Yale University
Department of Computer Science

P.O. Box 208205
New Haven, CT 06520–8285

Urn Automata

Dana Angluin[1] James Aspnes[1,2] Zoë Diamadi[1,3]
Michael J. Fischer[1,4] René Peralta[1,4]

YALEU/DCS/TR-1280
November 5, 2003

[1]Yale University Department of Computer Science.

[2]Supported in part by NSF grants CCR–9820888, CCR–0098078, and CCR–0305258.
[3]Supported in part by ONR grant N00014–01–1–0795.
[4]Supported in part by NSF grant CCR–0081823.

Urn Automata

Dana Angluin* James Aspnes*† Zoë Diamadi*‡ Michael J. Fischer*§
René Peralta*§

Abstract

Urn automata are a new class of automata consisting of an input tape, a finite-state controller, and an urn containing tokens with a finite set of colors, where the finite-state controller can sample and replace tokens in the urn but cannot control which tokens it receives. We consider the computational power of urn automata, showing that an urn automaton with $O(f(n))$ tokens can, with high probability, simulate a probabilistic Turing machine using $O(\log f(n))$ space and vice versa, as well as giving several technical results showing that the computational power of urn automata is not affected by variations in parameters such as the size of the state space, the number of tokens sampled per step, and so forth. Motivated by problems in distributed computing, we consider a special class of urn automata called **pairing automata** that model systems of finite-state machines that interact through random pairwise encounters. We show that pairing automata recognize precisely the symmetric languages recognized by urn automata.

1 Introduction

Most work in distributed algorithms assumes that the agents in a system are computationally powerful, capable of storing non-trivial amounts of data and carrying out complex calculations. But in systems consisting of massive amounts of cheap, bulk-produced hardware, or of small mobile agents that are tightly constrained by the systems they run on, the amount of storage available at each agent may be severely limited—possibly as little as a constant number of bits per agent. Such limitations are not crippling if the system designer has fine control over the interactions between agents, as even finite-state agents can be regimented into cellular automata with computational power equivalent to Turing machines. But if the system designer cannot control these interactions—if which agents interact with each other is unpredictable or even random—then it is not clear what, if anything, the system can compute.

The present work seeks to answer this question. We begin by defining (in Section 3) a new model of storage, called an **urn**, that reflects the lack of structure in the collective memory of diffuse distributed computing systems such as sensor networks. An urn is like a tape or pushdown stack without the ordering among its storage cells. All that is left once the cell structure is discarded is a set of storage cells, called **tokens**, each with a symbol written in it, called its **color**. Urns support two operations: `sample` removes and returns a token selected with equal probability from

*Yale University Department of Computer Science.
†Supported in part by NSF grants CCR–9820888, CCR–0098078, and CCR–0305258.
‡Supported in part by ONR grant N00014–01–1–0795.
§Supported in part by NSF grant CCR–0081823.

among the tokens in the urn; $\mathsf{insert}(t)$ inserts token t into the urn. The essentially random behavior of the sample operation differentiates urns from most previously-studied storage devices.

Urns can be used to model several different computational situations. In this paper, we consider two new (and related) automaton models built from urns: urn automata and pairing automata. An **urn automaton** (defined formally in Section 3) consists of an input source, a finite-state controller, and an urn containing some number of tokens, each with a color from a fixed set. A transition of the urn automaton depends on the current state, a possible input symbol, and a sequence of some number of tokens sampled at random from the urn. The transition causes the controller to enter a new state, insert some colored tokens into the urn, and possibly perform an action on the input source or an action to halt with an indication of acceptance or rejection. The urn thus plays the role of the stack in a push-down automaton or the work tape in a Turing machine.

We consider three variants of urn automata that differ only on how they receive their inputs: on a two-way read-only input tape, on a one-way read-only input tape, or with the input represented by tokens initially placed in the urn. Regardless of input convention, we focus on urn automata that have constant **width-**k, meaning that every transition consumes exactly k tokens from the urn, are **deterministic**, meaning that the transition chosen is completely determined by the current state, the current input symbol, and the k tokens sampled, and **conservative**, meaning that every transition puts the same number of tokens back into the urn as it consumes through its sampling.

