Efficient Checking of Link-reversal-based Concurrent Systems

Matthias Függer[1]⋆ and Josef Widder[2]⋆⋆

[1] TU Wien, Embedded Computing Systems Group
[2] TU Wien, Formal Methods in Systems Engineering Group

Abstract. Link reversal is an algorithmic method with various applications. Originally proposed by Gafni and Bertsekas in 1981 for routing in radio networks, it has been later applied also to solve concurrency related problems as mutual exclusion, resource allocation, and leader election. For resource allocation, conflicts can be represented by conflict graphs, and link reversal algorithms work on these graphs to resolve conflicts. In this paper we establish that executions of link reversal algorithms on large graphs are similar (a notion which we make precise in the paper) to executions on smaller graphs. This similarity then allows to verify linear time temporal properties of large systems, by verifying a smaller one.

1 Introduction

Model checking has been applied successfully to finite state hardware and software systems. Application of these techniques to concurrent systems that involve a possibly unbounded number of processes is still a major research question. In a seminal paper, Emerson and Namjoshi [12] showed that the problem is undecidable even in quite simple settings. Despite this discouraging result, Emerson and Namjoshi studied systems where verification is possible. In particular, they considered systems consisting of an arbitrary number of concurrent processes, where processes are organized in a logical ring and a token that circulates in the ring is used to coordinate special actions. They showed that verifying certain correctness properties of such systems of any size can be reduced to verifying small systems of that kind. The token circulation scheme ensures that certain actions of processes are scheduled in a strict round-robin fashion. Later, Clarke *et al.* [9] generalized the work of Emerson and Namjoshi by replacing the round-robin schedule by a more relaxed fairness assumption in which in each infinite run, each process receives the token infinitely often, while the frequency at which the token visits processes may vary between the processes.

While the techniques developed in [12] and [9] allow efficient verification, the results are limited to single token-based concurrent systems. The basic idea

⋆ Supported by projects P21694 and P20529 of the Austrian Science Fund (FWF).
⋆⋆ Supported in part by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) grant PROSEED, and by NSF grant 0964696.

behind structuring concurrent applications using a token is that the current holder of the token is privileged, and thus allowed to enter its critical section. Thus, tokens are a means to resolve conflicts over shared resources. However, the systems considered in [12] or [9] are based on the assumption that at each time at most one process may be privileged, which implicitly means that in both systems the (conservative) assumption is made that all processes are in conflict with all other processes. This assumption is usually overly pessimistic. Moreover, it drastically restricts the degree of concurrency in a system, as at each time at most one process may be scheduled to execute critical code, which basically boils down to serializing concurrent actions. Link reversal algorithms have been used to get rid of these limitations [6, 3], thus allowing to model systems with multiple tokens.

Substantial amount of literature is devoted to link-reversal algorithms (cf. [18] for an overview). Link reversal algorithms work on directed graphs, where to each node a process is associated. A process that is a *sink*, that is, all its incident links are incoming, reverses some of its incident links to be no longer a sink. Algorithms differ in which links are reversed. In this paper we will first focus on the *full reversal* (FR) algorithm in which always *all* incident links are reversed, and will then briefly discuss possible generalizations to the algorithm LR by using a recently introduced formalism [8].

The application of link-reversal algorithms ranges from routing and other problems in wireless networks [16, 15, 5] to resource allocation in concurrent systems [6, 3]. While in the routing problem, the communication graph is the underlying graph on which the algorithms work, in resource allocation, one considers the conflict graph: Let G be a *conflict graph*, that is, a directed graph, whose underlying non-directed graph is connected. If two processes have a conflict — for instance, access a common shared resource — there is a link between the two processes in G. If a link points from i to j, then process j is currently preferred to i. If a node is a sink, it is preferred with respect to all nodes it has a conflict with. As two neighbors cannot be sinks at the same time, a process associated to a sink may thus safely enter the critical section. Upon leaving the critical section it has to reverse the direction of some incident links.

This approach allows high degrees of concurrency in that processes that do not have a conflict may be in their critical sections at the same time. (Obviously, there may be multiple sinks in a directed graph.) Note that in the case where all processes have conflicts with all processes, this leads to a complete conflict graph in which there is (at most) one sink; link reversal thus generalizes the (round-robin) token based approach.

Link reversal algorithms determine the order in which processes take steps, that is, the *schedule*. These schedules induce executions of a transition system or Kripke structure for which temporal logic formulas can be verified. We are interested in linear time temporal logic properties that consider only some of the processes in a system.

Contributions. After recalling the FR algorithm and giving basic definitions, we provide preliminary analysis of FR executions in Section 2. Apart from provid-

ing a result on how steps of processes change the conflict graph, we explain how systems can be composed to ensure liveness and fairness. In Section 3, we define the model checking problem we are interested in, and will recall an important theorem that relates model checking of properties in the temporal logic LTL $\setminus$ X to stutter equivalence of traces. We can therefore concentrate on stutter equivalence of FR executions in the following. In Section 4, we characterize properties of conflict graphs which imply stutter equivalence of FR executions from these graphs. This analysis eventually leads to our major result in Corollary 1, which provides us with a tool to construct small conflict graphs that allow to verify properties of larger ones. For instance, properties that consider only two processes can be verified by considering just a chain graph. In Section 4 we give some examples. After discussing possible generalizations to other link reversal algorithms in Section 5, we close with conclusions that can be drawn from our results, for instance, concerning cut-off sizes.

