eventually entered from any initial state. In our case this can only be a single-point cycle consisting of a state e_{i^*} representing the halting configuration. In particular, the halting configuration is reached for every initial configuration. If the output of the algorithm is "not ergodic," then there exists more than one cycle, or there exist an unstable trajectory. In either case there exists a starting state, that corresponds to some configuration of a counter machine, from which the state e_{i^*} is never reached. This starting state must correspond to some configuration of a counter machine. As a result, there exists a configuration from which the halting configuration is never reached. We conclude, the walk is ergodic if and only if the halting configuration is reached for any initial configuration. However, the latter property is undecidable by part (b) of Theorem 1. □

The second part of Theorem 1 is also used in Blondel et al. (2001) to prove that a global stability of a piecewise affine action is not a decidable property. We use this result here to prove a similar result that stability of a homogeneous walk for all the initial states (global stability) is also an undecidable property. Thus, global stability is not decidable.

THEOREM 3. *There does not exist an algorithm which, on an input* $(\{\Delta(\Lambda)\}_\Lambda)$, *outputs "yes" if the deterministic walk* $Q(t)$ *with transition rules* $(\{\Delta(\Lambda)\}_\Lambda)$ *is stable for all the initial states* $Q(0)$ *and outputs "no" otherwise.*

PROOF. Given a counter machine we will construct a deterministic walk that is stable if and only if the counter machine halts for all the initial states. Then the result would follow from Theorem 1, part (b). Note that we cannot use the construction in the proof of Theorem 2 directly since the stability of the constructed walk from all the initial states only tells us whether the counter machine halts or loops from every initial configuration. We could check which one is the case for any individual initial configuration but not for all the initial configurations simultaneously. For this reason we modify a given counter machine (S, Γ) into a bigger counter machine $(\overline{S}, \overline{\Gamma})$ with the following property: If (S, Γ) halts for all the initial configurations, then $(\overline{S}, \overline{\Gamma})$ halts for all the initial configurations. If $(\underline{S}, \Gamma)$ does not halt for a certain initial configuration, then there exists a configuration in $(\overline{S}, \overline{\Gamma})$ starting from which one of the counters diverges to infinity (in particular, the machine does not halt).

Suppose the original counter machine (S, Γ) has states $S = \{s_1, s_2, \ldots, s_m\}$ and two counters z_1, z_2. Suppose the halting configuration is $(s^*, 0, 0)$. Our extended counter machine $(\overline{S}, \overline{\Gamma})$ has a state space S^2 and seven counters denoted by $z_1, z_2, z_1^+, z_2^+, z_1^-, z_2^-, z^\infty$. The first component s_{i_1} of the state $(s_{i_1}, s_{i_2}) \in S^2$ corresponds to the state of the original counter machine, and the first two counters z_1, z_2 correspond to the counters of the original counter machine.

The configuration update rules are split into several groups.

Case 1. $s_{i_1} \neq s_{i_2}$ or one of $z_1^+, z_2^+, z_1^-, z_2^-$ is nonzero. Suppose Γ updates the configuration (s_{i_1}, z_1, z_2) into $(s_{i_3}, z_1 + 1, z_2)$. We update the configuration in the extended machine as follows: $\overline{\Gamma}$: $(s_{i_1}, s_{i_2}) \to (s_{i_3}, s_{i_2})$ and $\overline{\Gamma}$: $z_1 \to z_1 + 1$, $\overline{\Gamma}$: $z_2 \to z_2$. In addition, if $z_1^- = 0$, then $\overline{\Gamma}$: $z_1^+ \to z_1^+ + 1$, and if $z_1^- > 0$, then $\overline{\Gamma}$: $z_1^- \to z_1^- - 1$. All the other counters remain

the same. As we will see later, the counters z_i^+ will represent the difference between the current reading of the counter z_1 and its original reading at some initial time. Specifically, if the difference is $\delta > 0$ (<0), we will arrange for $z_1^+ = \delta, z_1^- = 0$ ($z_1^+ = 0, z_1^- = -\delta$). If Γ: $(s_{i_1}, z_1, z_2) \to (s_{i_3}, z_1 - 1, z_2)$, then $\overline{\Gamma}$: $(s_{i_1}, s_{i_2}) \to (s_{i_3}, s_{i_2})$ and $\overline{\Gamma}$: $z_1 \to z_1 - 1, \overline{\Gamma}$: $z_2 \to z_2$. In addition, if $z_1^+ > 0$, then $\overline{\Gamma}$: $z_1^+ \to z_1^+ - 1$, and if $z_1^+ = 0$, then $\overline{\Gamma}$: $z_1^- \to z_1^- + 1$. All the other counters remain the same. If the second counter is changed in the original machine or both counters stay the same, then the updates in the extended machine are similar. Specifically, when both counters stay the same, the counters z_i^+, z_i^- do not change.

Case 2. $s_{i_1} = s_{i_2}$ and $z_1^+ = z_2^+ = z_1^- = z_2^- = 0$. The configuration update rule is exactly as above, but, in addition, $\overline{\Gamma}$: $z^\infty \to z^\infty + 1$.

In addition to the rules above we put: if $s_{i_1} = s^*$ and $z_1 = z_2 = 0$ (which corresponds to the halting configuration of the original machine), then $\overline{\Gamma}$: $(s^*, s_{i_2}) \to (s^*, s^*)$ and any of the nonzero counters of the extended machine is decremented by one. Finally, $(s_{i_1}, s_{i_2}) = (s^*, s^*)$ and $z_1 = z_2 = z_1^+ = z_2^+ = z_1^- = z_2^- = z^\infty = 0$ is set to be the halting configuration at which no configuration update occurs.

The constructed machine does not fit exactly the definition of a counter machine from §3, since it has more than two counters and more than one counter can be updated simultaneously. It is shown in Blondel et al. (2001) how to reduce a counter machine with more than two counters into an equivalent one with only two counters. The second problem can be circumvented simply by adding further new states and having only one counter update at a time. Neither of these changes is required here, since we can embed a counter machine with more than two counters, which updates several counters simultaneously, into a deterministic walk exactly as in the proof of Theorem 2.

Let us now analyze the dynamics of the constructed counter machine. Note that the first component s_{i_1} and the first two counters z_1, z_2 behave exactly like the original counter machine. Specifically from the rules above, the extended counter machine halts if and only if the original counter machine halts. However, we now show that, in addition, if the original machine does not halt starting from some configuration, then starting from some configuration the extended machine does not halt and the counter z^∞ diverges to infinity. Thus, suppose the original machine does not halt starting from some initial configuration. Then either one of the counters of the original (and, therefore, also of the extended) counter machine diverges to infinity, or the counter machine enters a cycle. In the first case we are done—the embedding of this machine into a deterministic walk corresponds to an unstable trajectory. Consider now the second case. Let $(s_{\hat{k}}, \hat{z}_1, \hat{z}_2)$ denote an arbitrary point from this cycle. Consider the following initial configuration of the extended counter machine: State $(s_{\hat{k}}, s_{\hat{k}})$ and counters $z_1 = \hat{z}_1, z_2 = \hat{z}_2, z_1^+ = z_2^+ = z_1^- = z_2^- = z^\infty = 0$. Note, from the rules of the extended counter machine, that starting from this configuration, the second s_{i_2} component of the state is always equal to $s_{\hat{k}}$. Also note that the difference between the counter value z_j at the current time and at time zero is represented by one of z_j^+, z_j^-. Specifically, if the difference is $\delta_j \geq 0$, then $z_j^+ = \delta_j, z_j^- = 0$, and if $\delta_j < 0$, then $z_j^+ = 0, z_j^- = -\delta_j$. Since, by assumption, the original machine starting from $(s_{\hat{k}}, \hat{z}_1, \hat{z}_2)$ enters a cycle and revisits this configuration infinitely often, then the extended machine will infinitely often be in a configuration with state $(s_{\hat{k}}, s_{\hat{k}})$ and $z_1^+ = z_2^+ = z_1^- = z_2^- = 0$. Every time this occurs, we increment z^∞ by one (see Case 2 above). In particular, the counter z^∞ diverges to infinity. Thus, if the original counter machine is not halting for some initial configuration, then the extended counter machine is also nonhalting for some initial configuration and, in addition, one of the counters diverges to infinity.