Because an urn lacks structure, it might seem intuitively to be weaker than a push-down store or a tape. Surprisingly, an urn can be considerably more powerful. We consider the computational power of urn automata in Section 4. These results split into two classes depending on whether we permit the urn automaton to make errors. If we insist that the computation be error-free and the input supplied in the initial urn, then no non-trivial language can be recognized. With a one-way input tape then error-free urn automata accept only regular languages; hence the urn adds no power to the finite state controller. With a two-way input tape, however, a error-free urn automata of size $O(n)$ can accept any language in **LOGSPACE**.If errors are permitted with small probability, then urn automata become much more powerful. With two-way input tape, a conservative width-1 deterministic urn automaton can recognize, with high probability, precisely those languages recognized with bounded two-sided error by a probabilistic Turing machine with space proportional to the logarithm of the number of tokens in the urn.

One of the primary motivations for studying urn automata is to model systems of interacting finite-state machines which we call **conjugating automata**. Imagine a society of mobile finite-state agents wandering about randomly. Whenever two such agents encounter each other, they each update their internal state as a function of their old state and the state of the other machine. They then proceed on their way. Such machines are described in [4].

Various possibilities come to mind for supplying inputs to conjugating automata. One possibility, natural in a distributed setting, is to supply each finite-state machine with its own input stream. Another possibility is to assume a global input tape that can be accessed by any machine. A third possibility is to ignore inputs altogether and investigate the behavior only as a function of the initial state vector of the system. This is the conventional setting for studying such problems as Byzantine agreement.

A system of conjugating automata with global input tape is easily modeled by a width-2 urn automaton that has only one state. We call such a machine a **pairing automaton**. Each finite-state machine in the conjugating system corresponds to a token in the urn with color corresponding to its state. An encounter between two conjugating automata is modeled by a transition that samples two tokens and replaces them with a new pair of tokens depending on the colors of the sampled pair. While there is a direct correspondence between computations of pairing automata and conjugating automata, we find pairing automata are a more tractable formalism to use in our studies.

The lack of global memory in pairing automata appears at first sight to be a major restriction on their power. Nevertheless, a width-1 conservative urn automaton can be simulated by a pairing automaton that elects some token leader and gives it the powers and responsibilities previously held by the now-deposed finite-state controller. We focus therefore in Section 5 on the problem of electing a leader using a pairing automaton under various conditions. In particular, electing a leader is not possible in general in the no-input-tape or one-way input case. However, with a two-way input tape, there is a leader election algorithm for a width-2 pairing automaton with polynomially small error bound. If we restrict the empty input from consideration, then leader election becomes possible in all three input models.

This paper only begins to explore this fascinating new model. In Section 6, we mention several variations on the model that are worth studying, together with sketches of some preliminary results we have about some of them.

2 Related work

Perhaps one way to approach urn automata is through the Petri net literature.[1] Our primary model corresponds to a form of randomized Petri net with the rule that when many transitions are simultaneously enabled, the one to fire is chosen with probability proportional to the number of distinct sequences of tokens it can consume from its input places. Randomized Petri nets were introduced by Volzer in [11] using a transition rule, however, that does not depend on the number of tokens in each input place; As noted in [6], most of the work on Petri nets focuses on properties such as reachability, liveness, and deadlock. An interesting open problem is whether techniques from the study of Petri nets can be applied to urn automata and whether the automata-theoretic approach favored in this paper can advance our understanding of the computational capabilities of Petri nets.

Milner's bigraphical reactive systems [9] address the issues of modeling locality and connectivity of agents by two distinct graph structures. In this work the primary focus is upon the expressiveness of the models, whereas we consider issues of computational power and resource usage.

