LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-361

A LATTICE-STRUCTURED PROOF TECHNIQUE APPLIED TO A MINIMUM SPANNING TREE ALGORITHM

Jennifer Lundelius Welch
Leslie Lamport
Nancy Lynch

June 1988

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

A Lattice-Structured Proof Technique Applied to a Minimum Spanning Tree Algorithm

Jennifer Lundelius Welch
Laboratory for Computer Science, Massachusetts Institute of Technology

Leslie Lamport
Digital Equipment Corporation, Systems Research Center

Nancy Lynch
Laboratory for Computer Science, Massachusetts Institute of Technology

Abstract: Highly-optimized concurrent algorithms are often hard to prove correct because they have no natural decomposition into separately provable parts. This paper presents a proof technique for the modular verification of such non-modular algorithms. It generalizes existing verification techniques based on a totally-ordered hierarchy of refinements to allow a partially-ordered hierarchy—that is, a lattice of different views of the algorithm. The technique is applied to the well-known distributed minimum spanning tree algorithm of Gallager, Humblet and Spira, which has until recently lacked a rigorous proof.

Keywords: Distributed algorithms, verification, modularity, partially-ordered refinements, liveness proofs, minimum spanning tree.

The work of Lynch and Welch was supported in part by the Office of Naval Research under Contract N00014-85-K-0168, by the National Science Foundation under Grant CCR-8611442, and by the Defense Advanced Research Projects Agency under Contract N00014-83-K-0125.

1. Introduction

The proliferation of distributed computer systems gives increasing importance to correctness proofs of distributed algorithms. Techniques for verifying sequential algorithms have been extended to handle concurrent and distributed ones—for example, by Owicki and Gries [OG], Manna and Pnueli [MP], Lamport and Schneider [LSc], and Alpern and Schneider [AS]. Practical algorithms are usually optimized for efficiency rather than simplicity, and proving them correct may be feasible only if the proofs can be structured. For a sequential algorithm, the proof is structured by developing a hierarchy of increasingly detailed versions of the algorithm and proving that each correctly implements the next higher-level version. This approach has been extended to concurrent algorithms by Lamport [L], Stark [S], Harel [H], Kurshan [K], and Lynch and Tuttle [LT], where a single action in a higher-level representation can represent a sequence of lower-level actions. The higher-level versions usually provide a global view of the algorithm, with progress made in large atomic steps and a large amount of nondeterminism allowed. At the lowest level is the original algorithm, which takes a purely local view, has more atomic steps, and usually has more constraints on the order of events.

With its totally ordered chain of versions, this hierarchical approach usually does not allow one to focus on a single task in the algorithm. The method described in this paper extends the hierarchical approach to a lattice of versions. At the bottom of the lattice is the original algorithm, which is a refinement of all other versions. However, two versions in the lattice may be incommeasurable, neither one being a refinement of the other.

Multiple higher-level versions of a communication protocol, each focusing on a different function, were considered by Lam and Shankar [LSh]. They called each higher-level version a "projection". If the original protocol is sufficiently modular, then it can be represented as the composition of the projections, and the correctness of the original algorithm follows immediately from the correctness of the projections. This approach was used by Fekete, Lynch, and Shrira [FLS] to prove the correctness of Awerbuch's synchronizer [A1].

Not all algorithms are modular. In practical algorithms, modularity is often destroyed by optimizations. The correctness of a non-modular algorithm is not an immediate consequence of the correctness of its higher-level versions. The method presented in this paper uses the correctness of higher-level versions of an algorithm to simplify its proof. The proofs of correctness of all the versions in the lattice (in which the original algorithm is the lowest-level version) constitute a structured proof of the algorithm.

Any path through our lattice of representations ending at the original algorithm is a totally-ordered hierarchy of versions that can be used in a conventional hierarchical proof. Why do we need the rest of the lattice? Each version in the lattice allows us to formulate and prove invariants about a separate task performed by the algorithm. These invariants will appear somewhere in any assertional proof of the original algorithm. Our method permits us to prove them at as high a level of abstraction as possible.

The method proceeds inductively, top-down through the lattice. First, the highest-level version is shown directly to have the original algorithm's desired property, which involves proving that it satisfies some invariant. Next, let A be any algorithm in the lattice, let $B_1, \ldots, B_i$ ($i \geq 1$) be the algorithms immediately above A in the lattice, and let $Q_1, \ldots, Q_i$ be their invariants. We prove that A satisfies the same safety properties as each B_j, and that a particular predicate P is invariant for A. The invariant P has the form $Q \wedge Q_1 \wedge \cdots \wedge Q_i$ for some predicate Q. In this way, the invariants Q_j are carried down to the proof of lower-level algorithms, and Q introduces information that cannot appear any higher in the lattice—information about details of the algorithm that do not appear at higher levels, and relations between the B_j. We provide two sets of sufficient conditions for verifying these safety properties, one set for the case $i = 1$, and the other for $i > 1$. We also provide three techniques for verifying liveness properties; only one of them makes use of the lattice structure.

The technique is used to prove Gallager, Humblet and Spira's distributed minimum spanning tree algorithm [GHS]. This algorithm has been of great interest for some time. There appears in [GHS] an intuitive description of why the algorithm should work, but no rigorous proof. There are several reasons for giving a formal proof. First, the algorithm has important applications in distributed systems, so its correctness is of concern. Second, the algorithm often appears as part of other algorithms [A2,AG], and the correctness of these algorithms depends upon the correctness of the minimum spanning tree algorithm. Finally, many concepts and techniques have been taken from the algorithm, out of context, and used in other algorithms [A2,CT,G]. Yet the pieces of the algorithm interact in subtle ways, some of which are not explained in the original paper. A careful proof of the entire algorithm can indicate the dependencies between the pieces.

Our proof method helped us to find the correct invariants; it allowed us to describe the algorithm at a high level, yet precisely, and to use our intuition about the algorithm to reason at an appropriate level of abstraction. A by-product of our proof was a better understanding of the purpose and importance of certain parts of the algorithm, enabling us to discover a slight optimization.

The complete proof of the correctness of this minimum spanning tree algorithm is very long and can be found in [W]. One reason for its length is the intricacy of the algorithm. Another reason is the duplication inherent in the approach: the code in all the versions is repetitive, because of carry-over from a higher-level version to its refinement, and because the original algorithm cannot be presented as a true composition of its immediate projections; the repetition in the code leads to repetition in the proof. The full proof also includes extremely detailed arguments—detailed enough so we hope that, in the not too distant, future, they will be machine-checkable. This level of detail seems necessary to catch small bugs in the program and the proof.

Two other proofs of this algorithm have recently been developed. Stomp and de Roever [SdR] used the notion of communication-closed layers, introduced by Elrad and Francez [EF]. Chou and Gafni [CG] prove the correctness of a simpler, more sequential version of the algorithm and then prove that every execution of the original algorithm is equivalent to an execution of the more sequential version.

2. Foundations

This section contains the definitions and results that form the basis for our lattice-structured proof method. Our method can be used with any state-based, assertional verification technique. In this paper, we formulate it in terms of the I/O automaton model of Lynch, Merritt, and Tuttle [LT,LM], which provides a convenient, ready-made "language" for our use. A summary of the I/O automaton model appears in the Appendix.

The first step is to design the lattice, using one's intuition about the algorithm. Each element in the lattice is a version of the algorithm, described as an I/O automaton, and has associated with it a predicate. The bottom element of the lattice is the original algorithm. Next, we must show that all the predicates in the lattice are invariants. The invariant for the top element of the lattice must be shown directly. Assuming that $Q_1, \ldots, Q_i$ are invariants for the versions $B_1, \ldots, B_i$ directly above A in the lattice, we verify that predicate $P = Q \wedge Q_1 \wedge \cdots \wedge Q_i$ is invariant for A, by demonstrating mappings that preserve Q and take executions of A to executions of $B_1, \ldots, B_i$ (thus preserve $Q_1 \wedge \cdots \wedge Q_i$). (Finding these mappings requires insight about the algorithm.) Finally, the lattice is used to show that the original algorithm solves the problem of interest by showing directly that the top element in the lattice solves the problem, and showing a path $A_1, \ldots, A_k$ in the lattice from top to bottom such that each version in the path *satisfies* its predecessor. To show that A_i satisfies A_{i-1}, we show that for every fair execution of A_i, there is a fair execution of A_{i-1} with the same sequence of external actions. The mapping used to verify the invariants takes executions to executions; by adding some additional constraints on the mapping, we can prove, using the invariants, that it takes fair executions to fair executions with the same sequence of external actions, i.e., that liveness properties are preserved.

Section 2.1 deals with safety properties. First, suppose there are two automata, A and B, where B is offered as a "more abstract" version of A. We define a mapping from executions of A to sequences of alternating states and actions of B; if the mapping obeys certain conditions, we say A *simulates* B. Lemma 1 proves that this definition preserves important safety properties, namely that executions of A map to executions of B, and that a certain predicate is an invariant for A. Next we suppose that there are several higher-level versions, A_1, A_2, etc., of one more concrete automaton A. There are situations in which it is difficult to show independently that A simulates A_1 and A simulates A_2, but invariants about states of A_2 can help show a mapping from A to A_1, and invariants about states of A_1 can help show a mapping from A to A_2. To capture this, we define a notion of *simultaneously simulates*, which Lemma 2 proves preserves the same safety properties as in Lemma 1. Of course, to be able to apply Lemma 2, we must know what the invariants of A_1 and A_2 are, which may require having already shown that A_1 and A_2 simulate other automata.

Section 2.2 considers liveness properties. Given automata A and B, and a locally-controlled action φ of B, a definition of A being *equitable* for φ is given; Lemmas 3 and 4 show that this definition implies that in the execution of B obtained from a fair execution of A by either of the simulation mappings, once φ becomes enabled, it either occurs or becomes disabled. We are on our way to verifying the fairness of the induced execution of B.

Three methods of showing that A is equitable for locally-controlled action φ of B are described. The first method is to show that there is an action ρ of A that is enabled whenever φ is, and whose occurrence implies φ's occurrence. (Cf. Lemma 5.)

The second method uses a definition of A being *progressive* for φ. The intuition behind the definition is that there is a set of "helping" actions of A that are guaranteed to occur, and which make progress toward an occurrence of φ in the induced execution of B. Lemma 6 shows that progressive implies equitable.

The third method for checking the equitable condition can be useful when various automata are arranged in a lattice. (See Figure 1.) Suppose B and C are more abstract versions of A, and D is a more abstract version of C. In order to show that A is equitable for action φ of B, we demonstrate an action ρ of D that is "similar" to φ, such that C is progressive for ρ using a set Ψ of helping actions, and A is equitable for all the helping actions in Ψ. (Cf. Lemma 7.)

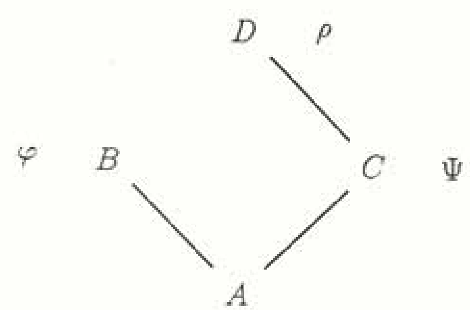

Figure 1

Theorems 8 and 9 in Section 2.3 relate the definitions of simulates, simultaneously simulates, and equitable to the notion of satisfaction.

2.1 Safety

Let A and B be automata. Throughout this paper, we only consider automata such that each locally-controlled action is in a separate class of the action partition. (The definitions and results of this section can be generalized to avoid this assumption, but the statements and proofs are more complicated, and the generalization is not needed for the proof of the [GHS] algorithm.) Let *alt-seq*(B) be the set of all finite sequences of alternating actions of B and states of B that begin and end with an action, including the empty sequence (and the sequence of a single action). An *abstraction mapping* $\mathcal{M}$ from A to B is a pair of functions, $\mathcal{S}$ and $\mathcal{A}$, where $\mathcal{S}$ maps *states*(A) to *states*(B) and $\mathcal{A}$ maps pairs (s, π), of states s of A and actions π of A enabled in s, to *alt-seq*(B).

Given execution fragment $e = s_0\pi_1 s_1 \ldots$ of A, define $\mathcal{M}(e)$ as follows.

- If $e = s_0$, then $\mathcal{M}(e) = \mathcal{S}(s_0)$.
- Suppose $e = s_0 \ldots s_{i-1}\pi_i s_i$, $i > 0$. If $\mathcal{A}(s_{i-1}, \pi_i)$ is empty, then $\mathcal{M}(e) = \mathcal{M}(s_0 \ldots s_{i-1})$. If $\mathcal{A}(s_{i-1}, \pi_i) = \varphi_1 t_1 \ldots t_{m-1}\varphi_m$, then $\mathcal{M}(e) = \mathcal{M}(s_0 \ldots s_{i-1})$ $\varphi_1 t_1 \ldots t_{m-1}\varphi_m \mathcal{S}(s_i)$. The t_j are called *interpolated* states of $\mathcal{M}(e)$.
- If e is infinite, then $\mathcal{M}(e)$ is the limit of $\mathcal{M}(s_0\pi_1 s_1 \ldots s_i)$ as i increases without bound.

We now define a particular kind of abstraction mapping, one tailored for showing inductively that a certain predicate is an invariant of A, and that executions of A map to (nontrivial) executions of B. (A *predicate* is a Boolean-valued function. If Q is a predicate on *states*(B), and $\mathcal{S}$ maps *states*(A) to *states*(B), then $(Q \circ \mathcal{S})$, applied to state s of A, is the predicate "Q is true in $\mathcal{S}(s)$," and is also written $(Q(\mathcal{S}(s)))$.) We give two sets of conditions on abstraction mappings, both of which imply that executions map to executions, with the same sequence of external actions. The first set of conditions applies when there is a single higher-level automaton immediately above. As formalized in Lemma 1, condition (2) ensures that the sequences of external actions are the same, and conditions (1) and (3) ensure that executions map to executions, and that a certain predicate is an invariant for the lower-level algorithm. A key point about this predicate is that it includes the higher-level invariant. Condition (1) is the basis step. Condition (3) is the inductive step, in which the predicate, including the high-level invariant, may be used; part (a) shows the low-level predicate is invariant, while parts (b) and (c) show executions map to executions, by ensuring that if there is no corresponding high-level action, then the high-level state is unchanged, and if there is a corresponding high-level action, then it is enabled in the previous high-level state and its effects are mirrored in the subsequent high-level state. Since executions map to executions, the high-level invariant, when composed with the state mapping, is also invariant for A.

Definition: Let A and B be automata with the same external action signature. Let $\mathcal{M} = (\mathcal{S}, \mathcal{A})$ be an abstraction mapping from A to B, P be a predicate on *states*(A), and Q be a predicate true of all reachable states of B. We say A *simulates* B *via* $\mathcal{M}$, P, and Q if the following three conditions are true.

(1) If s is in *start*(A), then

(a) $P(s)$ is true, and

(b) $\mathcal{S}(s)$ is in $start(B)$.

(2) If s is a state of A such that $Q(\mathcal{S}(s))$ and $P(s)$ are true, and π is any action of A enabled in s, then $\mathcal{A}(s,\pi)|ext(B) = \pi|ext(A)$.

(3) Let (s',π,s) be a step of A such that $Q(\mathcal{S}(s'))$ and $P(s')$ are true. Then
(a) $P(s)$ is true,
(b) if $\mathcal{A}(s',\pi)$ is empty, then $\mathcal{S}(s) = \mathcal{S}(s')$, and
(c) if $\mathcal{A}(s',\pi) = \varphi_1 t_1 \ldots t_{m-1}\varphi_m$, then $\mathcal{S}(s')\varphi_1 t_1 \ldots t_{m-1}\varphi_m \mathcal{S}(s)$ is an execution fragment of B. □

The first lemma verifies that if A simulates B via $\mathcal{M}$, then $\mathcal{M}(e)$ is an execution of B and a certain predicate is true of all states of e.

Lemma 1: *If A simulates B via $\mathcal{M} = (\mathcal{S},\mathcal{A})$, P and Q, then the following are true for any execution e of A.*

(1) $\mathcal{M}(e)$ is an execution of B.

(2) $(Q \circ \mathcal{S}) \wedge P$ is true in every state of e.

Proof: Let $e = s_0\pi_1 s_1 \ldots$. If (1) and (2) are true for every finite prefix $e_i = s_0 \ldots s_i$ of e, then (1) and (2) are true for e. We proceed by induction on i. We need to strengthen the inductive hypothesis for (1) to be the following:

(1) $\mathcal{M}(e_i)$ is an execution of B and $\mathcal{S}(s_i) = t$, where t is the final state in $\mathcal{M}(e_i)$.

(Throughout this proof, "conditions (1), (2) and (3)" refer to the conditions in the definition of "simulates".)

Basis: $i = 0$. (1) $\mathcal{M}(e_0) = \mathcal{S}(s_0)$. Since e_0 is an execution of A, s_0 is in $start(A)$. Condition (1b) implies that $\mathcal{S}(s_0)$ is in $start(B)$, so $\mathcal{M}(e_0)$ is an execution of B. Obviously, the assertion about the final states is true.

(2) Condition (1a) states that P is true in s_0. Since $\mathcal{S}(s_0)$ is in $start(B)$, it is a reachable state of B, and $Q(\mathcal{S}(s_0))$ is true.

Induction: $i > 0$. By the inductive hypothesis for (2), $Q(\mathcal{S}(s_{i-1}))$ and $P(s_{i-1})$ are true. Thus, conditions (3a), (3b) and (3c) are true.

(1) Let $\mathcal{M}(e_{i-1}) = t_0\varphi_1 t_1 \ldots t_j$ and $\mathcal{M}(e_i) = t_0\varphi_1 t_1 \ldots t_m$. Obviously, $m \geq j$.

Suppose $m = j$. Then $\mathcal{M}(e_i) = \mathcal{M}(e_{i-1})$ and is an execution of B by the inductive hypothesis for (1). We deduce that $\mathcal{A}(s_{i-1}, \pi_i)$ is empty, so by condition (3b), $\mathcal{S}(s_i) = \mathcal{S}(s_{i-1})$, and by the inductive hypothesis for (1), $\mathcal{S}(s_{i-1}) = t_j$.

Suppose $m > j$. By construction of $\mathcal{M}(e_i)$, $\mathcal{A}(s_{i-1}, \pi_i) = \varphi_{j+1}t_{j+1}\dots t_{m-1}\varphi_m$, and $t_m = \mathcal{S}(s_i)$. By the inductive hypothesis for (1), $\mathcal{S}(s_{i-1}) = t_j$. By condition (3c), $t_j\varphi_{j+1}\cdots\varphi_m t_m$ is an execution fragment of B. Thus, $\mathcal{M}(e_i)$ is an execution of B. Obviously, the assertion about the final states is true.