2 The full reversal algorithm

As mentioned above, the underlying structure of full reversal (FR) is a directed graph. The FR algorithm [13], consists of the following rule which can be applied by any node i that is a sink:

FR: All the links incident on i are reversed.

Note that the FR rule neither changes the set of nodes of the graph nor its undirected support. Let $G_0 = \langle V, E\rangle$ be a conflict graph, i.e., a directed graph whose underlying non-directed graph is connected, with the set of nodes V and the set of links E. An *FR execution* from G_0 is an infinite sequence G_0, S_1, ..., G_{t-1}, S_t, ... of alternating directed graphs and sets of nodes satisfying that for each $t \geq 1$, (i) if there is a sink in G_{t-1}, then S_t is a nonempty subset of the sinks in G_{t-1}, and $S_t = \emptyset$ otherwise, and (ii) G_t is obtained from G_{t-1} by requiring each node i in S_t to apply the FR rule. A sequence of subsets of V, $S = S_1, S_2, \ldots$ is called a *schedule*, and a schedule satisfying (i) and (ii) for initial graph G_0 is called an FR schedule from G_0. If i is in S_t, we say i *takes a step at iteration* t in schedule S. For a given schedule S and a node i, let $W_i(t)$ be the *work of* i by t, that is, the number of iterations $t' \leq t$ in which $i \in S_{t'}$. Formally, $W_i(t) = |\{t' \colon 1 \leq t' \leq t \wedge i \in S_{t'}\}|$. Initially, $W_i(0) = 0$ for all nodes i.

2.1 Basic Properties of FR-based Schedules

We start by introducing some notation. In the following, let $G = \langle V, E\rangle$ be an *acyclic* conflict graph. A *chain* is a sequence $i_0, \ldots, i_k$ of nodes in G, such that either (i_m, i_{m+1}) is in E or (i_{m+1}, i_m) is in E, for $0 \leq m < k$, where k is called the *length* of a chain c, denoted by len(c). A *circuit* is a chain with $i_0 = i_k$ and length greater than 0. A chain is *simple* if its nodes are pairwise distinct, except for the first and last node which may be equal. Let $C^s(i, j)$ be the set of simple

chains of nonzero length that start at i and end at j. A *path* is a chain $i_0, \ldots, i_k$ such that (i_m, i_{m+1}) is in E, for $0 \leq m < k$. The quantity $r_G(c)$, is the number of links in c that are directed "to the right." More formally, if $c = i_0, \ldots, i_k$ is a chain in the graph G, $r_G(c) = |\{(i_m, i_{m+1}) \in E \colon 0 \leq m < k\}|$. Clearly, $r_G(c) = \mathrm{len}(c)$ if and only if c is a path. As the FR algorithm only changes the direction of the links, but not the undirected support of the graph, we observe that for any FR schedule from a graph G_0, c is a chain in G_t if and only if c is a chain in G_{t+1}.

Let $\Sigma(G_0)$ be the *set of schedules*, and let $\Sigma_{FR}(G_0) \subseteq \Sigma(G_0)$ be the *set of FR schedules* from initial graph G_0. We obtain the following invariant of chains within FR executions:

Proposition 1. *Let $G_0, S_1, \ldots, G_t, S_{t+1}, G_{t+1}, \ldots$ be an FR execution. For any two nodes i and j in V, and any chain c in $C^s(i,j)$:*

(1) $r_{G_{t+1}}(c) = r_{G_t}(c)$, if $S_{t+1} \setminus \{i,j\} = S_{t+1}$,
(2) $r_{G_{t+1}}(c) = r_{G_t}(c) + 1$, if $i \in S_{t+1}$ and $j \notin S_{t+1}$,
(3) $r_{G_{t+1}}(c) = r_{G_t}(c) - 1$, if $i \notin S_{t+1}$ and $j \in S_{t+1}$, and
(4) $r_{G_{t+1}}(c) = r_{G_t}(c)$, otherwise, that is, if $\{i,j\} \subseteq S_{t+1}$.

Proof. Let the chain $c = i_0, \ldots, i_\ell$.

(1) Nodes that do not belong to c and take steps, or nodes that do not take steps have no influence on $r(c)$. As neither i nor j take a step at iteration $t+1$, we only have to consider nodes k with two distinct incoming links relative to c. These nodes reverse the directions of both links relative to c. As both links are reversed, the numbers of links pointing to left and right in c, respectively, remain unchanged, which proves (1).