Brand and Zafiropulo [3] define a model of communicating processes consisting of a collection of finite state machines that can communicate via pre-defined FIFO message queues, and focus on general properties of protocols defined in the model, such as the possibility of deadlock or loss of synchronization.

The Chemical Abstract Machine of Berry and Boudol [1] is an abstract machine designed to model a situation in which components move about a system and communicate when they come into contact, based on a metaphor of molecules in a solution governed by reaction rules. A concept of enforced locality using membranes to confine subsolutions allows the machines to implement classical process calculi or concurrent generalizations of the lambda calculus. No assumption is made about the mixing rule.

[1] See [5] for a survey of Petri nets.

3 Definitions

The class of **urn automata** is similar to many of the classes of automata defined in classical automata theory, with the difference that the deterministic external data structures used to augment the finite-state controller have been replaced by an urn, a container that imposes no structure on its contents and the behavior of which is probabilistic.

The **urn** contains tokens, each painted a color chosen from a finite set colors T. The base model does not specify the initial contents of the urn, which are treated as part of the input; this issue is discussed in more detail in Section 3.1. We can represent the contents of the urn by an element of T^*, which is a finite sequence of token colors; we will interpret this sequence (as well as other sequences of tokens that appear later) as a **multiset** of tokens by ignoring the order. We will use $x \sqsubseteq y$ to denote the case that x is a sub-multiset of y when both x and y are interpreted as multisets; this just means that each color c appears no more often in x than in y.

Managing the urn is a **finite-state controller**, with a state space Q containing a distinguished start state q_0. The finite-state controller also manages a finite **input tape**, each cell of which is labeled with a symbol from a finite alphabet Σ. The leftmost and rightmost positions on the tape are always labeled by special terminal symbols $\$_L$ and $\$_R$, which are always elements of Σ. We assume that the input head, which can move in either direction on the tape unless otherwise specified, initially starts on the cell just to the right of the $\$_L$ symbol.

A **transition** of an urn automaton removes zero or more tokens from the urn, reads the symbol under the head on the input tape, and then updates the state of the finite-state controller, while adding zero or more tokens back into the urn and executing an operation that either moves the input head one cell to the left (L), leaves the input head where it is (-), moves the input head once cell to the right (R), or causes the automaton to halt by either accepting (ACCEPT) or rejecting (REJECT). Transitions of the urn automaton are controlled by a **transition relation**, which specifies for each transition the previous state, a multiset of tokens to remove from the urn, the input symbol read, the resulting state, the operation to apply, and a multiset of new tokens to add to the urn.

We require that the urn automaton never moves the input tape left past the $\$_L$ symbol in the leftmost cell or right past the $\$_R$ symbol in the rightmost cell; this is an easily-enforced restriction on the transition relation.

Formally, an urn automaton is defined as follows.

Definition 1 *An urn automaton is a tuple $(Q, q_0, \Sigma, T, \Delta)$, where*

1. *Q is the finite set of states for the controller;*
2. *$q_0 \in Q$ is the initial state;*
3. *$\Sigma \supseteq \{\$_L, \$_R\}$ is the finite input alphabet;*
4. *T is the finite token alphabet, that is, the set of available token colors;*
5. *Δ is the finite transition relation, which is a subset of $Q \times \Sigma \times T^* \times Q \times T^* \times \{$L, -, R, ACCEPT, REJECT$\}$.*

A **state** of an urn automaton is a 3-tuple (q, x, i), where q is the state of the finite-state controller, x is a multiset of tokens from the token set T, and i is the position of the input tape head, with the leftmost position assigned index 0. In an **initial state**, $q = q_0$ and $i = 1$; and the initial contents of the urn are taken to be part of the input.

A **transition** $(q, x, i) \to (q', x', i')$ can occur if there exists a transition $(q, \sigma, t, q', t', \text{OP})$ such that:

1. The symbol in cell i of the input tape is σ;

2. The tokens in t are a submultiset of x;

3. The new urn state x' equals $x - t + t'$, where addition and subtraction of sequences of tokens from and to multisets of tokens are interpreted in the obvious way; and

4. Depending on the type of OP:

   (a) If OP is L, then $i' = i - 1$;

   (b) If OP is R, then $i' = i + 1$;

   (c) If OP is -, ACCEPT, or REJECT, then $i' = i$.