(2) By the inductive hypothesis for (2), $(Q \circ \mathcal{S}) \wedge P$ is true in every state of e_i, except (possibly) s_i. By condition (3a), $P(s_i)$ is true. The final state in $\mathcal{M}(e_i)$ is $\mathcal{S}(s_i)$. Since, by part (1), $\mathcal{M}(e_i)$ is an execution of B and $\mathcal{S}(s_i)$ equals the final state of $\mathcal{M}(e_i)$, $\mathcal{S}(s_i)$ is a reachable state of B. By definition of Q, $Q(\mathcal{S}(s_i))$ is true. □

Next we suppose that there are several higher-level versions, say B_1 and B_2, of automaton A, each focusing on a different task. There are situations in which it is impossible to show that A simulates B_1 without using invariants about B_2's task, and it is impossible to show that A simulates B_2 without using invariants about B_1's task. One could cast the invariants about B_2's task as predicates of A, and use the previous definition to show A simulates B_1, but this violates the spirit of the lattice. Instead, we define a notion of *simultaneously simulates*, which allows invariants about both tasks to be used in showing that A simulates B_1 and B_2. The definition differs from simply requiring A to simulate B_1 and A to simulate B_2 in one important way: steps of A only need to be reflected properly in each higher-level algorithm when *all* the higher-level invariants are true (cf. condition (3)).

Definition: Let I be an index set. Let A and A_r, $r \in I$, be automata with the same external action signature. For all $r \in I$, let $\mathcal{M}_r = (\mathcal{S}_r, \mathcal{A}_r)$ be an abstraction mapping from A to A_r, and let Q_r be a predicate true of all reachable states of A_r. Let P be a predicate on *states*(A). We say A *simultaneously simulates* $\{A_r : r \in I\}$ *via* $\{\mathcal{M}_r : r \in I\}$, P, and $\{Q_r : r \in I\}$ if the following three conditions are true.

(1) If s is in *start*(A), then
- (a) $P(s)$ is true, and
- (b) $\mathcal{S}_r(s)$ is in *start*(A_r) for all $r \in I$.

(2) If s is a state of A such that $\bigwedge_{r\in I} Q_r(\mathcal{S}_r(s))$ and $P(s)$ are true, and π is any action of A enabled in s then $\mathcal{A}_r(s,\pi)|ext(A_r) = \pi|ext(A)$ for all $r \in I$.

(3) Let (s', π, s) be a step of A such that $\bigwedge_{r \in I} Q_r(\mathcal{S}_r(s'))$ and $P(s')$ are true. Then
(a) $P(s)$ is true,
(b) if $\mathcal{A}_r(s', \pi)$ is empty, then $\mathcal{S}_r(s) = \mathcal{S}_r(s')$, for all $r \in I$, and
(c) if $\mathcal{A}_r(s', \pi) = \varphi_1 t_1 \dots t_{m-1} \varphi_m$, then $\mathcal{S}_r(s')\varphi_1 t_1 \dots t_{m-1}\varphi_m \mathcal{S}_r(s)$ is an execution fragment of A_r, for all $r \in I$. □

The statement "A simultaneously simulates $\{A_1, A_2\}$ via $\{\mathcal{M}_1, \mathcal{M}_2\}$, P and $\{Q_1, Q_2\}$" is weaker than the statement "A simulates A_1 via $\mathcal{M}_1$, P and Q_1, and A simulates A_2 via $\mathcal{M}_2$, P and Q_2" because the hypotheses of conditions (2) and (3) in the simultaneous definition require that a stronger predicate be true.

Lemma 2 shows that the safety properties of interest are still preserved.

Lemma 2: *Let I be an index set. If A simultaneously simulates $\{A_r : r \in I\}$ via $\{\mathcal{M}_r : r \in I\}$, P, and $\{Q_r : r \in I\}$, where $\mathcal{M}_r = (\mathcal{S}_r, \mathcal{A}_r)$ for all $r \in I$, then the following are true of any execution e of A.*

(1) $\mathcal{M}_r(e)$ is an execution of A_r, for all $r \in I$.

(2) $\bigwedge_{r \in I}(Q_r \circ \mathcal{S}_r) \wedge P$ is true in every state of e.

2.2 Liveness

The following notation is introduced to define the basic liveness notion, "equitable", and to verify that this definition has the desired properties.

We define an execution $e = s_0\pi_1 s_1 \dots$ of automaton A to satisfy $S \hookrightarrow (T, X)$, where S and T are subsets of $states(A)$ and X is a subset of $states(A) \times acts(A)$, if for all i with $s_i \in S$, there is a $j \geq i$ such that either $s_j \in T$ or $(s_j, \pi_{j+1}) \in X$. In words, starting at any state of e, eventually either a state in T is reached, or a state-action pair in X is reached.

If $\mathcal{M} = (\mathcal{S}, \mathcal{A})$ is an abstraction mapping from A to B, then for each locally-controlled action φ of B, we make the following definitions: E_φ is the set of all states s of A such that φ is enabled in $\mathcal{S}(s)$; D_φ is $states(A) - E_\varphi$; D'_φ is the set of all states t of B such that φ is not enabled in t; X_φ is the set of all pairs (s, π) of states s of A and actions π of A such that φ is in $\mathcal{A}(s, \pi)$; and X'_φ is $states(B) \times \{\varphi\}$.

Definition: Suppose $\mathcal{M}$ is an abstraction mapping from A to B. Let φ be a locally-controlled action of B. If every fair execution of A satisfies $states(A) \hookrightarrow (D_\varphi, X_\varphi)$, then A is *equitable* for φ via $\mathcal{M}$. If A is equitable for φ via $\mathcal{M}$ for every locally-controlled action φ of B, then A is *equitable* for B. □

The next lemma motivates the equitable definition — in the induced execution of B, if φ is ever enabled, then eventually φ either occurs or becomes disabled.

Lemma 3: *Suppose A simulates B via $\mathcal{M}$. Let φ be a locally-controlled action of B. If A is equitable for φ via $\mathcal{M}$, then $\mathcal{M}(e)$ satisfies $states(B) \hookrightarrow (D'_\varphi, X'_\varphi)$, for every fair execution e of A.*

Proof: Let $\mathcal{M} = (\mathcal{S}, \mathcal{A})$. Let $e = s_0\pi_1 s_1 \dots$ be a fair execution of A, and let $\mathcal{M}(e) = t_0\varphi_1 t_1 \dots$. For any $i \geq 0$, define $index(i)$ to be j such that $\mathcal{M}(s_0 \dots s_i) = t_0 \dots t_j$. Choose $i \geq 0$.

Case 1: t_i is not interpolated. Choose any l be such that $index(l) = i$. Then $t_i = \mathcal{S}(s_l)$, as argued in the proof of Lemma 1. Suppose there is an $m \geq l$ such that $s_m \in D_\varphi$. Then there is a $j = index(m) \geq i$ such that $t_j = \mathcal{S}(s_m)$, and by definition of D_φ, t_j is in D'_φ. Suppose there is an $m \geq l$ such that $(s_m, \pi_{m+1}) \in X_\varphi$. Then there is a $j = index(m) \geq i$ such that $\varphi_j = \varphi$, by definition of X_φ, and (t_j, φ_{j+1}) is in X'_φ.

Case 2: t_i is interpolated. Let i' be the smallest integer greater than i such that $t_{i'}$ is not interpolated. If either a state in D'_φ or φ occurs between i and i' in $\mathcal{M}(e)$, then we are done. Suppose not. Then the argument in Case 1, applied to $t_{i'}$, shows that eventually after $t_{i'}$, and thus after t_i, either a state in D'_φ or φ occurs in $\mathcal{M}(e)$. □

The next lemma is the analog of Lemma 3 for simultaneously simulates. (D'_φ and X'_φ are defined with respect to $\mathcal{M}_r$.)

Lemma 4: *Suppose A simultaneously simulates $\{A_r : r \in I\}$ via $\{\mathcal{M}_r : r \in I\}$. Let φ be a locally-controlled action of A_r for some r. If A is equitable for φ via $\mathcal{M}_r$, then $\mathcal{M}_r(e)$ satisfies $states(B) \hookrightarrow (D'_\varphi, X'_\varphi)$, for every fair execution e of A.*

The rest of this subsection describes three methods of verifying that A is equitable for action φ of B. Lemma 5 describes the first method, which is to identify an action of A that is essentially the "same" as φ.

Lemma 5: *Suppose $\mathcal{M} = (\mathcal{S}, \mathcal{A})$ is an abstraction mapping from A to B, φ is a locally-controlled action of B, and ρ is a locally-controlled action of A such that, for all reachable states s of A,*

(1) ρ is enabled in s if and only if φ is enabled in state $\mathcal{S}(s)$ of B, and

(2) if ρ is enabled in s, then φ is included in $\mathcal{A}(s, \rho)$.

Then A is equitable for φ via $\mathcal{M}$.

Proof: Let $e = s_0\pi_1 s_1 \ldots$ be a fair execution of A. Choose $i \geq 0$. If $s_i \in D_\varphi$, we are done. Suppose $s_i \in E_\varphi$. By assumption, ρ is enabled in s_i. Since e is fair, there exists $j > i$ such that either $\pi_j = \rho$, in which case $\mathcal{A}(s_{j-1}, \pi_j)$ includes φ, or else ρ is not enabled in s_j, in which case φ is not enabled in $\mathcal{S}(s_j)$. Thus, e satisfies $states(A) \hookrightarrow (D_\varphi, X_\varphi)$. □

The second method uses the following definition, which is shown in Lemma 6 to imply equitable.

Definition: Suppose $\mathcal{M} = (\mathcal{S}, \mathcal{A})$ is an abstraction mapping from A to B. If φ is a locally-controlled action of B, then we say A is *progressive* for φ via $\mathcal{M}$ if there is a set Ψ of pairs (s, ψ) of states s of A and locally-controlled actions ψ of A, and a function v from $states(A)$ to a well-founded set such that the following are true.

(1) For any reachable state $s \in E_\varphi$ of A, some action ψ is enabled in s such that (s, ψ) is in Ψ.

(2) For any step (s', π, s) of A, where s' is reachable and in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$,

(a) $v(s) \leq v(s')$,

(b) if $(s', \pi) \in \Psi$, then $v(s) < v(s')$, and

(c) if $(s', \pi) \notin \Psi$, ψ is enabled in s', and (s', ψ) is in Ψ, then ψ is enabled in s and (s, ψ) is in Ψ. □

Lemma 6: *If A is progressive for φ via $\mathcal{M}$, then A is equitable for φ via $\mathcal{M}$.*

Proof: Let $\mathcal{M} = (\mathcal{S}, \mathcal{A})$. By assumption, φ is a locally-controlled action of B, and there exist Ψ and v satisfying conditions (1) and (2) in the definition of "progressive".

Let $e = s_0\pi_1 s_1 \ldots$ be a fair execution of A. Choose $i \geq 0$. If $s_i \in D_\varphi$, we are done. Suppose $s_i \in E_\varphi$. Assume in contradiction that for all $j \geq i$, $(s_j, \pi_{j+1}) \notin X_\varphi$ and $s_j \in E_\varphi$. By condition (1), there is an action ψ enabled in s_i such that (s_i, ψ) is in Ψ. By condition (2c), as long as $(s_j, \pi_{j+1}) \notin \Psi$, ψ is enabled in s_{j+1} and $(s_{j+1}, \psi) \in \Psi$, for $j \geq i$. Since e is fair, there is $i_1 > i$ such that $(s_{i_1-1}, \pi_{i_1}) \in \Psi$. By conditions (2a) and (2b), $v(s_{i_1}) < v(s_i)$. Similarly, we can show that there is $i_2 > i_1$ such that $v(s_{i_2}) < v(s_{i_1})$. We can continue this indefinitely, contradicting the range of v being a well-founded set. □

The next lemma demonstrates a third technique for showing that A is equitable for locally-controlled action φ of B, in a situation when there are multiple higher-level algorithms. The main idea is to show that there is some action ρ of D that is "similar" to φ (cf. conditions (2) and (3)) such that C is progressive for ρ using certain helping actions (cf. condition (4)), and A is equitable for all the helping actions for ρ (cf. condition (5)). By "similar", we mean that if φ is enabled in the B-image of state s of A, then ρ is enabled in the D-image of the C-image of s; and if ρ occurs in the D-image of the C-image of the pair (s', π), then φ occurs in the B-image of (s', π). Condition (1) is needed for technical reasons. (For convenience, we define abstraction function $\mathcal{M}$ applied to the empty sequence to be the empty sequence. To avoid ambiguity, we add the superscript AB to E_φ, D_φ, and X_φ when they are defined with respect to the abstraction function from A to B.)

Lemma 7: *Let A, B, C and D be automata such that $\mathcal{M}_{AB} = (\mathcal{S}_{AB}, \mathcal{A}_{AB})$ is an abstraction function from A to B, and similarly for $\mathcal{M}_{AC}$ and $\mathcal{M}_{CD}$. Let φ be a locally-controlled action of B. Suppose the following conditions are true.*

(1) $\mathcal{M}_{AC}(e)$ is an execution of C for every execution e of A.

(2) There is a locally-controlled action ρ of D such that for any reachable state s of A, if $s \in E_\varphi^{AB}$, then $\mathcal{S}_{AC}(s) \in E_\rho^{CD}$.

(3) If (s', π, s) is a step of A, s' is reachable, and ρ is in $\mathcal{M}_{CD}(\mathcal{M}_{AC}(s'\pi s))$, then φ is in $\mathcal{A}_{AB}(s', \pi)$.

(4) C is progressive for ρ via $\mathcal{M}_{CD}$, using the set Ψ_ρ and the function v_ρ.

(5) A is equitable for ψ via $\mathcal{M}_{AC}$, for all actions ψ of C such that $(t, \psi) \in \Psi_\rho$ for some state t of C.

Then A is equitable for φ via $\mathcal{M}_{AB}$.

Proof: Let $e = s_0\pi_1 s_1 \ldots$ be a fair execution of A. Let $\mathcal{M}_{AC}(e) = t_0\varphi_1 t_1 \ldots$. By assumption (1), t_m is a reachable state of C for all $m \geq 0$. For any $i \geq 0$, define $index(i)$ to be m such that $\mathcal{M}_{AC}(s_0\pi_1 \ldots s_i) = t_0\varphi_1 \ldots t_m$.

Choose $i \geq 0$. If $s_i \in D_\varphi^{AB}$, we are done. Suppose $s_i \in E_\varphi^{AB}$. Assume in contradiction that for all $j \geq i$, $(s_j, \pi_{j+1}) \notin X_\varphi^{AB}$ and $s_j \in E_\varphi^{AB}$. Let $m = index(i)$. By assumption (2), there is a locally-controlled action ρ of D such that $t_n \in E_\rho^{CD}$ for all $n \geq m$. By assumption (3), $(t_n, \varphi_{n+1}) \notin X_\rho^{CD}$ for all $n \geq m$.

By assumption (4), C is progressive for ρ via $\mathcal{M}_{CD}$, using set Ψ_ρ and function v_ρ. Thus, there is a locally-controlled action ψ of C enabled in $\mathcal{S}_{AC}(s_i) = t_m$ such that $(t_m, \psi) \in \Psi_\rho$. By assumption (5), A is equitable for ψ via $\mathcal{M}_{AC}$. Since e is fair and $s_i \in E_\psi^{AC}$, by Lemma 3 there exists $i_1 > i$ such that either $(s_{i_1-1}, \pi_{i_1}) \in X_\psi^{AC}$ or $s_{i_1} \in D_\psi^{AC}$. Let $m_1 = index(i_1)$.

Case 1: $(s_{i_1-1}, \pi_{i_1}) \in X_\psi^{AC}$. Then $\mathcal{A}_{AC}(s_{i_1-1}, \pi_{i_1})$ includes ψ. Since t_n is reachable, $t_n \in E_\rho^{CD}$, and $(t_n, \varphi_{n+1}) \notin X_\rho^{CD}$ for all $n \geq m$, we conclude that $v_\rho(t_{m_1}) < v_\rho(t_m)$, by parts (2a) and (2b) of the definition of "progressive".

Case 2: $s_{i_1} \in D_\psi^{AC}$. Since t_n is reachable, $t_n \in E_\rho^{CD}$, and $(t_n, \varphi_{n+1}) \notin X_\rho^{CD}$ for all $n \geq m$, by part (2c) of the definition of " progressive", the only way ψ can go from enabled in t_m to disabled in t_{m_1} is for some action in Ψ_ρ to occur between φ_{m+1} and φ_{m_1}. By part (2b) of the definition of "progressive", $v_\rho(t_{m_1}) < v_\rho(t_m)$.

Similarly, we can show that there exists $i_2 > i_1$ such that $v_\rho(\mathcal{S}_{AC}(s_{i_2})) < v_\rho(\mathcal{S}_{AC}(s_{i_1}))$. We can continue this indefinitely, contradicting the range of v_ρ being a well-founded set. □

2.3 Satisfaction

The next theorem shows that our definitions of simulate and equitable are sufficient for showing that A satisfies B.

Theorem 8: *If A simulates B via $\mathcal{M}$, P and Q and if A is equitable for B via $\mathcal{M}$, then A satisfies B.*

Proof: We must show that for any fair execution e of A, there is a fair execution f of B such that $sched(e)|ext(A) = sched(f)|ext(B)$. Given e, let f be $\mathcal{M}(e)$. We verify that $\mathcal{M}(e)$ is a fair execution of B with the desired property. Lemma 1, part (1), implies that f is an execution of B. Choose any locally-controlled action φ of B. By Lemma 3, if φ is enabled in any state of f, then subsequently in f, either a state occurs in which φ is not enabled, or φ occurs. Thus, f is fair. Finally, $sched(e)|ext(A) = sched(f)|ext(B)$ because of condition (2) in the definition of "simulates". □

The next theorem is the analog of Theorem 7 for simultaneously simulates.

Theorem 9: *Let I be an index set. If A simultaneously simulates $\{A_r : r \in I\}$ via $\{\mathcal{M}_r : r \in I\}$, P and $\{Q_r : r \in I\}$, and if A is equitable for A_r via $\mathcal{M}_r$ for some $r \in I$, then A satisfies A_r.*

3. Problem Statement

We define the minimum spanning tree problem as an external schedule module.

For the rest of this paper, let G be a connected undirected graph, with at least two nodes and for each edge, a unique weight chosen from a totally ordered set. Nodes are $V(G)$ and edges are $E(G)$. For each edge (p,q) in $E(G)$, there are two *links* (i.e., directed edges), $\langle p,q\rangle$ and $\langle q,p\rangle$. The set of all links of G is denoted $L(G)$. The set of all links leaving p is denoted $L_p(G)$. The weight of (p,q) is denoted $wt(p,q)$; $wt(\langle p,q\rangle)$ is defined to be $wt(p,q)$; and $wt(nil)$ is defined to be ∞.

The following facts about minimum spanning trees will be useful.

Lemma 10: *(Property 2 in [GHS]) The minimum spanning tree of G is unique.*

Proof: Suppose in contradiction that T_1 and T_2 are both minimum spanning trees of G and $T_1 \neq T_2$. Let e be the minimum-weight edge that is in one of the trees but not both. Without loss of generality, suppose e is in $E(T_1)$. The set of edges $\{e\} \cup E(T_2)$ must contain a cycle, and at least one edge, say e', of this cycle is not in $E(T_1)$. Since $e \neq e'$ and e' is in one but not both of the trees, $wt(e) < wt(e')$. Thus replacing e' with e in $E(T_2)$ yields a spanning tree of G with smaller weight than T_2, contradicting the assumption. □

Let $T(G)$ be the (unique) minimum spanning tree of G.

An *external* edge (p,q) of subgraph F of G is an edge of G such that $p \in V(F)$ and $q \notin V(F)$.