To complete the proof, we embed the extended counter machine $(\overline{S}, \overline{\Gamma})$ into a deterministic walk exactly as in the proof of Theorem 2. For the faces Λ of the walk that do not correspond to any possible encoding of the counter machine we put $\Delta(\Lambda) = 0$. Specifically, starting from a state from such a face, the walk is stable. If the original counter machine halts for all the initial configurations, then the constructed walk is stable for all the initial

states. If there exists a configuration starting from which the original machine does not halt, then, by the argument above, there exists a state in the deterministic walk, starting from which it is unstable. If we had an algorithm for checking global stability (stability for all the initial states) of a deterministic walk, we would have an algorithm for checking "halting from all configuration" property for a counter machine, which contradicts Theorem 1, part (b). □

5. Stability of a generalized priority policy: The undecidability result. In this section we construct a queueing system and a generalized priority policy π, which mimics the behavior of a counter machine. We then argue that if we had an algorithm for checking stability of the queueing system, we could determine whether the counter machine halts or not, contradicting Theorem 1. In what follows we describe the queueing system and construct the reduction.

5.1. Description of the queueing system. Consider a counter machine (S, Γ) with $S = \{s_1, s_2, \ldots, s_m\}$. Our queueing system consists of $3m+24$ buffers and receives parts from $2m+7$ external sources. Figure 1 displays the system for the case $m = 3$. It should be understood that all the buffers belong to a single station (which we do not draw to avoid cluttering the figure). All the parts move from left to right horizontally. We have $2m$ buffers that are denoted by $A_1^0, A_1, A_2^0, A_2, \ldots, A_m^0, A_m$. For each $i = 1, 2, \ldots, m$ there is an external stream of parts coming into buffer A_i^0. After completing a service, these parts move into buffer A_i and then after service completion leave the system. The next m buffers, denoted by $B_1, \ldots, B_m$, receive parts externally. After receiving service, these parts leave the system. The subsequent six buffers are denoted by $C_1^0, C_1, C_1^1, C_2^0, C_2, C_2^1$. For each $i = 1, 2$, the parts are arriving from outside and visit buffers C_i^0, C_i, C_i^1 in that order and leave the system.

Finally, we have five types of special *monitor* parts ea, eb, ec_0, ec_1, ec_2. There will never be more than one monitor part of each type in the system at a time. The monitor part ea visits three buffers denoted by EA_1, EA_2, EA_3 respectively. The monitor part ec_0 visits buffers $EC_0^0, EC_1^0, EC_2^0, EC_3^0$. The parts ec_1, ec_2 visit buffers $EC_1^1, \ldots, EC_5^1$ and $EC_1^2, \ldots, EC_5^2$, respectively. Finally, monitor part eb visits one buffer EB. All parts arrive into the system simultaneously at times $0, I, 2I, 3I, \ldots$, where $I = 3m+26$. All the service times are equal to one unit of time.

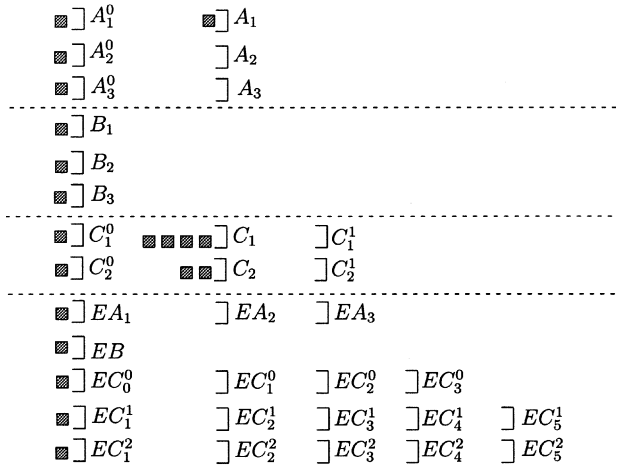

FIGURE 1. Beginning of subcycle $\mathcal{B}$.

5.2. Reduction. We first give a high-level description of our reduction. The detailed description will follow later. The content of buffers $A_1, \ldots, A_m$ and $B_1, \ldots, B_m$ are used for encoding the current state s and the updated state s' of the counter machine, respectively. Specifically, we will say that buffers $A_i, i = 1, 2, \ldots, m$ encode state $s = s_r$ at time t if, at time t, buffers A_i are empty for $i \neq r$, and buffer A_r contains exactly one part. Buffers $B_i, i = 1, 2, \ldots, m$ encode the states similarly. The buffers $A_1^0, \ldots, A_m^0$ are auxiliary and are only used for storing the parts so that they can be released into "content-full" buffers A_i whenever an update is required. The content of buffers C_1, C_2 will represent the content of the two counters of the counter machine. The remaining buffers C_i^j are auxiliary. We will ensure that only buffers C_i (representing counters) can have more than one part waiting for service at a time. An update $(s_r, z_1, z_2) \to (s_{r'}, z_1', z_2')$ of the counter machine will be represented by a cycle of service completions in our queueing system. It starts at time tI and ends at time $(t+1)I$, $t = 0, 1, 2, \ldots$. At the beginning of the cycle, buffers A_i represent the current state s of the counter machine; buffers C_1, C_2 represent the current content z_1, z_2 of the counters (that is, queue length in buffer C_i is z_i); buffers A_i^0, B_i, C_i^0 each contain exactly one part; each monitor part is in its first buffer; and all the remaining auxiliary buffers are empty. Figure 1, for example, represents the queueing system encoding configuration $(s_1, 4, 2)$ at the beginning of a cycle. If the counter machine goes through a sequence of configuration

$$(s_{r_0}, z_1^0, z_2^0) \to (s_{r_1}, z_1^1, z_2^1) \to (s_{r_2}, z_1^2, z_2^2) \to \cdots \to (s_{r_t}, z_1^t, z_2^t) \to \cdots,$$

then the queueing system will encode these configurations at times $0, I, 2I, \ldots, tI$. A complete cycle representing a generic transition $(s_r, z_1, z_2) \to (s_{r'}, z_1', z_2')$ is split into three subcycles, $\mathscr{B}$, $\mathscr{C}$, and $\mathscr{A}$. In the first subcycle, $\mathscr{B}$, the parts in buffers B_i are processed so that in the end they represent the updated state $s_{r'}$. This is achieved by processing all the parts in buffers $B_i, i \neq r'$ and not touching buffer $B_{r'}$. At this point buffers B_i "record" the updated state $s_{r'}$ for future references.

In the second subcycle, $\mathscr{C}$, the parts in buffers $C_i^0, C_i, C_i^1, i = 1, 2$, are processed so that the buffers C_i represent the updated counters z_1', z_2'. In the final subcycle, $\mathscr{A}$, the parts in buffers A_i^0, A_i are processed so that the buffers A_i also represent the updated state $s_{r'}$. This is done by "copying" the content of the buffers B_i into buffers A_i.