The transitive closure of $\to$ is written $\xrightarrow{*}$; we have $(q, x, i) \xrightarrow{*} (q', x', i')$ just in case there is a sequence of transitions that take (q, x, i) to (q', x', i'). A **computation** of an urn automaton with initial urn contents x_0 consists of a sequence transitions ending with the first transition that executes ACCEPT or REJECT. A computation is accepting/rejecting if this last operation is ACCEPT/REJECT.

The above model is quite general, and for the purposes of this paper we will make several additional assumptions, all of which hold unless otherwise specified:

- The automaton has **width** k for some integer k, which means that every transition $(q, t, \sigma, q', t', \text{OP})$ in Δ has $|t| = k$.

- The automaton is **conservative**, which means (following a similar usage in the Petri net literature) that $|t'| = |t|$ for each transition, or, equivalently, that the number of the tokens in the urn is fixed.

- The controller is **deterministic**, which means (for a width-k urn automaton) that for each q in Q, σ in Σ, each t in T^k there is exactly one transition of the form $(q, \sigma, t, q', t', \text{OP})$ in Δ.[2]

- The urn uses **uniform sampling**, which means that the probability of drawing a particular sequence of tokens t from the urn is obtained by assuming that each token is drawn uniformly at random without replacement.

The assumption of a deterministic finite-state controller and uniform sampling in the urn allows us to define a probability distribution on transitions; given the current state q and input symbol σ, the probability that the unique transition with previous state q, input symbol σ, and input tokens t occurs is just the probability of drawing the sequence of tokens t from the urn in order without replacement. Similarly, we can define the probability of acceptance/rejection given a particular input and initial urn contents as the probability that the computation eventually reaches an ACCEPT/REJECT operation. We will primarily be interested in urn automata that halt with probability 1, i.e., those for which the probability of acceptance and rejection sum to 1, but we can imagine urn automata that do not have this property.

[2] For variable-width urn automata, the more general definition is that for each q in Q and σ in Σ, the set $S_{q,\sigma} = \{t : (q, \sigma, t, \cdot, \cdot, \cdot) \in \Delta\}$ is maximally prefix-free, and for each q in Q, σ in Σ and t in $S_{q,\sigma}$, there is exactly one transition of the form $(q, t, \sigma, q', t', \text{OP})$ in Δ.

3.1 Urn automata and languages

We would like to compare the computation power of urn automata to that of such traditional devices of classical automata theory as finite-state machines, Turing machines, etc. To do so, we must define what it means for an urn automaton to accept a language L. Though we have been careful to define urn automata in a way that is similar to classical automata, some questions must yet be answered. These are:

How is the input word provided? There are two straightforward possibilities: it is written either on the cells of the input tape or the tokens in the initial urn. The first case splits further depending on what the finite-state controller can do with its input tape. If R operations are forbidden, we have a **one-way input tape**. If they are permitted, we have a **two-way input tape**, which turns out to be a surprisingly powerful tool for urn automata. If the input word is distributed across the urn, one symbol per token, and both L and R operations are forbidden, then we say that we have **no input tape**.

The two input tape models come closer to traditional automaton models, and allow an urn automaton to recognize languages that are not symmetric; but having no input tape may more natural in a distributed context, where it might be hard to justify a global input tape equally accessible to all the agents in the system. We will consider all three models in this paper.

What is the initial state of the urn? Because a conservative urn automaton cannot generate new tokens on its own, it must start with some initial population of tokens if the urn is to be useful. How many tokens do we start with, and what colors do they have? For the purposes of this paper, we will assume that for each urn automaton there is a function $f(n)$ of the input size n that determines the number of tokens in the urn (*i.e.*, the **size** of the urn); we will also assume that all $f(n)$ of these tokens, with the exception of the n tokens representing the input word when we have urn input, are initially a default blank color ν. (Some other possibilities are discussed in Section 6.)

