Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2011-022 April 14, 2011

Partial Reversal Acyclicity

Tsvetomira Radeva and Nancy Lynch

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Partial Reversal Acyclicity

Tsvetomira Radeva, Nancy Lynch
MIT, Cambridge, MA
{radeva, lynch}@csail.mit.edu

Abstract

Partial Reversal (PR) is a link reversal algorithm which ensures that the underlying graph structure is destination-oriented and acyclic. These properties of PR make it useful in routing protocols and algorithms for solving leader election and mutual exclusion. While proofs exist to establish the acyclicity property of PR, they rely on assigning labels to either the nodes or the edges in the graph. In this work we present simpler direct proof of the acyclicity property of partial reversal without using any external or dynamic labeling mechanism. First, we provide a simple variant of the PR algorithm, and show that it maintains acyclicity. Next, we present a binary relation which maps the original PR algorithm to the new algorithm, and finally, we conclude that the acyclicity proof applies to the original PR algorithm as well.

1 Introduction

The goal of link reversal algorithms is to ensure that all nodes in a directed acyclic graph (DAG) have a path to a particular destination node or nodes. Link reversal algorithms were first introduced by Gafni and Bertsekas in [4] as a way of providing an efficient graph structure for routing. They are also the main subject of [6] where Welch and Walter present multiple link reversal algorithms for solving problems such as routing, leader election and mutual exclusion.

Two particular link reversal algorithms proposed in [4], and also studied in [6], are *Full Reversal (FR)* and *Partial Reversal (PR)*. In both algorithms the initial graph is a DAG with a single destination node, where nodes do not necessarily have a path to the destination. The goal of the algorithms is to reverse particular edges in the graph so that each node has a directed path to the destination. A node executes a step of either algorithm only if it is a sink (all its incident edges are incoming). The destination node never takes any steps. In FR when a node is a sink it reverses all of its incident edges. In PR, each node keeps a list of the edges reversed by its neighbors the previous time they took a step. When a node is a sink, it reverses only the edges which are *not* in the list, and then clears the list. In other words, while in FR nodes always reverse all their edges, in PR it is possible to reverse fewer edges.

Even though PR seems to be much more efficient than FR, the worst case running time for both algorithms is the same. We measure the efficiency of both algorithms by comparing the total number of reversals performed by all nodes. For FR, the authors of [6], following an approach from [1] and [2], prove a tight bound of $\Theta(n_b^2)$ worst case total number of reversals, where n_b is the number of nodes that have no path to the destination initially. In [6], they also show that the same tight bound applies to PR. Since such a conclusion is surprising and counter-intuitive, Charron-Bost et al. [3] apply a game theoretical approach in showing that PR is more efficient than FR. The authors conclude that the strategy of FR is a Nash equilibrium, but it has the largest social cost among all Nash equilibria, while the strategy of PR is not necessarily a Nash equilibrium, but if it is, it achieves a global optimum and has the minimum social cost.

One of the key requirements of most link reversal algorithms is to always preserve the acyclicity of the underlying graph, because the presence of cycles is not desired in most of the applications (such as routing) of such algorithms. It is easy to show that FR maintains a graph with no cycles. To do so, suppose in contradiction that a cycle exists in some state of the execution of FR, and consider the last node which takes a step before the cycle is created. Since nodes always reverse all of their incident edges in FR, the last node to take a step results in having all outgoing edges after it takes that step. However, this is a contradiction to the assumption that a cycle exists.

In the case of PR, there exist a couple of proofs that the algorithm does not create any cycles. In the original Gafni and Bertsekas paper [4], each node is assigned a triple of integers, and each edge is directed from a node with a lexicographically larger value of the triple to a node with a smaller such value. They prove such an assignment exists, which forms a total order on the nodes, and therefore, no cycles exist in the graph. Another proof of the acyclicity property of PR is presented in [6], which uses a generalized algorithm – Binary Link Labels (BLL) – and provides conditions under which BLL maintains acyclicity. The BLL algorithm assumes that each edge in the graph is labeled, and reverses edges based on these labels. The condition under which BLL maintains acyclicity involves certain global properties of the number and type of edges in the graph. Finally, the authors show that PR is a special case of BLL, in which the acyclicity condition is satisfied.

In this paper we present a new simpler proof of the acyclicity property of PR, which does not use any mechanism of labeling nodes or edges. First, we introduce a new version of the original PR algorithm. In the original PR algorithm, each node keeps a dynamic list of neighbors which determines the set of edges to be reversed. However, if we observe the sets of edges reversed at each step, we notice that edges corresponding to the same sets of neighbors are reversed at every other step. Therefore, our new algorithm uses only the original sets of incoming and outgoing neighbors of each node, and reverses the corresponding set of edges, alternating between the two. Having such a simpler and more static algorithm, it is easier to prove that no cycles exist at any point of the execution. Our acyclicity proof relies on a few invariants based on the number of steps nodes have taken, and unlike existing proofs, does not use any

labeling of the nodes or edges of the graph.

Finally, since the new algorithm seems to be very similar to the original one, we provide a simulation relation from the original algorithm to the new one, to formally show a mapping between the two. The simulation relation establishes a correspondence between the different lists in the two algorithms, and concludes that for every step of the original algorithm, there exists a sequence of steps in the new algorithm, so that both algorithms result in the same directions of the edges in the graph. Additionally, such a relation shows that our new acyclicity proof carries over to the original PR algorithm.

The rest of this paper is organized as follows: Section 2 describes how we model the system; Section 3 presents the original PR algorithm in more detail, and shows some useful properties of the algorithm; Section 4 describes our new algorithm and some of its properties, including the acyclicity proof; Section 5 provides a simulation relation between the two algorithms, and presents the main conclusion that PR maintains acyclicity using our new proof; Section 6 summarizes our results.

2 System Model

We model the system as an undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. The graph has a single predetermined destination node $D \in V$. The set of neighbors of a particular node u in G is defined as $nbrs_u$. Since no nodes and edges are added or removed from the graph, G is constant throughout the execution of the algorithm. Let a directed version of G be denoted as $G' = (V, E')$, such that for a given edge $\{u, v\} \in E$ either $(u, v) \in E'$ or $(v, u) \in E'$, but not both. We also define an initial graph G'_{init} which represents the initial directed graph. Assuming G'_{init} is fixed, let $in\text{-}nbrs_u$ and $out\text{-}nbrs_u$ be the sets of nodes corresponding to incoming and outgoing edges of any node u in G'_{init}. Note that $nbrs_u$ is defined as the set of neighbors of u in G (the undirected graph), while $in\text{-}nbrs_u$ and $out\text{-}nbrs_u$ are defined with respect to G'_{init} (the initial directed graph). None of the sets $nbrs_u$, $in\text{-}nbrs_u$, and $out\text{-}nbrs_u$ changes throughout the execution of the algorithm, and so $nbrs_u = in\text{-}nbrs_u \cup out\text{-}nbrs_u$ at any state of the system.