In the end of the cycle the content of the buffers A_i represents the updated state $s_{r'}$, and the content of the buffers C_i represents the updated counters z_1', z_2'. The memory buffers B_i are cleared from parts, and another cycle representing the next transition $(s_{r'}, z_1', z_2') \to (s_{r''}, z_1'', z_2'')$ can begin at the moment of the next simultaneous arrival of parts. The correct order $\mathscr{B}, \mathscr{C}, \mathscr{A}$ of the subcycles is ensured via the monitor parts in buffers $EA_j, EB, EC_j^0, EC_j^1, EC_j^2$. Specifically, the presence of the monitor part eb in buffers EB indicates that subcycle $\mathscr{B}$ must be executed. The absence of the part eb and the presence of parts ec_0, ec_1, ec_2 indicate subcycle $\mathscr{C}$. Finally, the absence of parts eb, ec_1, ec_2 and the presence of ea indicate the execution of the last subcycle $\mathscr{A}$.

We now provide a detailed description of the generalized priority policy π that will represent the subcycles $\mathscr{B}, \mathscr{C}, \mathscr{A}$ described above. The subcycles are illustrated on Figures 1–10 on a sample transition $(s_1, 4, 2) \to (s_3, 4, 3)$. This transition corresponds to configuration update rule $\Gamma(s_1, (1, 1)) = (s_3, 2)$. We describe the policy π by specifying the state or the set of states of our queueing system (which buffers are empty, which buffers are nonempty) and the corresponding decision (the part that the server must work on). We also analyze the effect of applying the corresponding decisions.

Let (s_r, z_1, z_2) be a configuration of the counter machine. We consider our queueing system in the following state. All buffers $A_i^0, B_i, C_i^0, EA_1, EB, EC_1^0, EC_1^1, EC_1^2$, and buffer A_r contain exactly one part. The queue length in buffer C_i is equal to z_i, $i = 1, 2$. All the remaining buffers are empty. If the counter machine has a configuration (s, z_1, z_2) at the tth

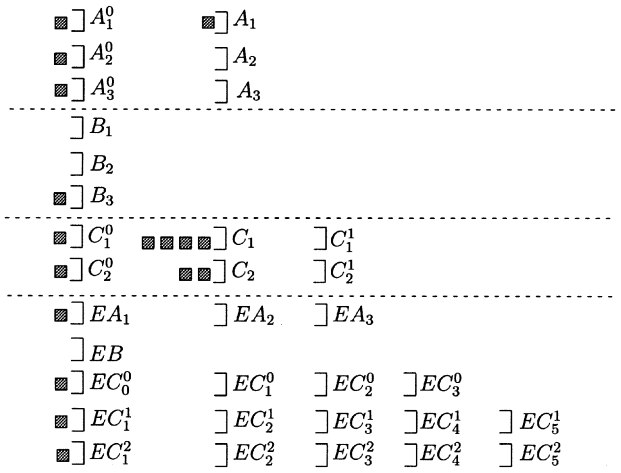

FIGURE 2. Beginning of subcycle $\mathscr{C}$. Step 1.

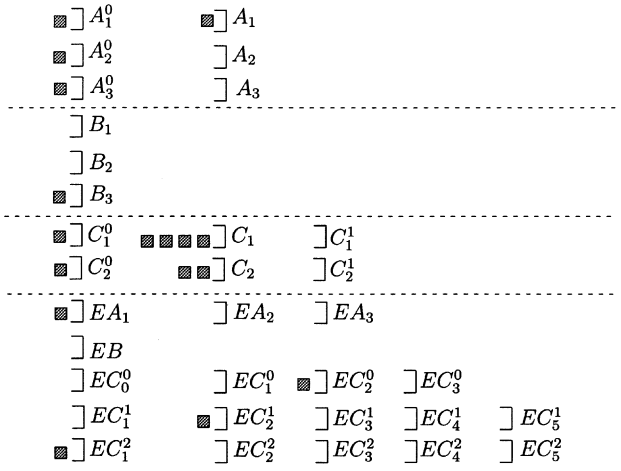

FIGURE 3. Subcycle $\mathscr{C}$. Step 2.

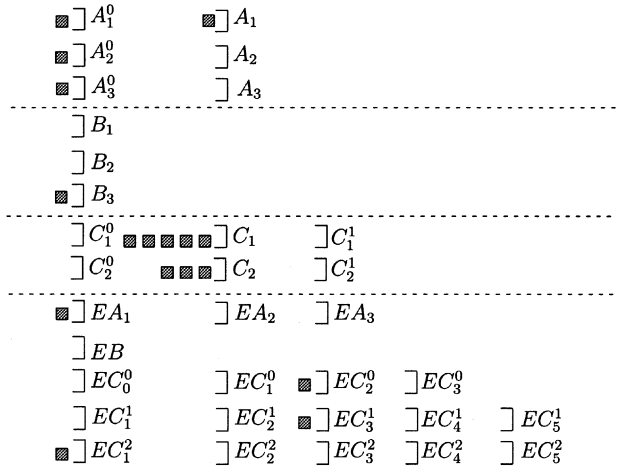

FIGURE 4. Subcycle $\mathscr{C}$. Step 3.

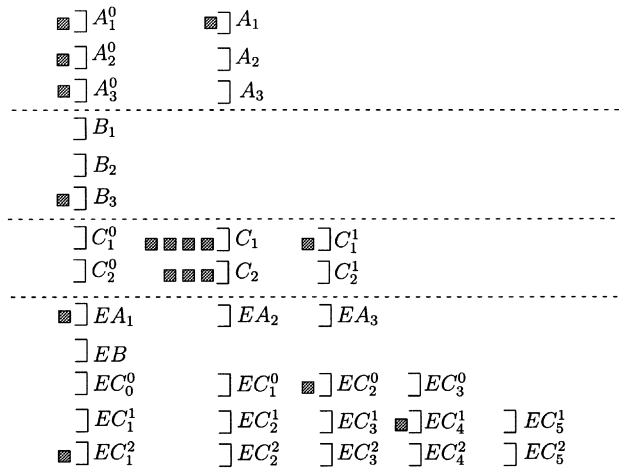

FIGURE 5. Subcycle $\mathscr{C}$. Step 4.

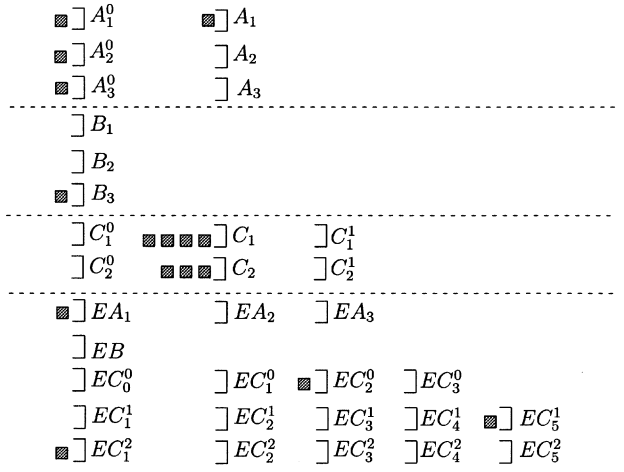

FIGURE 6. Subcycle $\mathscr{C}$. Step 5.

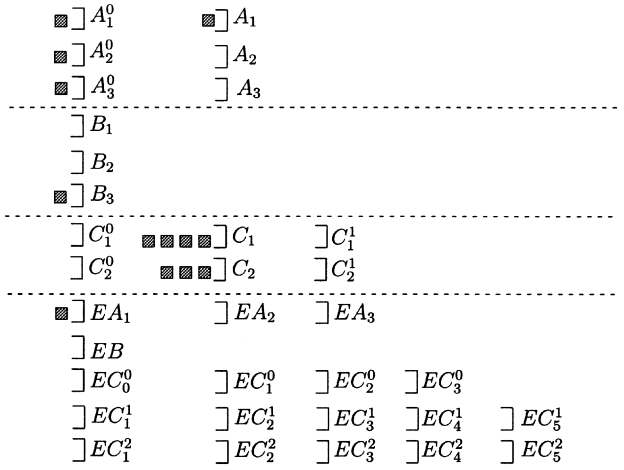

FIGURE 7. Subcycle $\mathscr{A}$. Step 1.