Having chosen an urn automaton with either tape input or urn input, and fixed the function $f(n)$ that determines the size of the urn, we have a complete specification of what the urn automaton does when presented with an input word w. We can thus define:

Definition 2 *A language $L \subseteq \Sigma^*$ is **accepted** by an urn automaton M with probability p if, for each $w \in L$, the probability that M accepts w is at least p, and for each $w \notin L$, the probability that M rejects w is at least p.*

4 Computational power of urn automata

In this section, we characterize the computational power of urn automata. We first consider the case where the urn automaton must produce the correct output with probability 1; here the power of an urn automaton depends strongly on the input model. We then consider urn automata that are allowed to err with small probability, and show that such automata can simulate (and be simulated by) Turing machines with a space bound proportional to the logarithm of the size of the urn.

4.1 Error-free urn automata

An urn automaton M is **error-free** if, for every input $w \in \Sigma^*$, M either accepts w with probability 1 or rejects w with probability 1. What languages are accepted by error-free urn automata?

The answer to this question depends very strongly on the input method. With the input on a one-way input tape, the urn adds little power to the finite-state controller: the languages accepted with probability 1 by an urn automaton with a one-way input tape are precisely the regular languages. With the input on a two-way input tape, an urn automaton can accept any language in **LOGSPACE** without errors. If the input appears in the initial state of the urn, then an urn automaton accepts no nontrivial languages without errors.

Details are given in the full paper. The key idea behind the negative results is that if x is a submultiset of y, then computations starting with urn contents x "lift" to computations with urn contents y, simply by ignoring any extraneous tokens. This is enough to show that an automaton with a one-way input tape can be simulated by a finite-state machine whose states represent both the state of the urn automaton controller and some $\sqsubseteq$-minimal element of the set of reachable urn states; and the set of such $\sqsubseteq$-minimal elements is finite by Higman's Lemma [7]. With no input tape we argue that if $x + \nu^i \sqsubseteq y + \nu^j$, then an error-free machine accepts x if and only if it accepts y, and then show (taking some care with the blank tokens) that all inputs are connected by chains of submultiset relations.

For the positive result, we show how to exploit the error-free counter given by the input head position to simulate error-free counters from error-prone probabilistic counters made out of urn tokens, which is enough to get **LOGSPACE** by simulating the work tape with counters.

The combined statement of these results is:

Theorem 3 *The class of languages accepted without error by urn automata:*

1. *Contains only $\emptyset$ and Σ^* with no input tape;*
2. *Is precisely the regular languages with a one-way input tape; and*
3. *Contains all of **LOGSPACE** with a two-way input tape.*

4.2 Urn automata and Turing machines

If errors are permitted with small probability, deterministic automata become much more powerful. For each function $f(n)$ of the input size n, let $\mathbf{URN}_0(f(n))$, $\mathbf{URN}_1(f(n))$, and $\mathbf{URN}_2(f(n))$ be the class of languages accepted with probability $1 - O(n^c)$ for any fixed $c > 0$ by a width-1 urn automaton of size $O(f(n))$ with no input tape, a one-way input tape, and a two-way input tape, respectively. Then

Theorem 4

1. *$\mathbf{URN}_0(f(n))$ consists of the symmetric languages in $\mathbf{URN}_1(f(n))$.*
2. *$\mathbf{URN}_1(f(n)) = \mathbf{URN}_2(f(n))$ when $f(n) = \Omega(|\Sigma|^n)$.*
3. *$\mathbf{URN}_2(f(n)) \supseteq \mathbf{SPACE}(\log f(n))$.*
4. *For any language L in $\mathbf{URN}_0(f(n))$, $\mathbf{URN}_1(f(n))$, or $\mathbf{URN}_2(f(n))$, there is a randomized Turing machine that uses space $O(\log f(n))$ and accepts L with polynomially small two-sided error.*

Sketch of proof: Working backwards through the list, it is easy to see that a randomized Turing machine can simulate an urn automaton using $O(\log f(n))$ space, by storing the state of the urn as a vector of counters, one for each color.