(2) Consider the case where $i = i_0$ takes a step at iteration $t+1$, but j does not. As i_0 takes a step, it is a sink in G_t and therefore (i_0, i_1) is not a link of G_t. Therefore i_1 is not a sink in G_t, and $i_1 \notin S_{t+1}$. Letting c' be the subchain $i_1, \ldots, i_\ell$ of c, we may therefore apply case (1) to c' and obtain

$$r_{G_{t+1}}(c') = r_{G_t}(c'). \tag{i}$$

As i_0 reverses all links in iteration $t+1$, (i_0, i_1) is a link G_{t+1}. As (i_0, i_1) is not a link of G_t, letting $c'' = i_0, i_1$ we obtain

$$r_{G_{t+1}}(c'') = r_{G_t}(c'') + 1. \tag{ii}$$

As c is the concatenation of the chains c'' and c', we obtain from (i) and (ii) that $r_{G_{t+1}}(c) = r_{G_{t+1}}(c'') + r_{G_{t+1}}(c') = r_{G_t}(c'') + 1 + r_{G_t}(c') = r_{G_t}(c) + 1$. The proposition follows in this case, and (3) can be proven analogously.

Similar arguments can be used for (4): Since (1) can be applied to $i_1, \ldots, i_{\ell-1}$, the number of right links stays constant in this subchain. As i and j take steps, the first link (i_1, i_0) and the last link $(i_{\ell-1}, i_\ell)$ in G_t are reversed in G_{t+1}. These two reversal cancel each other out, and (4) follows. □

For a graph G we define $R_G(i,j) = \min\{r_G(c) \mid c \in C^s(i,j)\}$. For the cases of Proposition 1 we thus observe, that in cases (1) and (4) $R_{G_{t+1}}(i,j) = R_{G_t}(i,j)$, in case (2) $R_{G_{t+1}}(i,j) = R_{G_t}(i,j)+1$, and in case (3) $R_{G_{t+1}}(i,j) = R_{G_t}(i,j)-1$. By repeated application of Proposition 1 we thus obtain:

Proposition 2. *If $G_0, S_1, \ldots, G_t, S_{t+1}, G_{t+1}, \ldots$ is an FR execution from G_0, then for any two nodes i and j in V, and any $t \geq 0$:*

$$R_{G_t}(i,j) = R_{G_0}(i,j) + W_i(t) - W_j(t).$$

Proposition 3. *Let $G_0, S_1, \ldots, G_t, S_{t+1}, G_{t+1}, \ldots$ be an FR execution from G_0. For any $t > 0$, if $j \in S_t$ and $i \in V$, then*

$$W_j(t-1) - W_i(t-1) < R_{G_0}(i,j).$$

Proof. As $j \in S_t$, node j is a sink in G_{t-1}. It follows that at least the last link in each chain ending at j is directed towards j and therefore $R_{G_{t-1}}(i,j) > 0$.

From Proposition 2 follows that

$$R_{G_{t-1}}(i,j) = R_{G_0}(i,j) + W_i(t-1) - W_j(t-1). \quad \text{(i)}$$

Now, assume by ways of contradiction that $W_j(t-1) - W_i(t-1) \geq R_{G_0}(i,j)$, that is,

$$0 \leq -R_{G_0}(i,j) + W_j(t-1) - W_i(t-1). \quad \text{(ii)}$$

Adding (i) and (ii), we obtain that $R_{G_{t-1}}(i,j) \leq 0$ which provides the required contradiction. □

2.2 Ensuring Liveness and Fairness

Emerson and Namjoshi [12], restricted the systems by requiring that processes are organized in a directed ring. In the link reversal approach processes can be organized in different ways. Two important properties of schedules that should be met by possible organizations are liveness and fairness: An FR schedule S from graph G_0 is called *live* if there is no $t' \geq 1$ such that $S_t = \emptyset$ for all $t \geq t'$. It is further called *fair* if each node in V takes an infinite number of steps in S. If the difference on the number of times processes are scheduled is bounded, we say a system ensures *strong fairness*.

To see when FR ensures liveness, we first observe that any acyclic conflict graph always contain at least one sink. Further, as in FR always all links incident to a sink are reversed, it is easy to see that FR maintains acyclicity. (This is a well known fact already used in [13]; a proof based on invariants is given in [8].) Hence, FR ensures that starting from an initial acyclic graph, all following graphs are acyclic, and thus contain at least one sink. For our purposes we obtain:

Proposition 4. *All FR schedules from an acyclic conflict graph are live.*

We next show that any FR schedule is not only fair but even provides stronger fairness guarantees:

Proposition 5. *Let G_0 be an acyclic conflict graph. All FR schedules from G_0 ensure strong-fairness: for any two nodes i and j, and any iteration $t \geq 1$,*

$$W_i(t) - W_j(t) \leq R_{G_0}(j, i).$$

Proof. Assume by means of contradiction that there is an FR execution from G_0 and an iteration $t \geq 1$ such that $W_i(t) - W_j(t) > R_{G_0}(j, i)$. Application of Proposition 2 yields, $R_{G_t}(j, i) = R_{G_0}(j, i) - (W_i(t) - W_j(t)) < 0$; a contradiction to $R_{G_t}(j, i)$ being by definition non-negative. The proposition follows. □

Since $R_{G_0}(j, i)$ is bounded by the diameter of the graph G_0, one immediately obtains that $|W_i(t) - W_j(t)|$ is at most the diameter of G_0 for all $t \geq 0$. We thus conclude that from a composability viewpoint, composition of FR instances without violating liveness and (strong) fairness requires just checking whether the resulting graph is acyclic and is therefore not significantly more complex than the composition of rings treated by Emerson and Namjoshi [12].