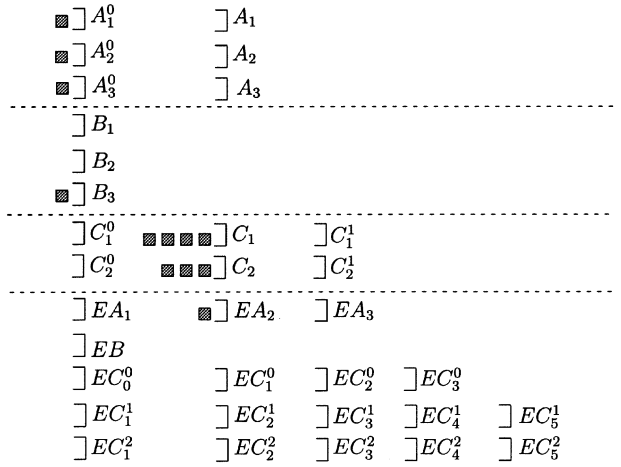

FIGURE 8. Subcycle $\mathcal{A}$. Step 2.

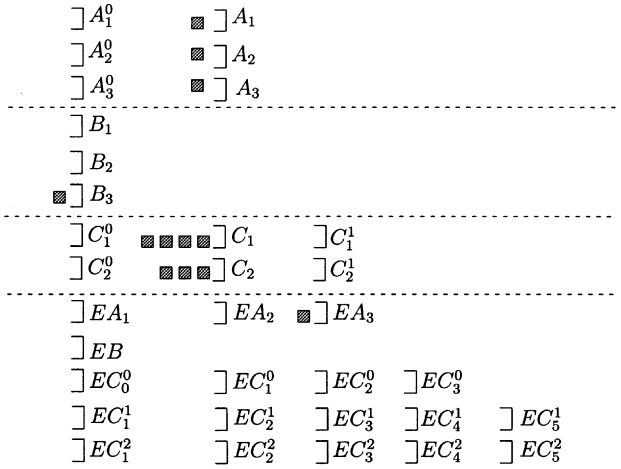

FIGURE 9. Subcycle $\mathcal{A}$. Step 3.

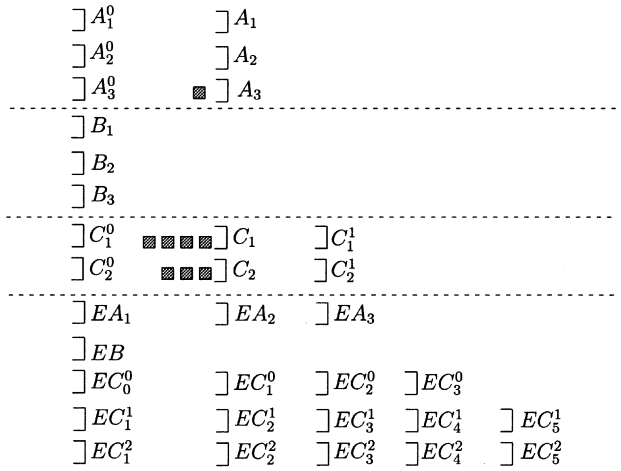

FIGURE 10. End of the full cycle.

step, our queueing system will be in the state described above at time tI. Let $(s_{r'}, z_1', z_2')$ be the configuration obtained from (s_r, z_1, z_2) using the update rule Γ of the counter machine.

Subcycle $\mathcal{B}$. The goal of this subcycle is to have buffers B_i encode the updated state $s_{r'}$. As will be seen, the set of rules corresponding to the cycle $\mathcal{B}$ is applicable whenever monitor part eb is present in the system. The states of the queueing system and corresponding decisions of the policy π are as follows:

State. Buffer EB nonempty (and contains monitor part eb).

Decision. Look at contents of the buffers $A_i, i = 1, 2, \ldots, m$ and compute the current state s of the counter machine. Specifically, $s = s_r$ if buffer A_r is nonempty and buffers A_i are empty for $i \neq r$. Look at the queue lengths Q_i at buffers $C_i, i = 1, 2$ and set $z_1 = Q_1, z_2 = Q_2$. Apply the rule Γ of the counter machine to the configuration (s_r, z_1, z_2) and obtain the updated state $s_{r'}$. Find the smallest $i \neq r'$ such that buffer B_i is nonempty. Process the part in the buffer B_i. If no such buffer exists, process the part in the buffer EB (remove monitor part eb from the system).

Analysis. All buffers $B_i, i \neq r'$ become empty. Buffer $B_{r'}$ contains one part. Thus, buffers B_i correctly encode the updated state $s_{r'}$.

Subcycle $\mathcal{C}$. The goal of this cycle is to have buffers C_i encode the updated counter contents z_1', z_2'. As will be seen, the set of rules corresponding to the cycle $\mathcal{C}$ is applicable whenever at least one of the buffers EC_i^j, $j = 0, 1, 2, i = 1, 2, \ldots, 5$ is nonempty but buffer EB is empty (cycle $\mathcal{B}$ is over).

We split the decision rules into two groups. The first corresponds to the increase by one or no change in one of the counters z_1, z_2. The second corresponds to the decrease by one in one of the counters z_1, z_2. Suppose, for example that the content z_1 increases by one and the content z_2 does not change. Then, the first group of steps is applicable. These steps first move one part from each buffer C_1^0, C_2^0 into buffers C_1, C_2. Then, exactly one part from buffer C_2 is sent to C_2^1. Finally this part is removed from C_2^1. These are Steps 1–4 below. The net result is an increase by one of the queue length in buffer C_1 and no change in buffer C_2.

Suppose now the z_1 decreases by one and the content z_2 does not change. Then, the second group of steps is applicable. We first send one part from each buffer C_1^0, C_2^0 to buffers C_1, C_2 and then to buffers C_1^1, C_2^1 and remove them from the system. Then in addition we move one part from buffer C_1 into buffer C_1^1 and remove it from the system. The net result is no change in the queue length in buffer C_2 and decrease by one of the queue length in buffer C_1. The correct order of all steps is ensured via movements of the monitor parts $ec_j, j = 0, 1, 2$. Specifically, movement of the part ec_1 corresponds to increasing one of the counters or no change in counters, and movement of ec_2 corresponds to decreasing the content of one of the counters. Part ec_0 encodes the index of the counter that needs to be updated.

The states of the queueing system and corresponding decisions of the policy π are as follows:

Step 1. State. Buffer EB empty, buffers EC_1^1, EC_1^2 nonempty (and contain monitor parts ec_1, ec_2 respectively).

Decision. Find the unique nonempty buffer A_r from the buffers $A_i, i = 1, 2, \ldots, m$. In other words, the current state of the counter machine is $s = s_r$. Construct a binary vector $b = (b_1, b_2)$, with $b_i = 1$ if the queue length in C_i is positive and $b_i = 0$ otherwise. Apply the rule Γ to the configuration (s_r, b_1, b_2) and get the corresponding counter update decision. Suppose the decision increases by one the content $z_1\,(z_2)$ and leaves $z_2\,(z_1)$ unchanged. Find the location of the monitor part type ec_0. If part ec_0 is in buffer $EC_0^0\,(EC_0^0, EC_1^0)$, then work on this part. If it is in $EC_1^0\,(EC_2^0)$, work on monitor part ec_1 in buffer EC_1^1.

Analysis. These steps bring monitor part ec_0 into buffer $EC_1^0\,(EC_2^0)$. Buffer $EC_1^0\,(EC_2^0)$ records the counter index $i = 1\,(i = 2)$ that needs to be updated. Then we move the monitor part ec_1 from buffer EC_1^1 into buffer EC_2^1. That way we indicate that the increase should occur.

