Self-Stabilizing Symmetry Breaking in Constant-Space

(EXTENDED ABSTRACT)

Alain Mayer* Yoram Ofek† Rafail Ostrovsky‡ Moti Yung§

Abstract

We investigate the problem of self-stabilizing round-robin token management scheme on an anonymous bidirectional ring of identical processors, where each processor is an asynchronous probabilistic (coin-flipping) finite state machine which sends and receives messages. We show that the solution to this problem is equivalent to symmetry breaking (i.e., leader election). Requiring only constant-size messages and message-passing model has practical implications: our solution can be implemented in high-speed networks using a universal fast hardware switches (i.e., finite state machines) of size *independent* of the size of the network.

Our automata-based message-passing model has inherent deadlock possibility (i.e., when all processors are waiting for a message) which we assume is detected by an external time-out mechanism. Provided that there is no deadlock to begin with, we show how starting from an arbitrary configuration, the system never enters a deadlock state and further stabilizes in polynomial time. We note that Dijkstra showed that the last problem does not have a deterministic solution (even when the identical processors possess an arbitrary power): starting from a ring with a multitude of tokens, any deterministic system will either not stabilize or will enter a deadlock state.

1 Introduction

We consider an arbitrary ring of anonymous identical processors, where each processor is a probabilistic finite-state machine whose size is fixed and independent of the size of the ring. We adopt asynchronous message-passing communication, where each processor can change state and send messages only upon receiving messages from one of its two adjacent neighbors.

* Computer Science Department, Brown University, Providence, RI, 02912. The research of this author was partly supported by ONR grant N00014-91-J-1613. Part of this work was performed while the author was at the IBM T.J. Watson Research Center. Author's e-mail address: ajm@cs.brown.edu

† IBM T.J. Watson Research Center, Yorktown Heights, NY 10598. Author's e-mail address: ofek@watson.ibm.com

‡ MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Supported by IBM Graduate Fellowship. Part of this work was done at IBM T.J. Watson Research Center. Author's e-mail address: raf@theory.lcs.mit.edu

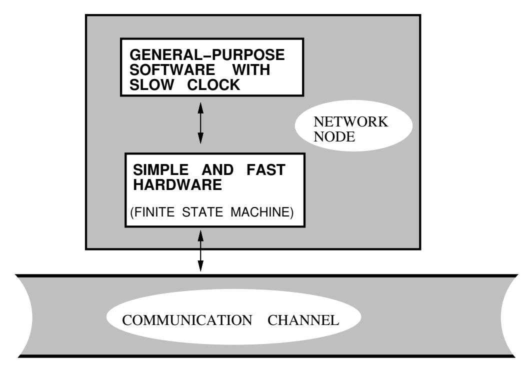
§ IBM T.J. Watson Research Center, Yorktown Heights, NY 10598. Author's e-mail address: moti@watson.ibm.com

1.1 Hardware-Based Protocol Design: an emerging technology

Our motivation for considering the above model comes from practical high-speed (fiber optic media) communication networks, where a constant-size messages can go through many high-speed hardware switches (i.e. finite-state machines) and through many communication links during a single software clock-tick.

In many such systems, the way fair access to the network is regulated is by passing a single *token* (i.e. a constant-bit control signal) in a round-robin fashion on a physical or embedded ring. When a network node needs a permission for a certain action, it waits for a token, holds it for a single unit of time and then passes it on to ensure fairness in case other processors are waiting.

It must be stressed that the efficiency of the above hardware-based approach heavily relies on the fact that the token is a control signal comprising of only a constant number of bits and that switching hardware is a finite state machine which can decide even before the token arrives if there is a need to just let it pass through or to hold it — depending only on its local state and the requirements of the network node. For example, let us consider the case when only a single processor on a ring competes for a resource access control. The fact that the processor must give up a token to ensure fairness (i.e. to check if there is some other processor waiting) *does not* incur a slowdown proportional to the size of the ring. That is, when the processor gives up a token it does not wait for the number of software clock-ticks proportional to the size of the ring: the constant-bit control message makes a full round without substantial delay by going through high-speed digital hardware switches at each node and not waiting for software clock-ticks at any of them. Moreover, the constant-bit control signal essentially does not decrease the bandwidth of the communication links, making the above scheme very much useful in practice.



Hardware–Based Protocol Design

1.2 Towards Hardware-Based Fault-Tolerant Token Management

An important concern in practice is to make the hardware-based round-robin token-management scheme fault-tolerant where the kind of fault-tolerance needed is an ability to recover from an arbitrary transient fault which puts the system in any (or even maliciously chosen) state. In fact, round-robin token-management in the shared memory model was considered in 1974 by Dijkstra in the seminal paper which introduced the notion *self-stabilization* [Dij74]. Informally, the goal of self-stabilization is for the system to reach "normal" operation, starting from an arbitrary initial state. Of course, while making the scheme fault-tolerant, we should *not* sacrifice efficiency during "normal" operation. That is, the scheme should still be *efficient* in both the speed and space requirements. In particular, the token, if not needed at a node, must be able to go through the switching hardware (i.e. through finite state machine) fast. That is, *the hardware should not rely on a software clock or other software mechanisms* and we should deal with faults while maintaining high-speed message-passing communication, especially of control information.

The inherent problem in any message-driven system is communication deadlock (i.e. all messages are lost), see for example [DIM91]. Also, any shared memory system of (even probabilistic) finite state automata for token-management can potentially deadlock in a state without tokens. Israeli and Jalfon [IJ90a] show that deadlock-freeness implies that $\Omega(log n)$ bits are needed to represent a token (where n is the size of the ring) for any deadlock-free solution.

In practice, in normal operation the high-speed switch can work while the software monitors it; which implies a two layer solution. An efficient solution should minimize the use of the monitoring software layer and let the fast switch do all the work. The slow software layer of a node has a clock and large memory and thus a node can introduce a new token into a system after a timeout which corresponds to a maximal round-trip delay of a token (i.e., when it detects that no token passed through a hardware switch the above period of time). From the system's speed point of view, it is important to let the slow software intervene only when a slow and rare event like a round-trip delay occurs. Note however, that nodes will time-out in an uncoordinated fashion, and consequently there might be many tokens in the system after these time-outs. We may try to detect and correct it at the slow software layer, however this means that this layer gets involved in the operation quite often, and also in non-faulty scenarios, which is an unacceptable solution.

Thus, our goal is to design an efficient self-stabilizing token management system which reduces an arbitrary plurality of tokens (i.e. one or more) to a single token traveling in a round-robin fashion in a completely asynchronous message passing ring of identical finite state machines. It must be stressed that in the above formulation we disallow all the so-called "solutions" which under any circumstances can bring the system from a non-deadlock state to a deadlock state. That is, we insist that if there are one or more tokens to begin with in an arbitrary initial configuration, we never eliminate all of them, and always end-up with exactly one token traveling in a cycle (we call this property *non-deadlocking*).

1.3 What constitutes a practical high-speed solution?

Dijkstra showed that round-robin token-management is impossible on a ring of unknown size if processors are uniform and deterministic.

However, the problem "whether a system of *probabilistic* automata can have round-robin token-management scheme" remained open. Israeli and Jalfon observed that if each token essentially uses symmetric random walk, where upon meetings, tokens conduct elimination tournament (i.e. when two tokens meet, one dies) then eventually only one token will remain [IJ90a]. Notice, however, that their solution gives up the round-robin property — it uses random-walk on the ring which enables token-management, but not round-robin. This relaxation has heavy implication on the actual use of the token after stabilization, since the token continues its drunkard walk at all times — thus its use is mainly for mutual exclusion (to solve contention), but not for regulation and fair-coordination of the network communication medium, which is the usual task of token systems. Indeed, Lamport and Lynch [LL90] mention that: "there is one important property that is harder to achieve in coordination problems than in contention – namely self-stabilization".

The random walk solution can be extended to achieve round-robin property in the following fashion: all tokens travel only in one direction (say clockwise, since self-stabilizing ring-orientation in constant space is possible [IJ90b]); at each step, each token flips a coin and either stays for a single clock-tick or advances. When two tokens meet they play elimination tournament. Again, eventually (and we pay in stabilization time) only one token will remain. Notice, however, that this solution is also totally unacceptable, as we need a steady state in which tokens can propagates in hardware without waiting for a clock or other delay (we call such a solution *fair*).

In this paper, we solve the problem of *fair* round-robin token management scheme by actually considering and solving a strictly stronger problem (in fact, equivalent to leader-election) of *automata-based bidirectional round-robin token management scheme*: tokens have "positive" or "negative" signs, and depending on the sign they travel in opposite directions on the ring. In the steady state, there are exactly two cycling tokens, one in each direction.

1.4 The Price of Self-Stabilizing Leader Election

In Dijkstra's original paper in addition to the impossibility of deterministic round-robin token management scheme, he presented a deterministic automata- based token- management algorithm in a presence of a leader. However, choosing a leader among identical processors (i.e., leader election) is one of the most basic techniques in distributed computing. That is, once a leader is chosen, it can coordinate any distributed task. How difficult is it to elect a leader in a self-stabilizing fashion?