3 Checking FR scheduled systems

We assume that each node i in V is equipped with a deterministic finite state machine on i's *local state* s_i, where s_i can attain values from state space $\sigma(i)$. The *global state* is defined to be a tuple of local states $s = (s_i)_{i \in V}$. In the following we denote by $I = \{i_j : 1 \leq j \leq |I|\}$ a nonempty subset of V. If s is a global state, then we denote by $s|I$ the projection of the global state s to I, that is, $s|I = (s_i)_{i \in I}$. We assume for simplicity that nodes change their local state, according to their state machine, only when scheduled. Thus given an initial global state $s^0 = (s_i^0)_{i \in V}$, a schedule S from $\Sigma(G_0)$ induces an *execution* from s^0, that is, an infinite sequence $s^0, s^1, \ldots$ of global states. From the initial global states and $\Sigma(G_0)$ one can define a transition system. Let AP be a set of atomic propositions, and λ_I be a function $\lambda_I : \sigma(i_1) \times \sigma(i_2) \times \cdots \times \sigma(i_{|I|}) \to 2^{AP}$. Then, λ is defined to be a labeling function for global states such that $\lambda(s) = \lambda_I(s|I)$. Fixing AP and λ, the transition system then defines a Kripke structure, which we denote by $M_{G_0|I}$. The Kripke structure then defines a set of sequences of atomic propositions, called *traces*. The model checking problem is whether $M_{G_0|I}$ is a model for some temporal logic formula φ over the atomic propositions AP, denoted by $M_{G_0|I} \models \varphi$.

Note that in an FR execution where $I \cap S_t = \emptyset$, $\lambda_I(s^{t-1}|I) = \lambda_I(s^t|I)$ and thus $\lambda(s^{t-1}) = \lambda(s^t)$. This behavior is called stuttering. In the following we shall use well established results regarding stutter equivalence to show how to efficiently verify FR scheduled systems. We need some more preliminaries. Stutter equivalence of two traces τ_1 and τ_2 is defined as follows: From a trace τ, the stutter free trace $\bar{\tau}$ is obtained by removing all successive repetitions. Then two traces are stutter equivalent if $\bar{\tau}_1 = \bar{\tau}_2$.

For each schedule $S = S_1, S_2, \ldots$ from $\Sigma(G_0)$ and each nonempty subset I of V, we define $S \mid I$ to be the infinite sequence $S_1 \cap I, S_2 \cap I, \ldots$ called the projection of S to I. Further for an infinite sequence S of subsets of V, denote

by $Co(S)$ the *condensed sequence of* S, that is obtained from S by removing all empty sets. We observe that if for two schedules S and S', $Co(S \mid I) = Co(S' \mid I)$ then the traces defined by S and S' are stutter equivalent.

In this paper we consider model checking against linear temporal logic formulas. More precisely, we consider the linear temporal logic without the "next time" operator LTL $\setminus$ X. For this temporal logic, there exists the following theorem, whose proof is given, for instance, in [2, page 534].

Theorem 1. *For any two infinite traces τ_1 and τ_2 over atomic propositions AP, and for any LTL$\setminus$X formula φ over AP, if τ_1 and τ_2 are stutter equivalent, then $\tau_1 \models \varphi$ if and only if $\tau_2 \models \varphi$.*

In the following we are therefore interested in stutter equivalence of schedules, and consequently stutter equivalence between the traces of the Kripke structures. In particular, we will show that the set of all condensed FR schedules from some conflict graph is equal to the set of all condensed FR schedules from a considerably simpler and smaller "reduced" graph.

4 Equivalence of FR schedules

In this section we will develop our central result in Corollary 1, that can be found at the end of this section. It considers two graphs G_0 and G_0', where the set of nodes U is contained in both graphs. We show that if certain properties of the directions on links along chains connecting nodes in U are satisfied in both, then $M_{G_0|U} \models \varphi$ if and only if $M_{G_0'|U} \models \varphi$. Then, we will show that the corollary gives us a tool to construct a small graph satisfying the same temporal logic formulas as a large graph.

We start by analyzing the relationship between schedules of FR executions, and initial link directions. We say a conflict graph G_0 is *U-oriented*, where U is a nonempty subset of nodes in G_0, if there exists a path from each node in G_0 to a node in U.