Step 2. State. EB, EC_1^1 empty, EC_2^1, EC_1^2 nonempty (monitor part ec_1 is in buffer EC_2^1).

Decision. Find $i = 1, 2$ such that buffer C_i^0 is nonempty and work on a part in this buffer. If no such buffer exists, work on buffer EC_2^1 and move part ec_1 from buffer EC_2^1 to the buffer EC_3^1.

Analysis. For each i, one part is moved from the buffer C_i^0 into buffer C_i and buffers C_i^0 are now empty. Then part ec_1 is moved to the buffer EC_3^1.

Step 3. State. Buffers EB, EC_1^1, EC_2^1 empty, buffers EC_3^1, EC_1^2 nonempty (monitor part ec_1 is in buffer EC_3^1).

Decision. Locate the monitor part ec_0. If it is in EC_1^0 (EC_2^0) and C_1^1, C_2^1 are empty, work on a part in buffer C_2 (C_1). Note that because of the previous step, both buffers C_1^0, C_2^0 are empty. If ec_0 is in EC_1^0 (EC_2^0) and C_2^1 (C_1^1) is nonempty, move part ec_1 from EC_3^1 to EC_4^1.

Analysis. Part ec_0 in EC_1^0 (EC_2^0) indicates that buffer $C_1(C_2)$ needs update. Part ec_1 in buffer EC_3^1 indicates that update should be an increase. We first process a part from buffer C_2 (C_1) (which should not increase) into buffer C_2^1 (C_1^1) and then move part ec_1 from EC_3^1 to EC_4^1.

Step 4. State. Buffers $EB, EC_1^1, EC_2^1, EC_3^1$ empty, buffers EC_4^1, EC_1^2 nonempty (monitor part ec_1 is in buffer EC_4^1).

Decision. Find which buffer C_1^1 or C_2^1 contains one part and process this part away from the system. If both C_2^1 and C_2^1 are empty move part ec_1 from EC_4^1 to EC_5^1.

Analysis. We clear buffers C_1^1, C_2^1 and move ec_1 to EC_5^1.

Step 5. State. Buffers $EB, EC_1^1, EC_2^1, EC_3^1, EC_4^1$ empty.

Decision. If buffer EC_5^1 is nonempty, work on monitor parts ec_1. If it is empty work on any of ec_0, ec_2, until they leave the system.

Analysis. Monitor parts ec_1 leave the system. Then the remaining monitor parts leave the system. Buffers C_1, C_2 contain updated content of the counters. Buffers C_i^j are empty. The subcycle $\mathcal{A}$ may begin.

This completes the group of decisions corresponding to increase of the counter z_1 (z_2) by one. When the counter needs to be decreased, similar decision rules are constructed, but instead monitor part ec_2 moves along the buffers $EC_1^2, \ldots, EC_5^2$. When the counters do not change, we use rules corresponding to the increase of one of the counters, except that monitor part ec_0 is placed in EC_3^0 in Step 1, instead of EC_1^0 or EC_2^0. In that case each buffer C_1, C_2 moves one part into buffers C_1^1, C_2^1 in Step 3, and each buffer C_1^1, C_2^1 processes one part in Step 4.

Subcycle $\mathcal{A}$. The goal of this subcycle is to copy the content of buffers B_i (updated state $s_{r'}$) into buffers A_i and clear the memory buffers B_i and auxiliary buffers A_i^0.

Step 1. State. EB, EC_k^j are empty, EA_1 nonempty.

Decision. Find the unique i such that the buffer A_i is nonempty and process a part in it. If all buffers A_i are empty move the monitor part ea from buffer EA_1 to the buffer EA_2.

Step 2. State. EB, EC_k^j, EA_1 empty, EA_2 nonempty.

Decision. Find the smallest i such that buffer A_i^0 is nonempty and process a part in it. If no such buffer exists, move the monitor part ea from buffer EA_2 to the buffer EA_3.

Step 3. State. EB, EC_k^j, EA_1, EA_2 empty, EA_3 nonempty.

Decision. Find a unique r' such that buffer $B_{r'}$ is nonempty. Find a smallest $i \neq r'$ such that A_i is nonempty. Process a part in this buffer. If no such buffer exists and if some buffer $B_{r'}$ is nonempty, work on the part in this buffer and remove it from the system. If all B_i are empty, remove the part ea from the system.

Analysis. The net result will be exactly one part in buffer $A_{r'}$, no parts in buffers A_i^0, B_i, and no parts in buffers $A_i, i \neq r'$. Thus, buffers A_i correctly encode the updated state $s_{r'}$ of the counter machine, and buffers B_i are cleared. Also, the last remaining monitor part ea is removed from the system.

Step 4. State. All the monitor buffers EA_k, EB, EC_k^j empty.

Decision. Idle.

This ends the full cycle.

The idling that starts at the end of the subcycle $\mathscr{A}$ continues until new parts arrive into the system at time $(t+1)I$, given that the cycle started at time tI. The beginning of a new cycle is indicated by an appearance of a monitor part eb in buffer EB. It is not hard to compute that the maximal total length of the entire cycle is smaller than $3m+26=I$ (the total number of buffers in the system plus 2, to account for a decrease of the queue length in one of the buffer C_1, C_2). Note that the scheduler does not need to "know" the correct order $\mathscr{B}, \mathscr{C}, \mathscr{A}$ of the subcycles. The correct order is ensured by presence/absence of the monitor parts ea, ec_j, eb. Note also that we have not specified the scheduling decisions for all $\{0,1\}^{3m+24}$ binary states that the system may potentially be in. We do not have to do that since we are guaranteed that none of these additional states are ever reached. For completeness, assume that the decision corresponding to these states is to "work on any available part."

In conclusion, if the counter machine goes through a sequence of configurations $(s_{r_0}, z_1^0, z_2^0) \to (s_{r_1}, z_1^1, z_2^1) \to (s_{r_2}, z_1^2, z_2^2) \to \ldots, (s_{r_t}, z_1^t, z_2^t) \to \ldots$, then these configurations are encoded via buffers A_i, C_i by our queueing systems at times $0, I, 2I, \ldots, tI, \ldots$, respectively. We have finished describing the reduction from a counter machine to a queueing system.

5.3. Main result. We now state and prove the main result of the section.

THEOREM 4. *There does not exist an algorithm which, on an input* $(\mathscr{Q}, \pi, Q(0))$, *outputs "yes" if the queueing system* $\mathscr{Q}$, *operating under generalized priority policy* π *and starting from initial vector of queue lengths* $Q(0)$, *is stable and outputs "no" otherwise. Thus, stability of a generalized priority policy is not decidable.*

PROOF. We prove the result by contradiction. Suppose such an algorithm $\mathscr{E}$ exists. Consider a counter machine with initial state (s_{r_0}, z_1^0, z_2^0) halting state $(s^*, 0, 0)$ and update rule Γ. We modify the counter machine slightly as follows: Γ: $(s^*, 0, 0) \to (s^*, 0, 0)$. In other words, if the counter machine reaches the halting configuration it keeps returning to it. We now consider a queueing system constructed in the previous subsection with initial state encoding configuration (s_{r_0}, z_1^0, z_2^0) and pass our queueing system through the stability oracle $\mathscr{E}$. If the oracle determines "unstable," we declare our counter machine nonhalting. This is accurate since, if the counter machine reaches the halting configuration $(s^*, 0, 0)$, it keeps returning to it. Then our queueing system always encodes $(s^*, 0, 0)$ at times tI for all sufficiently large t. Since the total number of parts in the queueing system at times $\tau \in [tI, (t+1)I]$ is not bigger than at time tI, then the queueing system is stable, which is a contradiction. If, on the other hand, the oracle determines "stable," then we simply compute the successive configurations of the counter machine until it either (1) enters a halting state or (2) repeats a nonhalting configuration. Note that one of these two possibilities must occur since by stability there is an upper bound on the maximal value in the counters of the counter machine. Thus, provided with stability checking algorithm $\mathscr{E}$ we would be able to check whether the counter machine halts—this contradicts Theorem 1. □

We now establish an analogue of Theorem 3 for a queueing system.