Lemma 11: *(Property 1 in [GHS]) If F is a subgraph of $T(G)$, and e is the minimum-weight external edge of F, then e is in $T(G)$.*

Proof: Suppose in contradiction that e is not in $T(G)$. Then a cycle is formed by e together with some subset of the edges of $T(G)$. At least one other edge e' of this cycle is also an external edge of F. By choice of e, $wt(e) < wt(e')$. Thus, replacing e' with e in the edge set of $T(G)$ produces a spanning tree of G with smaller weight than $T(G)$, which is a contradiction. □

The $MST(G)$ problem is the following external schedule module. Input actions are $\{Start(p) : p \in V(G)\}$. Output actions are $\{InTree(l), NotInTree(l) : l \in L(G)\}$. Schedules are all sequences of actions such that

- no output action occurs unless an input action occurs;

- if an input action occurs, then exactly one output action occurs for each $l \in L(G)$;
- if $InTree(\langle p, q \rangle)$ occurs, then (p, q) is in $T(G)$; and
- if $NotInTree(\langle p, q \rangle)$ occurs, then (p, q) is not in $T(G)$.

4. Proof of Correctness

The verification of Gallager, Humblet and Spira's minimum-spanning tree algorithm [GHS] uses several automata, arranged into a lattice as in Figure 2.

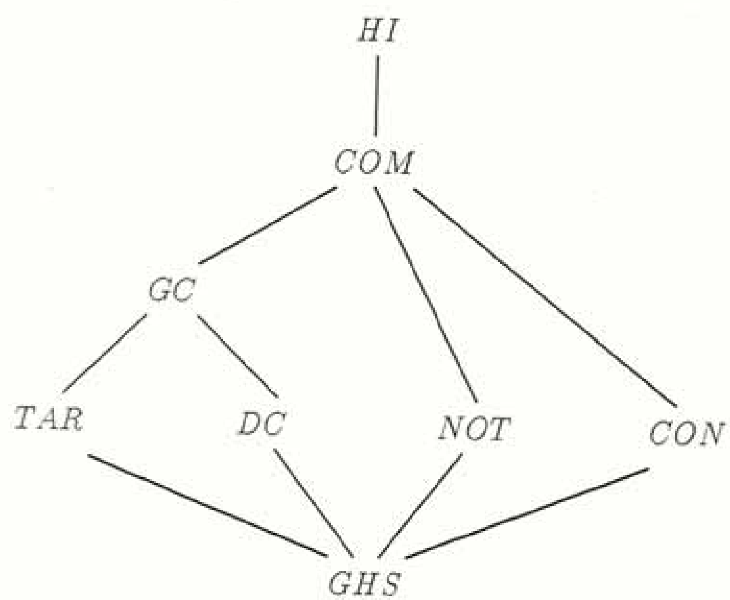

Figure 2: The Lattice

Each element of the lattice is a complete algorithm. However, the level of detail in which the actions and state of the original algorithm are represented varies. Working down the lattice takes us from a description of the algorithm that uses global information about the state of the graph, and powerful, atomic actions, to a fully distributed algorithm, in which each node can only access its local variables, and many actions are needed to implement a single higher level action. A brief overview of each algorithm is given below; a fuller description of each appears later.

HI is a very high-level description of the algorithm, and is easily shown in Section 4.1 to solve the $MST(G)$ problem. GHS is the detailed algorithm from

[GHS]. We show a path in the lattice from GHS to HI, where each automaton in the path satisfies the automaton above it. By transitivity of satisfaction, then GHS will have been shown to solve $MST(G)$.

The essential feature of the state of HI is a set of subgraphs of G, initially the set of singleton nodes of G. Subgraphs combine, in a single action, along minimum-weight external edges, until only one subgraph, the minimum spanning tree, remains.

The COM automaton introduces *fragments*, each of which corresponds to a subgraph of HI, plus extra information about the global *level* and *core* (or identity) of the subgraph. Two ways to combine fragments are distinguished, *merging* and *absorbing*, and two milestones that a fragment must reach before combining are identified. The first milestone is computing the minimum-weight external link of the fragment, and the second is indicating readiness to combine.

The GC automaton expands on the process of finding the minimum-weight external link of a fragment, by introducing for each fragment a set *testset* of nodes that are participating in the search. Once a node has found its local minimum-weight external link, it is removed from the testset.

TAR and DC expand on GC in complementary ways. DC focuses on how the nodes of a fragment cooperate to find the minimum-weight external link of the whole fragment in a distributed fashion. It describes the flow of messages throughout the fragments: first a broadcast informs nodes that they should find their local minimum-weight external links, and then a convergecast reports the results back. In contrast, TAR is unconcerned with specifying exactly when each node finds its local minimum-weight external link, and concentrates on the details of the protocol performed by a node to find this link.

NOT is a refinement of COM that expands on the method by which the global level and core information for a fragment is implemented by variables local to each node. Messages attempt to notify nodes of the level and core of the nodes' current fragment.

CON, an orthogonal refinement of COM, concentrates on how messages are used to implement what happens between the time the minimum-weight external link of an entire fragment is computed, and the time the fragment is combined with another one.

Finally, the entire, fully distributed, algorithm is represented in automaton GHS. It expands on and unites TAR, DC, NOT and CON.

The path chosen through the lattice is HI, COM, GC, TAR, GHS. Why this path? Obviously, GHS must be shown to satisfy one of TAR, DC, NOT and CON. However, it cannot be done in isolation; that is, invariants about the other three are necessary to show that GHS satisfies one. (As mentioned in Section 2.1, the invariants about the other three could be made predicates about GHS, but this approach does not take advantage of abstraction.) Thus, we show that GHS simultaneously simulates those four automata. To show this, however, we need to verify that certain predicates really are invariants for the four. In order to do this, we show that TAR and DC (independently) simulate GC, and that NOT and CON (independently) simulate COM. Likewise, in order to show these facts, we need to know that certain predicates are invariants of GC and COM, and the way we do that is to show that GC simulates COM, and that COM simulates HI. Thus, it is necessary to show safety relationships along every edge in the lattice.

The liveness relationships only need to be shown along one path from GHS to HI. After inspecting GHS and the four automata directly above it, we decided on pragmatic grounds that it would be easiest to show that GHS is equitable for TAR. One consideration was that the output actions have exactly the same preconditions in GHS and in TAR, and thus showing GHS is equitable for those actions is trivial. Once TAR was chosen, the rest of the path was fixed.

First, the necessary safety properties are verified in Section 4.2. We show that COM simulates HI (Section 4.2.1), that GC simulates COM (Section 4.2.2), that TAR simulates GC (Section 4.2.3), that DC simulates GC (Section 4.2.4), that NOT simulates COM (Section 4.2.5), that CON simulates COM (Section 4.2.6), and that GHS simultaneously simulates TAR, DC, NOT and CON (Section 4.2.7).

Section 4.3 contains the liveness arguments. To show the desired chain of satisfaction, we show that COM is equitable for HI (Section 4.3.1), that GC is equitable for COM (Section 4.3.2), that TAR is equitable for GC (Section 4.3.3), and that GHS is equitable for TAR (Section 4.3.6). In Section 4.3.6, the technique of Lemma 7 is used in several places; thus we need to show that DC is progressive for an action of GC (Section 4.3.4), and that CON is progressive for several actions of COM (Section 4.3.5).

Section 4.4 puts the pieces together to show that GHS solves $MST(G)$.

4.1 HI Solves MST(G)

The main feature of the HI state is the data structure FST (for "forest"), which consists of a set of subgraphs of G, partitioning $V(G)$. The idea is that the subgraphs of G are connected subgraphs of the minimum spanning tree $T(G)$. Two subgraphs can combine if the minimum-weight external link of one leads to the other. The *awake* variable is used to make sure that no output action occurs unless an input action occurs. The *answered* variables are used to ensure that at most one output action occurs for each link. *InTree*$(\langle p,q\rangle)$ can only occur if $\langle p,q\rangle$ is already in a subgraph, or is the minimum-weight external edge of a subgraph (i.e., is destined to be in a subgraph). *NotInTree*$(\langle p,q\rangle)$ can only occur if p and q are in the same subgraph but the edge between them is not.

Define automaton HI (for "High Level") as follows.

The state consists of a set FST of subgraphs of G, a Boolean variable *answered*(l) for each $l \in L(G)$, and a Boolean variable *awake*.

In the start state of HI, FST is the set of single-node graphs, one for each $p \in V(G)$, every *answered*(l) is false, and *awake* is false.

Input actions:

- *Start*(p), $p \in V(G)$
  Effects:
  awake := true

Output actions:

- *InTree*$(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  awake = true
  $(p,q) \in F$ or (p,q) is the minimum-weight external edge of F,
  for some $F \in FST$
  answered$(\langle p,q\rangle)$ = false
  Effects:
  answered$(\langle p,q\rangle)$:= true

- *NotInTree*$(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  awake = true

$p, q \in F$ and $(p, q) \notin F$, for some $F \in FST$
$answered(\langle p, q\rangle) =$ false
Effects:
$answered(\langle p, q\rangle) :=$ true

Internal actions:

- $Combine(F, F', e)$, $F, F' \in FST$, $e \in E(G)$
  Preconditions:
  $awake =$ true
  $F \neq F'$
  e is an external edge of F
  e is the minimum-weight external edge of F'
  Effects:
  $FST := FST - \{F, F'\} \cup \{F \cup F' \cup e\}$

Define the following predicates on $states(HI)$. (A *minimum spanning forest* of G is a set of disjoint subgraphs of G that span $V(G)$ and form a subgraph of a minimum spanning tree of G.)

- HI-A: Each F in FST is connected.
- HI-B: FST is a minimum spanning forest of G.

Let $P_{HI} =$ HI-A $\wedge$ HI-B. HI-B implies that the elements of FST form a partition of $V(G)$. Lemma 10 and HI-B imply that FST is a subgraph of $T(G)$.

Theorem 12: *HI solves the $MST(G)$ problem, and P_{HI} is true in every reachable state of HI.*

Proof: First we show that P_{HI} is true in every reachable state of HI. If s is a start state of HI, then P_{HI} is obviously true. Suppose (s', π, s) is a step of HI and P_{HI} is true in s'. If $\pi \neq Combine(F, F', e)$, then, since FST is unchanged, P_{HI} is obviously true in s as well.

Suppose $\pi = Combine(F, F', e)$. By the precondition, $F \neq F'$, e is the minimum-weight external edge of F', and e is an external edge of F in s'. By HI-A, F and F' are each connected in s'; thus, the new fragment formed in s by joining F and F' along e is connected, and HI-A is true. Since by HI-B and Lemma 10, F and F' are subgraphs of $T(G)$, and since by Lemma 11 e is in $T(G)$, the new FST is a minimum spanning forest of G, and HI-B is true.

We now show that HI solves $MST(G)$. Let e be a fair execution of HI. The use of the variable *awake* ensures that no output action occurs in e unless an input action occurs in e. The use of the variables *answered*(l) ensures that at most one output action occurs in e for each link l. Suppose *InTree*($\langle p,q\rangle$) occurs in e. Then in the preceding state, either (p,q) is in F or (p,q) is the minimum-weight external edge of F, for some $F \in FST$. By HI-B and Lemmas 10 and 11, (p,q) is in $T(G)$. Suppose *NotInTree*($\langle p,q\rangle$) occurs in e. Then in the preceding state, p and q are in F and (p,q) is not in F, for some $F \in FST$. By HI-A, there is path from p to q in F. By HI-B and Lemma 10, this path is in $T(G)$. Thus (p,q) cannot be in $T(G)$, or else there would be a cycle.

Suppose an input action occurs in e. We show that an output action occurs in e for each link. Let $e = s_0\pi_1 s_1 \ldots$. Obviously, π_1 is an input action. Only a finite number of output actions can occur in e. Choose m such that π_m is the last output action occurring in e. (Let $m = 1$ if there is no output action in e.) It is easy to see that $s_m = s_i$ for all $i \geq m$. Since an input action occurs in e before s_m, *awake* = true in s_m. $|FST| = 1$ in s_m, because otherwise some *Combine*(F, F', e') action would be enabled in s_m, contradicting e being fair. Let $FST = \{F\}$. By HI-A and HI-B, $F = T(G)$ in s_m. Furthermore, *answered*(l) is true in s_m for each l, because otherwise some output action for l would be enabled in s_m, contradicting e being fair. Yet the only way *answered*(l) can be true in s_m is if an output action for l occurs in e. □

4.2 Safety

Each algorithm in the lattice below HI is presented in a separate subsection. Each subsection is organized as follows. First, an informal description of the algorithm is given, together with a discussion of any particularly interesting aspects. Then comes a description of the state of the automaton, both explicit variables, and derived variables (if any). A *derived* variable is a variable that is not an explicit element of the state, but is a function of the explicit variables. We employ the convention that whenever the definition of a derived variable is not unique or sensible, then the derived variable is undefined. The actions of the automaton are specified next. Then predicates to be shown invariant for this automaton are listed. The abstraction mapping to be used for simulating the higher-level automaton is defined next. All our state mappings conform to the rule that variables with the same name have the same value in all the algorithms. The only potential problem that might arise with this rule is if a derived variable is mapped to an explicit variable, but the derived variable is undefined. Although we will prove that this situation never occurs in states we are interested in, for completeness of the definition of state mapping one can simply choose some default value for the explicit variable. Often it is useful to derive some predicates about this automaton's state that follow from the invariant for this automaton and the higher-level one; these predicates are true of any state of this automaton satisfying the invariant and mapping to a reachable state of the higher-level algorithm. The proof of simulation completes the subsection.

4.2.1 COM Simulates HI

The *COM* algorithm still takes a completely global view of the algorithm, but some intermediate steps leading to combining are identified, and the state is expanded to include extra information about the subgraphs. The *COM* state consists of a set of *fragments*, a data structure used throughout the rest of the lattice. Each fragment f has associated with it a subgraph of G, as well as other information: *level*(f), *core*(f), *minlink*(f), and *rootchanged*(f). Two milestones must be reached before a fragment can combine. First, the *ComputeMin*(f) action causes the minimum-weight external link of fragment f to be identified as *minlink*(f), and second, the *ChangeRoot*(f) action indicates that fragment f is ready to combine, by setting the variable *rootchanged*(f). This automaton distinguishes two ways that fragments (and hence, their associated subgraphs) can combine. The *Merge*(f, g) action causes two fragments, f and g, at the same level with the same minimum-weight external edge, to combine; the new fragment has a higher level and a new core (i.e., identifying edge). The *Absorb*(f, g) action causes a fragment g to be engulfed by the fragment f at the other end of *minlink*(g), provided f is at a higher level than g.

Define automaton *COM* (for "Common") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- *subtree*(f), a subgraph of G;
- *core*(f), an edge of G or *nil*;
- *level*(f), a nonnegative integer;
- *minlink*(f), a link of G or *nil*;
- *rootchanged*(f), a Boolean.

The state also contains Boolean variables, *answered*(l) one for each $l \in L(G)$, and Boolean variable *awake*.

In the start state of COM, *fragments* has one element for each node in $V(G)$; for fragment f corresponding to node p, *subtree*$(f) = \{p\}$, *core*$(f) = nil$, *level*$(f) = 0$, *minlink*(f) is the minimum-weight link adjacent to p, and *rootchanged*(f) is false. Each *answered*(l) is false and *awake* is false.

Two fragments will be considered the same if either they have the same single-node subtree, or they have the same nonnil core.

We define the following derived variables.

- For node p, *fragment*(p) is the element f of *fragments* such that p is in *subtree*(f).
- A link $\langle p, q\rangle$ is an *external* link of p and of *fragment*(p) if *fragment*$(p) \neq$ *fragment*(q); otherwise the link is *internal.*
- If *minlink*$(f) = \langle p, q\rangle$, then *minedge*$(f)$ is the edge (p, q), *minnode*$(f) = p$, and *root*(f) is the endpoint of *core*(f) closest to p.
- If $\langle p, q\rangle$ is the minimum-weight external link of fragment f, then *mw-minnode*(f) $= p$ and *mw-root*(f) is the endpoint of *core*(f) closest to p.
- *subtree*(p) is all nodes and edges of *subtree*(*fragment*(p)) on the opposite side of p from *core*(*fragment*(p)).
- q is a *child* of p if $q \in$ *subtree*(p) and $(p, q) \in$ *subtree*(*fragment*(p)).

Input actions:

- *Start*(p), $p \in V(G)$
  - Effects:
    - *awake* := true

Output actions:

- *InTree*$(\langle p, q\rangle)$, $\langle p, q\rangle \in L(G)$
  - Preconditions:
    - *awake* = true
    - $(p, q) \in$ *subtree*(*fragment*(p)) or $\langle p, q\rangle =$ *minlink*(*fragment*(p))

$answered(\langle p,q\rangle) = \text{false}$

Effects:

$answered(\langle p,q\rangle) := \text{true}$

- $NotInTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$

  Preconditions:

  $fragment(p) = fragment(q)$ and $(p,q) \notin subtree(fragment(p))$

  $answered(\langle p,q\rangle) = \text{false}$

  Effects:

  $answered(\langle p,q\rangle) := \text{true}$

Internal actions:

- $ComputeMin(f)$, $f \in fragments$

  Preconditions:

  $minlink(f) = nil$

  l is the minimum-weight external link of f

  $level(f) \leq level(fragment(target(l)))$

  Effects:

  $minlink(f) := l$

- $ChangeRoot(f)$, $f \in fragments$

  Preconditions:

  $awake = \text{true}$

  $rootchanged(f) = \text{false}$

  $minlink(f) \neq nil$

  Effects:

  $rootchanged(f) := \text{true}$

- $Merge(f,g)$, $f,g \in fragments$

  Preconditions:

  $f \neq g$

  $rootchanged(f) = rootchanged(g) = \text{true}$

  $minedge(f) = minedge(g)$

  Effects:

  add a new element h to $fragments$

  $subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$

  $core(h) := minedge(f)$

  $level(h) := level(f) + 1$

  $minlink(h) := nil$

$rootchanged(h) :=$ false
delete f and g from *fragments*

- $Absorb(f, g)$, $f, g \in fragments$
  Preconditions:
  $rootchanged(g) =$ true
  $level(g) < level(f)$
  $fragment(target(minlink(g))) = f$
  Effects:
  $subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$
  delete g from *fragments*

Define the following predicates on states of *COM*. (All free variables are universally quantified.)

- COM-A: If $minlink(f) = l$, then l is the minimum-weight external link of f, and $level(f) \leq level(fragment(target(l)))$.
- COM-B: If $rootchanged(f) =$ true, then $minlink(f) \neq nil$.
- COM-C: If $awake =$ false, then $minlink(f) \neq nil$, $rootchanged(f) =$ false, and $subtree(f) = \{p\}$ for some p.
- COM-D: If $f \neq g$, then $subtree(f) \neq subtree(g)$.
- COM-E: If $subtree(f) = \{p\}$ for some p, then $minlink(f) \neq nil$.
- COM-F: If $|nodes(f)| = 1$, then $level(f) = 0$ and $core(f) = nil$; if $|nodes(f)| > 1$, then $level(f) > 0$ and $core(f) \in subtree(f)$.

Let P_{COM} be the conjunction of COM-A through COM-F.

In order to show that *COM* simulates *HI*, we define an abstraction mapping $\mathcal{M}_1 = (\mathcal{S}_1, \mathcal{A}_1)$ from *COM* to *HI*. Define the function $\mathcal{S}_1$ from $states(COM)$ to $states(HI)$ as follows. In conformance with our convention (cf. the beginning of Section 4.2), the values of *awake* and $answered(l)$ (for all l) in $\mathcal{S}_1(s)$ are the same as in s. The value of FST in $\mathcal{S}_1(s)$ is the multiset $\{subtree(f) : f \in fragments\}$.