If every processor has a unique ID (i.e. at least a logarithmic memory at each node) then the problem of choosing a leader in a self-stabilizing manner becomes easy: the node with the smallest ID is a leader (i.e., [AKY90, APV91, AV91]). Notice, however, that the price of self-stabilization is high in both memory and communication. That is, in addition to logarithmic memory at each processor, the system must constantly send logarithmic size messages (i.e. containing ID's) *even in a steady state* to ensure proper stabilization. This does not give a constant size automata-based solution.

In this paper, we show that self-stabilizing bidirectional round-robin token management scheme already implies a leader election scheme and show how fair and efficient self-stabilizing bidirectional round-robin scheme can be achieved. We stress that our scheme utilizes only constant-size control signals and can be implemented in high-speed network hardware. That is, we present a uniform (i.e. universal) hardware switch which achieves fair round-robin token management scheme for rings of arbitrary size.

The bidirectional problem is strictly stronger than the unidirectional case: we show that on a unidirectional ring of finite-state machines, it is in fact impossible to elect a leader even in a randomized setting, while round-robin token management (with delays) is possible.

1.5 Previous Work on Self-Stabilizing Token Management

Previous solutions to the token problem include Dijkstra's original work [Dij74] which uses a leader to solve a uni- and bi-directional ring of automata with access to their neighbor's memory (shared-memory). Brown, Gouda, and Wu [BGW89] proposed a solution using a leader and introduced the use of additional control signals in the algorithm. Israeli and Jalfon proposed the first randomized token management scheme [IJ90a] in the shared memory model (they gave up the round-robin property by using random-walk on the ring.) Another token-management solution gave up generality by assuming a special size rings (prime-number size to avoid symmetries) by Burns and Pachl [BP88]; it is worthwhile mentioning that it presents a deterministic and uniform solution in this special case. Herman [Her90] presented a solution which requires strong synchrony and is also limited to special (odd) ring-size. Finally, Afek and Brown presented token-management in the message-passing model and round-robin fashion with the use of randomization, but still assumed that a leader exists.

1.6 Practical Applications

Our work is directly derived from practical systems. The main application is actual self-stabilizing mechanisms for future high-speed local area ring networks (LAN's) and embedded rings on general-topology networks. Note that many future fast (e.g., gigabit/sec.) LAN's: FDDI [R86], Meta-Ring [CO90], Cambridge LAN [HN88], Magnet [LTG90], ATM-ring [OMS89], etc., have ring-based topology. (Also, a recent suggestion for control on a general topology network, is to embed a virtual ring to support on-line high-speed control mechanism [OY90]). Having self-stabilizing mechanism can prevent costly centralized/duplicated monitoring and recovery protocols (for the high cost of such mechanism see e.g., [BCK+83].) As explained above, for future architectures e.g. [R86, CO90, OY90] we need hardware-oriented constant-time and constant-area control algorithm (finite automata) to support fast on-line processing at a low-level protocol.

Other applications (postponed till the full version) include computing a path (effectively embedding a bus-architecture in the ring), or the parity of the network, synchronization, counting, load balancing, various marking procedures, global regulation algorithms and similar control procedures.

The mechanism for *fair round-robin network access regulation* we achieve is an important regulation control task in existing (e.g., token ring) and future LAN's, e.g. MetaRing([CO90]). MetaRing employs a variant of a token, called *SAT-token*. The SAT-token distributes transmission-permits or quota to the active nodes, and as a result, multiple nodes can access the network at the same time. In this new scheme a node will hold the SAT-token only if it is not SATisfied (it could not send the quota given to it by the SAT-token during its previous visit). It has been shown that in high-speed implementations, the efficiency and effectiveness of this new fairness scheme increases as the transfer delay of the SAT-token decreases. Since the Meta-Ring (unlike token-ring) allows concurrent transmission, it has been further shown that when the SAT-token is used, the fairness and correct operation are preserved even during stabilization time, when there are several SAT-tokens in the system, as long as the token management scheme is round-robin and fair.

1.7 Software-Based Self-Stabilization

A recent set of results including general procedures for central fault-tolerance tasks, were designed in order to augment network protocols with self-stabilization capabilities (e.g., [KP90, AKY90, APV91, AV91]). These procedure are not limited by space or by processing and are therefore only suitable to augment high-layer (software) protocols but unsuitable to fast on-line (hardware) implementations. As explained above, it is crucial in the practical architectural needs for fast token (e.g. [CO90, OY90]) to have a constant delay per token at a node. Our motivation is therefore to deal with constant size (automata-based) processing resources (motivated both by practical needs and theoretical interest). We note that, the previous works mentioned above use either unbounded memory (and randomization), or require non-uniform processors with on-line access to protected unique ID's (which are too long for, or inaccessible to high switching components). The software-oriented methods are applicable to maintenance, while hardware-oriented methods are highly useful in on-line operation. For example, in a system like [OY90], the methods developed for general topology network, (e.g., [AKY90, APV91, AV91]) can be used to find and maintain a spanning tree over a general-topology network in a self-stabilizing fashion. Given a spanning tree, we can easily embed a virtual ring on it (as part of the self-stabilizing maintenance procedure). High-speed control information (as in [CO90]) is switched over the virtual ring in order to regulate the network in a fast, on-line, and fair fashion in a mechanism which is able to overcome transient faults.

2 The Model

Each processor is a probabilistic, finite state machine (PFSM). The PFSM can communicate with its two neighbors via asynchronous message-passing. Thus, the PFSM can receive constant-sized control messages from its neighbors. The PFSM can do a state-transition and send its own control messages only upon the receipt of such a message. Hence the processing at a node is done in a message-driven, asynchronous fashion. Two constant-sized buffers (ports) are used by each PFSM for communication with the left and right neighbor. (We can assume that the ring is oriented by using a self-stabilizing automata-based orientation protocol as suggested in [IJ90b].) A *bidirectional link* between two nodes is modeled as two independent asynchronous FIFO-channels which may lose messages. We allow neighboring nodes to detect *message collision* as a primitive event. That is, two control messages traveling in opposite directions are said to *collide*, if they physically cross each other on the ring[1].

The *global state* of the system is defined as the cross-product of the states of the PFSM's and the contents of the buffers and links. A *legal* state in our case is a global state in which there is exactly one token (in a buffer or on a link) in each direction and the state of each PFSM correctly reflects the last token passed through that node. An algorithm is said to be a self-stabilizing automata-based token management scheme, if starting in an arbitrary global state with at least one token, the scheme guarantees that (i) eventually a legal state is reached and (ii) every successor state of legal state is legal and tokens circulate in a round-robin fashion around the ring without delay. We postpone a more formal treatment to the final paper.

We note that the existence of such an algorithm implies that we can choose an *arbitrary* number of these PFSM's (think of a PFSM as a block of simple hardware), plug them together in a ring, and consequently obtain a correct token management scheme. Augmenting the mechanism with a global time-out mechanism implemented independently at each node, gives a full practical solution to the problem.

[1] In practice, if links are unit-capacity (i.e., similar to [APV91]), then a handshake protocol for the control messages to detect a collision can be implemented. Alternatively, if we assume an upper bound on the communication-delay for control-messages over a single link, we can detect collision by using a local time-out

3 The Algorithm

In this section we first present high-level ideas and then an exact formulation of an algorithm which achieves self-stabilizing round-robin token management on a bidirectional ring. The algorithm will stabilize to a state in which exactly one token circulates in each direction of the ring. Any mechanism which will make use of this algorithm is free to choose one of these tokens as its unique token with respect to the network-function associated with it (e.g. access-control). We call tokens circulating in one direction *positive* tokens and the token circulating in the opposite direction *negative* tokens.

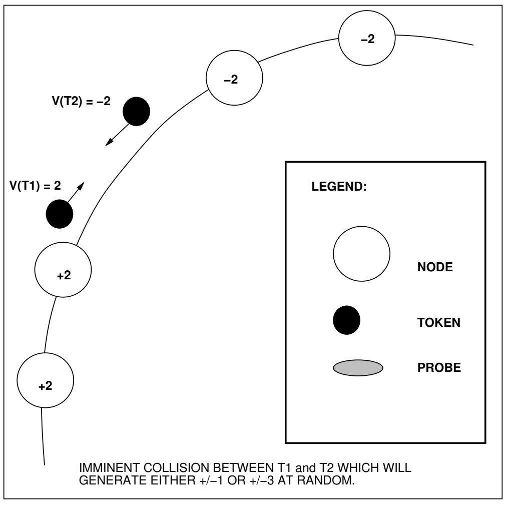
3.1 Basic Ingredients

The following type of control messages are used:

- Tokens: A token t is a constant-size message which carries a value $V(t) \in \{-3,-2,-1,1,2,3\}$. Tokens which go clockwise[2] always have a positive value and tokens which go counter-clockwise always have a negative value. When a token passes through a node, it leaves a "footprint" of its value on this node, overwriting any previous "footprint".
- Probes: A probe p is also constant-size message, which travels through the nodes, but it does not leave any "footprints". Probes also carry a value in $\{-2,-1,1,2\}$ and an additional enabled / disabled bit.