THEOREM 5. *There does not exist an algorithm which, on an input* $(\mathscr{Q}, \pi)$, *outputs "yes" if the queueing system* $\mathscr{Q}$, *operating under generalized priority policy* π, *is globally stable and outputs "no" otherwise.*

PROOF. Given a counter machine (S, Γ) we extend this counter machine to a newer one exactly as in the proof of Theorem 3. The extended counter machine halts for all the initial configurations if and only if the original counter machine halts for all the initial configurations. Moreover, if there exists a configuration starting from which the old counter

machine does not halt, then there exists a configuration in the new machine, starting from which one of the counters diverges to infinity. We embed the extended counter machine into a single station queueing system exactly as described in §§5.1 and 5.2. Note that if we start this queueing system from a state that does not correspond to an encoding of a counter machine then, since a work-conserving scheduling rule is used, the system stays stable or enters a state that does correspond to an encoding of some configuration. We conclude that the queueing system is globally stable if and only if the original counter machine halts for all the initial configurations. As a result no global stability checking algorithm for queueing systems can exist. □

6. Extensions. We discuss now some extensions of the results of the previous sections. Specifically, we discuss communication-type queueing networks and fluid models. We also describe a certain reformulation of the stability property that makes it decidable for most of the models considered in this paper.

6.1. Communication-type queueing network. A communication-type queueing network is described as an undirected graph (V, E), which represents the communication network topology. V and E are the set of nodes and edges, respectively. A set of simple paths (representing communication sessions) $\mathcal{P}$ is fixed. For each path $P \in \mathcal{P}$, jobs (communication packets) arrive into the network externally and cross all the edges of the path P. It takes one time unit to cross any given edge for any packet and only one packet can cross a given edge at a time. The remaining packets form a queue. Note that the paths are assumed to be simple and, as a result, no packet can cross the same edge twice. This is in contrast to the one-station queueing system model. We assume that packets that have to go through the path P have a deterministic interarrival time denoted by $1/\lambda_P$. The first arrival occurs at time $1/\lambda_P$ for all P. The rule by which a packet is chosen from a queue to cross an edge is called a scheduling policy. As in §2.2, we introduce a generalized priority policy. A scheduling policy π is defined to be a *generalized priority* policy if it operates as follows. Let n denote the number of edge-path pairs $(e, P) \in E \times \mathcal{P}$. For each $e \in E$ a map π_e: $\{0, 1\}^n \to \mathcal{P} \cup \{0\}$ is given. At each time $t = 0, 1, 2, \ldots$, the scheduler looks at the system and computes the binary vector $b = (b_{(e,P)})_{e \in E,\, P \in \mathcal{P}}$, where $b_{(e,P)} = 1$ if there is at least one packet following path P that is waiting to cross e and $b_{(e,P)} = 0$ otherwise. For each edge e the corresponding value $\pi_e(b)$ is computed. If $\pi_e(b) = P$, then a packet following path P is chosen for crossing the edge. If $\pi_e(b) = 0$, then the edge e idles and no packets are processed. As before, the policy is not necessarily work conserving. Note, again, that the generalized priority scheduling policy is defined in finitely many terms and is completely state dependent.

Let λ denote the vector (λ_P) of arrival rates. A queueing network $(V, E, \lambda, \mathcal{P}, \pi)$ with initial state (vector of queue lengths) $Q(0) = (Q_{(e,P)}(0))_{e,P}$ is defined to be stable if there exists $C > 0$ (which may depend on the initial state) such that the total number of packets does not exceed C for all times t. The definition of global stability is similar to the one for a single-station model and for a constrained random walk. The load condition necessary for stability is formulated for the network model as follows:

$$\rho_e \equiv \sum_{P: e \in P} \lambda_P \leq 1, \tag{2}$$

for each edge $e \in E$. We now show that the properties "$(V, E, \lambda, \pi, \mathcal{P}, Q(0))$ is stable" and "$(V, E, \lambda, \pi, \mathcal{P})$ is stable" are undecidable.

THEOREM 6. *There does not exist an algorithm that, on an input* $(V, E, \lambda, \mathcal{P}, \pi, Q(0))$, *outputs "yes" if a queueing network* $(V, E, \lambda, \mathcal{P})$ *with initial state* $Q(0)$, *operating under a generalized priority policy* π, *is stable and outputs "no" otherwise. Thus, stability of*

a generalized priority policy in a communication-type queueing network is not decidable. Similarly, global stability of a generalized priority policy in a communication-type queueing network is not decidable.

PROOF. Consider a reduction from a counter machine to a single-station queueing system constructed in §5. We construct a communication-type queueing network that is equivalent to our queueing system. For each $i = 1, 2, \ldots, m$, we consider three vertices a_i^0, a_i, a_i^1 connected by edges (a_i^0, a_i) and (a_i, a_i^1). These edges represent the buffers A_i^0, A_i of the original queueing system. In particular, for every i and for every time instance $tI, t = 0, 1, 2, \ldots$, a single packet arrives that has to go through the path a_i^0, a_i, a_i^1. Similarly, for every $i = 1, 2, \ldots, m$, there is pair of nodes b_i^0, b_i connected by an edge representing buffer B_i. Every time instance tI one packet comes that has to cross this edge only. We construct for each i four nodes c_i^0, c_i, c_i^1, c_i^2 connected by edges $(c_i^0, c_i), (c_i, c_i^1), (c_i^1, c_i^2)$ to represent buffers C_i^0, C_i, C_i^1. For each $i = 1, 2$, one packet arrives at times tI, $t = 0, 1, 2, \ldots$, that has to go through these edges. We construct nodes and edges to represent special buffers EA_j, EB, EC_i^j similarly. A policy π constructed for the queueing system is adopted to our network in an obvious way. For example, if the decision in the queueing system were to work on a part in buffer C_i, then the corresponding decision is to process a packet in the node c_i that needs to cross the edge (c_i, c_i^1). Clearly, the queueing network is stable if and only if the original queueing system is stable. Thus, by Theorem 4, stability of queueing networks under generalized priority policies for a given initial state is not decidable. □

6.2. Fluid models. Fluid models have proven to be an extremely useful technique for analyzing stability of constrained random walk and queueing systems (see Dai 1995, Dai and Vande Vate 2000, Malyshev 1993, Rybko and Stolyar 1992, Stolyar 1995). The approach is based on scaling state and time of the underlying process by a large constant and analyzing the behavior of the limiting process. Specifically, if $Q(t)$ is a homogeneous random walk in $\mathcal{Z}_+^d$, the process $\overline{Q}(t) = Q(nt)/n$ is considered for large n with initial state $\overline{Q}(0) = \lfloor nq \rfloor$ for some vector $q \in \Re_+^d$. Malyshev (1993) used fluid limits to construct deterministic dynamical systems on convex bounded regions and obtain stability conditions for constrained random walks in $\mathcal{Z}_+^d, d \leq 4$. Dai (1995) and Stolyar (1995) proved that in queueing systems if $\overline{Q}(t)$ becomes a zero vector after some finite time interval, then the queueing system is stable. While these results establish an important connection between discrete systems and fluid models, understanding the dynamics of the fluid limit process $\overline{Q}(t)$ itself is an open problem. In special cases (fluid models of feedforward fluid networks (see Down and Meyn 1997), networks with two stations (see Bertsimas et al. 1996, Dai and Vande Vate 2000) this can be done efficiently, but no constructive way of analyzing the behavior of fluid limits is available to this day. We now prove that computing the trajectory of a fluid limit is not possible.