Define the function $\mathcal{A}_1$ as follows. Let s be a state of *COM* and π an action of *COM* enabled in s.

- If $\pi = Start(p)$, $InTree(l)$, or $NotInTree(l)$, then $\mathcal{A}_1(s, \pi) = \pi$.

- If $\pi = ComputeMin(f)$ or $ChangeRoot(f)$, then $\mathcal{A}_1(s,\pi)$ is empty.
- If $\pi = Merge(f,g)$ or $Absorb(f,g)$, then $\mathcal{A}_1(s,\pi) = Combine(F,F',e)$, where $F = subtree(f)$ in s, $F' = subtree(g)$ in s, and $e = minedge(g)$ in s.

The following predicate is true in every state of COM satisfying $(P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$. (I.e., it is deducible from P_{COM} and the HI predicates.)

- COM-G: The multiset $\{subtree(f) : f \in fragments\}$ forms a partition of $V(G)$, and $fragment(p)$ is well-defined.

Proof: Let s be a state of COM satisfying $(P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$. In $\mathcal{S}_1(s)$, $FST = \{subtree(f) : f \in fragments\}$. By HI-B, FST forms a partition of $V(G)$. By COM-D, the multiset $\{subtree(f) : f \in fragments\} = FST$, and thus it forms a partition of $V(G)$. Consequently, $fragment(p)$ is well-defined. □

Lemma 13: *COM simulates HI via $\mathcal{M}_1$, P_{COM}, and P_{HI}.*

Proof: By inspection, the types of COM, HI, $\mathcal{M}_1$ and P_{COM} are correct. By Theorem 12, P_{HI} is a predicate true in every reachable state of HI.

(1) Let s be in $start(COM)$. Obviously, P_{COM} is true in s, and $\mathcal{S}_1(s)$ is in $start(HI)$.

(2) Obviously, $\mathcal{A}_1(s,\pi)|ext(HI) = \pi|ext(COM)$ for any state s of A.

(3) Let (s', π, s) be a step of COM such that P_{HI} is true of $\mathcal{S}_1(s')$ and P_{COM} is true of s'. We consider each possible value of π.

i) π is Start(p), InTree(l), or NotInTree(l). $\mathcal{A}_1(s',\pi) = \pi$. Obviously, P_{COM} is true in s, and $\mathcal{S}_1(s')\pi\mathcal{S}_1(s)$ is an execution fragment of HI.

ii) π is ComputeMin(f) or ChangeRoot(f). $\mathcal{A}_1(s',\pi)$ is empty. Obviously, $\mathcal{S}_1(s') = \mathcal{S}_1(s)$. Obviously, COM-A, COM-B, COM-D and COM-F are true in s. By COM-C for $ComputeMin(f)$ and by precondition for $ChangeRoot(f)$, $awake =$ true in s', and also in s; thus, COM-C is true in s.

Obviously, COM-E is true in s for any fragment $f' \neq f$. If $\pi = ComputeMin(f)$, then $minlink(f) \neq nil$ in s, and COM-E is vacuously true in s for f. If $\pi = ChangeRoot(f)$, then by COM-B, $minlink(f) \neq nil$ in s' and also in s, so COM-E is vacuously true in s for f.

iii) π is Merge(f,g).

(3c) $\mathcal{A}_1(s', \pi) = Combine(F, F', e)$, where $F = subtree(f)$ in s', $F' = subtree(g)$ in s', and $e = minedge(g)$ in s', for some fragments f and g.

Claims about s':

1. $f \neq g$, by precondition.
2. $rootchanged(f) = rootchanged(g) =$ true, by precondition.
3. $minedge(f) = minedge(g)$, by precondition.
4. $awake =$ true, by Claim 2 and COM-C.
5. $minedge(f) \neq nil$ and $minedge(g) \neq nil$, by Claim 2 and COM-B
6. $minlink(f)$ is an external link of f, by COM-A and Claim 5.
7. $minlink(g)$ is the minimum-weight external link of g, by COM-A and Claim 5.

Let $F = subtree(f)$, $F' = subtree(g)$ and $e = minedge(g)$.

Claims about $\mathcal{S}_1(s')$: (All depend on the definition of $\mathcal{S}_1$.)

8. $awake =$ true, by Claim 4.
9. $F \neq F'$, by Claim 1 and COM-D.
10. e is an external edge of F, by Claims 3 and 6.
11. e is the minimum-weight external edge of F', by Claim 7.

By Claims 8 through 11, $Combine(F, F', e)$ is enabled in $\mathcal{S}_1(s')$. Obviously, its effects are mirrored in $\mathcal{S}_1(s)$.

(3a) *More claims about s':*

12. $level(f) \geq 0$, by COM-F.
13. $subtree(f')$ and $subtree(g')$ are disjoint, for all $f' \neq g'$, by COM-G.

Claims about s:

14. $subtree(h) = subtree(f) \cup subtree(g) \cup minedge(f)$, by code.
15. $core(h) = minedge(f)$, by code.
16. $level(h) = level(f) + 1$, by code.
17. $minlink(h) = nil$, by code.
18. $rootchanged(h) =$ false, by code.
19. f and g are removed from *fragments*, by code.
20. $awake =$ true, by Claim 4.
21. $subtree(f')$ and $subtree(g')$ are disjoint, for all $f' \neq g'$, by Claims 13, 14 and 19.

22. $|nodes(h)| > 1$, by Claim 14.
23. $level(h) > 1$, by Claims 12 and 16.
24. $core(h) \in subtree(h)$, by Claims 14 and 15.

COM-A is vacuously true for h by Claim 17. COM-B is vacuously true for h by Claim 18. COM-C is vacuously true by Claim 20. COM-D is true by Claim 21. COM-E is vacuously true for h by Claim 22. COM-F is true for h by Claims 22, 23 and 24.

iv) π is Absorb(f,g).

(3c) $\mathcal{A}_1(s', \pi) = Combine(F, F', e)$, where $F = subtree(f)$ in s', $F' = subtree(g)$ in s', and $e = minedge(g)$ in s', for some fragments f and g.

Claims about s':

1. $rootchanged(g) =$ true, by precondition.
2. $level(g) < level(f)$, by precondition.
3. $fragment(target(minlink(g))) = f$, by precondition.
4. $f \neq g$, by Claim 2.
5. $minlink(g)$ is an external link of f, by Claims 3 and 4.
6. $minlink(g) \neq nil$, by Claim 3.
7. $minlink(g)$ is the minimum-weight external link of g, by Claim 6 and COM-A.
8. $awake =$ true, by Claim 1 and COM-C.

Let $F = subtree(f)$, $F' = subtree(g)$ and $e = minedge(g)$.

Claims about $\mathcal{S}_1(s')$: (All depend on the definition of $\mathcal{S}_1$.)

9. $awake =$ true, by Claim 8.
10. $F \neq F'$, by Claim 4 and COM-D.
11. e is an external edge of F, by Claim 5.
12. e is the minimum-weight external edge of F', by Claim 7.

By Claims 9 through 12, $Combine(F, F', e)$ is enabled in $\mathcal{S}_1(s')$. Obviously, its effects are mirrored in $\mathcal{S}_1(s)$.

(3a) COM-A: If $minlink(f) = nil$ in s', then the same is true in s, and COM-A is vacuously true for f. Suppose $minlink(f) = l$ in s'. Let $f' = fragment(target(l))$.

More claims about s':

13. $level(f) \leq level(f')$, by COM-A.
14. $f' \neq g$, by Claims 2 and 13.
15. $minedge(f) \neq minedge(g)$, by Claim 14.
16. $minlink(f)$ is the minimum-weight external link of f, by COM-A.
17. If $e' \neq minedge(g)$ is an external edge of g, then $wt(e') > wt(minedge(f))$. Pf: $wt(e') > wt(minedge(g))$ by Claim 7, and $wt(minedge(g)) > wt(minedge(f))$ by Claims 5, 15 and 16.

Since $minlink(f)$ is the same in s as in s', Claims 16 and 17 imply that in s, $minlink(f)$ is the minimum-weight external link of f. The only fragment whose *level* changes in going from s' to s is g (since g disappears). Thus, Claim 14 implies that in s, $level(f) \leq level(f')$. Finally, COM-A is true in s.

The next claims are used to verify COM-B through COM-F.

More claims about s':

18. $subtree(f')$ and $subtree(g')$ are disjoint, for all $f' \neq g'$, by COM-G.
19. $level(g) \geq 0$, by COM-F.
20. $level(f) > 0$, by Claims 2 and 19.
21. $|nodes(f)| > 1$, by Claim 20 and COM-F.
22. $core(f) \in subtree(f)$, by Claim 21 and COM-F.

Claims about s:

23. $awake =$ true, by Claim 1.
24. $subtree(f)$ in s is equal to $subtree(f) \cup subtree(g) \cup minedge(g)$ in s', by code.
25. $subtree(f')$ and $subtree(g')$ are disjoint, for all $f' \neq g'$, by Claims 18 and 24.
26. $|nodes(f)| > 1$, by Claims 21 and 24.
27. $level(f) > 0$, by Claim 20.
28. $core(f) \in subtree(f)$, by Claims 22 and 24.

COM-B is unaffected. COM-C is vacuously true by Claim 23. COM-D is true by Claim 25. COM-E is vacuously true for f by Claim 26. COM-F is true for f by Claims 26, 27 and 28. □

Let $P'_{COM} = (P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$.

Corollary 14: *P'_{COM} is true in every reachable state of COM.*

Proof: By Lemmas 1 and 13. □

4.2.2 GC Simulates COM

The *GC* automaton expands on the process of finding the minimum-weight external link of a fragment, by introducing for each fragment f a set *testset*(f) of nodes that are participating in the search. Once a node in f has found its minimum-weight external link, it is removed from *testset*(f). A new action, *TestNode*(p), is added, by which a node p atomically finds its minimum-weight external link — however, the fragment at the other end of the link cannot be at a lower level than p's fragment in order for this action to occur. The new variable *accmin*(f) (for "accumulated minlink") stores the link with the minimum weight over all links external to nodes of f no longer in *testset*(f). *ComputeMin*(f) cannot occur until *testset*(f) is empty. When an *Absorb*(f, g) action occurs, all the nodes formerly in g are added to *testset*(f) if and only if the target of *minlink*(g) is in *testset*(f). This version of the algorithm is still totally global in approach.

Define automaton *GC* (for "Global ComputeMin") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- *subtree*(f), a subgraph of G;
- *core*(f), an edge of G or *nil*;
- *level*(f), a nonnegative integer;
- *minlink*(f), a link of G or *nil*;
- *rootchanged*(f), a Boolean;
- *testset*(f), a subset of $V(G)$; and
- *accmin*(f), a link of G or *nil*.

The state also contains Boolean variables, *answered*(l), one for each $l \in L(G)$, and Boolean variable *awake*.

In the start state of *COM*, *fragments* has one element for each node in $V(G)$; for fragment f corresponding to node p, *subtree*(f) $= \{p\}$, *core*(f) $=$ *nil*, *level*(f) $=$ 0, *minlink*(f) is the minimum-weight link adjacent to p, *rootchanged*(f) is false, *testset*(f) is empty, and *accmin*(f) is *nil*. Each *answered*(l) is false and *awake* is false.

Input actions:

- $Start(p)$, $p \in V(G)$
  Effects:
  $awake :=$ true

Output actions:

- $InTree(\langle p,q \rangle)$, $\langle p,q \rangle \in L(G)$
  Preconditions:
  $awake =$ true
  $(p,q) \in subtree(fragment(p))$ or $\langle p,q \rangle = minlink(fragment(p))$
  $answered(\langle p,q \rangle) =$ false
  Effects:
  $answered(\langle p,q \rangle) :=$ true

- $NotInTree(\langle p,q \rangle)$, $\langle p,q \rangle \in L(G)$
  Preconditions:
  $fragment(p) = fragment(q)$ and $(p,q) \notin subtree(fragment(p))$
  $answered(\langle p,q \rangle) =$ false
  Effects:
  $answered(\langle p,q \rangle) :=$ true

Internal actions:

- $TestNode(p)$, $p \in V(G)$
  Preconditions:
  — let $f = fragment(p)$ —
  $p \in testset(f)$
  if $\langle p,q \rangle$, the minimum-weight external link of p, exists
  then $level(f) \leq level(fragment(q))$
  Effects:
  $testset(f) := testset(f) - \{p\}$
  if $\langle p,q \rangle$, the minimum-weight external link of p, exists
  and $wt(p,q) < wt(accmin(f))$
  then $accmin(f) := \langle p,q \rangle$

- $ComputeMin(f)$, $f \in fragments$
  Preconditions:
  $minlink(f) = nil$

$accmin(f) \neq nil$
$testset(f) = \emptyset$

Effects:

$minlink(f) := accmin(f)$
$accmin(f) := nil$

- $ChangeRoot(f)$, $f \in fragments$

Preconditions:

$awake =$ true
$rootchanged(f) =$ false
$minlink(f) \neq nil$

Effects:

$rootchanged(f) :=$ true

- $Merge(f, g)$, $f, g \in fragments$

Preconditions:

$f \neq g$
$rootchanged(f) = rootchanged(g) =$ true
$minedge(f) = minedge(g) \neq nil$

Effects:

add a new element h to *fragments*
$subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$
$core(h) := minedge(f)$
$level(h) := level(f) + 1$
$minlink(h) := nil$
$rootchanged(h) :=$ false
$testset(h) := nodes(h)$
$accmin(h) := nil$
delete f and g from *fragments*

- $Absorb(f, g)$, $f, g \in fragments$

Preconditions:

$rootchanged(g) =$ true
$level(g) < level(f)$
— let $p = target(minlink(g))$ —
$fragment(p) = f$

Effects:

$subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$
if $p \in testset(f)$ then $testset(f) := testset(f) \cup testset(g)$

delete g from *fragments*

Define the following predicates on the states of GC. (All free variables are universally quantified.)

- GC-A: If $accmin(f) = \langle p, q \rangle$, then $\langle p, q \rangle$ is the minimum-weight external link of any node in $nodes(f) - testset(f)$, and $level(f) \leq level(fragment(q))$.
- GC-B: If there is an external link of f, if $minlink(f) = nil$, and if $testset(f) = \emptyset$, then $accmin(f) \neq nil$.
- GC-C: If $testset(f) \neq \emptyset$, then $minlink(f) = nil$.

Let $P_{GC} = \text{GC-A} \wedge \text{GC-B} \wedge \text{GC-C}$.

In order to show that GC simulates COM, we define an abstraction mapping $\mathcal{M}_2 = (\mathcal{S}_2, \mathcal{A}_2)$ from GC to COM. Define the function $\mathcal{S}_2$ from $states(GC)$ to $states(COM)$ by simply ignoring the variables $accmin(f)$ and $testset(f)$ for all fragments f when going from a state of GC to a state of COM.

Define the function $\mathcal{A}_2$ as follows. Let s be a state of GC and π an action of GC enabled in s. If $\pi = TestNode(p)$, then $\mathcal{A}_2(s, \pi)$ is empty. Otherwise, $\mathcal{A}_2(s, \pi) = \pi$.

Recall that $P'_{COM} = (P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$. If $P'_{COM}(\mathcal{S}_2(s))$ is true, then the COM predicates are true in $\mathcal{S}_2(s)$, and the HI predicates are true in $\mathcal{S}_1(\mathcal{S}_2(s))$.

Lemma 15: *GC simulates COM* via $\mathcal{M}_2$, P_{GC}, and P'_{COM}.

Proof: By inspection, the types of GC, COM, $\mathcal{M}_2$, and P_{GC} are correct. By Corollary 14, P'_{COM} is a predicate true in every reachable state of COM.

(1) Let s be in $start(GC)$. Obviously, P_{GC} is true in s, and $\mathcal{S}_2(s)$ is in $start(COM)$.

(2) Obviously, $\mathcal{A}_2(s, \pi)|ext(COM) = \pi|ext(GC)$.

(3) Let (s', π, s) be a step of GC such that P'_{COM} is true of $\mathcal{S}_2(s')$ and P_{GC} is true of s'.

i) π **is Start(p), InTree(l), NotInTree(l), or ChangeRoot(f).** Obviously, $\mathcal{S}_2(s')\pi\mathcal{S}_2(s)$ is an execution fragment of COM, and P_{GC} is true in s.

ii) π **is ComputeMin(f).**

(3a) Obviously, P_{GC} is still true in s for any $f' \neq f$. GC-A is vacuously true for f in s, since $accmin(f)$ is set to nil. GC-B is vacuously true for f in s, since $minlink(f) \neq nil$. By COM-C, $awake$ = true in $\mathcal{S}_2(s')$ and thus in s'; the same is true in s, so GC-C(a) is true in s for f. GC-C(b) is vacuously true for f in s, since $testset(f) = \emptyset$.

(3c) $\mathcal{A}_2(s', \pi) = \pi$.

Claims about s':

1. $testset(f) = \emptyset$, by precondition.
2. $accmin(f) \neq nil$, by precondition.
3. $level(f) \leq level(fragment(target(accmin(f))))$, by Claim 2 and GC-A.
4. $accmin(f)$ is the minimum-weight external link of f, by Claim 2, GC-A, and Claim 1.
5. $level(f) \leq level(fragment(target(l)))$, where l is the minimum-weight external link of f, by Claims 3 and 4.

Using Claim 5, it is easy to see that $\mathcal{S}_2(s')\pi\mathcal{S}_2(s)$ is an execution fragment of *COM*.

iii) π is TestNode(p).

(3a) Obviously, P_{GC} is still true in s for any $f' \neq f$. Inspecting the code verifies that GC-A and GC-B are still true in s for f as well. By GC-C(b), $minlink(f) = nil$ in s'; GC-C is true for f in s because $minlink(f)$ is not changed.

(3b) $\mathcal{A}_2(s', \pi)$ is empty, and obviously $\mathcal{S}_2(s') = \mathcal{S}_2(s)$.

iv) π is Merge(f,g).

(3a) Obviously, P_{GC} is still true in s for any f' other than f and g. GC-A is vacuously true in s for h, since $accmin(h) = nil$. GC-B is vacuously true in s for h, since $testset(h) \neq \emptyset$. GC-C is true in s for h since $minlink(h) = nil$.

(3c) $\mathcal{A}_2(s', \pi) = \pi$. Obviously, $\mathcal{S}_2(s')\pi\mathcal{S}_2(s)$ is an execution fragment of *COM*.

v) π is Absorb(f,g).

(3a) Obviously, P_{GC} is still true in s for any f' other than f and g.

In going from s' to s, $testset(f)$ is either empty in both or non-empty in both, $minlink(f)$ remains the same, and the truth of the existence of an external link of

f either stays true or goes from true to false. Thus GC-B and GC-C are true in s for f.

We now deal with GC-A. If $accmin(f) = nil$ in s', then the same is true in s, so GC-A is vacuously true for f in s.

Assume $accmin(f) = \langle r,t \rangle$. Let $minlink(g) = \langle q,p \rangle$.