Nodes store "footprints" of passing-by tokens. When we say that node i has a certain "value" $V(i)$ we mean the value of its current "footprint".

Having introduced the basic elements, we present now high-level overview of the algorithm: (We note that line numbers used in this overview refer to lines in the code-like description of the algorithm, which can be found in the subsection 3.4.) As mentioned above, the most basic operation of a node i is to receive a token t, copy the value (or footprint) of t into its own register ($V(i) := V(t)$) and forward the token (Lines B11 and B12). In order to make the system self-stabilizing we need to add at least the following two mechanisms: *token-generation* and *token-elimination*. It should be noted at this point that both of these operations use local information only. As we will see later, this implies that our decisions are not always optimal with respect to the global state, but nevertheless, local information will prove sufficient to ensure convergence.



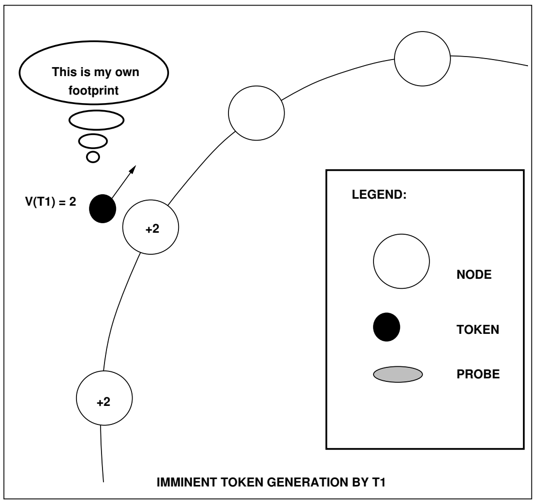
IMMINENT COLLISION BETWEEN T1 and T2 WHICH WILL GENERATE EITHER +/–1 OR +/–3 AT RANDOM.

In order to implement elimination, we introduce the notion of a *token collision*. Every time a positive and a negative token (t_1 and t_2) meet on the ring, they execute a collision-operation. A collision between t_1 and t_2 consists of executing the following operations: If t_1 and t_2 have identical absolute values ($|V(t_1)| = |V(t_2)|$) then the tokens flip an unbiased coin to agree on a new, distinct value (Lines A4, A5) while maintaining their sign. In other words, they agree on new, up to the sign identical, footprints.

In the other case, where their absolute values are different (e.g. $|V(t_1)| < |V(t_2)|$), the token with the smaller absolute value (e.g. t_1) is eliminated (Line A2). The intuition behind the collision-operation is the following: if two tokens have distinct values and these values originate from a previous collision, then there must be more than one pair of tokens in the system. If the (absolute) values are equal, we still cannot be sure that there is really just one pair, so we must continue to check, i.e choose a new random value. Note that the above is the only condition on which a token is eliminated and thus the non-deadlocking property of the algorithm follows easily (i.e. the last token in the system is never eliminated introducing a deadlock as part of the operation).

Now let us focus on the generation operation: When a token t arrives at node i and sees a footprint identical to its own ($V(t) = V(i)$), it assumes that the footprint at node i was left behind by t itself. If this assumption is correct, this implies that neither token t nor node i has seen another token since t left i at its last visit. Thus t can conclude that it is the only token left in the system and generates a new token of opposite sign. The idea of token-generation based on marking, has been used by [Mis83] for unidirectional rings, though his method is different and actually uses $O(\log n)$ space.

So far we have described the collision - (and thus the elimination -) and the generation-operation. Unfortunately this is not enough to ensure convergence. To see why, consider a state in which all tokens travel in the same direction (we call such a state *skewed*). In such a state there are only positive (or only negative) tokens left; if each pair of neighboring tokens leaves distinct footprints behind, neither a collision nor a generation will ever take place.

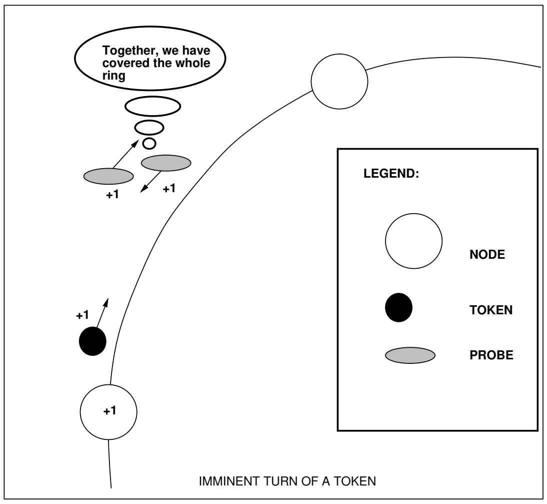


IMMINENT TOKEN GENERATION BY T1

To overcome this difficulty we introduce "exploration" by a second kind of messages, called *probes*. As we will show now, probes are used to explore the current state and, if appropriate, turn around a token from positive to negative (or vice versa). Probes are always generated by tokens ("fathers") in pairs ("siblings"). Siblings travel in opposite directions to each other. A probe traveling in the same direction as its father-token is called a *forward-probe*; analogously, a probe traveling in the opposite direction is called a *backward-probe*. Now, every time a token suspects that the system is in a skewed state, it sends a forward- and backward-probe to "explore" the global situation. The condition to send probes is enabled when a token seeing foot-

[2] WLOG, the ring is oriented, with one direction labeled "clockwise" and the other counter-clockwise. The self-stabilizing ring orientation in constant space was shown in [IJ90b].

prints (values) of a token of the same kind (direction) which are larger (in absolute values) than its own. So, for example, a positive token t will send out probes, if, upon arriving at some node i, the condition $V(t) < V(i)$ holds (Line B6). A probe p is basically carrying the value of its father t ($V(p) = V(t)$) and an enabled/disabled bit (initially enabled).



IMMINENT TURN OF A TOKEN

After a probe p has been generated (and p is enabled), one of the following two events will occur: (i) p meets or collides with a token t which has a value of opposite sign ($V(p)V(t) < 0$) or (ii) p collides with a probe p' which has the same value as p and p thus assumes that p' is its sibling.

In the first case, p has obtained evidence (i.e the token t) that the current state is not skewed and thus p will be disabled (Lines D1-D2, E1-E2). In the second case, p has not obtained any evidence that the current state is not skewed, so it will rely on the experience of p', since if p' is indeed p's sibling, these two probes have together covered the whole ring. If both are still enabled, then the probes conclude that the current state is indeed skewed, and thus the backward-probe, whose next collision is with its father-token, will change the sign (and direction) of its father (Line F3). Otherwise, they conclude (correctly), that the current state still contains tokens of opposite sign, and thus they eliminate each other (Line F6).

3.2 How to Piece the Basic Operations Together

It can be easily reconstructed how a lone token will generate a token of opposite sign. So we provide now an intuition on how the operations described above work together to reduce the number of tokens, if there are more tokens in the system than just one positive and one negative.

Our main tool which we are using below and a number of times in the correctness arguments of the next section is the following simple observation: Let $r, v \in \{1, 2, 3\}$. Let r be a random value and v either an arbitrary ("initial") value or another, independently obtained random value. Thus $Pr(r \neq v) > 0$. Thus in an infinite sequence of comparisons of two such values we get a mismatch with probability 1.

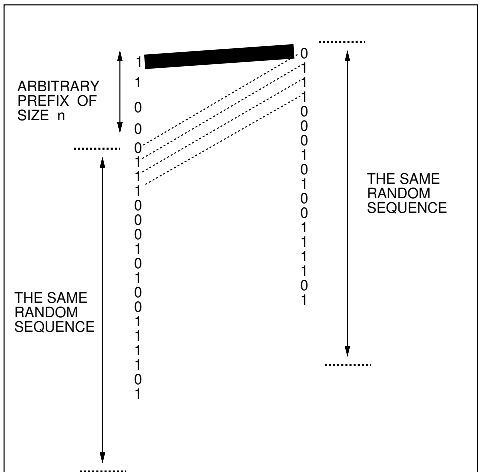
Now, if in a collision, at least one of the token carries a value obtained in a previous collision (and is thus random), then exactly the comparison operation described above will be executed and a token will be eliminated if there is a mismatch. Thus our strategy should be to ensure that this kind of collision is repeatedly going to take place as long as the system has not stabilized to one pair of tokens.

In order to show this we have to make sure that probes which originate from an "initial' state or have been produced erroneously are not an obstacle by causing tokens to repeatedly turn and thus preventing these useful collisions. This will be done by again using the simple observation mentioned above with respect to comparing values when (erroneous) backward-probes and tokens collide. Thus we will be able to conclude that despite the presence of arbitrary probes, the system stabilizes to exactly one pair of tokens.

Now, all we have to add to make the algorithm complete are conditions, on which disabled probe who do not have a sibling-probe (anymore), can be eliminated from the system. Note that this task is facilitated by the fact that erroneous decisions are not affecting the convergence of the tokens.