To show that $\mathbf{URN}_2(f(n)) \supseteq \mathbf{SPACE}(f(n))$, we use Minsky's simulation of a Turing machine by a counter machine [10, Section14.1], where each half of the work tape is Gödel-encoded by a polynomial $c_k x^k c_{k-1} x^{k-1} + \cdots + c_0$, reading the symbol under the work tape head involves finding c_0 by taking a mod operation, and moving the head multiplies or divides the polynomial by x. In our simulation, each counter is represented in unary by the number of tokens of a particular color in the urn. The only complication is that an urn automaton cannot test for emptiness, because it has no way to tell if it has found all the tokens of a particular color. We solve this problem probabilistically by inserting a single dummy token in the urn and testing for zero by waiting to see the dummy k times in a row without seeing any tokens of the other color, a test which errs with probability $O(1/N^{k-1})$ where $N = f(n)$ is the size of the urn. Unfortunately this probability of error is too high to use for every zero test in Minsky's original simulation, so we have to look at the probability of missing a token at any time during a multiply, divide, or mod operation, which we show is $O(\log N/N^{k-1})$. The result follow by an appropriate choice of k.

To show $\mathbf{URN}_1(f(n)) \supseteq \mathbf{URN}_2(f(n))$, we use the same Gödel-numbering technique to read the entire input tape into a counter of size $O(|\Sigma|^n)$, and then simulate the $\mathbf{URN}_2(f(n))$ machine using this counter in place of the two-way input tape.

Finally, to show $\mathbf{URN}_0(f(n))$ contains all the symmetric languages in $\mathbf{URN}_1(f(n))$, we simulate the $\mathbf{URN}_1(f(n))$ input tape by finding and marking as used an input-symbol token whenever the $\mathbf{URN}_1(f(n))$-automaton moves to a new input cell. ∎

5 Computational power of pairing automata

Our original goal in defining urn automata was to model populations of finite state machines with unpredictable interaction. With this goal in mind, the tokens in the urn have a natural role representing the states of the finite-state machines, but the finite-state controller is an unfortunate asymmetry in the model whose existence appears to be justified only by the need to attach an input tape somewhere to allow comparison between urn automata and the language classes of classical automata theory. In this section we consider what happens if we reduce the finite-state controller's importance, restoring the original ideal of an interacting population of equals, with no single machine chosen at birth to rule over the others. We do so by limiting the controller to the minimum state space $\{q_0\}$.

We concentrate on width-2 one-state urn automata, the smallest width that permits interaction between tokens, and call these **pairing automata** for short. Our goal is to show that pairing automata can simulate width-1 urn automata without the state restriction, by filling the leadership void created by the abdication of the finite-state controller with a unique leader token that simulates the controller in its interactions with other tokens. But because we cannot expect the input to provide a unique leader, the question becomes whether we can elect one early in the execution of the automaton.

Let us begin by observing that, unless we assume constant urn size, it is impossible to elect a leader without exploiting the input in some way, where by electing a leader we mean that the automaton eventually produces a token with color in some class $C \subseteq T$, and from that time on always has exactly one such token. We will then show that electing a leader in this sense is generally not possible for an one-state automaton with no input tape. that it is trivial with a two-way tape if we are allowed to exploit the $\$_L$ token to simulate an extra state, and that it is possible under any input tape assumption without such trickery if we exclude the empty input.