3 Original Algorithm

3.1 Algorithm Description

In this section we present the original PR algorithm [4] and express it as an I/O automaton (PR).

The entire system is modeled as a single I/O Automaton (as described in [5]) with a single set of actions – $reverse(S)$. The set S represents all nodes that are taking a step together, where each one of these nodes reverses a set of edges to its neighbors. The destination node D does not reverse its incident edges, and so it is never in S. For each node u, PR has a state variable $list[u]$

which contains all the neighbors of u which took a step since the last time u took a step. Initially $list[u]$ is empty. Additionally, the PR automaton has a state variable, $dir[u, v]$, one for each ordered pair (u, v), which represents the direction of the edge between u and v from u's perspective.

Algorithm 1 PR automaton

Signature:

$reverse(S)$, $S \subseteq V$, $S \neq \emptyset$, $D \notin S$

States:

for each u, v where $\{u, v\} \in E$:
  $dir[u, v] \in \{in, out\}$, initially in if $v \in in\text{-}nbrs_u$ or
    out if $v \in out\text{-}nbrs_u$
  $dir[v, u] \in \{in, out\}$, initially in if $u \in in\text{-}nbrs_v$ or
    out if $u \in out\text{-}nbrs_v$
for each u, $list[u]$, a set of nodes $W \subseteq nbrs_u$, initially empty

Transitions:

$reverse(S)$
  Precondition:
    for each $u \in S$
      for each $v \in nbrs_u$, $dir[u, v] = in$
  Effect:
    for each $u \in S$
      if $list[u] \neq nbrs_u$ then
        for each $v \in nbrs_u \setminus list[u]$
          $dir[u, v] := out$
          $dir[v, u] := in$
          $list[v] := list[v] \cup \{u\}$
      else
        for each $v \in nbrs_u$
          $dir[u, v] := out$
          $dir[v, u] := in$
          $list[v] := list[v] \cup \{u\}$
      $list[u] := \emptyset$

Tasks:

$\{reverse(S), S \subseteq V, S \neq \emptyset, D \notin S\}$

The only precondition for the $reverse(S)$ action is that all nodes in S are sinks. The effect of the reversal is that the edge between u and each neighbor of u *not* in $list[u]$ is reversed (from *in* to *out*). However, if $list[u]$ contains all neighbors of u, then the edges to all neighbors are reversed. Also, each neighbor v of u that has its edge to u reversed, adds u to $list[v]$. Finally, after reversing the particular edges, u empties $list[u]$.

3.2 Properties

The following invariants establish some basic properties of the algorithm above. Invariant 3.1 ensures the consistency of edge directions with respect to both endpoints of the edge. Invariant 3.2 shows the possible contents of $list[u]$ for any node u. Corollary 3.3 follows directly from Invariant 3.2 concluding that if u is not a sink, then $list[u]$ is a subset of either $in\text{-}nbrs_u$ or $out\text{-}nbrs_u$. Corollary 3.4 states that $list[u]$ must be equal to either the set of $in\text{-}nbrs_u$ or the set of $out\text{-}nbrs_u$, whenever u is a sink.

Invariant 3.1 *In every reachable state of PR, for each u and v where $\{u, v\} \in E$, $dir[u, v] = in$ iff $dir[v, u] = out$.*

Proof Initially, each $dir[u, v]$ variable is set according to $in\text{-}nbrs_u$, $out\text{-}nbrs_u$, $in\text{-}nbrs_v$, and $out\text{-}nbrs_v$, so if the edge $\{u, v\}$ is directed from v to u, then $dir[u, v] = in$, and $dir[v, u] = out$.

Assuming this property is true in some state s, we now show that it remains true in any state s' that is reachable from s in a single step of the algorithm. If neither u nor v reverses the edge between them, then $dir[u, v]$ and $dir[v, u]$ remain the same, so the invariant remains correct in s'. If u takes a step and reverses its edge to v, then $s.dir[u, v] = in$ because u is a sink in s. Therefore, $s.dir[v, u] = out$. When u executes a step of the algorithm, it sets $s'.dir[u, v] = out$ and $s'.dir[v, u] = in$. Therefore, the property remains true in s'. If v takes a step in s, then $s.dir[v, u] = in$. When v reverses the edge, it sets $s'.dir[v, u] = out$ and $s'.dir[u, v] = in$, and the property remains true. ■

The following invariant states that at any state of the system $list[u]$ consists of either only $in\text{-}nbrs_u$ or $out\text{-}nbrs_u$. Also, since all nodes in the list already reversed their edges back to u, all edges corresponding to nodes in the list are incoming. We also show that if the list consists of $in\text{-}nbrs_u$ ($out\text{-}nbrs_u$), then all $out\text{-}nbrs_u$ ($in\text{-}nbrs_u$) have incoming edges to u.

Invariant 3.2 *In every reachable state of PR, for each node u, exactly one of the following is true:*

1. *For each $w \in out\text{-}nbrs_u$, $dir[u, w] = in$ and $list[u] = \{v | v \in in\text{-}nbrs_u$ and $dir[u, v] = in\}$.*
2. *For each $w \in in\text{-}nbrs_u$, $dir[u, w] = in$ and $list[u] = \{v | v \in out\text{-}nbrs_u$ and $dir[u, v] = in\}$.*

Proof (by induction on the number r of completed steps)

Initially, the list is empty. Part 2 is true because all $in\text{-}nbrs_u$ initially have incoming edges to u, and also because no $out\text{-}nbrs_u$ initially have incoming edges to u. We also need to show that part 1 is false. If u is a source, part 1 does not hold because the direction of the edges to all $out\text{-}nbrs_u$ is out. If u is not a source, part 1 is false because $list[u]$ is empty initially.

Assuming the property is true after r steps, we now show that it is true after $r+1$ steps. Let the state of the system after r steps be s, and the state of the system after $r+1$ steps be s'.

Case 1: The $r+1$'st step of the execution includes a step of u.

Case 1.1: $s.list[u] \neq nbrs_u$ and part 1 is true in s.

We show that part 2 is true in s', and part 1 is false in s'.

Since part 1 is true in s, by the inductive hypothesis $s.list[u] = \{v | v \in \textit{in-nbrs}_u \text{ and } dir[u,v] = in\}$. Also, u is a sink in s, so all edges to nodes in $\textit{in-nbrs}_u$ are incoming. Therefore, $s.list[u] = \textit{in-nbrs}_u$. Because $s.list[u] \neq nbrs_u$, when u takes a step, it reverses $nbrs_u \setminus \textit{in-nbrs}_u = \textit{out-nbrs}_u$, and so all nodes in $\textit{in-nbrs}_u$ still have incoming edges to u in s'. Also, $s'.list[u] = \emptyset$, and part 2 is true because no $\textit{out-nbrs}_u$ have incoming edges to u in s'. Moreover, part 1 is not true in s' because u has outgoing edges to all nodes in $\textit{out-nbrs}_u$, and since $s.list[u] \neq nbrs_u$ and $s.list[u] = \textit{in-nbrs}_u$, it follows that that $\textit{out-nbrs}_u \neq \emptyset$.