A way to view how we achieve progress towards convergence given an initial global state is via the following basic argument which can be called a "sliding rule" argument. Consider two copies of the same infinite sequence with one of the two sequences preceded with n arbitrary bits.

Notice that without a prefix, if we look at consecutive bits of the the sequence, they will always match (indicated by dotted lines). Our goal is to achieve a "matched" state, starting from another state which is not "matched". Essentially, the sliding argument says that if you "wiggle" the top pair at random (i.e. if mismatched, kill at random one of the mismatched bits and try again) then eventually they will match. This is used to argue that given a global state our algorithm will eliminate initial probes which could potentially be a part of an initial state, and which "do not match" any other probes.



The "Sliding" Rule

Note that from the arguments above, it also immediately follows that a node is permitted to "lose" incoming probes if their number exceeds its (constant-size) buffer-space without affecting the correctness.

3.3 Formal Description of the Algorithm

In the following we present the different types of messages used in the algorithm:

- *positive tokens*: circulating clockwise on the ring.
- *negative tokens*: circulating counter-clockwise on the ring.
- *probes*: tokens can send out probes to avoid getting stuck in an skewed state. A token will send out probes in both directions. A probe circulating in the same direction as its father-token is called a *forward-probe*, otherwise it is called a *backward-probe*.

Next we present the data-structures used by the algorithm.

- $V(i)$: value of node i ($1 \leq |V(i)| \leq 3$).
- $V(t)$: value of token t ($1 \leq |V(t)| \leq 3$).
- $S(t)$: bit indicating that token t is currently sending out probes.
- $V(p)$: value of probe p. $V(p)$ is composed of two integers: $V(p).v_1$ and $V(p).v_2$ ($1 \leq |V(p).v_1| \leq 2$, $2 \leq |V(p).v_2| \leq 3$) indicating respectively the value of probe

p's father-token and the value of the last node the father-token had visited (at the time p was generated).

- $E(p)$: bit indicating whether the probe p is enabled.
- $C(p)$: bit indicating: if p is backward-probe then whether it is about to turn its father; if p is forward-probe then whether it already has seen one matching backward-probe (it needs to see two matching backward-probes in order to decide to turn its father).

positive token t_1 and negative token t_2 collide:
A1: IF $|V(t_1)| \neq |V(t_2)|$ THEN
A2: eliminate token with smaller absolute value
A3: ELSE
A4: $V(t_1) := \text{RANDOM}(\{1,2,3\} - \{V(t_1)\})$;
A5: $V(t_2) := -V(t_1)$
A6: END;
A7: $S(t_1)$:= FALSE; $S(t_2)$:= FALSE;

Node i receives positive token t :
B1: IF $V(i) = V(t)$ THEN
B2: generate token t' (* negative *);
B3: $V(t) := V(t) + 1$;
B4: $V(t') := -V(t)$
B5: $S(t)$:= FALSE; $S(t')$:= FALSE
B6: ELSIF $V(t) < V(i)$ THEN
(* t follows a larger-value pos. token *)
B7: send forward-probe p_1 and backward-probe p_2 with $V(p_1).v_1 = V(p_2).v_1 = V(t)$ and $V(p_1).v_2 = V(p_2).v_2 = V(i)$
B8: $S(t)$:= TRUE
B9: ELSE $S(t)$:= FALSE
B10: END
B11: $V(i) := V(t)$;
B12: send positive token t (and negative token if one newly generated)

Node i receives negative token t :
symmetric to the receipt of a positive token

Node i receives forward-probe p:
C1: send probe p

Node i receives backward-probe p:
D1: IF $((V(i))(V(p).v_1) < 0)$ AND $E(p)$ THEN
D2: $E(p)$:= FALSE
D3: END
D4: send probe p

Token t and probe p collide:
E1: IF $(V(t))(V(p).v_1) < 0$ THEN
(* p is a forward-probe *)
E2: IF $E(p)$ THEN $E(p)$:= FALSE
E3: ELSE eliminate p
E4: END
E5 ELSE (* p is a backward-probe *)
E6: IF $\neg E(p)$ THEN
eliminate p
E7: ELSIF $E(p)$ AND $C(p)$ THEN
E8: IF $(V(p).v_1 = V(t))$ AND $S(t)$ THEN
E9: turn t;
E10: $V(t) := -V(p).v_2$;
E11: $S(t)$:= FALSE
E12: END;
E13: eliminate p
E14: END
E15: END

forward-probe p_1 and backward-probe p_2 collide:
F1: IF $V(p_1) = V(p_2)$ THEN
F2: IF $E(p_1)$ AND $E(p_2)$ THEN
F3: IF $C(p_1)$ THEN $C(p_2)$:= TRUE; eliminate p_1
F4: ELSE $C(p_1)$:= TRUE; eliminate p_2
F5: END;
F6: ELSE eliminate p_1 and p_2
F7: END
F8: END

Note that we make use of a function RANDOM(S) which returns a random value drawn from from the set S. We will only use it for sets of size 2.

4 Analysis

In this section we give a formal correctness-proof of the round-robin token management scheme provided in the previous sections.

Theorem: (MAIN) There is a randomized self-stabilizing algorithm for *fair round-robin token management* on an asynchronous bidirectional ring of oblivious (name-less) finite automata processors, using constant size buffers and messages, starting from an arbitrary non-deadlocked state.

Since the proof will be quite involved, we will precede its presentation with a subsection introducing all necessary definitions and a subsection giving an overview of the proof.

4.1 Definitions

We start with defining a global state of the system and the transitions among those global states. A global state is basically the value of each node, the value of each token and the relative position of the tokens among each other on the ring. A transition from a state to its successor-state consists either of a collision among two tokens, or a turn of a token, or a generation of a token. Further we define a local state of a ring segment with respect to the probes present on it. Note that probes are "invisible" in the global state. One can think of a local state as putting a magnifying glass on a segment of the ring.

Definition 1 *(local and global state, state-transition, segment)*

1. The *global state* of the system is defined as:
   - for each token t its value $V(t)$.
   - the relative position of the tokens on the ring.
   - for each node i its value $V(i)$.
2. *(global) state transition* from state s to state s' is defined by a *successor* function $S : s' = S(s)$ where s' results from s by applying one of the following transitions: {collision, turn, generation} (See previous section for a detailed description of these transitions). Let $S^j(s)$ be the jth successor of s.
3. $|s|$ denotes the number of token in state s.
4. A sequence of nodes starting at some node i and ending at some node j, such that $V(i) = V(i+1) = \ldots = V(j)$ is called a *segment* σ of value $V(\sigma) = V(i)$. We will say that a token or a probe is *on* segment σ, if it is on a node k, $i \leq k \leq j$ or if it is on a link $(l, l+1)$, $i \leq l < j$.

   We will use the symbol t alternatively for the token itself or the segment the token t is creating.
5. The *local state* of a segment σ in a global state s is defined as:

- its value $V(\sigma)$.
- for each probes p on σ its value $V(p)$ and its position. Let $f^{\sigma}(s)$ ($b^{\sigma}(s)$) denote the set of enabled forward-probes (backward-probes) on σ.

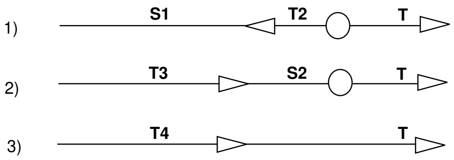
In terms of the above definition we can now define the correctness condition that any round-robin token management scheme must follow:

Definition 2 *(correctness)*
For an *arbitrary* state s with $|s| \geq 1$ there are infinitely many successor-states and, in particular, there is a state $S^{l}(s)$, such that $|S^{l}(s)| = 2$ and such that every state $S^{l_2}(s), (l_2 > l)$ results from a collision of two tokens with identical absolute value.

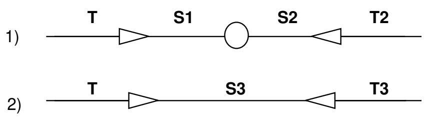
Since "initially" we have no control over the values stored in the nodes or on the tokens, we will define now a predicate over states, which will indicate that, although the system may not have stabilized yet (and indeed might have more tokens than "initially"), we have reached a certain level of order in the system. Intuitively, this means that every node has at least seen one token and thus the neighborhood of a token cannot any longer be be influenced by the "initial" state.

Definition 3 *(cover)*
We say that a state s has the predicate *cover*, i.e. cover(s) if:

- $|s| \geq 2$.
- the ring is a concatenation of segments, such that the segment boundaries are exactly the tokens and a subset of their previous collision-points. Below, we enumerate first all possible neighborhoods to the left of a (positive) token t and then to the right of t ("$\lhd$", "$\rhd$" denote tokens (and their direction), "$\bigcirc$" denote previous collision-points. s's denote segments and ts denote tokens and the segment they are creating). Tokens traveling from left to right are positive tokens and thus their value is always positive. Tokens traveling from right to left are negative tokens and carry a negative value.



The values in the above scenarios are constrained as follows: $|V(t2)| = |V(t)|$, $|V(s1)| \neq |V(t)|$, $|V(s2)| = |V(t)|$, $V(s2) < 0$, $|V(t3)| \neq |V(s2)|$, $V(t4) \neq V(t)$.