Lemma 5 *Let M be a width-2 one-state urn automaton whose initial urn size $N = f(n)$ grows without bound. Suppose that the initial tokens in M's urn are all blank, and that no transition of M executes* L *or* R*. Then the probability that M elects a unique leader converges to zero in the limit.*

Sketch of proof: The proof is based on examining the development of an **interaction graph**, with vertices representing tokens and edges representing pairs of tokens that have interacted by being sampled by the finite-state controller in the same step. We argue that by time $N^{1-\epsilon}$, for any $\epsilon > 0$, that no pair of token has interacted twice with probability 1 in the limit; this means that before this time the interaction graph captures the full structure of the computation except for the order and direction of interactions.

We then consider the smallest tree-structured sequences of interactions that produce elements of the leader set C, and show using standard results from the theory of random graphs [8, 2] that unboundedly many such copies of each tree appear as components by time $N^{1-1/k+\delta}$, where k is the this of the tree and δ is a small constant. Since each such tree generates a leader with probability $2^{-k}/k! = \Theta(1)$, we get unboundedly many leaders in the limit. ∎

It follows that we cannot elect a unique leader in the no-input-tape model unless we exclude the empty input:

Corollary 6 *Let M be a width-2 one-state urn automaton with no input tape whose initial urn size $N = f(n)$ grows without bound. Then if M elects a leader with nonzero probability on an empty input, the probability that M elects a unique leader converges to zero in the limit.*

Proof: Apply the proof of Lemma 5 to executions in which no non-blank tokens are sampled during the first $N^{1-1/k+\delta}$ transitions; observe that the existence of a minimal leader-electing tree using only non-blank tokens follows from the empty-input case. ∎

A more sophisticated version of this argument, given in the full paper, shows that a similar excess of leadership arises with a one-way input tape on inputs of the form a^n for any single symbol a.

In contrast, a one-state automaton with a two-way input tape does not suffer much from its lack of extra states:

Theorem 7 *For any k, there is a leader election algorithm for a width-2 one-state urn automaton with a two-way input tape that succeeds with probability $1 - O(N^{-k})$.*

Sketch of proof: The trick is to generate the leader on the first transition, which consumes two unmarked blank tokens and executes L to move onto the $\$_L$ symbol. The leader can then mark all the blank tokens (waiting to see some unique dummy token it generates $k+1$ times in a row to be sure it has found them all) before moving off $\$_L$. ∎

If we do exclude the empty input, then leader election becomes possible even without exploiting $\$_L$ in all three input models, provided we have enough blank tokens. The intuition is that if the non-blank tokens make up a polynomial fraction of the blank tokens, then each non-blank token can declare itself a candidate, emit a single dummy token for termination testing, and then declare itself leader when it has seen k dummy tokens in row without encountering (and killing) a rival candidate. With no input tape, the non-blank tokens are just the initial tokens representing the input. With a one-way or two-way input tape, the finite-state controller walks across the input generating non-blank tokens, reducing to the no-input case when $\$_R$ is reached.

Theorem 8 *For any $k > 0$ and $\epsilon > 0$, there exists an urn automaton M that elects a unique leader with probability $1 - O(n^{-k})$ for all inputs except the empty input, provided the urn size is $\Omega(n^{1+\epsilon})$.*

Note that reducing to the no-tape case destroys all information about the order of the input symbols in the one-way tape case. It is an interesting (and difficult) open problem whether it is possible to preserve the ordering perfectly.

6 Conclusion and open problems

We have defined urn automata, a new model of randomized computation based on urns, and have obtained a partial characterization of the power of urn automata under several natural assumptions. But many questions remain open. Some in particular are suggested by relaxing the assumptions made throughout this paper. In particular, we know little about what happens with:

Other sampling rules We have assumed throughout the present work that tokens are drawn from the urn by uniform sampling. For some applications it may make sense to consider other sampling rules. These might include **weighted sampling**, where the probability of drawing a token from the urn is proportional to its weight (which may vary from token to token), or the still more general **fair sampling**, where (following typical notions of fairness used in the theory of distributed computing) there is no probability distribution on tokens, but the finite-state controller is guaranteed to see any sequence of tokens that can be formed from the urn if it waits long enough. Our suspicion is that with some minimal restrictions on the weights, weighted sampling yields the same power as uniform sampling, while fair sampling is strictly weaker; but determining precisely the relative power afforded by different sampling rules requires further study.