Case 1.2: $s.list[u] \neq nbrs_u$ and part 2 is true in s.

We show that part 1 is true in s', and part 2 is false in s'.

Since part 2 is true in s, by the inductive hypothesis $s.list[u] = \{v | v \in \textit{out-nbrs}_u \text{ and } dir[u,v] = in\}$. Also, u is a sink in s, so all edges to nodes in $\textit{out-nbrs}_u$ are incoming. Therefore, $s.list[u] = \textit{out-nbrs}_u$. Because $s.list[u] \neq nbrs_u$, when u takes a step, it reverses $nbrs_u \setminus \textit{out-nbrs}_u = \textit{in-nbrs}_u$, and so all nodes in $\textit{out-nbrs}_u$ still have incoming edges to u in s'. Also, $s'.list[u] = \emptyset$, and part 1 is true because no $\textit{in-nbrs}_u$ have incoming edges to u in s'. Moreover, part 2 is not true in s' because u has outgoing edges to all nodes in $\textit{in-nbrs}_u$, and since $s.list[u] \neq nbrs_u$ and $s.list[u] = \textit{out-nbrs}_u$, it follows that that $\textit{in-nbrs}_u \neq \emptyset$.

Case 1.3 $s.list[u] = nbrs_u$ and part 1 is true in s.

We show that part 1 true in s', and part 2 is false in s'.

Since part 1 is true in s, by the inductive hypothesis $s.list[u] = \{v | v \in \textit{in-nbrs}_u \text{ and } dir[u,v] = in\}$. Also, u is a sink in s, so all edges to nodes in $\textit{in-nbrs}_u$ are incoming Therefore, $s.list[u] = \textit{in-nbrs}_u$. Because $s.list[u] = nbrs_u$, when u takes a step, it reverses $\textit{in-nbrs}_u$, so that all nodes in $\textit{in-nbrs}_u$ have outgoing edges from u in s'. Therefore, part 2 is false because its first condition is false. Moreover, the first condition of part 1 is satisfied because $\textit{out-nbrs}_u = \emptyset$. Additionally, no $\textit{in-nbrs}_u$ have incoming edges to u, so $s'.list[u] = \emptyset$, and thus part 1 is true.

Case 1.4 $s.list[u] = nbrs_u$ and part 2 is true in s.

We show that part 2 is true in s', and part 1 is false in s'.

Since part 2 is true in s, by the inductive hypothesis $s.list[u] = \{v | v \in \textit{out-nbrs}_u \text{ and } dir[u,v] = in\}$. Also, u is a sink in s, so all edges to nodes in $\textit{out-nbrs}_u$ are incoming. Therefore, $s.list[u] = \textit{out-nbrs}_u$. Because $s.list[u] = nbrs_u$, when u takes a step, it reverses $\textit{out-nbrs}_u$, so that all nodes in $\textit{out-nbrs}_u$ have outgoing edges from u in s'. Therefore, part 1 is false because its first condition is false. Moreover, the first condition of part 2 is satisfied because $\textit{in-nbrs}_u = \emptyset$. Additionally, no $\textit{out-nbrs}_u$ have incoming edges to u, so $s'.list[u] = \emptyset$, and thus part 2 is true.

Case 2: The $r+1$'st step of the execution includes a step of some node $v \in nbrs_u$.

Note that Case 2 is disjoint from Case 1 because no two neighboring nodes can be sinks at the same time. Let $T = nbrs_u \cap S$, that is, T is the set of neighbors v of u such that the $r+1$'st step of the execution includes a step of v. By the definition of the case, $T \neq \emptyset$.

All neighbors of u that take a step in s (all nodes in T) are added to $s'.list[u]$. Let v be an arbitrary neighbor of u in T. In s, the edge between u and v must be from u to v, while in s' the direction of the edge must be from v to u.

Case 2.1: Part 1 is true in s.

We show that part 1 is true in s', and part 2 is false in s'.

By part 1, $s.list[u] \subseteq in\text{-}nbrs_u$ and all nodes in $out\text{-}nbrs_u$ have incoming edges to u. Therefore, $v \notin out\text{-}nbrs_u$, and so $v \in in\text{-}nbrs_u$. When v is added to $list[u]$, it is true that $s'.list[u] = \{v | v \in in\text{-}nbrs_u \text{ and } dir[u,v] = in\}$. No edges to $out\text{-}nbrs_u$ are reversed in this step, so part 1 is satisfied. Part 2 is not true in s' because $s'.list[u]$ contains at least one node, $v \in in\text{-}nbrs_u$, which was just added to $s'.list[u]$ in step $r+1$.

Case 2.2: Part 2 is true in s.

We show that part 2 is true in s', and part 1 is false in s'.

By part 2, $s.list[u] \subseteq out\text{-}nbrs_u$ and all nodes in $in\text{-}nbrs_u$ have incoming edges to u. Therefore, $v \notin in\text{-}nbrs_u$, and so $v \in out\text{-}nbrs_u$. When v is added to $list[u]$, it is true that $s'.list[u] = \{v | v \in out\text{-}nbrs_u \text{ and } dir[u,v] = in\}$. No edges to $in\text{-}nbrs_u$ are reversed in this step, so part 2 is satisfied. Part 1 is not true in s' because $s'.list[u]$ contains at least one node, $v \in out\text{-}nbrs_u$, which was just added to $s'.list[u]$ in step $r+1$.

Case 3: Neither u nor any $v \in nbrs_u$ takes a step during the $r+1$'st step of the execution.

Since only u or its neighbors can change $list[u]$, in this case $s.list[u] = s'.list[u]$. Moreover, none of u's incident edges are reversed during this step, so the property remains true.

■

Corollary 3.3 *In any reachable state of PR, for any node u, $list[u] \subseteq in\text{-}nbrs_u$ or $list[u] \subseteq out\text{-}nbrs_u$ (or both if $list[u] = \emptyset$).*

Corollary 3.4 *In any reachable state of PR, if u is a sink, then $list[u] = in\text{-}nbrs_u$ or $list[u] = out\text{-}nbrs_u$.*

4 New Algorithm

4.1 Algorithm Description

In this algorithm, nodes use only the initial *in-nbrs* and *out-nbrs* sets to determine which edges to reverse in each step. Whenever a node is a sink, it reverses the edges corresponding to either its *in-nbrs* or *out-nbrs* set, alternating between the two. In order to determine which set is about to be reversed, each node keeps track of the parity of the number of steps taken so far. If the node has taken an even number of steps, then it reverses the set of *in-nbrs*; if it has

taken an odd number of steps then the set of *out-nbrs* is reversed. Initially, nodes have taken zero steps, so they reverse their *in-nbrs* the first time they take a step.