Proposition 6. *Let G be an acyclic conflict graph and U a nonempty subset of nodes in G. If only nodes in U are sinks in G, then G is U-oriented.*

Proof. Assume that only nodes in U are sinks. Choose an arbitrary node i_0 in G. In case i_0 is in U, there exists a path from i_0 to a node in U, namely the empty path, and we are done. Otherwise, i_0 is not a sink in G. Thus there exists a node i_1 such that (i_0, i_1) is a link in G. Again, either i_1 in U in which case there exists a path from i_0 to a node in U, or i_1 is not a sink. By repeated application of the above arguments, we obtain a sequence of nodes $i_0, i_1, \ldots$ whose nodes are pairwise distinct because G is acyclic. As V is finite, one eventually ends up in a node in U. Since the finite sequence is a path in G and its last node is in U, the proposition follows. □

Using Proposition 6, we shall next establish the relation between FR executions from a graph G_0, and the direction of links on chains connecting two nodes

in G_0. Intuitively, node i may take a step before node j if and only if on each chain connecting j and i, at least one link is directed towards i.

Proposition 7. *Let G_0 be an acyclic conflict graph. For any two disjoint subsets I and J of V, where I is nonempty, the following statements are equivalent:*

(A) *There exists an FR execution from G_0 such that all nodes in I take their first step at the same iteration $t \geq 1$, and no node in J takes a step before or at iteration t.*

(B) *For all nodes $i \in I$ and $j \in I \cup J$, $R_{G_0}(j,i) > 0$.*

Proof. To show that Statement (A) implies (B), let $G_0, S_1, G_1, \ldots$ be an FR execution from G_0, where all nodes in I take their first step at iteration t, and no node in J takes a step before or at iteration t. Then, all nodes i in I are sinks in G_{t-1}. Thus all chains ending at i have at least one link directed towards i and for all i in I and j in $I \cup J$, $R_{G_{t-1}}(j,i) > 0$. Proposition 2 yields

$$R_{G_0}(j,i) + W_j(t-1) - W_i(t-1) = R_{G_{t-1}}(j,i) > 0.$$

From $W_i(t-1) = W_j(t-1) = 0$, one finally obtains $R_{G_0}(j,i) > 0$.

To show that Statement (B) implies (A), let $G_0, S_1, G_1, \ldots, S_{t-1}, G_{t-1}$, with $t-1 \geq 0$, be a (finite) prefix of an FR execution from G_0, where for all t', $1 \leq t' \leq t-1$, no node in $I \cup J$ takes a step at iteration t', and only nodes in $I \cup J$ are sinks in G_{t-1}. Such a prefix must exist as otherwise in all FR executions from G_0, where nodes $I \cup J$ do not take steps, there exists a node u in $V \setminus (I \cup J)$ that takes an infinite number of steps, which contradicts strong-fairness of Proposition 5. Proposition 6 further yields that G_{t-1} is $(I \cup J)$-oriented.

To show that Statement (A) follows, it is thus sufficient to show that all nodes in I are sinks in G_{t-1}: Assume by means of contradiction that there is a node i in I that is not a sink in G_{t-1}. Then there exists a neighbor u of i, such that, (i,u) is a link in G_{t-1}. Since G_{t-1} is $(I \cup J)$-oriented, there must be a path from u to some j' in $I \cup J$. Thus there is a path from i to j'. It follows that $R_{G_{t-1}}(j',i) = 0$. Further, by Proposition 2, $R_{G_0}(j',i) + W_{j'}(t-1) - W_i(t-1) = R_{G_{t-1}}(j',i)$. Because $W_{j'}(t-1) = W_i(t-1) = 0$, it holds that $R_{G_0}(j',i) = 0$, a contradiction to Statement (B). □

For a nonempty set of nodes $U \subseteq V$, Proposition 7 allows to determine whether there exist FR schedules S from initial graph G_0 such that the condensed schedule $Co(S \mid U)$ starts with a set $I \subseteq U$. For example, in case $U = \{i,j\}$, there exists a condensed schedule $Co(S \mid U)$ that starts with $\{i,j\}$ if and only if $R_{G_0}(i,j) > 0$ and $R_{G_0}(j,i) > 0$.

Repeated application of Propositions 2 and 7, finally allows us to determine the set of all possible condensed schedules $Co(S \mid U)$, where S is an FR schedule from G_0 and $U \subseteq V$. This set is called the *set of U-condensed FR schedules from G_0*.

Theorem 2. *Let G_0 be an acyclic conflict graph, U a nonempty subset of its nodes, and $S = S_1, S_2, \ldots$ a schedule of nodes in U. The following statements are equivalent:*

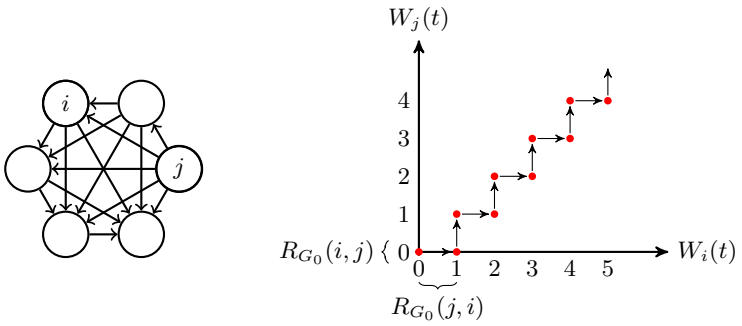

Fig. 1. Complete conflict graph G_0 and set of $\{i, j\}$-condensed FR schedules from G_0.