Other initial urn contents The urn automaton model does not specify the initial contents of the urn, treating it as part of the input. We have considered two possibilities for the initial contents of the urn: $u(n)$ tokens, all blank; or $n + u(n)$ tokens, of which n carry the symbols of the input word and the remaining $u(n)$ are all blank. Some other intriguing possibilities are a **random urn**, where each of $u(n)$ tokens is initially labeled with a color chosen uniformly at random from T; or an **adversarial urn**, where the colors of the tokens are chosen by an adversary. These assumptions could be used to characterize the self-stabilization properties of urn automata.

Nonconservative urn automata In this paper, we have concentrated on conservative urn automata, as these most closely match our motivating examples involving interacting finite-state machines. But we can ask what happens if we permit an urn automaton or pairing automaton to be **nonconservative**. The interesting case is when interactions between k tokens generate more than k tokens, and the size of the urn increases. Curiously, both the one-way case of Theorem 3 (limiting one-way error-free urn automata to accepting regular languages) and Lemma 5 (excluding certain leader election algorithms) continue to apply in this case, as the bad outcome in the proof of each theorem occurs before a significant number of new tokens can be added to the urn. An analogue to Theorem 4 applies as well, but now an urn automaton, by generating arbitrarily many tokens, can represent a Turing machine with no space limitation. Still open is the question of whether a nonconservative urn automaton can exploit its ability to adjust the size of the urn to improve time efficiency by dynamically removing unneeded blank tokens without significantly increasing the error rate.

Nondeterministic urn automata Suppose that we relax the requirement that the finite-state controller be deterministic, and say that an urn automaton accepts an input w if there is any computation path starting with $u(n)$ blank tokens in the urn that ends with an ACCEPT operation. Though there are some complications in how we approach the interaction between the nondeterminism in the finite-state controller and the probabilistic behavior of the urn, we can quickly see that such nondeterministic automata are strictly more powerful than deterministic urn automata in the error-free case, as a nondeterministic urn automaton that emits an a token when it reads an a and consumes one when it reads a b can recognize the non-regular language $\{a^i b^j | a \geq b\}$. But characterizing the full power of nondeterministic urn automata, even in the error-free case, remains open.

Nonuniform urn automata Suppose that the number of states or tokens is not constant, but is allowed to grow slowly as a function of n. Now what can we compute? It is not hard to see that $O(f(n))$ token colors are enough to simulate a Turing machine with $f(n)$ space with a width-2 urn automaton, by the simple expedient of assigning one token, labeled with cell contents and index, to represent each cell of the Turing machine tape. But are there more subtle ways of augmenting urn automaton power that require only small growth in the number of tokens?

References

[1] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[2] B. Bollobás. *Random Graphs.* Cambridge University Press, second edition, 2001.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, Apr. 1983.

[4] Z. Diamadi and M. J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1–2):72–82, Mar. 2001. Also appears as Yale Technical Report TR–1207, January 2001, available at URL `ftp://ftp.cs.yale.edu/pub/TR/tr1207.ps`.

[5] J. Esparza. Decidability and complexity of petri net problems-an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic models.*, pages 374–428. Springer Verlag, 1998. Published as LNCS 1491.

[6] J. Esparza and M. Nielsen. Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994.

[7] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(2):326–366, 1952.

[8] S. Janson, T. Łuczak, and A. Ruciński. *Random Graphs.* John Wiley & Sons, 2000.

[9] R. Milner. Bigraphical reactive systems: basic theory. Technical report, University of Cambridge, 2001. UCAM-CL-TR-523.

[10] M. L. Minsky. *Computation: Finite and infinite machines.* Prentice-Hall series in automatic computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[11] H. Volzer. Randomized non-sequential processes. In *Proceedings of CONCUR 2001-Concurrency Theory*, pages 184–201, Aug. 2001.