THEOREM 7. *Given a homogeneous deterministic walk $Q(t)$ in $\mathcal{Z}_+^d$, with initial state $Q(0) = \lfloor nq \rfloor$, let $\overline{Q}(t) \in \Re_+^d$ denote a set of limit points of $Q(nt)/n$ as $n \to \infty$, for each fixed time t. There does not exist an algorithm which, on an input $(\{\Delta(\Lambda)\}_\Lambda, t)$, outputs "yes" if $\overline{Q}(t) = \{0\}$ and outputs "no" otherwise. Thus, computing fluid limit is not a decidable problem. Likewise, computing a fluid limit for a single-station queueing system operating under a generalized priority policy is not a decidable problem.*

PROOF. Consider a counter machine with states $s_1, s_2, \ldots, s_m$ and with initial and halting configurations $(s_{i_0}, z_1^0, z_2^0), (s_{i^*}, 0, 0)$, respectively. We can assume, without loss of generality, that $z_1^0 = z_2^0 = 0$. (This can be easily achieved by extending the state space of the machine; we omit the details.) We introduce several simple modifications to the machine. We add two counters z_3, z_4 and embed the counter machine into a deterministic walk as we did in the proof of Theorem 2. Specifically, the state of the walk at time t is denoted

by $(Q_1(t), \dots, Q_{m+4}(t))$. The initial state of the walk is set to be $Q_{m+3}(0) = n$, $Q_i(0) = 0$, $i \neq m+3$. In addition to the rules of the counter machine, for $\Lambda = \{m+3\}$ we set $\Delta_{m+3}(\Lambda) = -1$, $\Delta_i(\Lambda) = 0$, $i \neq m+3$. Thus, the walk starts in state with $\|Q(0)\| = n$ and after n steps ends in a state $Q(n) = 0$. Then, we set $\Delta_{i_0}(\varnothing) = 1$, $\Delta_i(\varnothing) = 0$, $i \neq i_0$. Thus, $Q(n+1)$ (to be exact $(Q_1(n+1), \dots, Q_{m+2}(n+1))$ corresponds to the encoding of the initial configuration. Finally, we set that for any Λ that corresponds to an encoding of a counter machine configuration, $\Delta_{m+4}(\Lambda) = 1$ if the encoding does not correspond to the halting configuration; $\Delta_{m+4}(\Lambda) = -1$ if the encoding corresponds to the halting configuration and $m+4 \in \Lambda$; and $\Delta_{m+4}(\Lambda) = 0$ if the encoding corresponds to the halting configuration and $m+4 \notin \Lambda$. Thus, starting from $t = n+1$ the walk simulates the dynamics of the counter machine, and the coordinate $m+4$ is incremented by one at every step. If the machine does not halt, then this goes on forever, and $\|Q(t)\| \geq Q_{m+4}(t) = t - (n+1)$. Specifically, $\|Q(2n)\|/n \geq (2n-(n+1))/n$ and $\liminf_n \|Q(2n)\|/n \geq 1$. On the other hand, if the machine halts in, say, t_0 steps (note that t_0 is independent from n), the component $m+4$ starts decreasing until it reaches zero, while the counter machine keeps reentering the halting configuration. Then $(Q_1(n+1+2t_0), \dots, Q_{m+2}(n+1+2t_0))$ encodes the halting configuration and $Q_{m+3}(n+1+2t_0) = Q_{m+4}(n+1+2t_0) = 0$, and the walk stays in this state forever. Then $\|Q(2n)\| = 1$ and $\lim_n \|Q(2n)\|/n = 0$. We see that for $t = 2$, $\overline{Q}(t) \geq 1$ if the machine does not halt, and $\overline{Q}(t) = 0$ if the machine halts. This completes the proof. The proof for a single-station queueing system is similar. $\square$

6.3. A decidable reformulation of the stability property. Since the models we consider in this paper are intractable in the very strongest possible sense, a natural question is: What are the necessary minimal modifications of the problem that would make them tractable? We first provide some reformulations that restrict the class of models considered but still do not avoid the "undecidability curse." Then we propose a different formulation of stability that makes, almost trivially, a decidable and sometimes an easily checkable property.

Given a constrained random walk $Q(t)$ in $\mathcal{Z}_+^d$ with transition probabilities $p(\Lambda, \Delta)$, suppose we further restrict the transition probabilities to be strictly less than one. Note that this leaves out the deterministic walk we constructed directly from a counter machine. Unfortunately, this restriction does not make the problem easier, as the following observation shows: Consider the deterministic walk constructed, based on a counter machine, as was done in §4. For any face Λ and direction vector Δ such that $p(\Lambda, \Delta) = 1$ we put $\tilde{p}(\Lambda, \Delta) = \frac{1}{2}$ and $\tilde{p}(\Lambda, 0) = \frac{1}{2}$. In other words, the walk moves in the direction Δ or stays in the same state, both with probability $\frac{1}{2}$. This new walk with transition probabilities $\tilde{p}(\Lambda, \Delta)$ has exactly the same trajectories as the original deterministic walk but follows these trajectories with expected rate twice as small as the original one. The new random walk is stable if and only if the original walk is stable. Thus, stability of a random walk with the restriction described above is also undecidable.

Another possible restriction is nonuniqueness of the trajectories. Suppose that for our constrained random walk $Q(t)$ there exist states $x, x_1, x_2 \in \mathcal{Z}_+^d$ such that both transitions from x to x_1 and from x to x_2 have positive probability. This is clearly violated by the random walk constructed above, since only one state x' different from x can be reached in a single transition. Unfortunately, even this restriction still leaves the problem undecidable, as the following construction shows. Given a counter machine with m states we consider the following random walk in $\mathcal{Z}_+^{2m+2}$. The state s_i is encoded by unit vectors e_{2i} and e_{2i+1}, with $i = 1, 2, \dots, m$. When the state s_i is changed to the state s_j in the counter machine, the walk moves along the vectors Δ_1 or Δ_2 with probability $\frac{1}{2}$, where Δ_1 brings the state $Q(t)$ into state $Q(t+1)$ with $Q_{2j}(t+1) = 1$, $Q_i(t+1) = 0$, $i \neq 2j$, $i \leq 2m$ and Δ_2 brings $Q(t)$ into state $Q(t+1)$ with $Q_{2j+1}(t+1) = 1$, $Q_i(t+1) = 0$, $i \neq 2j+1$, $i \leq 2m$. In other words,

we have pairs of states of the walk that encode the same configuration of the machine. We omit the details.

A final restriction we consider is to assume that, for each state, the transition probabilities to all of the neighboring states are positive. We are not able to prove that the problem remains undecidable but suspect that it is.

CONJECTURE 1. *Consider a homogeneous random walk $Q(t)$ with transition probabilities $\{p(\Lambda, \Delta)\}_{\Lambda, \Delta}$, such that if $\Delta_i \geq 0$ for all $i \notin \Lambda$, then $p(\Lambda, \Delta) > 0$. In other words, the transitions to all of the legal neighboring states occur with positive probability. The (global) stability property of this random walk is not decidable.*

REMARK. Note that the random walk with this property is irreducible. As a result if it is stable for some initial state, it is also stable for all the initial states. Thus, there is no distinction between the stability and global stability.

We now consider a certain reformulation of the stability property.

DEFINITION 1. Given a positive value $\theta > 0$, a deterministic walk $Q(t)$ is defined to be θ-stable if

$$\sum_{i=1}^{d} Q(t) \leq \theta, \tag{3}$$

for all times t. Similarly, θ-stability is defined for deterministic queueing systems and queueing networks.

Trivially, the walk is not θ-stable if $Q(0) > \theta$. The definition above is not quite suitable for constrained random walks or stochastic queueing systems since the event $\sum_{i=1}^{d} Q(t) > \theta$ occurs infinitely often with probability one, as long as the underlying Markov chain can reach any state with some positive probability.