(A) *Schedule S is a U-condensed FR schedule from G_0.*
(B) *For all $t \geq 1$, (B.i) S_t is nonempty, and (B.ii) for each node i in S_t, and each j in U, $W_i(t-1) - W_j(t-1) < R_{G_0}(j,i)$.*

Proof. To show that Statement (A) implies (B), first observe that (B.i) immediately follows from liveness and fairness of FR schedules from acyclic conflict graphs. Further, (B.ii) follows from Proposition 3.

To show that Statement (B) implies (A), we prove by induction on $t \geq 1$, that for all $t \geq 1$ there exists an FR schedule S' from initial graph G_0 such that $Co(S' \mid U)$ has a prefix of length t equal to $S_1, S_2, \ldots, S_t$. From this it follows that $Co(S' \mid U) = S$, and thus (A) holds.

Induction basis ($t = 1$). Let $I = S_1$ and $J = U$. For all $i \in I$ and $j \in J$ follows from our assumption that $R_{G_0}(j,i) > W_i(0) - W_j(0)$. As $W_i(0) = W_j(0) = 0$, it follows that $R_{G_0}(j,i) > 0$. Therefore, we may apply Proposition 7 with initial graph G_0, in order to obtain that there exists an FR schedule S' from initial graph G_0 such that schedule $Co(S' \mid U)$ starts with $I = S_1$.

Inductive step ($t - 1 \to t$). Assume that there is an FR schedule S' such that $Co(S' \mid U)$ has a prefix of length $t-1$ equal to $S_1, S_2, \ldots, S_{t-1}$. Letting $I = S_t$ and $J = U$, we obtain from Proposition 2 that for all i in I and j in J,

$$R_{G_{t-1}}(j,i) = R_{G_0}(j,i) - (W_i(t-1) - W_j(t-1)) > 0 \ . \tag{1}$$

Application of Proposition 7 with G_{t-1} as initial graph, and the sets I and J as defined above, together with Equation (1) yields that there exists an FR schedule S' from initial graph G_0 such that schedule $Co(S' \mid U)$ starts with $S_1, S_2, \ldots, S_t$. The theorem follows. □

Figures 1 and 2 show examples of condensed schedules as characterized by Theorem 2. Figure 1 depicts a complete conflict graph and Figure 2 a chain

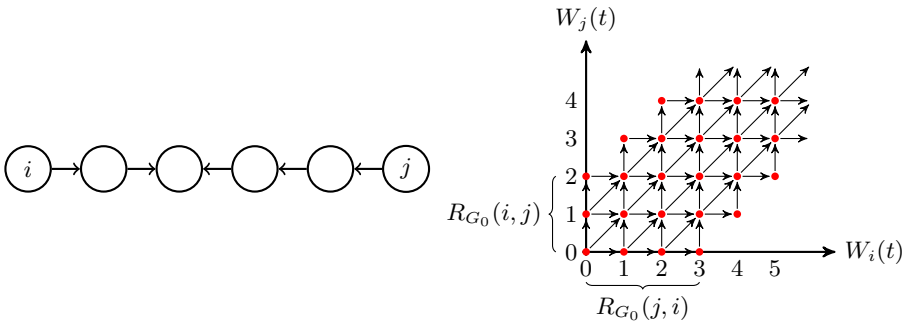

Fig. 2. Chain conflict graph G_0 and set of $\{i,j\}$-condensed FR schedules from G_0.

conflict graph, respectively, together with a graphical representation of the set of all $\{i,j\}$-condensed FR schedules from the respective graphs: Hereby a schedule is represented by a path in the infinite lattice. For example the (only) path in Figure 1 corresponds to the (only) $\{i,j\}$-condensed FR schedule $\{i\},\{j\},\{i\},\ldots$ from the complete conflict graph.

From the understanding of FR schedules we obtained from Theorem 2, we are now in the position to state the main theorem of this paper. It is an exact characterization of all conflict graphs G_0 for which the set of U-condensed FR schedules from G_0 is the same.

Theorem 3. *Let $G_0 = \langle V, E\rangle$ and $G_0' = \langle V', E'\rangle$ be acyclic conflict graphs and let $U \subseteq V \cap V'$ be nonempty. If for all i and j in U, $R_{G_0}(i,j) = R_{G_0'}(i,j)$, then the set of U-condensed FR schedules from G_0 is identical to the set of U-condensed FR schedules from G_0'.*

Proof. According to Theorem 2, the set of U-condensed FR schedules from G_0 is the set of schedules satisfying that for all $t \geq 1$ (i) S_t is nonempty, and (ii) for each node i in $S_t \subseteq U$, and each node j in U, $W_i(t-1) - W_j(t-1) < R_{G_0}(j,i)$. This condition depends on G_0 only by the value of $R_{G_0}(j,i)$, for any i and j in U. As by our assumption, $R_{G_0}(i,j) = R_{G_0'}(i,j)$, for all i and j in U, the theorem follows. □