In addition to the above two cases, the right neighborhood of token T can also be like a left neighborhood of another positive token (i.e. see the three cases in the previous picture).

The values in the above scenarios are constrained as follows: $|V(s1)| \neq |V(t)|$, $V(s1) < 0$, $V(s2) = -V(s1)$, $|V(t2)| \neq |V(s2)|$, $|V(s3)| \neq |V(t)|$, $|V(t3)| \neq |V(s3)|$.

The cases for a negative token are symmetric.

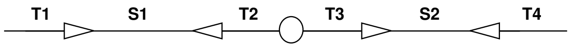
We conclude this section with the following sequence of simple definitions:

Definition 4 *(fragment)*
A *fragment* is a largest concatenation of segments, such that the structural requirement of a cover is still fulfilled and at least one token is present on this part of the ring.

Definition 5 *(block)*
We say that a segment σ has the predicate *block* in state s, i.e. block(σ, s) if all probes in $b^{\sigma}(s)$ have identical values.

Definition 6 *(state as a sequence of pairs)*
If for state s cover(s) holds, then we can view s alternatively as a sequence of pair of numbers. Every pair represents one token and the segment it is traveling on (through its *token*-component and *segment*-component).

For example the following cover corresponds to the sequence $s = (t1, s1)(t2, s)(t3, s2)(t4, s2)$:



4.2 Overview of the Proof

The proof will be structured as follows: For every state s with $s \geq 1$ we define a *subgoal* and show that this subgoal will indeed be reached. For every possible sequence of these subgoals, the last one is always to reach stability. Thus if the algorithm reaches for each state the desired subgoal, correctness is assured. The following is now a list of different states and their subgoals:

1. For a state s with $|s| \geq 1$ and $\neg cover(s)$ the subgoal is to show that there is a successor state $S^{l_1}(s)$ for which $cover(S^{l_1}(s))$. Note that it is very well possible that $|S^{l_1}(s)| > |s|$ and thus at first glance, the successor- state seems to be farther away from our ultimate goal. However, the predicate cover implies that we have reached a higher level of order in the system, since as soon as cover holds, no generation-transition can take place anymore and cover is a stable predicate. Thus it guarantees that the system does not move away anymore from its goal-state (by generating additional tokens). Another important fact is that any state with a cover has a successor-state. If the state is skewed, this property holds thanks to our probes-scheme.

2. For a state s with $cover(s)$, $|s| > 2$, $odd(|s|)$ the subgoal is to show that there is a successor state $S^{l_2}(s)$ such that either $|S^{l_2}(s)| < |s|$ or $Pr(|S^{l_2}| < |s|) > 0$. If the first disjunctive part of our goal holds, then we have made direct progress towards the goal state, otherwise, we must have at least reached a state in which we could have made progress, i.e. there was a nonzero probability to reduce the number of tokens in the transition to this state. This probability follows from the fact that in a state with an odd number of tokens, there will be always at least one token which collides consecutively with at least two other tokens. Since in the first collision, it picked up a random value independent of the rest of the system, the subgoal follows.

3. For a state s with $cover(s)$, $|s| > 2$, $even(s)$, the subgoal is to show that either $|S^{l_3}(s)| < |s|$ or $Pr\ (|S^{l_3}(s)| < |s|) > 0$ or that there is a pair of tokens (t_1, t_2) of the following form $(t_1, \sigma_1)(t_2, \sigma_2)$ such that $Pr(block(\sigma_2, S^{l_3}(s))) > 0$. Thus in the case in which there is an even number of tokens we restrict ourselves to an even more modest goal. This is necessary since tokens can (at least temporarily) form pairs, such that every token only collide with its partner and then turns and repeats this cycle. However note that in order to turn, a token needs to be supplied with probes. If now a *block* of probes is on the adjacent segment of a token, then the token, which carries a random value independent of the rest of the system has a nonzero probability to mismatch these probes and thus will break out of its local cycle.

4. For a state s with $|s| = 2, cover(s)$, the subgoal is to show that the left-over probes will eventually be removed from the the system. This task is considerably simplified by the fact that as long as the system is in a state with more than two tokens, we are allowed to make "mistakes" with respect to this task, since we have shown convergence to a state with two tokens for arbitrary configurations of probes.
5. For a state s with $s = (t_1, \sigma_1)(t_2, \sigma_2)$, we show that if there are no left-over probes, we have reached stability.

The rest of the proof is basically to tie all the above subgoals together to show that in an infinite sequence of states, the probability is 1 that the desired stable state is reached.

4.3 Correctness Proof

In order to prove our main theorem, we will introduce a series of lemmas. The first lemma addresses the non-deadlocking property of the algorithm:

Lemma 1 For every state s for which $|s| \geq 1$: $\forall l > 0 : |S^l(s)| \geq 1$.

Proof: The only condition on which a token is eliminated is the collision of two tokens (Line A1 in the algorithm). Thus every time a token is eliminated another token is surviving. ∎

The following lemma proves that we reach our first subgoal, a state s for which $cover(s)$:

Lemma 2 For every state s, for which $|s| \geq 1$, there exists a finite $l > 0$ such that cover($S^l(s)$).

Proof: First we need to show that every state s, for which $\neg cover(s)$, has a successor-state: (i) s is skewed, $|s| \geq 2$: Assume that no transition takes places. Then eventually $s = (t_1, \sigma_1)(t_2, \sigma_2) \ldots (t_{|s|}, \sigma_{|s|})$, where $V(t_i)$ are either all positive or all negative and $\sigma_i = t_{i+1}$. Thus, as it can be easily seen: $cover(s)$. (ii) s is not skewed, $|s| \geq 2$: there are always two opposite tokens facing each other with no other token in between. Thus either a turn- or a collision-transition is imminent. (iii) $|s| = 1$: if the token does not turn, then a generation-transition will take place after at most one round.

Consider a link $(i, i+1)$ as a pair of values of its end-nodes $(V(i), V(i+1))$ together with the values of tokens which might be in transit on this link. Based on this information, we can decide for every link whether it could be a part of a fragment, i.e. the link is *consistent.*

First note that if a token has crossed a link and updated the value of the destination-node, the link will become consistent. Thus, a token can enlarge a fragment by crossing the first inconsistent link outside of its fragment. In that case, we say that the token is at the *border* of its fragment. Further, every generation-, collision-, and turn-transition preserves the fragment in which (or at whose border) it occurs (see Lines A1-A6, B2-B4, E9-E10). Hence the consistency of a link is a stable property.

It remains to show that eventually every link will be crossed by a token and that at least two tokens will be present: By Lemma 1 all successor-states have at least one token and thus at least one fragment. Assume that there is no successor-state of s, such that a given link has been crossed by a token. Then there must be a state from which on no fragment grows. This implies in turn that there must be infinitely many turns on both ends of every fragment. Every turn consumes two probes. But every gap between two fragments contains at least one node such that backward-probes, which are needed for a turn, cannot cross the gap in at least one direction without becoming disabled (compare condition on Line D1 in algorithm). Thus eventually there will be one such segment σ empty of backward-probes. And the next time a token travels on σ, it will not turn and thus it will enlarge its fragment, which contradicts our assumption.

If a state is reached with a single fragment spanning the whole ring and just one token is alive, then after at most one round a second token will be generated (compare condition on line B1) and thus a cover will have been reached. ∎

The next few lemmas show that the cover-predicate is stable and well-defined with respect to the successor-function. We also show that no generation can take place in a state in the cover-predicate holds.

Lemma 3 Every state s, for which cover(s), has a successor-state.

Proof:

- s is skewed: If no transition takes place then eventually $s = (t_1, \sigma_1)(t_2, \sigma_2) \ldots (t_{|s|}, \sigma_{|s|})$, where $V(t_i)$ are either all positive or all negative and $\sigma_i = t_{i+1}$. Thus, for at least one token t_i the condition to send out probes (Line B7 or symmetric case) will be true every time it visits a node. t_i sends forward-probes on σ_i and backward-pros on t_i. Given the structure of s, these probes cannot be disabled (i.e. conditions on lines D1, C1, and E1 cannot become true). Thus there will be at least one segment σ_k, on which there is at least one forward-probe which matches t_i's backward-probes and thus there will be a successor-state of s, resulting from a turn-transition of token t_k.
- s is not skewed and consequently there is a fragment $(t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1) > 0$, $V(t_2) < 0$. Thus a turn-transition or a collision-transition is imminent.

∎

Lemma 4 For every state s for which cover(s): $|S(s)| \leq |s|$.

Proof: By definition of a cover, the only condition on which the algorithm generates a new token (Line B1 in the algorithm) cannot be enabled in s. ∎

Lemma 5 For every state s for which cover(s): cover($S(s)$).

Proof: By Lemma 4, there is no generation-transition leading out of s. As it can be easily verified, every possible turn-transition and, given that $|s| > 2$, every possible collision-transition preserves the cover (see Lines A1-A6, B2-B4, E9-E10). Also, if $|s| = 2$ and cover(s), then, by definition of a cover, all collisions will take place between tokens of equal absolute value. ∎

Lemma 6 For every state s, for which cover(s), there exists a finite $l > 0$ such that $S^l(s)$ results from a collision-transition.