Algorithm 2 $NewPR$ automaton

Signature:
$reverse(u)$, $u \in V$, $u \neq D$

States:
for each u, v where $\{u, v\} \in E$:
$dir[u, v] \in \{in, out\}$, initially in if $v \in in\text{-}nbrs_u$ or out if $v \in out\text{-}nbrs_u$
$dir[v, u] \in \{in, out\}$, initially in if $u \in in\text{-}nbrs_v$ or out if $u \in out\text{-}nbrs_v$
for each node u, $count[u]$, integer, initially 0

Derived State:
for each node u, $parity[u] \in \{even, odd\}$, $even$ if $count[u]$ is even
odd if $count[u]$ is odd

Transitions:
$reverse(u)$
Precondition:
for each $v \in nbrs_u$, $dir[u, v] = in$
Effect:
if $parity[u] = even$ then
for each $v \in in\text{-}nbrs_u$
$dir[u, v] := out$
$dir[v, u] := in$
else
for each $v \in out\text{-}nbrs_u$
$dir[u, v] := out$
$dir[v, u] := in$
$count[u] := count[u] + 1$

Tasks:
$\{reverse(u), u \in V, u \neq D\}$

The entire system is modeled as a single I/O Automaton with a single set of actions – $reverse(u)$ – where u is any node in V, which is currently a sink. The destination node never reverses any of its incident edges, so $u \neq D$. Moreover, associated with each node are two variables: $dir[u, v]$ which represents the direction of the edge between nodes u and v, and history variable $count[u]$ which keeps track of the number of steps u has taken so far. There is also has a derived variable $parity[u]$, which is a function of $count[u]$ that represents its parity; it is used to keep track of which set of neighbors is to be reversed next.

The precondition for a node u to perform a $reverse(u)$ action is that it is

a sink. The effect of the reversal is that depending on the value of $parity[u]$, either the edges corresponding to nodes in $in\text{-}nbrs_u$ or $out\text{-}nbrs_u$ are reversed. Also, $count[u]$ is incremented, which results in flipping the parity bit.

Note that it is possible that in the $reverse(u)$ action u does not reverse any edges because either $in\text{-}nbrs_u = \emptyset$ or $out\text{-}nbrs_u = \emptyset$. This case occurs only when nodes are initially sinks or sources. When such an action is performed, all u does is increment the step counter (flip the parity bit) without reversing any edges. In this case u remains a sink but now the parity has the correct value, so u can perform a regular $reverse(u)$ action the next time it takes a step.

It is important to notice the main differences between PR and $NewPR$:

- In PR, each node keeps one list of neighbors which changes as edges are reversed, while in $NewPR$ nodes have two constant lists, *in-nbrs* and *out-nbrs*, and a *parity* bit to alternate between the lists.
- In PR, there are two possible ways nodes reverse their edges (depending on whether all neighbors are in the list or not), and so whenever a node is a sink it reverses some edges and empties the list. In $NewPR$, however, it is possible that a node is a sink but the parity does not have the right value to reverse the corresponding set of edges. This happens to nodes that are originally sinks or sources. During this "dummy" step, a node does not reverse any edges but only increments its step count, so the next time it takes a step, the parity corresponds to the list of edges to be reversed. This extra step in $NewPR$ causes it to incur a greater cost in certain situations, compared to PR.
- In PR, a set of nodes takes a step at once, while in $NewPR$ only one node at a time can take a step.

It is important to note that PR keeps a dynamic list of nodes in order to determine which edges to reverse, while $NewPR$ is a lot more static because it always reverses one of two constant sets. We believe that describing the algorithm in such a way simplifies it and makes it easier to understand. Moreover, the dummy step in $NewPR$ helps treat all nodes equivalently and thus makes it possible to state nice invariants based on the number of steps nodes have taken. On the other hand, the increased number of steps, and the restriction of only one node taking a step at a time, affect the complexity of the algorithm, but we are not concerned with this issue in this paper.

4.2 Acyclicity Property

The proof of the acyclicity property of $NewPR$ consists of Invariant 4.1 and Invariant 4.2, which are then combined into Theorem 4.3 concluding that PR maintains acyclicity.

Since the input to the PR algorithm is a DAG, we can embed it in a plane, ensuring all edges are initially directed from left to right. Therefore, for each node u all edges associated with nodes in $in\text{-}nbrs_u$ are to the left of u, and all nodes associated with edges in $out\text{-}nbrs_u$ are to the right of u.

Invariant 4.1 states that if the *parity* of two neighboring nodes is the same, then we can determine whether the edge between them is directed from left to right, or right to left.

Invariant 4.1 *In any reachable state, if u and v are neighbors, then:*

(a) If $parity[u] = parity[v] = even$, then the edge $\{u, v\}$ is directed from left to right.

(b) If $parity[u] = parity[v] = odd$, then the edge $\{u, v\}$ is directed from right to left.

Proof (by induction on the number r of total number of steps taken by all nodes)

In the initial state $parity[u] = parity[v] = even$. Part (b) is vacuously true, and part (a) is true because initially all edges are directed from left to right.

Assume both properties are true after r steps. Let the state of the system after r steps be s. We need to show that the properties are true after $r + 1$ steps. Let the state of the system after $r + 1$ steps be s'.

Note that an arbitrary node can take the $r + 1$'st step. If neither u nor v takes a step, then both properties remain true. Therefore, we are concerned only with cases in which either u or v takes a step. Since the two properties are symmetric with respect to u and v, without loss of generality, assume u is taking the $r + 1$'st step.

If $s'.parity[u] = s'.parity[v] = even$, part (b) is vacuously true, so we show part (a). Since u takes a step, then it must be a sink in s, so the edge $\{u, v\}$ is directed from v to u in s. Since u takes the $r+1$'st step, then $s.parity[u] = odd$.

Since $s.parity[u] = odd$, by the second case of the code of the $reverse(u)$ action, the edges corresponding to $out\text{-}nbrs_u$ (to the right of u) are reversed. If v is to the right of u, then the edge $\{u, v\}$ is reversed and is now directed from left to right in s'. If v is to the left of u, the edge $\{u, v\}$ is not reversed and remains directed from left to right.

Similarly, for the proof of part (b), we assume $s'.parity[u] = s'.parity[v] = odd$, which implies that part (a) is vacuous, and we use the same arguments to show that part (b) is satisfied. ■

Invariant 4.2 has four parts, establishing different properties of the number of steps that nodes have taken. Part (a) gives a range of the possible number of steps of a node v, given the number of steps its neighbor, node u, has taken. Parts (b) and (c) show two possible cases in which it can be concluded that two neighboring nodes have taken the same number of steps. Part (d) states that if one node has taken strictly more steps that its neighbor, then the edge between them is directed from the node which has taken more steps to the node which has taken fewer steps. Combined together the invariants 4.1 and 4.2 give us a way of using the number of steps and directions of edges to show that it is not possible to create a cycle in the graph.