Combination of Theorem 3 and Theorem 1 thus allows us to check properties on executions induced by an FR schedule from the simple reduced graph of the original graph G_0:

Corollary 1. *Let $G_0 = \langle V, E\rangle$ and $G_0' = \langle V', E'\rangle$ be acyclic conflict graphs and let $U \subseteq V \cap V'$ be nonempty. Further let φ be a LTL $\setminus$ X formula over AP. If, for all nodes i and j in U, $R_{G_0}(i,j) = R_{G_0'}(i,j)$, then $M_{G_0|U} \models \varphi$ if and only if $M_{G_0'|U} \models \varphi$.*

Corollary 1 provides us with a way to construct from G_0 simpler graphs G_0' that allow to verify the same properties. We just have to ensure that $R_{G_0}(i,j) =$

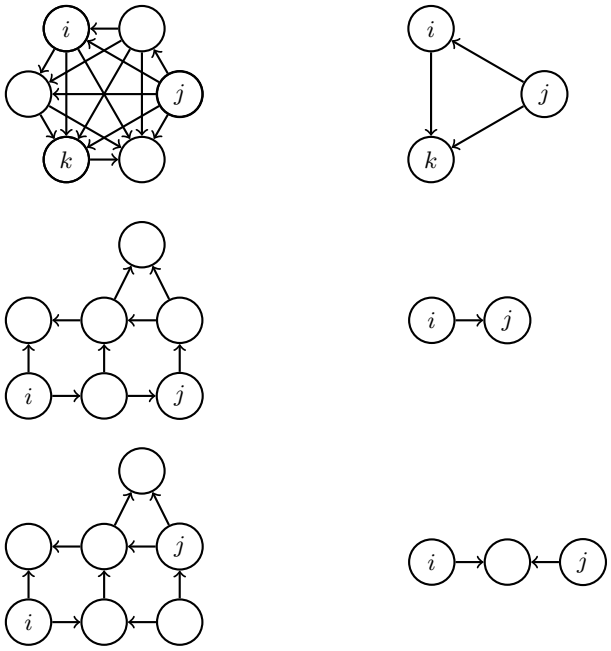

Fig. 3. Conflict graphs (on the left) and their small reduced graphs (on the right).

$R_{G'_0}(i,j)$. Interestingly there exists a very simple graph in the case U comprises of two distinct nodes i and j of G_0, only: We denote by $\mathit{Red}_{ij}(G_0)$ a *reduced graph of* G_0. It is a chain graph that starts with node i, ends with node j, and has $R_{G_0}(j,i)$ links pointing towards i and $R_{G_0}(i,j)$ links pointing towards j in it. Then, the set of $\{i,j\}$-condensed FR schedules from G_0 is identical to the set of $\{i,j\}$-condensed FR schedules from $\mathit{Red}_{ij}(G_0)$.

One can easily generalize graph $\mathit{Red}_{ij}(G_0)$ to the case where U is an arbitrary nonempty subset of nodes in G_0: For each pair i,j of nodes in U, there is a chain in $\mathit{Red}_U(G_0)$ that starts with node i, ends with node j, and has $R_{G_0}(j,i)$ links pointing towards i and $R_{G_0}(i,j)$ links pointing towards j in it. Any two such chains have distinct nodes except possibly for the first or last node. Figure 3 shows some examples of how small reduced graphs can be constructed from conflict graphs G_0.

5 Generalizing FR

Recently, Charron-Bost *et al.* [8] introduced a new formalism called LR that generalizes FR and another link reversal algorithm introduced by Gafni and Bertsekas [13] called partial reversal (PR). In PR only those links are reversed that have not been reversed since the last time a node was a sink.

The LR algorithm works on directed graphs whose links are labeled with 0 or 1. Each node i can apply the following (mutually exclusive) rules if it is a sink:

R1: If at least one link incident on i is labeled 0, then all the links incident on node i that are labeled with 0 are reversed, the other incident links are not reversed, and the labels on all the incident links are flipped.

R2: If all the links incident on node i are labeled 1, then all the links incident on i are reversed, but none of their labels is changed.

This approach generalizes FR and PR via different initial link labelings: if the links are all initially labeled with 1, links remain labeled with 1 and a sink can execute R2 only. The generated executions are FR executions. Otherwise, if all links are initially labeled with 0, then certain nodes may apply R1. One can show that the generated executions are PR executions. Non-uniform labelings may lead to executions different from FR and PR.

Ensuring Liveness and Fairness. We have discussed above that composing two graphs without violating liveness and fairness in the FR case requires just to ensure acyclicity of the resulting initial graph. In the general LR case liveness and fairness do not follow from the initial graph being acyclic. However, in [8] we introduced a simple property (AC) on graphs that guarantees both liveness and fairness of LR schedules from graphs where (AC) is satisfied. It was shown in [7] that checking the (AC) condition for a graph can be reduced to checking the acyclicity of a transformed graph and can thus be achieved efficiently. Moreover, in [8] we discussed a specialization of (AC) that can be easily implemented: for each node, all incoming labels are locally uniformly labeled (either by 0 or 1). Finding working initial labelings can thus be done efficiently.