Claims about s':

1. $level(g) < level(f)$, by precondition.
2. $fragment(p) = f$, by precondition.
3. $level(f) \leq level(fragment(t))$, by GC-A.
4. $fragment(t) \neq g$, by Claims 1 and 3.
5. $\langle q,p \rangle \neq \langle t,r \rangle$, by Claim 4 and COM-A.
6. $wt(q,p) < wt(l)$, for any $l \neq \langle q,p \rangle$ that is an external link of g, by COM-A.
7. If $p \notin testset(f)$, then $wt(r,t) < wt(q,p)$, by Claim 5 and GC-A.
8. If $p \notin testset(f)$, then $wt(r,t) < wt(l)$, for any l that is an external link of g, by Claims 6 and 7.

If $p \notin testset(f)$ in s', then any node $p' \in nodes(f)$ is not in $testset(f)$ in s exactly if, in s', p' is either in $nodes(f) - testset(f)$ or in $nodes(g)$. Claim 8 implies that in s, $\langle r,t \rangle$ is still the minimum-weight external link of any node in f that is not in $testset(f)$.

If $p \in testset(f)$ in s', then any node $p' \in nodes(f)$ is not in $testset(f)$ in s exactly if p' is in $nodes(f) - testset(f)$ in s'. Thus in s, $\langle r,t \rangle$ is still the minimum-weight external link of any node in f that is not in $testset(f)$.

Since g is the only fragment whose *level* changes in going from s' to s, Claim 4 implies that $level(f) \leq level(fragment(t))$ in s. Thus, since $accmin(f) = \langle r,t \rangle$ in s, GC-A is true in s for f.

(3c) $\mathcal{A}_2(s,\pi) = \pi$. Obviously $\mathcal{S}_2(s')\pi\mathcal{S}_2(s)$ is an execution fragment of *COM*.

□

Let $P'_{GC} = (P'_{COM} \circ \mathcal{S}_2) \wedge P_{GC}$.

Corollary 16: *P'_{GC} is true in every reachable state of GC.*

Proof: By Lemmas 1 and 15. □

4.2.3 TAR Simulates GC

This automaton expands on the method by which a node finds its local minimum-weight external link. Some local information is introduced in this version, in the form of node variables and messages. Three FIFO message queues are associated with each link $\langle p, q\rangle$: $tarqueue_p(\langle p, q\rangle)$, the outgoing queue local to p; $tarqueue_{pq}(\langle p, q\rangle)$, modelling the communication channel; and $tarqueue_q(\langle p, q\rangle)$, the incoming queue local to q. The action *ChannelSend*(l, m) transfers a message m from the outgoing local queue of link l to the communication channel of l; and the action *ChannelRecv*(l, m) transfers a message m from the communication channel of link l to the incoming local queue of l.

Each link l is classified by the variable *lstatus*(l) as branch, rejected, or unknown. Branch means the link will definitely be in the minimum spanning tree; rejected means it definitely will not be; and unknown means that the link's status is currently unknown. Initially, all the links are unknown.

The search for node p's minimum-weight external link is initiated by the action *SendTest*(p), which causes p to identify its minimum-weight unknown link as *testlink*(p), and to send a TEST message over its testlink together with information about the level and core (identity) of p's fragment. If the level of the recipient q's fragment is less than p's, the message is requeued at q, to be dealt with later (when q's level has increased sufficiently). Otherwise, a response is sent back. If the fragments are different, the response is an ACCEPT message, otherwise, it is a REJECT message. An optimization is that if q has already sent a TEST message over the same edge and is waiting for a response, and if p and q are in the same fragment, then q does not respond — the TEST message that q already sent will inform p that the edge (p, q) is not external.

When a REJECT message (or a TEST in the optimized case described above) is received, the recipient marks that link as rejected, if it is unknown. It is possible that the link is already marked as branch, in which case it should not be changed to rejected.

When a *ChangeRoot*(f) occurs, *minlink*(f) is marked as branch; when an *Absorb*(f, g) occurs, the reverse link of *minlink*(g) is marked as branch. As soon as a link l is classified as branch, the *InTree*(l) output action can occur; as soon as a link l is classified as rejected, the *NotInTree*(l) output action can occur.

The requeuing of a message is a delicate aspect of this (as well as the original) algorithm. When p receives a message that it is not yet ready to handle, it cannot simply block receiving any more messages on that link, but instead it must allow other messages to jump over that message, as the following example shows. Suppose p is in a fragment at level 3, q is in a fragment at level 4, p sends a TEST message to q with parameter 3, and before it is received, q sends a TEST message to p with parameter 4. When p receives q's TEST message, it is not ready to handle it. When q receives p's TEST message, it sends back an ACCEPT message. In order to prevent deadlock, p must be able to receive this ACCEPT message, even though it was sent after the TEST message. Thus, the correctness of the algorithm depends on a subtle interplay between FIFO behavior, and occasional, well-defined, exceptions to it.

The following scenario demonstrates the necessity of checking that $lstatus(l)$ is unknown before changing it to rejected, when a TEST or REJECT is received. (The reason for the check, which also appears the full algorithm, is not explained in [GHS].) Suppose p is in fragment f with level 8 and core c, q is in fragment g with level 4 and core d, and $\langle q, p\rangle$ is the minimum-weight external link of g. First, q determines that $\langle q, p\rangle$ is its local minimum-weight external link. Then p sends a TEST$(8, c)$ message to p, which is requeued, since $8 > 4$. Eventually, $ComputeMin(g)$ occurs, and $minlink(g)$ is set equal to $\langle q, p\rangle$. Then $ChangeRoot(g)$ occurs, and $\langle q, p\rangle$ is marked as branch. Then $Absorb(f, g)$ occurs, and $\langle p, q\rangle$ is marked as branch. The next time that q tries to process p's TEST$(8, d)$ message, it succeeds, determines that $\langle q, p\rangle$ is not external, since d is the core of q's fragment, and sends REJECT to q. But q had better not change the classification of $\langle q, p\rangle$ from branch to rejected. Similarly, when p receives q's REJECT message, it had better not change the classification of $\langle p, q\rangle$ from branch to rejected.

Define automaton *TAR* (for "Test-Accept-Reject") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- $subtree(f)$, a subgraph of G;
- $core(f)$, an edge of G or nil;
- $level(f)$, a nonnegative integer;
- $minlink(f)$, a link of G or nil;
- $rootchanged(f)$, a Boolean; and
- $testset(f)$, a subset of $V(G)$.

For each node p, there is a variable $testlink(p)$, which is either a link of G or nil.

For each link $\langle p,q\rangle$, there are associated four variables:

- $lstatus(\langle p,q\rangle)$, which takes on the values "unknown", "branch" and "rejected";
- $tarqueue_p(\langle p,q\rangle)$, a FIFO queue of messages from p to q waiting at p to be sent;
- $tarqueue_{pq}(\langle p,q\rangle)$, a FIFO queue of messages from p to q that are in the communication channel; and
- $tarqueue_q(\langle p,q\rangle)$, a FIFO queue of messages from p to q waiting at q to be processed.

The set of possible messages M is $\{\text{TEST}(l,c) : l \geq 0, c \in E(G)\} \cup \{\text{ACCEPT}, \text{REJECT}\}$.

The state also contains Boolean variables, $answered(l)$, one for each $l \in L(G)$, and Boolean variable $awake$.

In the start state of TAR, $fragments$ has one element for each node in $V(G)$; for fragment f corresponding to node p, $subtree(f) = \{p\}$, $core(f) = nil$, $level(f) = 0$, $minlink(f)$ is the minimum-weight link adjacent to p, $rootchanged(f)$ is false, and $testset(f)$ is empty. For all p, $testlink(p)$ is nil. For each link l, $lstatus(l) =$ unknown. The message queues are empty. Each $answered(l)$ is false and $awake$ is false.

The derived variable $tarqueue(\langle p,q\rangle)$ is defined to be $tarqueue_p(\langle p,q\rangle) \,||\, tarqueue_{pq}(\langle p,q\rangle) \,||\, tarqueue_q(\langle p,q\rangle)$. [1]

The derived variable $accmin(f)$ is defined as follows. If $minlink(f) \neq nil$, or if there is no external link of any $p \in nodes(f) - testset(f)$, then $accmin(f) = nil$. Otherwise, $accmin(f)$ is the minimum-weight external link of all $p \in nodes(f) - testset(f)$.

Input actions:

- $Start(p)$, $p \in V(G)$
  Effects:

[1] Given two FIFO queues q_1 and q_2, define $q_1||q_2$ to be the FIFO queue obtained by appending q_2 to the end of q_1. Obviously this operation is associative.

$awake := \text{true}$

Output actions:

- $InTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  $lstatus(\langle p,q\rangle) = \text{branch}$
  $answered(\langle p,q\rangle) = \text{false}$
  Effects:
  $answered(\langle p,q\rangle) := \text{true}$

- $NotInTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  $lstatus(\langle p,q\rangle) = \text{rejected}$
  $answered(\langle p,q\rangle) = \text{false}$
  Effects:
  $answered(\langle p,q\rangle) := \text{true}$

Internal actions (and a procedure):

- $ChannelSend(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  Preconditions:
  m at head of $tarqueue_p(\langle p,q\rangle)$
  Effects:
  dequeue($tarqueue_p(\langle p,q\rangle)$)
  enqueue($m, tarqueue_{pq}(\langle p,q\rangle)$)

- $ChannelRecv(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  Preconditions:
  m at head of $tarqueue_{pq}(\langle p,q\rangle)$
  Effects:
  dequeue($tarqueue_{pq}(\langle p,q\rangle)$)
  enqueue($m, tarqueue_q(\langle p,q\rangle)$)

- $SendTest(p)$, $p \in V(G)$
  Preconditions:
  $p \in testset(fragment(p))$
  $testlink(p) = nil$
  Effects:
  execute procedure $Test(p)$

- Procedure $Test(p)$, $p \in V(G)$

  — let $f = fragment(p)$ —

  if l, the minimum-weight link of p with $lstatus(l)$ = unknown, exists then [

  $testlink(p) := l$

  enqueue(TEST($level(f), core(f)$), $tarqueue_p(l)$)]

  else [

  remove p from $testset(f)$

  $testlink(p) := nil$]

- $ReceiveTest(\langle q,p\rangle, l, c)$, $\langle p,q\rangle \in L(G)$

  Preconditions:

  TEST(l, c) at head of $tarqueue_p(\langle q,p\rangle)$

  Effects:

  dequeue($tarqueue_p(\langle q,p\rangle)$)

  if $l > level(fragment(p))$ then

  enqueue(TEST(l, c),$tarqueue_p(\langle q,p\rangle)$)

  else

  if $c \neq core(fragment(p))$ then

  enqueue(ACCEPT, $tarqueue_p(\langle p,q\rangle)$)

  else [

  if $lstatus(\langle p,q\rangle)$ = unknown then $lstatus(\langle p,q\rangle)$:= rejected

  if $testlink(p) \neq \langle p,q\rangle$ then

  enqueue(REJECT,$tarqueue_p(\langle p,q\rangle)$)

  else execute procedure $Test(p)$]

- $ReceiveAccept(\langle q,p\rangle)$, $\langle q,p\rangle \in L(G)$

  Preconditions:

  ACCEPT at head of $tarqueue_p(\langle q,p\rangle)$

  Effects:

  dequeue($tarqueue_p(\langle q,p\rangle)$)

  $testlink(p) := nil$

  remove p from $testset(fragment(p))$

- $ReceiveReject(\langle q,p\rangle)$, $\langle q,p\rangle \in L(G)$

  Preconditions:

  REJECT at head of $tarqueue_p(\langle q,p\rangle)$

  Effects:

  dequeue($tarqueue_p(\langle q,p\rangle)$)

  if $lstatus(\langle p,q\rangle)$ = unknown then $lstatus(\langle p,q\rangle)$:= rejected

execute procedure $Test(p)$

- $ComputeMin(f)$, $f \in fragments$

  Preconditions:

  $minlink(f) = nil$

  $accmin(f) \neq nil$

  $testset(f) = \emptyset$

  Effects:

  $minlink(f) := accmin(f)$

- $ChangeRoot(f)$, $f \in fragments$

  Preconditions:

  $awake =$ true

  $rootchanged(f) =$ false

  $minlink(f) \neq nil$

  Effects:

  $rootchanged(f) :=$ true

  $lstatus(minlink(f)) :=$ branch

- $Merge(f, g)$, $f, g \in fragments$

  Preconditions:

  $f \neq g$

  $rootchanged(f) = rootchanged(g) =$ true

  $minedge(f) = minedge(g)$

  Effects:

  add a new element h to *fragments*

  $subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$

  $core(h) := minedge(f)$

  $level(h) := level(f) + 1$

  $minlink(h) := nil$

  $rootchanged(h) :=$ false

  $testset(h) := nodes(h)$

  delete f and g from *fragments*

- $Absorb(f, g)$, $f, g \in fragments$

  Preconditions:

  $rootchanged(g) =$ true

  $level(g) < level(f)$

  — let $\langle q, p \rangle = minlink(g)$ —

  $fragment(p) = f$

Effects:

$subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$
if $p \in testset(f)$ then $testset(f) := testset(f) \cup nodes(g)$
$lstatus(\langle p,q \rangle) :=$ branch
delete g from *fragments*

A message m is defined to be a *protocol message* for link $\langle p,q \rangle$ in a state if m is one of the following:
(a) a TEST message in $tarqueue(\langle p,q \rangle)$ with $lstatus(\langle p,q \rangle) \neq$ rejected.
(b) an ACCEPT message in $tarqueue(\langle q,p \rangle)$
(c) a REJECT message in $tarqueue(\langle q,p \rangle)$
(d) a TEST message in $tarqueue(\langle q,p \rangle)$ with $lstatus(\langle q,p \rangle) =$ rejected.
A protocol message for $\langle p,q \rangle$ can be considered a message that is actively helping p to discover whether $\langle p,q \rangle$ is external.

Define the following predicates on states of TAR. (All free variables are universally quantified.)

- TAR-A:

  (a) If $lstatus(\langle p,q \rangle) =$ branch, then either $(p,q) \in subtree(fragment(p))$ or *minlink* $(fragment(p)) = \langle p,q \rangle$.

  (b) If $(p,q) \in subtree(fragment(p))$, then $lstatus(\langle p,q \rangle) = lstatus(\langle q,p \rangle) =$ branch.

- TAR-B: If $lstatus(\langle p,q \rangle) =$ rejected, then $fragment(p) = fragment(q)$ and $(p,q) \notin subtree(fragment(p))$.

- TAR-C: If $testlink(p) \neq nil$, then
  (a) $testlink(p) = \langle p,q \rangle$ for some q;
  (b) $p \in testset(fragment(p))$;
  (c) there is exactly one protocol message for $\langle p,q \rangle$;
  (d) if $lstatus(\langle p,q \rangle) \neq$ branch, then $\langle p,q \rangle$ is the minimum-weight link of p with *lstatus* unknown;
  (e) if $lstatus(\langle p,q \rangle) =$ branch, then $lstatus(\langle q,p \rangle) =$ branch and $testlink(q) \neq \langle q,p \rangle$.

- TAR-D: If there is a protocol message for $\langle p,q \rangle$, then $testlink(p) = \langle p,q \rangle$.

- TAR-E: If TEST(l,c) is in $tarqueue(\langle p,q \rangle)$ then
  (a) $(p,q) \neq core(fragment(p))$;

(b) if *lstatus*($\langle p,q\rangle$) $\neq$ rejected, then c = *core*(*fragment*(p)) and l = *level*(*fragment*(p)); and

(c) if *lstatus*($\langle p,q\rangle$) = rejected, then c = *core*(*fragment*(q)) and l = *level*(*fragment*(q)).

- TAR-F: If ACCEPT is in *tarqueue*($\langle p,q\rangle$), then *fragment*(p) $\neq$ *fragment*(q) and *level* (*fragment*(p)) $\geq$ *level*(*fragment*(q)).
- TAR-G: If REJECT is in *tarqueue*($\langle p,q\rangle$), then *fragment*(p) = *fragment*(q) and *lstatus* ($\langle p,q\rangle$) $\neq$ unknown.
- TAR-H: *rootchanged*(f) is true if and only if *lstatus*(*minlink*(f)) = branch.
- TAR-I: If $p \notin$ *testset*(*fragment*(p)), then either no $\langle p,q\rangle$ has *lstatus*($\langle p,q\rangle$) = unknown, or else there is an external link $\langle r,t\rangle$ of *fragment*(p) with *level*(*fragment*(p)) $\leq$ *level*(*fragment*(t)).
- TAR-J: If *awake* = false, then *lstatus*($\langle p,q\rangle$) = unknown.

Let P_{TAR} be the conjunction of TAR-A through TAR-J.

In order to show that TAR simulates GC, we define an abstraction mapping $\mathcal{M}_3 = (\mathcal{S}_3, \mathcal{A}_3)$ from TAR to GC. Define the function $\mathcal{S}_3$ from *states*(TAR) to *states*(GC) by ignoring the message queues, and the *testlink* and *lstatus* variables. The derived variables *accmin* of TAR map to the (non-derived) variables *accmin* of GC. Define the function $\mathcal{A}_3$ as follows. Let s be a state of TAR and π an action of TAR enabled in s. The GC action *TestNode*(p) is simulated in TAR when p receives the message that tells p either that this link is external or that p has no external links.

- If π = *ReceiveAccept*($\langle q,p\rangle$), then $\mathcal{A}_3(s,\pi)$ = *TestNode*(p).
- If π = *SendTest*(p) or *ReceiveReject*($\langle q,p\rangle$), then $\mathcal{A}_3(s,\pi)$ = *TestNode*(p) if there is no link $\langle p,r\rangle$, $r \neq q$, with *lstatus*($\langle p,r\rangle$) = unknown in s; otherwise, $\mathcal{A}_3(s,\pi)$ is empty.
- If π = *ReceiveTest*($\langle q,p\rangle, l, c$), then $\mathcal{A}_3(s,\pi)$ = *TestNode*(p) if $l \leq$ *level*(*fragment*(p)), c = *core*(*fragment*(p)), *testlink*(p) = $\langle p,q\rangle$, and there is no link $\langle p,r\rangle$, $r \neq q$, with *lstatus*($\langle p,r\rangle$) = unknown in s; otherwise, $\mathcal{A}_3(s,\pi)$ is empty.
- If π = *ChannelSend*($\langle p,q\rangle, m$) or *ChannelRecv*($\langle p,q\rangle, m$), then $\mathcal{A}_3(s,\pi)$ is empty.

- For all other values of π, $\mathcal{A}_3(s, \pi) = \pi$.

The following predicates are true in every state of TAR satisfying $(P'_{GC} \circ \mathcal{S}_3) \wedge P_{TAR}$. Recall that $P'_{GC} = (P'_{COM} \circ \mathcal{S}_2) \wedge P_{GC}$. If $P'_{GC}(\mathcal{S}_3(s))$ is true, then the GC predicates are true in $\mathcal{S}_3(s)$, the COM predicates are true in $\mathcal{S}_2(\mathcal{S}_3(s))$, and the HI predicates are true in $\mathcal{S}_1(\mathcal{S}_2(\mathcal{S}_3(s)))$. Thus, these predicates are derivable from P_{TAR}, together with the HI, COM and GC predicates.

- TAR-K: If $testlink(p) = \langle p, q \rangle$, then $lstatus(\langle p, q \rangle) \neq$ rejected.

Proof: By TAR-C(d) and TAR-C(e).