Invariant 4.2 *In any reachable state, if u and v are neighbors, then:*

(a) If $count[u] = n$*, then* $count[v] \in \{n-1, n, n+1\}$*.*

(b) If $count[u] = n$*, where* n *is odd, and* v *is to the right of* u*, then* $count[v] = n$*.*

(c) If $count[u] = n$*, where* n *is even, and* v *is to the left of* u*, then* $count[v] = n$*.*

(d) If $count[u] > count[v]$*, then the edge* $\{u, v\}$ *is directed from* u *to* v*.*

Proof (by induction on the number r of total number of steps taken by all nodes)

In the initial configuration no node has taken any steps yet, so $count[u] = count[v] = 0$. Therefore, all four parts are true initially.

Suppose all properties are true after r steps. Let the state of the system after r steps be s. We need to show that all properties are true after $r+1$ steps. Let the state of the system after $r+1$ steps be s'.

Note that an arbitrary node can take the $r+1$'st step. If neither u nor v takes a step, then all properties remain true. Therefore, we are concerned only with cases in which either u or v takes a step.

Assume $s'.count[u] = k$.

Case 1: u takes the $r+1$'st step. Therefore, u is a sink in s and the edge $\{u, v\}$ is directed from v to u. Also, $s.count[u] = k-1$, and by the inductive hypothesis part (a), $s'.count[v] = s.count[v] \in \{k-2, k-1, k\}$. By the inductive hypothesis part (d), $s.count[v] \geq s.count[u]$, and therefore $s'.count[v] = s.count[v] \in \{k-1, k\}$.

Part (a): $s'.count[u] = k$, and so it is true that $s'.count[v] \in \{k-1, k, k+1\}$.

Part (b): Assume k is odd, and v is to the right of u in s'. If $s.count[v] = k-1$, then $s.count[u] = s.count[v] = k-1$, which is even, so by Invariant 4.1 (a), the edge $\{u, v\}$ is directed from u to v, a contradiction. So $s.count[v] = s'.count[v] = k$.

Part (c): The proof for part (c) is analogous to that of part (b). By Invariant 4.1 (b), $s.count[v] \neq k-1$. Therefore, $s'.count[v] = k$.

Part (d): Assume $s'.count[u] > s'.count[v]$, so $s'.count[v] \neq s'.count[u]$. If k is odd, by part (b) applied to s', v must be to the left of u. Also, since k is odd, $k-1$ is even, so when u takes a step, it reverses its left edges. Thus, the edge $\{u, v\}$ is reversed and is now directed from u to v. Similarly, if k is even, part (c) applied to s' implies that v must be to the right of u. Since $k-1$ is odd when u takes a step, it reverses all the edges to its right, and so the edge $\{u, v\}$ is now directed from u to v.

Case 2: v takes the $r+1$'st step. Therefore, v is a sink in s, so the edge $\{u, v\}$ is directed from u to v. Also, $s.count[u] = s'.count[u] = k$, and by the inductive hypothesis of part (a), $s.count[v] \in \{k-1, k, k+1\}$. If $s.count[v] = k+1$, then $s.count[v] > s.count[u]$, and by the inductive hypothesis part (d) the edge $\{u, v\}$ is directed from v to u, a contradiction. Therefore, $s.count[v] \in \{k-1, k\}$, and so $s'.count[v] \in \{k, k+1\}$.

Part (a): From the facts above it follows that $s'.count[v] \in \{k, k+1\}$.

Part (b): Assume k is odd, and v is to the right of u in s'. If $s.count[v] = k$, then $s.count[u] = s.count[v] = k$, which is odd, so by Invariant 4.1 (b), the

edge $\{u, v\}$ is directed from v to u, a contradiction. So, $s.count[v] = k - 1$, and therefore $s'.count[v] = k$.

Part (c): The proof for part (c) is analogous to part (b). By Invariant 4.1 (a), $s.count[v] \neq k$. Therefore, $s.count[v] = k - 1$, and $s'.count[v] = k$.

Part (d): Assume $s'.count[u] > s'.count[v]$. By part (a) applied to s', $s'.count[v] \in \{k - 1, k, k + 1\}$, so $s'.count[v] = k - 1$. Since v takes a step in r, then $s.count[v] = k-2$. This is a contradiction to the inductive hypothesis of part (a), and therefore it is not possible for v to take the $r + 1$'st step in this case.

■

The next theorem uses Invariant 4.1 and part (d) of Invariant 4.2 to show that nodes in a circuit can never form a cycle because of the relation between the edge directions and the number of steps the nodes have taken.

Let $s.G' = (V, E')$ be the directed graph in state s, where V is the same set of nodes as in the undirected graph G, and E' is the set of directed edges determined using the *dir* variables as follows. The edge between any pair of nodes u and v is directed from u to v if and only if $dir[u, v] = out$.

Theorem 4.3 *In any reachable state s of the execution of $NewPR$ the underlying directed graph $s.G'$ is acyclic.*

Proof Suppose in contradiction that there exists a cycle in some reachable state s of the system. Let $s.G'$ be the directed graph in state s. Therefore, there is a sequence of nodes: $u, v_1, v_2, \ldots, v_n, u$ such that the edges between these nodes are directed from u to v_1, from v_n to u, and from v_i to v_{i+1}, for all $1 \leq i < n$. By Invariant 4.2 (d) the number of steps of the nodes in the sequence is non-increasing: $s.count[u] \geq s.count[v_1] \geq s.count[v_2] \geq \ldots \geq s.count[v_n] \geq s.count[u]$. Since node $s.count[u]$ is both in the beginning and the end of the sequence, it follows that $s.count[u] = s.count[v_1] = s.count[v_2] = \ldots = s.count[v_n] = s.count[u]$.

Let v_i be the rightmost node of the cycle. Then there must be some subsequence of nodes v_{i-1}, v_i, v_{i+1}, such that the edge $\{v_{i-1}, v_i\}$ is directed from left to right, and the edge $\{v_i, v_{i+1}\}$ is directed from right to left. We also know that $s.count[v_{i-1}] = s.count[v_i] = s.count[v_{i+1}]$. By the definition of $parity[u]$, $s.parity[v_{i-1}] = s.parity[v_i] = s.parity[v_{i+1}] = p$. By Invariant 4.1 (b) applied to v_{i-1} and v_i, it follows that $p = even$. By Invariant 4.1 (a) applied to v_i and v_{i+1}, it follows that $p = odd$, a contradiction. ■

5 Simulation Relation

In this section we show that PR simulates $NewPR$. First, we introduce a slight modification of the PR algorithm – instead of allowing a set of nodes to take a step at the same time, we now require only one node to take a step at a time. Let this modified version of PR be $OneStepPR$. We use $OneStepPR$ as an intermediate step in showing that PR simulates $NewPR$. To do so, first, we

provide a binary relation from PR to $OneStepPR$, and then another binary relation from $OneStepPR$ to $NewPR$. The main guarantee of both relations is to preserve the same directed version G' of the graph.