Proof: By Lemma 3 we have always a "next" transition, and by Lemma 4 there is no generation-transition. If all successor-states of s were results of turn-transitions, then the sum of the values of all tokens would steadily increase. Since this sum is bounded, this is not possible. ∎

Lemma 7 Starting in any state s for which $cover(s)$, no token will be in two or more consecutive turn-transitions.

Proof: After a token t has been in a turn-transition, its absolute value ($|V(t)|$) has been increased. Furthermore its is traveling on a segment whose value is t's old value and thus as long as it is on this segment, t will not turn again. t can leave this segment only through a collision-transition. The lemma follows. ∎

Lemma 8 For every state s, for which cover(s), and for every token t in s, there exists a finite $l_t > 0$ such that $S^{l_t}(s)$ results from a collision-transition involving token t.

Proof: By induction on $|s|$:

- Basis: $|s| = 2$: follows immediately from Lemma 6.
- Hypothesis: $|s| = n$.
- Step: $|s| = n + 1$. Assume that token t will never be involved in a collision-transition. Then by Lemma 7, t can be involved in only one turn-transition. Thus token t will be neither in a turn-transition nor in a collision-transition for arbitrary many successor-states. By definition of a cover, token t will thus eventually be in the following fragment: $(t, \sigma_1)(t', \sigma_2)$ where $V(t)$ and σ_1 are either both positive or both negative and $t' = \sigma_1$. Now, if t' turns, then t is in a collision. If t' is in a collision where one token is eliminated, then we can apply the hypothesis. If no token eliminated then t is in collision. Thus t' is neither in a turn-transition nor in a collision-transition. Thus it will eventually be in a analogous fragment as t. Then we can look at t' right neighbor etc. Finally we will get a contradiction with Lemma 3 and thus our assumption that t will never be involved in a collision-transition was wrong.

∎

The next lemma shows that any token which is involved in at least two consecutive collisions creates a nonzero probability that a token will be eliminated:

Lemma 9 The following two statements concern probabilistic elimination of tokens and hold in any state s with $cover(s)$:

1. if, starting in state s, there is at least one token which consecutively collides with at least two opposite tokens (let $S^k(s)$ be the state right after the second collision), then $Pr(|S^k(s)| < |s|) > 0$.
2. if $Pr(|S^k(s)| < |s|) = 0$, then every token collided with at most one opposite token in all transitions leading from s to $S^k(s)$.

Proof: (1) The value the token carries after the first collision is random and independent of any other value in the system. Thus there is a nonzero probability that at the second collision the tokens carry distinct absolute values and consequently there is a nonzero probability that one of the tokens involved in the collision will be eliminated (see Lines A1-A2). In addition, no new tokens are generated (by Lemma 4). (2) follows from the contraposition of 1) and Lemma 7. Note that a token needs to turn at least once between any two collisions to ensure that $Pr(.) = 0$, and by Lemma 7 a token can turn at most once. ∎

The next lemma shows that our second subgoal will be reached:

Lemma 10 For every state s with odd($|s|$) and cover(s), exists a finite $l > 0$, such that $(|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0)$.

Proof: By Lemma 8 every token t present in s will be eventually involved in a collision-transition with another token t'. If one of these tokens is eliminated then we are done. Otherwise the fragment of these two tokens right after the collision looks as follows: $(t, \sigma_1)(t', \sigma_2)$, where $V(t), V(\sigma_2) < 0$ and $V(t'), V(\sigma_1) > 0$. Since the number of tokens in s is odd, there will be a token t which after its first collision with t' will be involved in a collision with a different token t'', $(t'' \neq t')$. By Lemma 7, there cannot be two or more consecutive turns by a token. Thus t must have been involved in two consecutive collisions. Now we can use Lemma 9 and we are done. ∎

Definition 7 *(partner, pair,local cycle, neighbor)*

- Starting in some state s for which $cover(s)$, $even(|s|)$, $|s| > 2$ we call two tokens involved in their first collision *partners* with respect to state s, and the function partner$_s(t)$ yields the partner of token t with respect to state s. By Lemma 8 this function is well defined. Both partners together form a *pair*.
- A *local cycle* of a pair is a sequence of the three transitions which brings both tokens back to the same relative position: Each token is involved in a turn-transition and in a collision-transition with its partner. Note that if both tokens started out with the same value, this is exactly the scenario for which the probability of decreasing the number of tokens is zero (Lemma 9).
- The two tokens on the left and on the right (which might coincide) of a pair are called *neighbors* of that pair.

The next few lemmas provide useful facts about pairs of tokens and the local state of adjacent segments:

Lemma 11 Let s be a state, such that $cover(s)$, $|s| > 2$, $even(|s|)$. If in s the neighbors of a pair are not a pair, or if a pair does not correspond to one of the following fragments, then exists a finite $l > 0$, such that $(|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0)$. For any pair the fragment looks like: $(t, \sigma_1)(partner_s(t), \sigma_2)$ with the following restrictions:

1. $V(t) > 0, V(partner_s(t)) < 0, |V(t)| = |V(partner_s(t))|$, $\sigma_1 = \sigma_2$ (* about to collide *)
2. $V(t) < 0, V(partner_s(t)) > 0, |V(t)| = |V(partner_s(t))|$ (* right after collision *)
3. $V(t) > 0, V(partner_s(t)) > 0, |V(t)| > |V(partner_s(t))|$, $|V(\sigma_2)| = |V(t)|$ (* both in pos dir *)
4. $V(t) < 0, V(partner_s(t)) < 0, |V(t)| < |V(partner_s(t))|$, $|V(\sigma_1)| = |V(partner_s(t))|$ (* both in neg dir *)

Proof: If the neighbors are not a pair, a similar argument as in the proof of Lemma 10 shows that a token must be involved in two consecutive collisions. If a pair is not in one of the scenarios above, a collision-transition with an elimination is imminent. ∎

Lemma 12 If in state s (cover(s), $|s| > 2$, $even(|s|)$), a pair is in the following form: $(t, \sigma_1)(partner_s(t), \sigma_2)$, where $|V(t)| = |V(partner_s(t))|$, $V(t) < 0$, $V(partner_s(t)) > 0$ and $V(\sigma_1) < 0$, then for every probe $p \in b^{\sigma_1}(s)$ the value $V(p)$ is mutually independent from $V(t)$.

Proof: For every probe $p \in b^{\sigma_1}(s)$ we can conclude that token t cannot have produced p since its last collision. The lemma follows now since the value $V(t)$ is random and mutually independent from every other value in the system. ∎

Lemma 13 If in state s (cover(s), $|s| > 2$, $even(|s|)$), a pair is in the following form: $(t, \sigma_1)(partner_s(t), \sigma_2)$, where $|V(t)| = |V(partner_s(t))|$, $V(t) < 0$, $V(partner_s(t)) > 0$ and for at least one of t or $partner_s(t)$, the next transition is not a turn, then exists a finite $l > 0$, such that $(|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0)$.

Proof: Assume wlog that t's next transition is not a turn. By Lemma 8 every token will be involved in a collision leading into a successor-state (and thus every token is involved in a future transition). From this follows that t will be involved in a future transition and the only transition which is possible is a collision. If the other token involved in this particular transition is not $partner_s(t)$ then we can apply Lemma 9, if this token is indeed $partner_s(t)$, then, since $|s| > 2$, $partner_s(t)$ must have been in at least two consecutive collisions, and thus we can also apply Lemma 9 with respect to $partner_s(t)$. ∎

The next lemma show that our third subgoal is reached:

Lemma 14 If the system is in state s, cover(s), $|s| > 2$, $even(|s|)$, and a pair is in the following form: $(t, \sigma_1)(partner_s(t), \sigma_2)$, where $|V(t)| = |V(partner_s(t))|$, $V(t) < 0$, $V(partner_s(t)) > 0$ and $V(\sigma_1) < 0$, $V(\sigma_2) > 0$, then exists a finite $l > 0$, such that either $(|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0)$ or $Pr(block(\sigma_2, s^l)) > 0$.

Proof: If either t's or $partner_s(t)$'s next transition is not a turn then we can apply Lemma 13. Otherwise the transitions of t and $partner_s(t)$ form a local cycle. In order to maintain this cycle both tokens must continuously turn and thus use probes in $|b^{\sigma_1}(s)|$ and $|b^{\sigma_2}(s)|$.