- TAR-L: If $minlink(f) = nil$ and l is an external link of f, then $lstatus(l) =$ unknown.

Proof: By TAR-A(a), if $lstatus(l) =$ branch, then l is internal. By TAR-B, if $lstatus(l) =$ rejected, then l is internal. □

- TAR-M: If $\text{TEST}(l, c)$ is in $tarqueue(\langle p, q \rangle)$, then $l \geq 1$ and $c \neq nil$.

Proof: Let $f = fragment(p)$ and $g = fragment(q)$.
1. $\text{TEST}(l, c)$ is in $tarqueue(\langle p, q \rangle)$, by assumption.

Case 1: $lstatus(\langle p, q \rangle) \neq$ rejected.
2. $lstatus(\langle p, q \rangle) \neq$ rejected, by assumption.
3. $c = core(f)$ and $l = level(f)$, by Claim 2 and TAR-E(b).
4. $testlink(p) = \langle p, q \rangle$, by Claims 1 and 2 and TAR-D.
5. $p \in testset(f)$, by Claim 4 and TAR-C(b).
6. $minlink(f) = nil$, by Claim 5 and GC-C.
7. $subtree(f) \neq \{p\}$, by Claim 6 and COM-E.
8. $core(f) \neq nil$ and $level(f) \neq 0$, by Claim 7 and COM-F.
9. $level(f) \geq 1$, by Claim 8 and COM-F.
10. $c \neq nil$ and $l \geq 1$, by Claims 3, 8 and 9.

Case 2: $lstatus(\langle p, q \rangle) =$ rejected.
11. $lstatus(\langle p, q \rangle) =$ rejected, by assumption.
12. $c = core(g)$ and $l = level(g)$, by Claim 11 and TAR-E(c).
13. $testlink(q) = \langle q, p \rangle$, by Claims 1 and 11 and TAR-D.
14. $q \in testset(g)$, by Claim 13 and TAR-C(b).
15. $minlink(g) = nil$, by Claim 14 and GC-C.
16. $subtree(g) \neq \{q\}$, by Claim 15 and COM-E.

17. $core(g) \neq nil$ and $level(g) \neq 0$, by Claim 16 and COM-F
18. $level(g) \geq 1$, by Claim 17 and COM-F.
19. $c \neq nil$ and $l \geq 1$, by Claims 12, 17 and 18. □

- TAR-N: If $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p\rangle)$ and $c = core(fragment(p))$, then $fragment(p) = fragment(q)$.

Proof:

1. $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p\rangle)$, by assumption.
2. $c = core(fragment(p))$, by assumption.
3. $c \neq nil$, by Claim 1 and TAR-M.
4. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $c = core(fragment(q))$, by TAR-E(b).
5. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $fragment(q) = fragment(p)$, by Claims 2, 3 and 4, and COM-F.
6. If $lstatus(\langle q, p\rangle) =$ rejected, then $fragment(q) = fragment(p)$, by TAR-B. □

- TAR-O: If $minlink(f) \neq nil$, then there is no protocol message for any link of any node in $nodes(f)$.

Proof:

1. $minlink(f) \neq nil$, by assumption.
2. $testset(f) = \emptyset$, by Claim 1 and GC-C.
3. $testlink(p) = nil$ for all $p \in nodes(f)$, by Claim 2 and TAR-C(b).
4. There is no protocol message for any link $\langle p, q\rangle$, $p \in nodes(f)$, by Claim 3 and TAR-D. □

- TAR-P: If $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p\rangle)$, $c = core(fragment(p))$, $testlink(p) = \langle p, q\rangle$, and $lstatus(\langle q, p\rangle) \neq$ rejected, then a $\text{TEST}(l', c')$ message is in $tarqueue(\langle p, q\rangle)$ and $lstatus(\langle p, q\rangle) =$ unknown.

Proof:

1. $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p\rangle)$, by assumption.
2. $c = core(fragment(p))$, by assumption.
3. $testlink(p) = \langle p, q\rangle$, by assumption.
4. $lstatus(\langle q, p\rangle) \neq$ rejected, by assumption.
5. $fragment(p) = fragment(q)$, by Claims 1 and 2 and TAR-N.
6. No ACCEPT message is in $tarqueue(\langle q, p\rangle)$, by Claim 5 and TAR-F.
7. The $\text{TEST}(l, c)$ message in $tarqueue(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by Claim 4.
8. $testlink(q) = \langle q, p\rangle$, by Claim 7 and TAR-D.

9. $lstatus(\langle q,p\rangle) \neq$ branch, by Claims 3, 8 and TAR-C(e).
10. $lstatus(\langle q,p\rangle) =$ unknown, by Claims 4 and 9.
11. No REJECT message is in $tarqueue(\langle q,p\rangle)$, by Claim 10 and TAR-G.
12. There is exactly one protocol message for $\langle p,q\rangle$, by Claim 3 and TAR-C(c).
13. A TEST(l',c') message is in $tarqueue(\langle p,q\rangle)$ and $lstatus(\langle p,q\rangle) \neq$ rejected, by Claims 6, 7, 11 and 12.
14. $lstatus(\langle p,q\rangle) \neq$ branch, by Claims 3 and 8 and TAR-C(e).
15. $lstatus(\langle p,q\rangle) =$ unknown, by Claims 13 and 14.

Claims 13 and 15 give the result. □

Lemma 17: *TAR simulates GC via $\mathcal{M}_3$, P_{TAR}, and P'_{GC}.*

Proof: By inspection, the types of TAR, GC, $\mathcal{M}_3$, and P_{TAR} are correct. By Corollary 16, P'_{GC} is a predicate true in every reachable state of COM.

(1) Let s be in $start(TAR)$. Obviously, P_{TAR} is true in s, and $\mathcal{S}_3(s)$ is in $start(GC)$.

(2) Obviously, $\mathcal{A}_3(s,\pi)|ext(GC) = \pi|ext(TAR)$.

(3) Let (s',π,s) be a step of TAR such that P'_{GC} is true of $\mathcal{S}_3(s')$ and P_{TAR} is true of s'. Condition (3a) is only shown below for those predicates that are not obviously true in s.

i) π is ChannelSend($\langle$p,q$\rangle$,m) or ChannelRecv($\langle$p,q$\rangle$,m). $\mathcal{A}_3(s',\pi)$ is empty. (3a) and (3b) are obviously true.

ii) π is Start(p) or InTree(l) or NotInTree(l).

(3c) $\mathcal{A}_3(s',\pi) = \pi$. If $\pi = InTree(l)$, then by TAR-J and TAR-A(a), π is enabled in $\mathcal{S}_3(s')$. If $\pi = NotInTree(l)$, then by TAR-J and TAR-B, π is enabled in $\mathcal{S}_3(s')$. Thus, $\mathcal{S}_3(s')\pi\mathcal{S}_3(s)$ is an execution fragment of GC.

(3a) Obviously, P_{TAR} is still true in s.

iii) π is SendTest(p). Let $f = fragment(p)$ in s'.

Case 1: There is a link $\langle p,q\rangle$ with $lstatus(\langle p,q\rangle) =$ unknown in s'.

(3b) $\mathcal{A}_3(s',\pi)$ is empty. It is easy to see that $\mathcal{S}_3(s') = \mathcal{S}_3(s)$.

(3a) By TAR-D and precondition that $testlink(p) = nil$, there is no protocol message for any link of p in s'.

TAR-C(c): In s, there is exactly one protocol message for $\langle p, q\rangle$, namely the TEST message in $tarqueue(\langle p, q\rangle)$.

TAR-D: The TEST message added in s is a protocol message for $\langle p, q\rangle$, and is not a protocol message for any other link. By the code, $testlink(p) = \langle p, q\rangle$.

TAR-E(a): By TAR-A(b), $(p, q) \notin subtree(f)$. By COM-F, $(p, q) \neq core(f)$.

Case 2: There is no link $\langle p, q\rangle$ with $lstatus(\langle p, q\rangle) =$ unknown in s'.

(3c) $\mathcal{A}_3(s', \pi) = TestNode(p)$.

Claims about s':

1. $p \in testset(f)$, by precondition.
2. $minlink(f) = nil$, by Claim 1 and GC-C.
3. There is no external link of p, by Claim 2, TAR-L, and assumption.

By Claims 1 and 3, $TestNode(p)$ is enabled in $\mathcal{S}_3(s')$.

Claims about s:

4. $p \notin testset(f)$, by code.
5. There is no external link of p, by Claim 3 and code.
6. $accmin(f)$ does not change, by Claim 5.

By Claims 4, 5, and 6, the effects of $TestNode(p)$ are mirrored in $\mathcal{S}_3(s)$.

(3a) TAR-I: By assumption for Case 2, p has no unknown links in s', and the same is true in s.

iv) π is ReceiveTest(⟨q,p⟩,l,c). Let $f = fragment(p)$ in s'.

Case 1: $l \leq level(f)$, $c = core(f)$, $testlink(p) = \langle p, q\rangle$, and there is no link $\langle p, r\rangle$, $r \neq q$, with $lstatus(\langle p, r\rangle) =$ unknown in s'.

(3c) $\mathcal{A}_3(s', \pi) = TestNode(p)$.

Claims about s':

1. $c = core(f)$, by assumption.
2. $testlink(p) = \langle p, q \rangle$, by assumption.
3. There is no link $\langle p, r \rangle$, $r \neq q$, with $lstatus(\langle p, r \rangle) =$ unknown, by assumption.
4. $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p \rangle)$, by preconditions.
5. $p \in testset(f)$, by Claim 2 and TAR-C(b).
6. $minlink(f) = nil$, by Claim 5 and GC-C.
7. No link $\langle p, r \rangle$, $r \neq q$, is external, by Claims 6 and 3 and TAR-L.
8. $\langle p, q \rangle$ is not external, by Claims 2, 3 and 4 and TAR-N.

By Claims 5, 7 and 8, *TestNode*(p) is enabled in s'.

Claims about s:

9. $p \notin testset(f)$, by code.
10. There is no external link of p, by Claims 7 and 8 and code.
11. $accmin(f)$ does not change, by Claim 10.

By Claims 9, 10 and 11, the effects of *TestNode*(p) are mirrored in s.

(3a) TAR-B: The only case of interest is when $lstatus(\langle p, q \rangle)$ changes from unknown in s' to rejected in s. By TAR-N, $f = fragment(q)$ in s' and the same is still true in s. By TAR-A(b), $(p, q) \notin subtree(f)$ in s', and the same is still true in s.

TAR-D:

Claims about s′:

1. $\text{TEST}(l, c)$ is in $tarqueue(\langle q, p \rangle)$, by precondition.
2. $c = core(f)$, by assumption.
3. $testlink(p) = \langle p, q \rangle$, by assumption.
4. There is exactly one protocol message for $\langle p, q \rangle$, by Claim 3 and TAR-C(c).
5. There is no protocol message for any link $\langle p, r \rangle$, $r \neq q$, by Claim 3 and TAR-D.

Case A: $lstatus(\langle q, p \rangle) =$ rejected. The $\text{TEST}(l, c)$ message in $tarqueue(\langle q, p \rangle)$ is the protocol message for $\langle p, q \rangle$ in s'. Since it is removed in s, by Claims 4 and 5 there is no protocol message for any link of p in s. Concerning q: by TAR-K, $testlink(q) \neq \langle q, p \rangle$; thus, the predicate is still true for q in s, even if $lstatus(\langle p, q \rangle)$ is changed to rejected.

Case B: $lstatus(\langle q, p \rangle) \neq$ rejected.

6. A $\text{TEST}(l', c')$ is in $tarqueue(\langle p, q \rangle)$ and $lstatus(\langle p, q \rangle) =$ unknown, by Claims 1, 2, 3, assumptions for Case B, and TAR-P.

7. $testlink(q) = \langle q, p\rangle$, by Claim 1, assumption for Case B and TAR-D.

In s, the TEST(l', c') message in $tarqueue(\langle p, q\rangle)$, which exists by Claim 6, becomes a protocol message for $\langle q, p\rangle$, since $lstatus(\langle p, q\rangle)$ is changed to rejected. By Claim 7, $testlink(q)$ has the correct value. By Claims 4 and 5, the predicate is vacuously true for p in s.

TAR-E(c): The only case of interest is when $lstatus(\langle p, q\rangle)$ goes from unknown in s' to rejected in s, while there is a TEST(l', c') message in $tarqueue(\langle p, q\rangle)$. By TAR-E(b), $c' = core(f)$ and $l' = level(f)$ in s'. By TAR-N, $fragment(q) = f$. Thus $c' = core(fragment(q))$ and $l' = level(fragment(q))$.

TAR-I: By the assumption for Case 1 and code, p has no unknown links in s.

TAR-J: The TEST message in $tarqueue(\langle q, p\rangle)$ is a protocol message for either $\langle p, q\rangle$ or $\langle q, p\rangle$. Without loss of generality, suppose for $\langle p, q\rangle$. By TAR-D, $testlink(p) = \langle p, q\rangle$, and by TAR-C(b), $p \in testset(f)$. Thus, by GC-C, $minlink(f) = nil$, and by COM-C $awake =$ true.

Case 2: $l > level(f)$, or $c \neq core(f)$, or $testlink(p) \neq \langle p, q\rangle$, or there is a link $\langle p, r\rangle$, $r \neq q$, with $lstatus(\langle p, r\rangle) =$ unknown in s'.

(3b) $\mathcal{A}_3(s', \pi)$ is empty. The only variables that are possibly changed are $lstatus(\langle p, q\rangle)$, $tarqueue$'s, and $testlink(p)$, none of which is reflected (directly) in the state of GC. Thus $accmin(f)$ does not change and $\mathcal{S}_3(s') = \mathcal{S}_3(s)$.

(3a) TAR-B: As in Case 1.

TAR-C(b): If $testlink(p) \neq nil$ in s, then by inspecting the code, the same is true in s'. So the predicate is true in s because it is true in s'.

TAR-C(c): If $l > level(f)$ in s', nothing affecting the predicate changes in going from s' to s. Suppose $l \leq level(f)$ in s'.

Claims about s':

1. TEST(l, c) is in $tarqueue(\langle q, p\rangle)$, by precondition.

Case A: $c \neq core(f)$.

2. $lstatus(\langle q, p\rangle) \neq$ rejected, by TAR-E(c).

3. The TEST(l, c) message in *tarqueue*$(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by Claim 2.

The ACCEPT message added in s is a protocol message for $\langle q, p\rangle$. There is no change that affects the truth of the predicate for p.

Case B: $c = core(f)$.

Case B.1: $testlink(p) \neq \langle p, q\rangle$.

4. There is no protocol message for $\langle p, q\rangle$, by TAR-D.
5. The TEST(l, c) message in *tarqueue*$(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by Claim 4.

The REJECT message added in s is a protocol message for $\langle q, p\rangle$. No change affects the truth of the predicate for p.

Case B.2: $testlink(p) = \langle p, q\rangle$.

6. There is a link $\langle p, r\rangle$, $r \neq q$, with $lstatus(\langle p, r\rangle) =$ unknown, by assumption for Case B.2.
7. There is no protocol message for $\langle p, r\rangle$, by Claim 6 and TAR-D.

Case B.2.1: $lstatus(\langle q, p\rangle) \neq$ rejected.

8. There is a TEST(l', c') message in *tarqueue*$(\langle p, q\rangle)$ and $lstatus(\langle p, q\rangle) =$ unknown, by assumptions for Case B.2.1 and TAR-P.
9. The TEST(l, c) message in *tarqueue*$(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by assumptions for Case B.2.1.

The TEST(l', c') message of Claim 8 becomes a protocol message for $\langle q, p\rangle$ in s, since $lstatus(\langle p, q\rangle)$ is changed to rejected. Concerning p: $testlink(p) = \langle p, r\rangle$ in s, and a TEST message is added to *tarqueue*$(\langle p, r\rangle)$ and is the sole protocol message for $\langle p, r\rangle$ by Claim 7.

Case B.2.2 $lstatus(\langle q, p\rangle) =$ rejected.

10. The TEST(l, c) message in *tarqueue*$(\langle q, p\rangle)$ is the protocol message for $\langle p, q\rangle$, by assumptions for Case B.2.2.
11. $testlink(q) \neq \langle q, p\rangle$, by assumption for Case B.2.2 and TAR-K.

The predicate is true for p in s because the TEST(l, c) message, which was the sole protocol message for $\langle p, q\rangle$ by Claim 10, is removed in s; $testlink(p)$ is now $\langle p, r\rangle$, and $\langle p, r\rangle$ has exactly one protocol message, by inspecting the code. No change is made that affects the truth of the predicate for q, by Claim 11.

TAR-D: If $l > level(f)$ in s', nothing affecting the predicate changes in going from s' to s. Suppose $l \leq level(f)$ in s'.

Claims about s'*:*

1. TEST(l, c) is in $tarqueue(\langle q, p\rangle)$, by precondition.

Case A: $c \neq core(f)$.

2. $lstatus(\langle q, p\rangle) \neq$ rejected, by assumption for Case A and TAR-E(c).
3. $testlink(q) = \langle q, p\rangle$, by Claims 1 and 2 and TAR-D.

Then $testlink(q)$ is still $\langle q, p\rangle$ in s, and there is an ACCEPT message in $tarqueue(\langle p, q\rangle)$. No change affects the truth of the predicate for p.

Case B: $c = core(f)$.

Case B.1: $testlink(p) \neq \langle p, q\rangle$.

4. The TEST(l, c) message in $tarqueue(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by assumptions for Case B.1 and TAR-D.
5. $testlink(q) = \langle q, p\rangle$, by Claim 4 and TAR-D.

Then in s, there is a REJECT message in $tarqueue(\langle p, q\rangle)$ and $testlink(q)$ is still $\langle q, p\rangle$. No change affects the truth of the predicate for p.

Case B.2: $testlink(p) = \langle p, q\rangle$.

6. There is a link $\langle p, r\rangle$, $r \neq q$, with $lstatus(\langle p, r\rangle) =$ unknown, by assumption for Case 2.
7. There is exactly one protocol message for $\langle p, q\rangle$, by TAR-C(c).

Case B.2.1: $lstatus(\langle q, p\rangle) =$ rejected.

8. $testlink(q) \neq \langle q, p\rangle$, by TAR-K.

No changes affect the truth of the predicate for q. For p: The TEST(l, c) message in $tarqueue(\langle q, p\rangle)$ is the protocol message for $\langle p, q\rangle$. It is removed in s. A TEST message is added to $tarqueue(\langle p, r\rangle)$ in s, where $lstatus(\langle p, r\rangle) =$ unknown, and $testlink(p) = \langle p, r\rangle$ by code.

Case B.2.2: $lstatus(\langle q,p\rangle) \neq$ rejected.

9. A TEST(l',c') message is in $tarqueue(\langle p,q\rangle)$ and $lstatus(\langle p,q\rangle) =$ unknown, by Claim 1, the assumption for Case B.2.2 and TAR-P.
10. $testlink(q) = \langle q,p\rangle$, by Claim 8 and TAR-D.

For q: In s, since $lstatus(\langle q,p\rangle)$ is changed to rejected, the TEST(l',c') message in $tarqueue(\langle p,q\rangle)$ (of Claim 9) becomes a protocol message for $\langle q,p\rangle$. This is OK by Claim 10.

For p: The TEST(l',c') message of Claim 9 is the protocol message for $\langle p,q\rangle$. The rest of the argument is as in Case B.2.1.