5.1 Description of $OneStepPR$

$OneStepPR$ is very similar to PR. It has the same state variables (dir and $list$), and a similar set of actions. Instead of allowing a set of nodes S to take a step together, in $OneStepPR$, only a single node u performs a $reverse(u)$ action. The precondition for this action is that u is a sink, and the effect of the action is that, similarly to PR, u reverses the edges to its neighbors which are not in $list[u]$. However, if $list[u] = nbrs_u$, then all edges incident to u are reversed.

Algorithm 3 $OneStepPR$ automaton

Signature:

$reverse(u)$, $u \in V$, $u \neq D$

States:

for each u, v where $\{u, v\} \in E$:
  $dir[u, v] \in \{in, out\}$, initially in if $v \in in\text{-}nbrs_u$ or out if $v \in out\text{-}nbrs_u$
  $dir[v, u] \in \{in, out\}$, initially in if $u \in in\text{-}nbrs_v$ or out if $u \in out\text{-}nbrs_v$
for each u, $list[u]$, a set of nodes $W \subseteq nbrs_u$, initially empty

Transitions:

$reverse(u)$
  Precondition:
    for each $v \in nbrs_u$, $dir[u, v] = in$
  Effect:
    if $list[u] \neq nbrs_u$ then
      for each $v \in nbrs_u \setminus list[u]$
        $dir[u, v] := out$
        $dir[v, u] := in$
        $list[v] := list[v] \cup \{u\}$
    else
      for each $v \in nbrs_u$
        $dir[u, v] := out$
        $dir[v, u] := in$
        $list[v] := list[v] \cup \{u\}$
    $list[u] := \emptyset$

Tasks:

$\{reverse(u), u \in V, u \neq D\}$

5.2 Relation between PR and $OneStepPR$

We now define a binary relation R' from reachable states of PR to reachable states of $OneStepPR$, in order to show that both algorithms preserve the same directed version G' of the graph. Let s be a reachable state of PR and t be a reachable state of $OneStepPR$. We define $(s,t) \in R'$ if:

1. $s.G' = t.G'$
2. For each node u, $s.list[u] = t.list[u]$.

Lemma 5.1 *(a) For each initial state s of PR, there exists an initial state t of $OneStepPR$ such that $(s,t) \in R'$.*

(b) For each pair of reachable states s of PR, and t of $OneStepPR$, with $(s,t) \in R'$, and for every step (s,s') of PR, there exists a finite sequence of steps of $OneStepPR$ starting with t and ending with some t' such that $(s',t') \in R'$.

Proof Initially, both directed graphs are the same and all nodes' lists are empty, so part (a) of the lemma is true.

To show that part (b) is true, assume $(s,t) \in R$ where s is a reachable state of PR, and t is a reachable state of $OneStepPR$. We need to show that for each step $(s, reverse(S), s') \in trans(PR)$, there exists a finite sequence of steps of $OneStepPR$ starting with t and ending with some t' such that $(s',t') \in R$. Let the corresponding sequence of steps of $OneStepPR$ consist of a $reverse(u)$ action for each $u \in S$. Let $S = \{u_1, u_2, \cdots, u_n\}$; then the sequence of steps in $OneStepPR$ is $(reverse(u_1), reverse(u_2), \cdots, reverse(u_n))$, and $(t = t_0, t_1, t_2, \cdots, t_{n-1}, t_n = t')$ is the corresponding sequence of states.

Consider an arbitrary node $u_i \in S$. In PR, u_i is a sink and reverses a particular set of incident edges determined by the contents of $s.list[u_i]$. First, we show that the $reverse(u_i)$ action is enabled in state t_{i-1} by proving that u_i is a sink in t_{i-1}. We know u_i is a sink in t, and u_i does not take a step until t_{i-1}. No other node could have reversed u_i's edges from incoming to outgoing in the interval $[t, t_{i-1}]$, and so u_i is a sink in each state in $[t, t_{i-1}]$.

Part 1: Here we show that $s'.G' = t'.G'$. To show this, we argue that the same sets of edges are reversed in both algorithms. The sets of edges to be reversed depend only on the contents of the list, so we need to show that $s.list[u_i] = t_{i-1}.list[u_i]$. By part (2) of the relation we know that $s.list[u_i] = t.list[u_i]$, and we also showed that u_i is a sink in each state in $[t, t_{i-1}]$. Therefore, no neighbor of u_i is a sink in this interval, because no two neighboring nodes can be sinks at the same time. Since no neighbor of u_i is a sink, then no neighbor of u_i takes a step in $[t, t_{i-1}]$. Therefore, u_i's list remains the same, and so $s.list[u_i] = t.list[u_i] = t_{i-1}.list[u_i]$. Because the sets of edges reversed in any state depend only on the contents of the lists in that state, it follows that the same sets of edges are reversed in both algorithms. By part (1), $s.G' = t.G'$, so after the same sets of edges are reversed in both graphs, it follows that $s'.G' = t'.G'$. Therefore, part (1) is satisfied.

Part 2: Here we show that $s'.list[u] = t'.list[u]$ for all u. Fix an arbitrary node u. Depending on which nodes take steps in s, there are three possible cases:

Case 1: If $u \in S$, then we know that in both algorithms the lists are emptied after each reversal, so $s'.list[u] = t'.list[u] = \emptyset$.

Case 2: $u \notin S$ but some of u's neighbors are in S. Let $T = nbrs_u \cap S$, $T \neq \emptyset$, that is, T is the set of neighbors of u which take a step together. In PR, all nodes in T are added to $s'.list[u]$. Therefore, $s'.list[u] = s.list[u] \cup T$. In $NewPR$, all nodes in T take a step one at a time, and are added to $list[u]$ one at a time. Therefore, for some arbitrary $u_i \in T$, $t_i.list[u] = t_{i-1}.list[u] \cup \{u_i\}$. Consequently, $t'.list[u] = t.list[u] \cup T$. By part (2) we know that $s.list[u] = t.list[u]$, and therefore, $s'.list[u] = t'.list[u]$.

Case 3: $u \notin S$ and none of u's neighbors are in S. Since $list[u]$ can be modified only by u and its neighbors, and neither u nor any of its neighbors take a step, it follows that $s'.list[u] = s.list[u]$. Similarly, $t'.list[u] = t.list[u]$. Therefore, by part 2 of the relation, it follows that $s'.list[u] = s.list[u] = t.list[u] = t'.list[u]$.

Both parts of the relation are satisfied for s' and t', so $(s', t') \in R'$.

■

Theorem 5.2 *For any reachable state s of PR there exists a reachable state t of $OneStepPR$ such that $(s, t) \in R'$.*

Proof We prove the following statement, which immediately implies the theorem: For any non-negative integer k, and for any state s that is the final state of a k-step execution of PR, there exists a reachable state t of $OneStepPR$ such that $(s, t) \in R'$. The proof is by induction on k.