So let us consider how $|b^{\sigma_2}(S^k(s))|$ can change for $k > 0$: First note that also the neighbors will form a local cycle (otherwise we can again invoke Lemma 13). This implies that there will be always new "incarnations" of σ_2 with initially $|b^{\sigma_2}(S^k(s))| = 0$. Note that all probes entering σ_2 are either generated by the neighbor, or else passed through the neighbors which form the following fragment: (by Lemma 11, Scenario 3): $(t', \sigma_3)(partner_s(t'), \sigma_4)$ $V(t') > 0, V(partner_s(t')) > 0, \sigma_3 = partner_s(t'), V(\sigma_4) = V(t)$. In order to pass through this fragment, the incoming backwards-probes must mismatch all probes in $f^{\sigma_4}(s)$ all of which originated at token $partner_s(t')$. If incoming backwards-probes originated with $partner_s(t')$ then they obviously match, otherwise they match with nonzero probability, since the forward-probe carries a random value independent of the value of the backward-probe (By Lemma 12). Now if the first incoming probe matches, only probes generated by $partner_s(t')$ will be on $partner_s(t')$ and consequently on σ_2. Thus for some finite $l > 0$: $Pr(block(\sigma_2, s^l)) > 0$, or, in other words, the neighboring pair acts as a *probabilistic filter* with respect to incoming backwards-probes.

Note that if there are any disabled probes on σ_2 which match any of the enabled probes, then they will merely eliminate the enabled probe and thus not interfere with the block-predicate. ∎

Lemma 15 For every state s $(cover(s), |s| > 2, even(s))$, exists a finite $l > 0$, such that either $(|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0)$.

Proof: By Lemma 8, there is a successor-state in which a pair is guaranteed to be in the following scenario (i.e. right after a collision): $(t, \sigma_1)(partner_s(t), \sigma_2)$, where $|V(t)| = |V(partner_s(t))|$, $V(t) < 0$, $V(partner_s(t)) > 0$ and $V(\sigma_1) < 0$, $V(\sigma_2) > 0$. By Lemma 14, if $\neg((|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0))$ then there is a nonzero probability that there is a block on at least one of σ_1 or σ_2. By Lemma 12 the value of the block is independent of the token's value. Thus there is a nonzero probability that the block has a different value than the token and thus the next transition of the token is not a turn. Using Lemma 13 and noting that the probability for a block on σ_2 and the probability that there will be an elimination in the subsequent collision (second in a row) are independent, we are done. ∎

The last few lemmas concern the last two subgoals, i.e.states with two tokens:

Lemma 16 for every state s with $cover(s)$, $|s| = 2$, there a finite $l > 0$, such that $S^l(s) = (t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1)$, $V(\sigma_2) > 0$, $V(t_2), V(\sigma_1) < 0$ and $V(t_1) < V(\sigma_2)$.

Proof: Lemma 6 guarantees that there will be always a state s' as described above except maybe for the condition $V(t_1) < V(\sigma_2)$. If this condition does not hold, then by simply following the algorithm, we can conclude that the full condition holds in either $S(s')$ or $S^2(s')$. ∎

Lemma 17 If there are no enabled probes in a state s of the form of Lemma 16, then the system has reached stability.

Proof: Remember that $s = (t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1)$, $V(\sigma_2) > 0$, $V(t_2), V(\sigma_1) < 0$ and $V(t_1) < V(\sigma_2)$. Assume wlog that just before the next transition $s = (t_1, \sigma_2)(t_2, \sigma_2)$. Thus t_1 is sending out probes ($S(t_1) =$ TRUE). All of these forward-probes are in $f^{\sigma_2}(s)$ and all of its backward-probes are in $b^{t_1}(s)$. Following the algorithm, we get $S(s) = (t_2, \sigma_3)(t_1, \sigma_4)$, where $V(\sigma_3) > 0, V(\sigma_4) < 0$. For all probes which were in $f^{\sigma_2}(s)$ the conditions on Line E1 and E2 became true before the transition and thus have been disabled. All these forward-probes are now on σ_4. All enabled backward-probes are now in $b^{\sigma_3}(S(s))$. Thus before the next transition, all backwards-probes will have been in a collision with a disabled forward-probe and thus be eliminated. Also, all disabled forward-probes which have not been in a collision with two backward-probes, will be eliminated before system enters state $S^2(s)$ (condition on Line E3 in the algorithm is enabled on the collision of t_2 with t_1). ∎

Lemma 18 For every state as in the form of Lemma 16, the number of enabled probes is not larger than the previous time the state of the system was of this form. There is a nonzero probability that the number is strictly smaller.

Proof: Trivially, $|b^{t_1}(s)| = |f^{t_1}(s)| = |b^{t_2}(s)| = |f^{t_2}(s)| = 0$. Wlog let us assume that just before the next transition $s = (t_1, \sigma_2)(t_2, \sigma_2)$. Thus we concentrate on $b^{\sigma_2}(s)$. If before t_1 reached σ_2, $|b^{\sigma_2}(s)| \neq 0$, then there is a nonzero probability that some of these probes will match t_1's probes (since is $V(t_1)$ is random and independent). Thus, assume that k $(1 \leq k \leq |b^{\sigma_2}(s)|)$ probes match and that l is the number of probes t_1 is going send out before it turns. Then if $k > l$ there will be at most $k - l$ enabled probes left in the next state of the form of Lemma 16. If $l < k$, then no enabled probes will be left (by simply following the algorithm). ∎

Lemma 19 The lifetime of a disabled probe is finite.

Proof: Just follow algorithm. ∎

We now get to our main theorem which ties all the previous lemmas together and show the correctness of our algorithm:

Theorem 1 *(Correctness)* There is a randomized self-stabilizing algorithm for *fair round-robin token management* on an asynchronous bidirectional ring of oblivious (name-less) finite automata processors, using constant size buffers and messages, starting from an arbitrary non-deadlocked state. Furthermore, the lifetime of any control-signal, except for one pair of a positive and a negative token, is finite.

Proof: The following chain of arguments proves that starting the system in an arbitrary state s, $|s| > 0$ will lead with probability $= 1$ to a stable state $S^l(s)$ with $|S^l(s)| = 2$ $(l \geq 0)$:

By Lemma 2, $cover(S^{l_1}(s))$ for a finite l_1. If $|S^{l_1}(s)| > 2$, then by Lemma 10 and Lemma 15 there exists a finite l_2, such that $(|S^{l_2}(s)| < |S^{l_1}(s)|) \vee (Pr(|S^{l_2}(s)| < |S^{l_1}(s)|) > 0)$. Also, by Lemma 4 and by Lemma 5, $|S^k(s)| \leq |S^{l_1}(s)|, \forall k > l_1$. Suppose now we have an infinite sequence of states such that $|S^m(s)| = |S^{l_1}(s)|$ $(\forall m \geq 0)$. The number of different states in the system is finite. So some state s' for which $|Pr(|S(s')| < |s|) > 0$ is visited infinitely often. Therefore the probability of the computation we have supposed is zero. This argument can be repeated until for some l_3 we have: $|S^{l_3}(s)| = 2$. By Lemma 17,Lemma 16 and an argument similar to the above, there is now an l_4 for which $S^{l_4}(s)$ is stable. Using Lemma 19 together with the argument above, we can conclude that the lifetime of every signal except for one positive and one negative token is indeed finite. ∎

5 Stabilization Time

The stabilization time is the time-interval starting at the end of the last transient fault and ending when the system has reached a legal state.

Theorem 2 The expected worst-case stabilization time of the token management scheme is polynomial in the actual size of the ring.

We postpone the proof until the final version, it involves analysis of the time to get rid of the initial unmatched system's elements, and analysis of convergence of a "clean" system.

6 Symmetry Breaking

Symmetry breaking is an important procedure in many parallel and distributed computing scenarios (see e.g. [AAHK86, AB89, AN90, Ang80, ASW85, CV86, Dij74, It90, IR81, FL84, FS86, PKR82, RL81, STT89, SS89, V84]) In this section we demonstrate a reduction from the problem of self-stabilizing leader-election (breaking symmetry among the ring processors) to the round-robin token management.

Since we assume that the nodes in the ring are identical, the task of electing a unique leader and thus breaking the symmetry of the ring is an important task which will serve as a building block for other applications. First we need the following definition:

Definition 8 *(Leader)*
We augment every node i with one bit $(l^i(s))$ indicating that in state s node i claims to be the leader. Let $L(s) = \sum_{i=1}^{n} l^i(s)$.

The obvious goal of electing a leader is, starting in a state s with arbitrary $L(s)$ to converge to a state s' where $L(s') = 1$ and where the location of the leader eventually remains fixed. This is in fact what we achieve, given a by-directional round-robin token management scheme as a subroutine. That is, we use the round-robin token management as a basic building block for leader election, such that once tokens have reached a stable state, exactly one stationary leader will be elected. We use collisions as a way to make conclusions about the current value of L. The following is a high- level description of our approach: Every token carries an additional state indicating whether it has seen (1) no leader, (2) exactly one leader, or (3) more than one leader since its last collision, (4) the token is eliminating all leaders it visits until its next collision. Thus, on a collision the following actions will be carried out:

1. (Create a leader if no leader exists):

   IF both tokens have not seen a leader since the last collision THEN generate a leader; set the state of the tokens to (1, 2).

2. (It was ok in the previous "cycle", lets check again):

   IF the sum of leaders seen by both tokens since the last collision is exactly one THEN set the state of the tokens to (1,1).

3. (Something could be wrong, lets clean the whole ring:)

   IF the sum of leaders seen by both tokens since the last collision is larger than one THEN generate a leader; set the state of the tokens to (4,4) (cleaning mode).