TAR-E: (a) Suppose a TEST message is added to $tarqueue(\langle p,r\rangle)$. As in $\pi = SendTest(p)$, Case 1. (c) As in Case 1.

TAR-F: The only case of interest is when an ACCEPT message is added to $tarqueue(\langle p,q\rangle)$ in s.

Claims about s':

1. TEST(l,c) is in $tarqueue(\langle q,p\rangle)$, by precondition.
2. $l \leq level(f)$, by assumption.
3. $c \neq core(f)$, by assumption.
4. $lstatus(\langle q,p\rangle) \neq$ rejected, by Claims 1 and 3 and TAR-E(c).
5. $c = core(fragment(q))$, by Claims 1, 4 and TAR-E(b).
6. $l = level(fragment(q))$, by Claims 1, 4 and TAR-E(b).
7. $core(f) \neq core(fragment(q))$, by Claims 3 and 5.
8. $level(f) \leq level(fragment(q))$, by Claims 2 and 6.

Claims 7 and 8 are still true in s.

TAR-G: The only case of interest is when a REJECT message is added to $tarqueue(\langle p,q\rangle)$.

Claims about s':

1. TEST(l,c) is in $tarqueue(\langle q,p\rangle)$, by precondition.
2. $c = core(f)$, by assumption.
3. $testlink(p) \neq \langle p,q\rangle$, by assumption.
4. If $lstatus(\langle q,p\rangle) \neq$ rejected, then $c = core(fragment(q))$, by Claim 1 and TAR-E(b).

5. If $lstatus(\langle q,p \rangle) \neq$ rejected, then $f = fragment(q)$, by Claim 4 and COM-F.
6. If $lstatus(\langle q,p \rangle) =$ rejected, then $f = fragment(q)$, by TAR-B.
7. $f = fragment(q)$, by Claims 5 and 6.

Claim 7 is still true in s.

TAR-I: The only case of interest is when p is removed from $testset(f)$. But when that happens, there are no unknown links of p.

TAR-J: Suppose $lstatus(\langle p,q \rangle)$ is changed to rejected. As in Case 1.

v) π is ReceiveAccept(⟨q,p⟩). Let $f = fragment(p)$ in s'.

(3c) $\mathcal{A}_3(s', \pi) = TestNode(p)$.

Claims about s':

1. ACCEPT is in $tarqueue(\langle q,p \rangle)$, by precondition.
2. $fragment(q) \neq f$, by Claim 1 and TAR-F.
3. $level(f) \leq level(fragment(q))$, by Claim 1 and TAR-F.
4. $\langle p,q \rangle$ is an external link of f, by Claim 2.
5. $testlink(p) = \langle p,q \rangle$, by Claim 1 and TAR-D.
6. $p \in testset(f)$, by Claim 5 and TAR-C(b).
7. $minlink(f) = nil$, by Claim 6 and GC-C.
8. $lstatus(\langle p,q \rangle) \neq$ branch, by Claims 4 and 7 and TAR-L.
9. $\langle p,q \rangle$ is the minimum-weight link of p with $lstatus$ unknown, by Claims 5 and 8 and TAR-C(d).
10. $\langle p,q \rangle$ is the minimum-weight external link of p, by Claims 7 and 9 and TAR-L.

By Claims 6, 10, and 3, $TestNode(p)$ is enabled in s'.

Claims about s:

11. $p \notin testset(f)$, by code.
12. $\langle p,q \rangle$ is the minimum-weight external link of p, by Claim 10.
13. If $wt(p,q) < wt(accmin(f))$ in s', then $accmin(f) = \langle p,q \rangle$ in s, by Claims 11 and 12.

By Claims 11 and 13, the effects of $TestNode(p)$ are mirrored in s.

(3a) TAR-D: In s', ACCEPT in *tarqueue*$(\langle q,p\rangle)$ is a protocol message for $\langle p,q\rangle$. By TAR-C(c) and TAR-D, it is the only protocol message for any link of p in s'. Thus in s, there is no protocol message for any link of p, and the predicate is vacuously true in s for p. No other node is affected.

TAR-I: By Claims 3 and 4, it is OK to remove p from *testset*(f).

vi) π is ReceiveReject(⟨q,p⟩). Let $f = \mathit{fragment}(p)$ in s'.

Case 1: There is a link $\langle p,r\rangle$, $r \neq q$, with *lstatus*$(\langle p,r\rangle) =$ unknown.

(3b) $\mathcal{A}_3(s',\pi)$ is empty. Obviously $\mathcal{S}_3(s') = \mathcal{S}_3(s)$.

(3a) *Claims about s':*

1. REJECT is in *tarqueue*$(\langle q,p\rangle)$, by assumption.
2. The REJECT in *tarqueue*$(\langle q,p\rangle)$ is a protocol message for $\langle p,q\rangle$, by Claim 1.
3. *testlink*$(p) = \langle p,q\rangle$, by Claim 2 and TAR-D.
4. There is only one protocol message for $\langle p,q\rangle$, by Claim 3 and TAR-C(c).
5. There is no protocol message for any other link of p, by Claim 3 and TAR-D.
6. $p \in \mathit{testset}(f)$, by Claim 3 and TAR-C(b).

TAR-B: Suppose *lstatus*$(\langle p,q\rangle)$ goes from unknown in s' to rejected in s. By TAR-G, $f = \mathit{fragment}(q)$ in s'. By TAR-A(b), $(p,q) \notin \mathit{subtree}(f)$ in s'. Both facts are still true in s.

TAR-C(b): By Claim 6.

TAR-C(c): In s, *testlink*$(p) = \langle p,r\rangle$, and the TEST message is the sole protocol message for $\langle p,r\rangle$ by Claim 5.

TAR-D: In s, the REJECT message is removed and a TEST message is added to *tarqueue*$(\langle p,r\rangle)$ with *lstatus*$(\langle p,r\rangle) =$ unknown. So there is a protocol message for $\langle p,r\rangle$ and no other link of p by Claims 4 and 5. By code, *testlink*$(p) = \langle p,r\rangle$.

TAR-E(a): Suppose a TEST messge is added to some *tarqueue*$(\langle p,r\rangle)$. As in $\pi = \mathit{SendTest}(p)$, Case 1.

TAR-E(c): The only case of interest is when *lstatus*$(\langle p,q\rangle)$ goes from unknown in s' to rejected in s. But by Claims 2 and 4, there is no TEST message in *tarqueue*$(\langle p,q\rangle)$ in s' if *lstatus*$(\langle p,q\rangle) =$ unknown.

TAR-I: By Claim 6, the predicate is vacuously true.

TAR-J: Suppose *lstatus*$(\langle p, q \rangle)$ is changed from unknown to rejected. Similar to $\pi = ReceiveTest(\langle q, p \rangle, l, c)$, Case 1, with REJECT being the protocol message for $\langle p, q \rangle$.

Case 2: There is no link $\langle p, r \rangle$, $r \neq q$, with *lstatus*$(\langle p, r \rangle) =$ unknown.

(3c) $\mathcal{A}_3(s', \pi) = TestNode(p)$.

Claims about s':

1. REJECT is in *tarqueue*$(\langle q, p \rangle)$, by precondition.
2. *testlink*$(p) = \langle p, q \rangle$, by Claim 1 and TAR-D.
3. $p \in testset(f)$, by Claim 2 and TAR-C(b)
4. *minlink*$(f) = nil$, by Claim 3 and GC-C.
5. *fragment*$(q) = f$, by Claim 1 and TAR-G.
6. $\langle p, q \rangle$ is not external, by Claim 5.
7. There is no external link $\langle p, r \rangle$, $r \neq q$, of p, by Claim 4, TAR-L, and assumption for Case 2.

By Claims 3, 6 and 7, *TestNode*(p) is enabled in s'.

Claims about s:

8. $p \notin testset(f)$, by code.
9. There is no external link of p, by Claims 6 and 7 and code.
10. *accmin*(f) does not change, by Claim 9.

By Claims 8, 9 and 10, the effects of *TestNode*(p) are mirrored in s.

(3a) TAR-B: Same as Case 1.

TAR-D: In s, *testlink*$(p) = nil$. We must show there is no protocol message for any link of p. In s', the REJECT message in *tarqueue*$(\langle q, p \rangle)$ is the sole protocol message for any link of p, as in Case 1. The REJECT message is removed in s and no protocol message is added.

TAR-E(c): As in Case 1.

TAR-I: By assumption for Case 2 and code, there are no unknown links of p in s.

TAR-J: As in Case 1.

vii) π is ComputeMin(f).

(3c) $\mathcal{A}_3(s', \pi) = \pi$. Since $accmin(f) = nil$ in s because $minlink(f) = nil$ in s, it is easy to see that π is enabled in $\mathcal{S}_3(s')$ and that its effects are mirrored in $\mathcal{S}_3(s)$.

(3a) TAR-H: By GC-A, $accmin(f) = l$ is an external link of f in s'. Since $minlink(f) = nil$ in s', $lstatus(l) \neq$ branch by TAR-A(a). Also, by COM-B, $rootchanged(f) =$ false in s'. Thus in s, $rootchanged(f) =$ false and $lstatus(minlink(f)) \neq$ branch.

viii) π is ChangeRoot(f).

(3c) $\mathcal{A}_3(s', \pi) = \pi$. It is easy to see that π is enabled in $\mathcal{S}_3(s')$ and that its effects are mirrored in $\mathcal{S}_3(s)$.

(3a) Only TAR-A(a), TAR-H and TAR-J are affected. Obviously TAR-A(a) and TAR-H are still true in s. For TAR-J: by precondition $awake =$ true in s', and is still true in s.

ix) π is Merge(f,g).

(3c) $\mathcal{A}_3(s', \pi) = \pi$. After noting that $accmin(h) = nil$ in s because $testset(h) = nodes(h)$ in s, it is easy to see that π is enabled in $\mathcal{S}_3(s')$ and that its effects are mirrored in $\mathcal{S}_3(s)$.

(3a) TAR-A(b): The predicate is true for h by TAR-H.

TAR-B: The predicate is true for h by TAR-H.

TAR-C: By GC-C, no r in $nodes(f)$ or $nodes(g)$ is in $testset(f)$ or $testset(g)$ in s'. By TAR-C(b), $testlink(r) = nil$ for all such r. So the predicate is vacuously true in h.

TAR-E(a): By TAR-O, there is no TEST message in $tarqueue(\langle p,q\rangle)$ or in $tarqueue(\langle q,p\rangle)$, where $\langle p,q\rangle = minlink(f)$, in s'. Since $(p,q) = core(h)$ in s, done.

TAR-E(b): By TAR-O, there is no TEST(l,c) message in $tarqueue(\langle p,q\rangle)$ with $lstatus(\langle p,q\rangle) \neq$ rejected in s', for any p in $nodes(f)$ or $nodes(g)$. Thus, the same is true in s for any p in $nodes(h)$, and the predicate is vacuously true in s for h.

TAR-E(c): If TEST(l, c) is in *tarqueue*$(\langle p, q\rangle)$ and *lstatus*$(\langle p, q\rangle)$ = rejected in s', then it is a protocol message for $\langle q, p\rangle$ in s'. By TAR-O, *fragment*(q) is neither f nor g in s'. So the predicate is still true in s.

TAR-F: If ACCEPT is in *tarqueue*$(\langle p, q\rangle)$ in s', it is a protocol message for $\langle q, p\rangle$ in s'. By TAR-O, *fragment*(q) is neither f nor g in s'. If *fragment*(p) is neither f nor g in s', then the predicate is still true in s. Without loss of generality, suppose *fragment*$(p) = f$ in s'. By TAR-F, *level*$(f) \geq$ *level*(*fragment*(q)) in s'. Then *fragment*$(p) = h \neq$ *fragment*(q) in s, and *level*(h) (in s) $>$ *level*(f) (in s') $\geq$ *level*(*fragment*(q)) (in s' and s).

TAR-H: By code, *rootchanged*(h) = false. Since *minlink*$(h) = nil$ by code, *lstatus* (*minlink*(f)) $\neq$ branch.

TAR-I: For nodes in h, the predicate is vacuously true since *testset*(h) = *nodes*(h). For nodes not in h, the predicate is still true since the level of every node formerly in *nodes*(f) or *nodes*(g) is increased.

x) π is Absorb(f,g).

(3c) $\mathcal{A}_3(s', \pi) = \pi$. It is easy to see that π is enabled in $\mathcal{S}_3(s')$. Below we show that *accmin*(f) is the same in s as in s', which together with inspecting the code, shows that the effects of π are mirrored in $\mathcal{S}_3(s)$.

Let $\langle q, p\rangle$ = *minlink*(g). If $p \in$ *testset*(f) in s', then every node in *nodes*(g) in s' is added to *testset*(f) in s. No change is made to any of the criteria for defining *accmin*(f).

Suppose $p \notin$ *testset*(f) in s'. If *minlink*$(f) \neq nil$ in s', then the same is true in s, and *accmin*$(f) = nil$ in s' and s. Suppose *minlink*$(f) = nil$ in s'.

Claims about s':

1. *level*$(f) <$ *level*(g), by precondition.
2. $p \in$ *nodes*(f), by precondition.
3. $p \notin$ *testset*(f), by assumption.
4. *minlink*$(f) = nil$, by assumption.
5. $q \in$ *nodes*(g), by COM-A.
6. $f \neq g$, by Claim 1.
7. *accmin*$(f) = \langle r, t\rangle$, for some r and t, by Claims 2 through 6.
8. *fragment*$(t) \neq g$, by Claims 1 and 7 and GC-A.

9. $\langle r,t\rangle \neq \langle p,q\rangle$, by Claims 5 and 8.
10. $wt(r,t) < wt(p,q)$, by Claims 2, 3, 5, 6, 7, and 9 and GC-A.
11. $wt(p,q) \leq wt(u,v)$ for any external link $\langle u,v\rangle$ of g, by COM-A.
12. $wt(r,t) < wt(u,v)$ for any external link $\langle u,v\rangle$ of g, by Claims 10 and 11.

By Claims 7, 8 and 12, $accmin(f) = \langle r,t\rangle$ in s.

(3a) TAR-A(b): The predicate is true in s for f by TAR-H.

TAR-B: The predicate is true in s for f by TAR-H.

TAR-C(b): By GC-C, since $minlink(g) \neq nil$, $testset(g) = \emptyset$ in s'. By TAR-C(b), $testlink(p) = nil$ in s' for all $p \in nodes(g)$. There is no change for $p \in nodes(f)$ in s' in going from s' to s. Thus the predicate is true in s for f.

TAR-C(e): Suppose $\langle q,p\rangle = minlink(g)$ in s' and $lstatus(\langle p,q\rangle)$ becomes branch in s. By TAR-H, $lstatus(\langle q,p\rangle) =$ branch in s'. As in TAR-C(b), $testlink(q) \neq \langle q,p\rangle$, so the predicate is still true in s.

TAR-E(a): OK because $core(f)$ does not change.

TAR-E(b): Let $\langle q,p\rangle = minlink(g)$ in s'. If we can show $lstatus(\langle p,q\rangle) \neq$ rejected in s', we'd be done. If $lstatus(\langle p,q\rangle) =$ rejected in s', then $fragment(p) = fragment(q)$. This contradicts $level(g) < level(f)$, which implies that $g \neq f$.

TAR-E(c): Suppose TEST(l,c) is in $tarqueue(\langle p,q\rangle)$ and $lstatus(\langle p,q\rangle) =$ rejected in s', for some link $\langle p,q\rangle$ in $L(G)$. This is a protocol message for $\langle q,p\rangle$. By TAR-O, $fragment(q) \neq g$ in s'. Thus $fragment(q)$ is the same in s' and s, and $c = core(fragment(q))$ and $l = level(fragment(q))$ in s.

TAR-F: Suppose ACCEPT is in $tarqueue(\langle p,q\rangle)$ in s', for some link $\langle p,q\rangle$ in $L(G)$. This is a protocol message for $\langle q,p\rangle$. By TAR-O, $fragment(q) \neq g$ in s'. By TAR-F, $fragment(p) \neq fragment(q)$ in s'. By preconditions, $level(g) < level(f)$, so it cannot be the case that $fragment(p) = g$ and $fragment(q) = f$.

Suppose $fragment(p) = g$. Since $level(fragment(p))$ in s is greater than it is in s', and since $fragment(q) \neq f$ in s', the predicate is still true in s.

Suppose $fragment(q) = f$. Since $fragment(q)$ is the same in s as in s', and since $fragment(p) \neq g$ in s', the predicate is still true in s.

If $fragment(p) \neq g$ and $fragment(q) \neq f$ in s', the predicate is obviously still true in s.

TAR-G: Suppose REJECT is in $tarqueue(\langle p, q\rangle)$ in s', for some link $\langle p, q\rangle$ in $L(G)$. This is a protocol message for $\langle q, p\rangle$. By TAR-O, $fragment(q) \neq g$ in s'. By TAR-G, $fragment(p) \neq g$ in s', since otherwise $fragment(p) = fragment(q) = g$ in s'. So the predicate is still true in s.

TAR-H: Let $\langle q, p\rangle = minlink(g)$. Since $level(f) > level(g)$ by COM-A, $\langle p, q\rangle \neq minlink(g)$. So it is OK to set $lstatus(\langle p, q\rangle)$ to branch.

TAR-I: First note that if there is some node $r \in nodes(f) - testset(f)$ in s' with an unknown link, then by TAR-I there is an external link $\langle t, u\rangle$ of f, and $level(f) \leq level(fragment(u))$. Thus $fragment(u) \neq g$, so in s, the predicate is still true for nodes that were in $nodes(f)$ in s'.

To show that the predicate is true in s for nodes that were in $nodes(g)$ in s': we only need to consider the case when $p \notin testset(f)$ in s', i.e., when nodes formerly in $nodes(g)$ are not added to $testset(f)$. Since $level(f) > level(g)$, $minlink(f) \neq \langle p, q\rangle$, by COM-A. Thus, by TAR-A(a) and TAR-B, $lstatus(\langle p, q\rangle) =$ unknown, and the argument in the previous paragraph holds.

To show that the predicate is true in s for nodes that are not in either $nodes(f)$ or $nodes(g)$ in s', it is enough to note that the only relevant change is that the level of every node formerly in $nodes(g)$ is increased. □

Let $P'_{TAR} = (P'_{GC} \circ S_3) \wedge P_{TAR}$.

Corollary 18: *P'_{TAR} is true in every reachable state of TAR.*

Proof: By Lemmas 1 and 17. □

4.2.4 DC Simulates GC

This automaton focuses on how the nodes of a fragment cooperate to find the minimum-weight external link of the fragment in a distributed fashion. The variable *minlink*(f) is now a derived variable, depending on variables local to each node, and the contents of message queues. There is no action *ComputeMin*(f). The two nodes adjacent to the core send out FIND messages over the core. These messages are propagated throughout the fragment. When a node p receives a FIND message, it changes the variable *dcstatus*(p) from unfind to find, relays FIND messages, and records the link from which the FIND was received as its *inbranch*(p). Then the node atomically finds its local minimum-weight external link using action *TestNode*(p) as in GC, and waits to receive REPORT(w) messages from all its "children" (the nodes to which it sent FIND). The variable *findcount*(p) records how many children have not yet reported. Then p takes the minimum over all the weights w reported by its children and the weight of its own local minimum-weight external link and sends that weight to its "parent" in a REPORT message, along *inbranch*(p); the weight and the link associated with this minimum are recorded as *bestwt*(p) and *bestlink*(p), and *dcstatus*(p) is changed back to unfind. When a node adjacent to the core has heard from all its children, it sends a REPORT over the core. This message is not processed by the recipient until its *dcstatus* is set back to unfind. When a node p adjacent to the core receives a REPORT(w) over the core with $w >$ *bestwt*(p), then *minlink*(f) becomes defined, and is the link found by following *bestlinks* from p.