Base Case: In the base case where $k = 0$, the final state of a k-step execution is the unique initial state of the PR algorithm. By Lemma 5.1 (a), for each initial state s of PR, there exists an initial state t of $OneStepPR$ such that $(s, t) \in R'$. Since t is an initial state of $OneStepPR$, it is a reachable state.

Inductive Step: Assume that for any state s that is the final state of a k-step execution of PR, there exists a reachable state t of $OneStepPR$ such that $(s, t) \in R'$. We need to show that for any state s' that is the final state of a $k+1$-step execution of PR, there exists a reachable state t' of $OneStepPR$ such that $(s', t') \in R'$.

Fix a state s' which is the final state of a $k+1$-step execution of PR. Let $(s'', reverse(u), s')$ be the final step of this execution. Then s'' is the final state of a k-step execution of PR. By the inductive hypothesis, it follows that there exists a reachable state t'' of $OneStepPR$, such that $(s'', t'') \in R'$. Now we apply Lemma 5.1 (b) to $(s'', t'') \in R'$ and (s'', s') being a step of PR. It follows that there exists a sequence of steps of $OneStepPR$ starting with t'' and ending in some state t' such that $(s', t') \in R'$. We append this sequence of steps to some execution of $OneStepPR$ which ends in t''. The resulting execution of $OneStepPR$ ends in state t', and therefore, t' is a reachable state in $OneStepPR$. We have shown that for state s', which is the final state of a

$k+1$-step execution of PR, there exists a reachable state t' of $OneStepPR$ such that $(s', t') \in R'$.

■

5.3 Relation between $OneStepPR$ and $NewPR$

We now define a binary relation from states of $OneStepPR$ to states of $NewPR$, which satisfies specific properties outlined in Lemma 5.3. The main guarantee of the relation is to preserve the equivalence of the directed graphs in both algorithms. Let s be a reachable state of $OneStepPR$ and t be a reachable state of $NewPR$. We define $(s, t) \in R$ if all of the following conditions hold:

1. $s.G' = t.G'$
2. For each node u, if $t.parity[u] = even$ then $s.list[u] \subseteq out\text{-}nbrs_u$.
3. For each node u, if $t.parity[u] = odd$ then $s.list[u] \subseteq in\text{-}nbrs_u$.

Lemma 5.3 *(a) For each initial state s of $OneStepPR$, there exists an initial state t of $NewPR$ such that $(s, t) \in R$.*

(b) For each pair of reachable states s of $OneStepPR$, and t of $NewPR$, with $(s, t) \in R$, and for every step (s, s') of $OneStepPR$, there exists a finite sequence of steps of $NewPR$ starting with t and ending with some t' such that $(s', t') \in R$.

Proof Initially, both graphs are the same, so part 1 of the relation is satisfied. Also initially, $list[u] = \emptyset$, which implies that parts 2 and 3 are true. This proves part (a) of the lemma.

To show that part (b) of the lemma is true, assume $(s, t) \in R$ where s is a state of $OneStepPR$, and t is a state of $NewPR$. We need to show that for each step $(s, reverse(w), s') \in trans(OneStepPR)$, there exists a finite sequence of steps of $NewPR$ starting with t and ending with some t' such that $(s', t') \in R$. This sequence consists of either one or two consecutive $reverse(w)$ steps. If $s.list[w] \neq nbrs_w$, the corresponding sequence of steps of $NewPR$ is a single $reverse(w)$ step. Otherwise, $NewPR$ executes two consecutive $reverse(w)$ steps. The first $reverse(w)$ action is enabled because by part 1 $s.G' = t.G'$, and since w is a sink in s, it is also a sink in t. The second $reverse(w)$ action is enabled because w did not reverse any edges in the previous step, so it is still a sink.

We now show that $(s', t') \in R$, which involves proving that the three parts of R hold for s' and t'.

Part 1: We prove that $t.G' = s.G'$.

Case 1: $t.parity[w] = even$

Since part 2 is true with respect to s and t, $s.list[w] \subseteq out\text{-}nbrs_w$. By Corollary 3.4, because w is a sink, $s.list[w] = out\text{-}nbrs_w$.

Case 1.1: $s.list[w] \neq nbrs_w$. The corresponding step in $NewPR$ is a $reverse(w)$ action.

When w takes a step in $OneStepPR$ it reverses the edges to all nodes in $nbrs_w \setminus s.list[w] = in\text{-}nbrs_w$. Node w reverses the same set of edges in $NewPR$ because $t.parity[w] = even$. Therefore, since $s.G' = t.G'$, and the set of edges reversed in going from s to s' is the same as the set of edges reversed in going from t to t', it follows that $s'.G' = t'.G'$.

Case 1.2: $s.list[w] = nbrs_w$. The corresponding steps in $NewPR$ are two consecutive $reverse(w)$ actions.

In $OneStepPR$, w reverses all edges corresponding to nodes in $out\text{-}nbrs_w$. In $NewPR$, when w executes the first $reverse(w)$ action, since $t.parity[w] = even$ and $in\text{-}nbrs_w = \emptyset$, w does not reverse any edges to neighbors, but only increments its step counter. The result of that action is that $t.parity[w]$ is flipped from *even* to *odd*. Next, w performs the second $reverse(w)$ action. Since $parity[w]$ is *odd*, w reverses all edges corresponding to nodes in $out\text{-}nbrs_w$. Therefore, since $s.G' = t.G'$, and the set of edges reversed in going from s to s' is the same as the set of edges reversed in going from t to t', it follows that $s'.G' = t'.G'$.

Case 2: $t.parity[w] = odd$

Since part 3 is true with respect to s and t, $s.list[w] \subseteq in\text{-}nbrs_w$. By Corollary 3.4, because w is a sink, $s.list[w] = in\text{-}nbrs_w$.

Case 2.1: $s.list[w] \neq nbrs_w$. The corresponding step in $NewPR$ is a $reverse(w)$ action.

When w takes a step in $OneStepPR$ it reverses the edges to all nodes in $nbrs_w \setminus s.list[w] = out\text{-}nbrs_w$. Node w reverses the same set of edges in $NewPR$ because $t.parity[w] = odd$. Therefore, since $s.G' = t.G'$, and the set of edges reversed in going from s to s' is the same as the set of edges reversed in going from t to t', it follows that $s'.G' = t'.G'$.

Case 2.2: $s.list[w] = nbrs_w$. The corresponding steps in $NewPR$ are two consecutive $reverse(w)$ actions.

In $OneStepPR$, w reverses all edges corresponding to nodes in $in\text{-}nbrs_w$. In $NewPR$, when w executes the first $reverse(w)$ action, since $t.parity[w] = odd$ and $out\text{-}nbrs_w = \emptyset$, w does not reverse any edges to neighbors, but only increments its step counter. The result of that action is that $t.parity[w]$ is flipped from *odd* to *even*. Next, w performs the second $reverse(w)$ action. Since $parity[w]$ is *even*, w reverses all edges corresponding to nodes in $in\text{-}nbrs_w$. Therefore, since $s.G' = t.G'$, and the set of edges reversed in going from s to s' is the same as the set of edges reversed in going from t to t', it follows that $s'.G' = t'.G'$.