4. (We eliminated at least one leader, lets check the state:)

   IF at least one token is eliminating THEN set the state of the tokens to (1,1).

We stress that the above solution provides us with unique leader with essentially the same stabilization time as the underlying problem of self-stabilizing token scheme.

In this extended abstract we only give an outline of high-level steps of the correctness Proof. Let s be the state leading into a collision. When the round-robin token management has stabilized and every token has already been in at least one collision, then in every subsequent collision the tokens can make accurate conclusions about the value $L(s)$ as it was right after the system entered state s. In fact, we can reduce self-stabilizing token management to self-stabilizing leader election, thus establishing:

Theorem 3 *(Equivalence)* Given either self-stabilizing *leader election* or self-stabilizing *round-robin token management* algorithm with polynomial time stabilization, there exists a polynomial time self-stabilizing algorithm for the other problem on an asynchronous bidirectional ring of oblivious finite automata processors and constant size messages.

7 Unidirectional Case

We assume now a uni-directional ring with asynchronous links and show that it is very much different from by- directional case. As was mentioned in the introduction, the following randomized solution is possible: every token flips a coin and either stays for the duration of the software clock-tick or advances. When two tokens meet, one is eliminated. Notice that this solution achieves round-robin property (i.e. tokens do travel in a cycle) however the scheme is not *fair*: the token has an expected delay every two consecutive steps. Thus, the solution is not efficient (i.e. for the hardware implementation, where the symbol must travel without delays), nevertheless, it is interesting that there is a randomized non-deadlocking *token management* algorithm on an asynchronous (unknown but bounded link delay) unidirectional ring of oblivious finite automata processors, using constant size messages.

Even though token management in a uni-directional case is possible, leader election is not possible. That is, in the full version of the paper we show that there is no randomized *leader election* algorithm on an asynchronous (bounded delay) *unidirectional*ring of oblivious finite automata processors, using constant size messages. We postpone the details to the final version.

Acknowledgments

We are happy to thank Yehuda Afek, Baruch Awerbuch, Amotz Bar-Noy, Shay Kutten, Sergio Rajsbaum, and Baruch Schieber for helpful discussions.

References

[AAHK86] Abrahamson, Adler, Higham, and Kirkpatrick, Probabilistic solitude verification on a ring. *Proc. 5th ACM Symp. on Principles of Distributed. Computing* (1986).

[AB89] Y. Afek and G.M. Brown, Self-stabilization over unreliable communication media, manuscript. (Previous version appeared as: Self-stabilization of the alternating-bit protocol, *Proc. 8th IEEE Symp. on Reliable Distributed Systems* (1989), 10–12.

[AKY90] Y. Afek, S. Kutten, and M. Yung, Memory-efficient self-stabilization on general networks, *Proc. 4th International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, Vol 486, Springer-Verlag, New York, (1989), 12–28.

[AN90] N. Alon and M. Naor, Coin-flipping games immune against linear-sized coalition, *Proc. 31st IEEE Symp. on Foundations of Computer Science* (1990), 46–54.

[Ang80] D. Angluin, Local and global properties in networks of processors, *Proc. 12th ACM Symp. on Theory of Computing* (1980), 82–93.

[ASW85] C. Attiya, M. Snir and M. Warmuth, Computing on an anonymous ring, *Proc. 4th ACM Symp. on Principles of Distributed. Computing* (1985), 196–203. (see also: *Journal of the ACM* **35**(4) (1988), 845–875.

[APV91] B. Awerbuch, B. Patt, and G. Varghese, Self-stabilization by local checking and correction *Proc. 33nd IEEE Symp. on Foundations of Computer Science* (1991), 268–277.

[AV91] B. Awerbuch, and G. Varghese, Distributed program checking: a paradigm for building self-stabilizing distributed protocols, *Proc. 33nd IEEE Symp. on Foundations of Computer Science* (1991), 258–267.

[BGW89] G.M. Brown, M.G Gouda, and C.L. Wu, Token systems that self-stabilize, *IEEE Transactions on Computers* **38**(6) (1989), 845–852.

[BP88] J.E. Burns and J. Pachl, Uniform self-stabilizing rings, *Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science*, Vol 319, Springer-Verlag, New York, (1988), 391–400. (See also: *ACM TOPLAS* **11**(2) (1989), 330–344.)

[BCK+83] W. Bux, F.H. Closs, K. Kümmerle, H.J. Keller and H.R. Müller, Architecture and design of a reliable token-ring network, *IEEE J. on Selected Areas in Comm.*, **1**(5), (1983), 756–765.

[CV86] R. Cole and U. Vishkin, Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms, *Proc. 18th ACM Symp. on Theory of Computing* (1986), 206–219.

[CO90] I. Cidon and Y. Ofek, MetaRing - A full-duplex ring with fairness and spatial reuse, *Proc INFOCOM'90* 969–981. (Also: Accepted to IEEE Tras. on Comm.).

[Dij74] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, *Communications of the ACM* **17**(11) (1974), 643–644.

[DIM91] S. Dolev, A. Israeli and S. Moran, Resource bounds for self stabilizing message driven protocols, *Proc. 10th ACM Symp. on Principles of Distributed. Computing* (1991), 281–293.

[FL84] G.N. Frederickson and N. Lynch, The impact of synchronous communication on the problem of electing a leade*Proc. 16th ACM Symp. on Theory of Computing* (1984), 493–503. (see also: *J. of the ACM* **34**(1) (1987), 98–115.)

[FS86] G.N. Frederickson and N. Santoro, Breaking Symmetry in synchronous networks, *Proc. 1st AWOC Lecture Notes in Computer Science*, Vol 227, Springer-Verlag, New York, (1986), 82–93.

[Her90] T. Herman, Probabilistic self-stabilization, *Information Processing Letters* **35** (1990), 63–67.

[HN88] A. Hopper and R. M. Needham, The Cambridge fast ring networking system, *IEEE Transactions on Computers*, **37**(10) (1988), 1214–1223.

[IJ90a] A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, *Proc. 9th ACM Symp. on Principles of Distributed. Computing* (1990), 119–130.

[IJ90b] A. Israeli and M. Jalfon, Self-Stabilizing Ring Orientation, *Proc. 4th International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, Vol 486, Springer-Verlag, New York, (1990), 1–14.

[IR81] A. Itai and M. Rodeh, Symmetry breaking in distributive networks, *Proc. 22nd IEEE Symp. on Foundations of Computer Science* (1981), 150–159.

[It90] A. Itai, On the power needed to elect a leader *Proc. 4th International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, Vol 486, Springer-Verlag, New York, (1989), 29–40.

[KP90] S. Katz and K.J. Perry, Self-stabilizing extensions for message-passing systems, *Proc. 9th ACM Symp. on Principles of Distributed. Computing* (1990), 91–101.

[LTG90] A. A. Lazar, A. T. Temple and R. Gidron, MAGNET II: A metropolitan area network based on asynchronous time sharing, *IEEE J. on Selected Areas in Comm.*, **8**(8), (1990), 1582–1594.

[LL90] L. Lamport and N.A. Lynch, Distributed computing models and methods, *Handbook on Theoretical Computer Science*, North-Holland, 1990, 1159–1199.

[MZ86] Y. Mansour and S. Zaks, On the bit complexity of distributed computations in a ring with a leader, *Proc. 5th ACM Symp. on Principles of Distributed. Computing* (1986), 131–140.

[Mis83] J. Misra, Detecting termination of distributed computations using markers, *Proc. 2nd ACM Symp. on Principles of Distributed. Computing* (1983), 290–294.

[OMS89] H. Ohnishi, N. Morita, and S. Suzuki, ATM ring protocol and performance, *Proc. ICC'89, 13.1*, (1989), 394–398.

[OY90] Y. Ofek and M. Yung, Principles for high-speed network control *Proc. 9th ACM Symp. on Principles of Distributed. Computing*, (1990).

[PKR82] J. Pachl, E. Korach and D. Rotem, A technique for proving lower bounds for distributed maximum-finding algorithms, *Proc. 14th ACM Symp. on Theory of Computing* (1982), 378–382.

[R86] F. E. Ross, FDDI - a Tutorial, *IEEE Communication Magazine*, **24**(5), (1986), 10–17.

[RL81] M. O. Rabin and D. Lehmann, On the advantage of free choice: a symmetric solution to the dining philosophers problem, *Proc. ACM Symp. on Principles of Programming Languages.* (1981), 133–138.

[SS89] B. Schieber and M. Snir, Calling names on nameless networks, *Proc. 8th ACM Symp. on Principles of Distributed. Computing.*

[STT89] P. Spirakis, B. Tampakas and A. Tsiolis, Symmetry breaking in asynchronous rings in O(n) messages, *Proc. 3d International Workshop on Distributed Algorithms, Lecture Notes in Computer Science*, Vol 393, Springer-Verlag, New York, (1989), 233–241.

[V84] P. Vitanyi, Distributed elections in an Archimedian ring of processors, *Proc. 16th ACM Symp. on Theory of Computing* (1984), 542–547.