For composition purposes, note that joining two graph components A and B can be done by connecting nodes in A with nodes in B. If all links are directed from A to B and labeled with 1, (AC) is satisfied if (AC) is satisfied in A and B.

Hence, from a composability viewpoint the more general link reversal approach is not significantly more complex than FR.

Checking LR scheduled systems. We have recently shown [7] that any LR execution from a labeled conflict graph G_0 is equivalent to an FR execution from a (non labeled) transformed conflict graph G_0^{FR}: For each node i in G_0, there is either one node i' or two nodes i'_0 and i'_1 in graph G_0^{FR}. Node i' respectively nodes i'_0 and i'_1 are called the corresponding nodes of i. Whether there are one or two corresponding nodes of i depends on the labels and directions of links incident on i only. In [7], we have shown that there is a bijection from the set of LR schedules from G_0 to the set of FR schedule from G_0^{FR} such that, node i takes a step at iteration t in the LR execution if and only if one of the corresponding nodes of i takes a step at iteration t in the FR execution.

Consequently, we can apply Corollary 1 to conflict graph G_0^{FR}, which easily generalizes our approach to LR. Note, that since G_0^{FR} has at most twice the nodes of G_0, this approach is still efficient.

6 Discussions

We recalled the link reversal approach for scheduling concurrent systems, and presented new results regarding the verification of systems based on this approach. The work is closely related to parameterized model checking [1, 4, 14, 17, 10, 11] where one tries to verify properties of systems of any size. In this context, Emerson and Namjoshi [12] and Clarke *et al.* [9] presented constant size cut-offs — that is, graphs consisting of two to five nodes — for restricted classes of temporal logic formulas. They proved for token-based systems that if some temporal logic formula can be verified for the cut-off graphs, then they hold for systems of any size. One important property of the restricted classes of temporal logic is that their satisfiability has to be independent of the system size. This is why usually the "next time" operator has to be forbidden.

For link-reversal-based systems, one consequence of our Corollary 1 is that system size cannot be determined by our logic: For each graph G_0, all nodes i and j in U, one can easily create a graph G'_0 containing more nodes (for instance by appending a chain of arbitrary length to one node), that still satisfies $R_{G_0}(i,j) = R_{G'_0}(i,j)$. With this respect, the correctness of some systems of arbitrary size follows from the correctness of systems that are scheduled according to our reduced graphs.

With respect to cut-off sizes, assume each node i's state machine can represent — for some constant C — a counter c_i, for $0 \leq c_i < C$. Upon each step of node i the counter c_i is increased by one. One can then define a constant D, satisfying $D < C$, and a labeling function that maps the global states satisfying $c_i - c_j \leq D$ to some atomic proposition p. Consider the problem of verifying the property $\varphi \equiv \mathbf{G}p$, which means that p is always satisfied. Proposition 5 shows that the difference of the work done by two nodes $W_i(t) - W_j(t)$ is bounded by the diameter, from which follows that any graph with diameter at most D satisfies φ, while it is easy to construct a graph with a diameter greater than D that violates φ. We conclude, that there cannot be a constant size cut-off for link-reversal-based systems even if one wants to verify properties that depend on two processes only. This is in sharp contrasts to the results on single token-based systems mentioned above.

Acknowledgments. We are grateful to Igor Konnov for valuable discussions and comments on earlier versions of the paper.

References

1. Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.
2. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.
3. Valmir C. Barbosa and Eli Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Trans. Program. Lang. Syst.*, 11(4):562–584, 1989.

4. Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Reasoning about networks with many identical finite state processes. *Inf. Comput.*, 81(1):13–31, 1989.
5. Costas Busch, Srikanth Surapaneni, and Srikanta Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 210–219, 2003.
6. K. Mani Chandy and Jayadev Misra. The drinking philosopher's problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, 1984.
7. Bernadette Charron-Bost, Matthias Függer, Jennifer L. Welch, and Josef Widder. Partial is full. In *SIROCCO*, volume 6796 of *LNCS*, pages 113–124, 2011.
8. Bernadette Charron-Bost, Antoine Gaillard, Jennifer L. Welch, and Josef Widder. Routing without ordering. In *Proceedings of the 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 145–153, 2009.
9. Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *CONCUR*, volume 3170 of *LNCS*, pages 276–291. Springer, 2004.
10. E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254. Springer, 2000.
11. E. Allen Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *CSL*, volume 3210 of *LNCS*, pages 325–339. Springer, 2004.
12. E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 85–94, 1995.
13. Eli Gafni and Dimitri P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18, January 1981.
14. Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.
15. Navneet Malpani, Jennifer L. Welch, and Nitin Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, 2000.
16. Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *16th Conference on Computer Communications (Infocom)*, pages 1405–1413, April 1997.
17. Amir Pnueli, Jessie Xu, and Lenore D. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In *CAV*, volume 2404 of *LNCS*, pages 107–122. Springer, 2002.
18. Jennifer L. Welch and Jennifer E. Walter. *Link Reversal Algorithms*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.