Part 2: Here we show that for each node u, if $t.parity[u] = even$ then $s.list[u] \subseteq out\text{-}nbrs_u$.

Fix an arbitrary node u. Assume $t'.parity[u] = even$ because otherwise part 2 is vacuously true.

Case 1: $u = w$

Then u is the node that takes the step, so $s'.list[u] = \emptyset$, which implies part 2 for s' and t'.

Case 2: $u \neq w$ and $w \in nbrs_u$. Since u does not take a step, it follows that $t.parity[u] = t'.parity[u] = even$.

Claim 5.4: $w \in \textit{out-nbrs}_u$

Case 2.1: $t.\textit{parity}[w] = \textit{even}$

By Invariant 4.1 (a), the edge between u and w is directed from left to right. Also, because w is a sink in s and t, the edge is directed from u to w. Therefore, w is to the right of u, and so $w \in \textit{out-nbrs}_u$.

Case 2.2: $t.\textit{parity}[w] = \textit{odd}$

Since $t.\textit{parity}[u] \neq t.\textit{parity}[w]$, then $t.\textit{count}[u] \neq t.\textit{count}[w]$. By Invariant 4.2 (c), w is to the right of u, and so $w \in \textit{out-nbrs}_u$.

So far, in Claim 5.4, we established that $w \in \textit{out-nbrs}_u$. Since w is added to $s'.\textit{list}[u]$ in the step of *OneStepPR*, $s'.\textit{list}[u] = s.\textit{list}[u] \cup \{w\}$. By Corollary 3.3, $\textit{list}[u]$ is always a subset of either $\textit{in-nbrs}_u$ or $\textit{out-nbrs}_u$. Since $w \in \textit{out-nbrs}_u$ and $w \in s'.\textit{list}[u]$, it has to be the case that $s'.\textit{list}[u] \subseteq \textit{out-nbrs}_u$. Part 2 remains true with respect to s' and t' because we have assumed that $t'.\textit{parity}[u] = \textit{even}$ and we just showed that $s'.\textit{list}[u] \subseteq \textit{out-nbrs}_u$.

Case 3: $u \neq w$ and $w \notin \textit{nbrs}_u$

Since only u and its neighbors can change the contents of the list, and neither u nor any of its neighbors take a step, $s.\textit{list}[u] = s'.\textit{list}[u]$. Also because u does not take a step, $t.\textit{parity}[u] = t'.\textit{parity}[u]$, and so part 2 remains true for s' and t'.

Part 3: We show that for each node u, if $t.\textit{parity}[u] = \textit{odd}$ then $s.\textit{list}[u] \subseteq \textit{in-nbrs}_u$. The proof is symmetric to the proof of part 2.

■

Theorem 5.4 *For any reachable state s of OneStepPR there exists a reachable state t of NewPR such that $(s, t) \in R$.*

Proof We prove the following statement, which immediately implies the theorem: For any non-negative integer k, and for any state s that is the final state of a k-step execution of *OneStepPR*, there exists a reachable state t of *NewPR* such that $(s, t) \in R$. The proof is by induction on k.

Base Case: In the base case where $k = 0$, the final state of a k-step execution is the unique initial state of the *OneStepPR* algorithm. By Lemma 5.3 (a), for each initial state s of *OneStepPR*, there exists an initial state t of *NewPR* such that $(s, t) \in R$. Since t is an initial state of *NewPR*, it is a reachable state.

Inductive Step: Assume that for any state s that is the final state of a k-step execution of *OneStepPR*, there exists a reachable state t of *NewPR* such that $(s, t) \in R$. We need to show that for any state s' that is the final state of a $k+1$-step execution of *OneStepPR*, there exists a reachable state t' of *NewPR* such that $(s', t') \in R$.

Fix a state s' which is the final state of a $k+1$-step execution of *OneStepPR*. Let $(s'', \textit{reverse}(u), s')$ be the final step of this execution. Then s'' is the final state of a k-step execution of *OneStepPR*. By the inductive hypothesis, it follows that there exists a reachable state t'' of *NewPR*, such that $(s'', t'') \in$

R. Now we apply Lemma 5.3 (b) to $(s'', t'') \in R$ and (s'', s') being a step of $OneStepPR$. It follows that there exists a sequence of steps of $NewPR$ starting with t'' and ending in some state t' such that $(s', t') \in R$. We append this sequence of steps to some execution of $NewPR$ which ends in t''. The resulting execution of $NewPR$ ends in state t', and therefore, t' is a reachable state in $NewPR$. We have shown that for state s', which is the final state of a $k+1$-step execution of $OneStepPR$, there exists a reachable state t' of $NewPR$ such that $(s', t') \in R$.

■

Theorem 5.5 *In any reachable state s of the execution of PR the underlying directed graph $s.G'$ is acyclic.*

Proof Let s be any reachable state of PR. By Theorem 5.2, there exists a reachable state r of $OneStepPR$ such that $(s, r) \in R'$. By Theorem 5.4, there exists a reachable state t of $NewPR$ such that $(r, t) \in R$. By the definition of R', $s.G' = r.G'$, and by the definition of R, $r.G' = t.G'$. It follows that $s.G' = t.G'$. By Theorem 4.3, $t.G'$ is acyclic, and therefore, $s.G'$ is acyclic too.

■

6 Conclusion

We have presented an alternative algorithm for Partial Reversal, and proved that it does not create any cycles in the graph. Our proof does not assume any labels on either the nodes or the edges of the graph, and uses only properties of the PR algorithm to establish the acyclicity property. We have also defined two binary relation between the original PR algorithm and a modified version of it, and between the modified version and our new algorithm. These relations imply that the acyclicity property of our new algorithm applies to the original PR algorithm.

A possible extension of this result is showing a binary relation in the reverse direction too (from the new algorithm to the original one). Such a relation would imply that not only does the acyclicity property apply to the original algorithm too, but also that the two algorithms are equivalent with respect to the direction of the edges in the graph.

References

[1] C. Busch, S. Surapaneni, and S. Tirthapura. *Analysis of link reversal routing algorithms for mobile ad hoc networks.* In Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pages 210219, 2003.

[2] C. Busch and S. Tirthapura. *Analysis of link reversal routing algorithms.* SIAM Journal on Computing, 35(2):305326, 2005.

[3] B. Charron-Bost, J. L. Welch, and J. Widder. *Link reversal: How to play better to work less.* In Proceedings of the 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks, 2009.

[4] E. Gafni and D. Bertsekas. *Distributed algorithms for generating loop-free routes in networks with frequently changing topology.* IEEE Transactions on Communications, C-29(1):1118, 1981.

[5] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[6] J. Welch, J. Walter. *Link Reversal Algorithms.* Morgan Claypool, *Synthesis Lectures on Distributed Computing Theory* (to appear).