THEOREM 8. *Given a constrained deterministic walk $Q(t)$ in $\mathcal{Z}_+^d$, with initial state $Q(0)$ and given $\theta > 0$, the θ-stability is a decidable property. It can be checked in no more than $\lceil \theta \rceil^d$ computation steps.*

PROOF. The algorithm for checking θ-stability is very simple. We first check if $\sum_{i=1}^{d} Q_i(0) \leq \theta$. If not, the walk is not θ-stable. Otherwise, compute the trajectory $Q(t)$ for $t = 1, 2, \ldots, \lceil \theta \rceil^d$. If at any time t, (3) is violated, then the walk is not θ-stable. Otherwise, since the number of states satisfying (3) is not bigger than $\lceil \theta \rceil^d$, a certain state will be repeated. Then the walk will enter an infinite loop, contained entirely within the set, defined by (3). As such the walk is θ-stable. □

Note that we can also check whether the walk is θ-stable for all the initial states $Q(0)$ satisfying (3), since there is only a finite number of them. If d is understood to be a constant, then the stability checking algorithm runs in polynomial in θ time. The extension of θ-stability and of Theorem 8 to queueing systems is immediate. The θ-stability notion could also be of more practical use, since a certain upper bound on the maximal size of the buffers in a queueing system is typically dictated by reality.

7. Conclusions. We have introduced in this paper a class of generalized priority scheduling policies and proved that stability of these policies is not algorithmically decidable. To the best of our knowledge, this is the first result showing that stability of scheduling policies in queueing systems can be undecidable. We also proved that stability of a homogeneous random walk in $\mathcal{Z}_+^d$ is not decidable for general d. This settles in a somewhat unexpected way an open problem of characterizing stable random walks in $\mathcal{Z}_+^d$ for $d \geq 5$.

A very important extension of this work would be establishing a similar result for less "exotic," more common policies like FIFO or priority policies. To achieve this, one is forced to look at queueing networks, since these policies, like any other work-conserving policy

are stable in a single-station queueing system. We conjecture that the stability of FIFO, LIFO, and priority policies is not algorithmically decidable.

Another direction for investigation is understanding the stability of constrained deterministic or random walks for a fixed dimension d. Note, that, assuming that the dimension d is a constant, the number of different deterministic walks (the number of systems of transition rules $\{\Delta(\Lambda)\}_\Lambda$) is also a constant, upper bounded by $3^d 2^d$. In fact, for each Λ and each coordinate i, the corresponding $\Delta_i(\Lambda)$ can potentially take three values $-1, 0, 1$. Also there are 2^d different faces Λ. It is a standard corollary of the decidability theory that a system with only constantly many examples is decidable; see Sipser (1997). However, if the initial state of the walk $Q(0)$ is a part of the input, then it is not clear whether stability is decidable even for, say, $d = 5$. Another interesting challenge would be establishing undecidability for constrained random walks, with strictly positive transitions to neighboring states. As we mentioned above, stability for a specific initial state and global stability are equivalent for this model due to irreducibility.

Acknowledgments. The author wishes to thank Dimitris Bertsimas, John Tsitsiklis, Vadim Malyshev, Michael Shub, Bruce Kitchens, and Jay Sethuraman for many helpful discussions on the decidability issues and corrections of the earlier draft.

References

Andrews, M., B. Awerbuch, A. Fernandez, Jon Kleinberg, T. Leighton, Z. Liu. 2001. Universal stability results for greedy contention-resolution protocols. *J. Assoc. Comput. Mach.* **48**(1) 39–69.

Bertismas, D., D. Gamarnik, J. Tsitsiklis. 1996. Stability conditions for multiclass fluid queueing networks. *IEEE Trans. Automatic Control* **41** 1618–1631.

Blondel, V. D., O. Bournez, P. Koiran, C. H. Papadimitriou, J. N. Tsitsiklis. 2001. Deciding stability and mortality of piecewise affine systems. *Theoret. Comput. Sci.* **255**(1–2) 687–696.

Borodin, A., J. Kleinberg, P. Raghavan, M. Sudan, D. Williamson. 2001. Adversarial queueing theory. *J. Assoc. Comput. Mach.* **48**(1) 13–38.

Bramson, M. 1994. Instability of FIFO queueing networks. *Ann. Appl. Probab.* **2** 414–431.

Dai, J. G. 1995. On the positive Harris recurrence for multiclass queueing networks: A unified approach via fluid models. *Ann. Appl. Probab.* **5** 49–77.

———, J. H. Vande Vate. 2000. On the stability of two-station fluid networks. *Oper. Res.* **48** 721–744.

———, J. J. Hasenbein. 1999. Stability of a three-station fluid network. *Queueing Systems* **33** 293–325.

———, G. Weiss. 1996. Stability and instability of fluid models for certain re-entrant lines. *Math. Oper. Res.* **21** 115–134.

Down, D. D., S. P. Meyn. 1997. Piecewise linear test functions for stability and instability of queueing networks. *Queueing Systems* **27** 205–226.

Fayolle, G. 1989. On random walks arising in queueing systems: Ergodicity and transience via quadratic forms as Lyapunov functions—Part II. *Queueing Systems* **5** 167–183.

Gamarnik, D. 1999. Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks. *Proc. 31st ACM Sympsos. Theory of Comput.*, Atlanta, GA. *SIAM J. Comput.* Forthcoming.

———. 2000. Using fluid models to prove stability of adversarial queueing networks. *IEEE Trans. Automatic Control* **4** 741–747.

Goel, A. 1999. Stability of networks and protocols in the adversarial queueing model for packet routing. *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms.* Networks, Baltimore, MD.

Henzinger, T., P. Kopke, A. Puri, P. Varaiya. 1995. What's decidable about hybrid automata. *Proc. 27th ACM Sympos. on Theory of Comput.*, Las Vegas, NV.

Hooper, P. 1966. The undecidability of the Turing machine immortality problem. *J. Symbolic Logic* **2** 219–234.

Hopcroft, J., J. Ullman. 1969. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, MA.

Ignatyuk, I. A., V. A. Malyshev. 1993. Classification of random walks in $\mathbb{Z}_+^d$. *Selecta Math.* **12** 129–194.

Koiran, P., M. Cosnard, M. Garzon. 1994. Computability properties of low-dimensional dynamical systems. *Theoret. Comput. Sci.* **132** 113–128.

Lu, S. H., P. R. Kumar. 1991. Distributed scheduling based on due dates and buffer priorities. *IEEE Trans. Automatic Control* **36** 1406–1416.

Malyshev, V. A. 1972. Classification of two-dimensional positive random walks and almost linear semimartingales. *Dokl. Akad. Nauk SSSR* **202** 526–528.

———. 1993. Networks and dynamical systems. *Adv. Appl. Probab.* **25** 140–175.

Menshikov, M. V. 1974. Ergodicity and transience conditions for random walks in the positive octant of space. *Soviet. Math. Dokl.* **15** 1118–1121.

Rybko, A., A. Stolyar. 1992. On the ergodicity of stochastic processes describing open queueing networks. *Problemi Peredachi Informatsii* **28** 3–26.

Seidman, T. I. 1994. First come first serve can be unstable. *IEEE Trans. Automatic Control* **39** 2166–2170.

Sipser, M. 1997. Introduction to the Theory of Computability. PWS Publishing Company, Boston, MA.

Stolyar, A. 1995. On the stability of multiclass queueing networks: A relaxed sufficient condition via limiting fluid processes. *Markov Processes and Related Fields* 491–512.

D. Gamarnik: IBM T. J. Watson Research Center, Yorktown Heights, New York 10598; e-mail: gamarnik@watson.ibm.com