The *ChangeRoot*(f) action is the same as in GC. When two fragments merge, a FIND message is added to one link of the new core. A new action, *AfterMerge*(p, q), adds a FIND message to the other link of the new core. When an *Absorb*(f, g) action occurs, a FIND message is directed toward the old g along the reverse link of *minlink*(g) if and only if the target of *minlink*(g) is in *testset*(f) and its *dcstatus* is find.

This algorithm (as well as the original one) correctly handles "leftover" REPORT messages. Recall that a REPORT message is sent in both directions over the core (p, q) of a fragment f. Suppose the root p receives its REPORT message first, and the other REPORT message, the "leftover" one, which is headed toward q, remains in the queue until after f merges or is absorbed. Since the queues are FIFO relative to REPORT and FIND messages, the state of q remains such that when the leftover REPORT message is received, the only change is the removal of the message.

Define automaton DC (for "Distributed ComputeMin") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- $subtree(f)$, a subgraph of G;
- $core(f)$, an edge of G or nil;
- $level(f)$, a nonnegative integer;
- $rootchanged(f)$, a Boolean; and
- $testset(f)$, a subset of $V(G)$.

For each node p, there are the following variables:

- $dcstatus(p)$, either find or unfind;
- $findcount(p)$, a nonnegative integer;
- $bestlink(p)$, a link of G or nil;
- $bestwt(p)$, a weight or ∞; and
- $inbranch(p)$, a link of G or nil.

For each link $\langle p, q\rangle$, there are associated three variables:

- $dcqueue_p(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at p to be sent;
- $dcqueue_{pq}(\langle p, q\rangle)$, a FIFO queue of messages from p to q that are in the communication channel; and
- $dcqueue_q(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at q to be processed.

The set of possible messages M is $\{\text{REPORT}(w) : w \text{ a weight or } \infty\} \cup \{\text{FIND}\}$.

The state also contains Boolean variables, $answered(l)$, one for each $l \in L(G)$, and Boolean variable $awake$.

In the start state of DC, $fragments$ has one element for each node in $V(G)$; for fragment f corresponding to node p, $subtree(f) = \{p\}$, $core(f) = nil$, $level(f) = 0$, $rootchanged(f)$ is false, and $testset(f)$ is empty. For each p, $dcstatus(p) =$ unfind, $findcount(p) = 0$, $bestlink(p)$ is the minimum-weight external link of p, $bestwt(p)$ is the weight of $bestlink(p)$, and $inbranch(p) = nil$. The message queues are empty. Each $answered(l)$ is false and $awake$ is false.

The derived variable $dcqueue(\langle p,q\rangle)$ is defined to be $dcqueue_q(\langle p,q\rangle) \parallel dcqueue_{pq}(\langle p,q\rangle) \parallel dcqueue_p(\langle p,q\rangle)$.

A REPORT(w) message is *headed toward* p if either it is in $dcqueue(\langle q,p\rangle)$ for some q, or it is in some $dcqueue(\langle q,r\rangle)$, where $q \in subtree(r)$ and $r \in subtree(p)$. A FIND message is *headed toward* p if it is in some $dcqueue(\langle q,r\rangle)$ and p is in $subtree(r)$. A message is said to be *in* $subtree(f)$ if it is in some $dcqueue(\langle q,p\rangle)$ and $p \in nodes(f)$.

Now $minlink(f)$ is a derived variable, defined as follows. If $nodes(f) = \{p\}$, then $minlink(f)$ is the minimum-weight external link of p. Suppose $nodes(f)$ contains more than one node. If f has an external link, if $dcstatus(p) =$ unfind for all $p \in nodes(f)$, if no FIND message is in $subtree(f)$, and if no REPORT message is headed toward $mw\text{-}root(f)$, then $minlink(f)$ is the first external link reached by starting at $mw\text{-}root(f)$ and following bestlinks; otherwise, $minlink(f) = nil$.

Also $accmin(f)$ is a derived variable, defined as in TAR as follows. If $minlink(f) \neq nil$, or if there is no external link of any $p \in nodes(f) - testset(f)$, then $accmin(f) = nil$. Otherwise, $accmin(f)$ is the minimum-weight external link of all $p \in nodes(f) - testset(f)$.

Note below that $ReceiveFind(\langle q,p\rangle)$ is only enabled if $AfterMerge(p,q)$ is not enabled; without this precondition on $ReceiveFind$, p could receive the FIND before sending a FIND to q, and thus q's side of the subtree would not participate in the search.

Input actions:

- $Start(p)$, $p \in V(G)$
  - Effects:
    - $awake :=$ true

Output actions:

- $InTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  - Preconditions:
    - $awake =$ true
    - $(p,q) \in subtree(fragment(p))$ or $\langle p,q\rangle = minlink(fragment(p))$
    - $answered(\langle p,q\rangle) =$ false

Effects:
$answered(\langle p,q\rangle) :=$ true

- $NotInTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  $fragment(p) = fragment(q)$ and $(p,q) \notin subtree(fragment(p))$
  $answered(\langle p,q\rangle) =$ false
  Effects:
  $answered(\langle p,q\rangle) :=$ true

Internal actions:

- $ChannelSend(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  Preconditions:
  m at head of $dcqueue_p(\langle p,q\rangle)$
  Effects:
  dequeue($dcqueue_p(\langle p,q\rangle)$)
  enqueue(m, $dcqueue_{pq}(\langle p,q\rangle)$)

- $ChannelRecv(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  Preconditions:
  m at head of $dcqueue_{pq}(\langle p,q\rangle)$
  Effects:
  dequeue($dcqueue_{pq}(\langle p,q\rangle)$)
  enqueue(m, $dcqueue_q(\langle p,q\rangle)$)

- $TestNode(p)$, $p \in V(G)$
  Preconditions:
  — let $f = fragment(p)$ —
  $p \in testset(f)$
  if $\langle p,q\rangle$, the minimum-weight external link of p, exists
  then $level(f) \leq level(fragment(q))$
  $dcstatus(p) =$ find
  Effects:
  $testset(f) := testset(f) - \{p\}$
  if $\langle p,q\rangle$, the minimum-weight external link of p, exists then
  if $wt(p,q) < bestwt(p)$ then [
  $bestlink(p) := \langle p,q\rangle$
  $bestwt(p) := wt(p,q)$]
  execute procedure $Report(p)$

- $ReceiveReport(\langle q,p\rangle, w)$, $\langle q,p\rangle \in L(G)$

  Preconditions:

  REPORT(w) message at head of $dcqueue_p(\langle q,p\rangle)$

  Effects:

  dequeue($dcqueue_p(\langle q,p\rangle)$)
  if $\langle p,q\rangle \neq inbranch(p)$ then [
  $findcount(p) := findcount(p) - 1$
  if $w < bestwt(p)$ then [
  $bestwt(p) := w$
  $bestlink(p) := \langle p,q\rangle$]
  execute procedure $Report(p)$]
  else
  if $dcstatus(p) =$ find then enqueue(REPORT(w), $dcqueue_p(\langle q,p\rangle)$)

- $ReceiveFind(\langle q,p\rangle)$, $\langle q,p\rangle \in L(G)$

  Preconditions:

  FIND message at head of $dcqueue_p(\langle q,p\rangle)$
  $AfterMerge(p,q)$ not enabled

  Effects:

  dequeue($dcqueue_p(\langle q,p\rangle)$)
  $dcstatus(p) :=$ find
  $inbranch(p) := \langle p,q\rangle$
  $bestlink(p) := nil$
  $bestwt(p) := \infty$
  — let $S = \{\langle p,r\rangle : (p,r) \in subtree(fragment(p)), r \neq q\}$ —
  $findcount(p) := |S|$
  enqueue(FIND, $dcqueue_p(l)$) for all $l \in S$

- Procedure $Report(p)$, $p \in V(G)$

  if $findcount(p) = 0$ and $p \notin testset(fragment(p))$ then [
  $dcstatus(p) :=$ unfind
  enqueue(REPORT($bestwt(p)$), $dcqueue_p(inbranch(p))$)]

- $ChangeRoot(f)$, $f \in fragments$

  Preconditions:

  $awake =$ true
  $rootchanged(f) =$ false
  $minlink(f) \neq nil$

  Effects:

$rootchanged(f) :=$ true

- $Merge(f, g)$, $f, g \in fragments$

  Preconditions:

  $f \neq g$

  $rootchanged(f) = rootchanged(g) =$ true

  $minedge(f) = minedge(g)$

  Effects:

  add a new element h to $fragments$

  $subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$

  $core(h) := minedge(f)$

  $level(h) := level(f) + 1$

  $rootchanged(h) :=$ false

  $testset(h) := nodes(h)$

  — let $\langle p, q\rangle = minlink(f)$ —

  enqueue(FIND, $dcqueue_p(\langle p, q\rangle)$)

  delete f and g from $fragments$

- $AfterMerge(p, q)$, $p, q \in V(G)$

  Preconditions:

  $(p, q) = core(fragment(p))$

  FIND message in $dcqueue(\langle q, p\rangle)$

  no FIND message in $dcqueue(\langle p, q\rangle)$

  $dcstatus(q) =$ unfind

  no REPORT message in $dcqueue(\langle q, p\rangle)$

  Effects:

  enqueue(FIND, $dcqueue_p(\langle p, q\rangle)$)

- $Absorb(f, g)$, $f, g \in fragments$

  Preconditions:

  $rootchanged(g) =$ true

  $level(g) < level(f)$

  — let $\langle q, p\rangle = minlink(g)$ —

  $fragment(p) = f$

  Effects:

  $subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$

  if $p \in testset(f)$ then [

  $testset(f) := testset(f) \cup nodes(g)$

  if $dcstatus(p) =$ find then [

enqueue(FIND, $dcqueue_p(\langle p, q\rangle)$)
$findcount(p) := findcount(p) + 1$]]
delete g from *fragments*

Define the following predicates on $states(DC)$, using these definitions.

A child q of p is *completed* if no node in $subtree(q)$ is in $testset(fragment(p))$, and no REPORT is headed toward p in $subtree(q)$ or in $dcqueue(\langle q, p\rangle)$. Node p is *up-to-date* if either $subtree(fragment(p)) = \{p\}$, or the following two conditions are met: (1) following *inbranches* from p leads along edges of $subtree(fragment(p))$ toward and over $core(f)$, and (2) if $p \in testset(fragment(p))$, then $dcstatus(p) =$ find. Given node p, define C_p to be the set $\{r :$ either $r = p$ and $p \notin testset(fragment(p))$, or r is in $subtree(q)$ for some completed child q of $p\}$.

All free variables are universally quantified, except that $f = fragment(p)$, in these predicates. (The fact that an old REPORT message, in a link that was formerly the core of a fragment, can remain even after that fragment has merged or been absorbed, complicated the statement of some of the predicates.)

- DC-A: If REPORT(w) is in $dcqueue(\langle q, p\rangle)$ and $inbranch(p) \neq \langle p, q\rangle$, then
  (a) if $(p, q) = core(f)$, then a FIND message is ahead of the REPORT in $dcqueue(\langle q, p\rangle)$;
  (b) $\langle q, p\rangle = inbranch(q)$;
  (c) $bestwt(q) = w$;
  (d) $dcstatus(q) =$ unfind;
  (e) every child of q is completed;
  (f) $q \notin testset(f)$; and
  (g) if $(p, q) \neq core(f)$, then $dcstatus(p) =$ find, and q is a child of p.

- DC-B: If REPORT(w) is in $dcqueue(\langle q, p\rangle)$ and $inbranch(p) = \langle p, q\rangle$, then
  (a) either $(p, q) = core(f)$ or p is a child of q; and
  (b) if $(p, q) \neq core(f)$, then $dcstatus(p) =$ unfind.

- DC-C: If REPORT(w) is in $dcqueue(\langle q, p\rangle)$ and $(p, q) = core(f)$, then
  (a) q is up-to-date;
  (b) $dcstatus(q) =$ unfind; and
  (c) $bestwt(q) = w$.

- DC-D: If FIND is in $dcqueue(\langle q, p\rangle)$, then
  (a) if $(p, q) \neq core(f)$ then p is a child of q and $dcstatus(q) =$ find;

(b) $dcstatus(p)$ = unfind; and
(c) every node in $subtree(p)$ is in $testset(f)$.

- DC-E: If $p \in testset(f)$, then a FIND message is headed toward p, or $dcstatus(p)$ = find, or $AfterMerge(q,r)$ is enabled, where $p \in subtree(r)$.

- DC-F: If $(p,q) = core(f)$ and $inbranch(q) \neq \langle q,p\rangle$, then either a FIND is in $dcqueue(\langle p,q\rangle)$, or $AfterMerge(p,q)$ is enabled.

- DC-G: If $AfterMerge(p,q)$ is enabled, then every node in $subtree(q)$ is in $testset(f)$.

- DC-H: If $dcstatus(p)$ = unfind, then
  (a) $dcstatus(q)$ = unfind for all $q \in subtree(p)$; and
  (b) $findcount(p) = 0$.

- DC-I: If $dcstatus(p)$ = find, then
  (a) p is up-to-date; and
  (b) either a REPORT message is in $subtree(p)$ headed toward p, or some $q \in subtree(p)$ is in $testset(f)$.

- DC-J: If $dcstatus(p)$ = find and $core(f) = (p,q)$, then a FIND message is in $dcqueue(\langle p,q\rangle)$, or $dcstatus(q)$ = find, or a REPORT message is in $dcqueue(\langle q,p\rangle)$.

- DC-K: If p is up-to-date, then
  (a) $findcount(p)$ is the number of children of p that are not completed;
  (b) if $bestlink(p) = nil$, then $bestwt(p) = \infty$, and there is no external link of any node in C_p.
  (c) if $bestlink(p) \neq nil$, then following $bestlinks$ from p leads along edges in $subtree(f)$ to the minimum-weight external link l of all nodes in C_p; $wt(l) = bestwt(p)$, and $level(fragment(target(l))) \geq level(f)$.

- DC-L: If $inbranch(p) \neq nil$, then $inbranch(p) = \langle p,q\rangle$ for some q, and $(p,q) \in subtree(f)$.

- DC-M: $findcount(p) \geq 0$.

- DC-N: If $mw\text{-}minnode(f)$ is not in $testset(f)$, then $mw\text{-}minnode(f)$ is up-to-date.

- DC-O: The only possible values of $dcqueue(\langle p,q\rangle)$ are empty, or FIND, or REPORT, or FIND followed by REPORT (only if $(p,q) = core(f)$), or REPORT followed by FIND (only if $(p,q) \neq core(f)$).

Section 4.2.4: DC Simulates GC

Let P_{DC} be the conjunction of DC-A through DC-O.

In order to show that DC simulates GC, we define an abstraction mapping $\mathcal{M}_4 = (\mathcal{S}_4, \mathcal{A}_4)$ from DC to GC.

Define the function $\mathcal{S}_4$ from $states(DC)$ to $states(GC)$ by ignoring the message queues, and the variables *dcstatus*, *findcount*, *bestlink*, *bestwt*, and *inbranch*. The derived variables *minlink* and *accmin* of DC map to the (non-derived) variables *minlink* and *accmin* of GC.

Define the function $\mathcal{A}_4$ as follows. Let s be a state of DC and π an action of DC enabled in s. The GC action $ComputeMin(f)$ is simulated in DC when a node adjacent to the core, having already heard from all its children, receives a REPORT message over the core with a weight larger than its own *bestwt*. Then the node knows that the minimum-weight external link of the fragment is on its own side of the subtree.

- Suppose $\pi = ReceiveReport(\langle q,p\rangle, w)$. If $(p,q) = core(f)$ and $dcstatus(p) =$ unfind and $w > bestwt(p)$, then $\mathcal{A}_4(s,\pi) = ComputeMin(fragment(p))$. Otherwise $\mathcal{A}_4(s,\pi)$ is empty.
- If $\pi = ChannelSend(\langle q,p\rangle, m)$, $ChannelRecv(\langle q,p\rangle, m)$, $ReceiveFind(\langle q,p\rangle)$ or $AfterMerge(p,q)$, then $\mathcal{A}_4(s,\pi)$ is empty.
- For all other values of π, $\mathcal{A}_4(s,\pi) = \pi$.

The following predicates are true in any state of DC satisfying $(P'_{GC} \circ \mathcal{S}_4) \wedge P_{DC}$. Recall that $P'_{GC} = (P'_{COM} \circ \mathcal{S}_2) \wedge P_{GC}$. If $P'_{GC}(\mathcal{S}_4(s))$ is true, then the GC predicates are true in $\mathcal{S}_4(s)$, the COM predicates are true in $\mathcal{S}_2(\mathcal{S}_4(s))$, and the HI predicates are true in $\mathcal{S}_1(\mathcal{S}_2(\mathcal{S}_4(s)))$. Thus, these predicates are deducible from P_{DC}, together with the GC, COM and HI predicates.

- DC-P: If REPORT(w) is at the head of $dcqueue(\langle q,p\rangle)$ and $(p,q) = core(f)$ and $dcstatus(p) =$ unfind, then

  (a) if $w < bestwt(p)$, then the minimum-weight external link l of f is closer to q than to p, and $wt(l) = w$;

  (b) if $w > bestwt(p)$, then the minimum-weight external link l of f is closer to p than to q, and $wt(l) = bestwt(p)$; and

  (c) if $w = bestwt(p)$, then $w = \infty$ and there is no external link of f.

Proof:

1. REPORT(w) is at head of $dcqueue(\langle q,p\rangle)$, by assumption.
2. $dcstatus(p)=$ unfind, by assumption.
3. $(p,q)=core(f)$, by assumption.
4. q is up-to-date, by Claims 1 and 3 and DC-C(a).
5. $dcstatus(q)=$ unfind, by Claims 1 and 3 and DC-C(b).
6. $w=bestwt(q)$, by Claims 1 and 3 and DC-C(c).
7. $q\notin testset(f)$, by Claims 4 and 5.
8. No FIND is in $dcqueue(\langle q,p\rangle)$, by Claims 1 and 3 and DC-O.
9. p is up-to-date, by Claims 2, 3, 4 and 8 and DC-T.
10. $p\notin testset(f)$, by Claims 2 and 9.
11. $findcount(p)=0$, by Claim 2 and DC-H(b).
12. $findcount(q)=0$, by Claim 5 and DC-H(b).
13. All children of p are completed, by Claims 9 and 11 and DC-K(a).
14. All children of q are completed, by Claims 4 and 12 and DC-K(a).
15. If $bestwt(p)=\infty$, then there is no external link of $subtree(p)$, by Claims 9, 10 and 13 and DC-K(b) and (c).
16. If $bestwt(p)\neq\infty$, then following *bestlinks* from p leads to the minimum-weight external link l of $subtree(p)$ and $wt(l)=bestwt(p)$, by Claims 9, 10 and 13, and DC-K(b) and (c).
17. If $bestwt(q)=w=\infty$, then there is no external link of $subtree(q)$, by Claims 4, 6, 7 and 14 and DC-K(b) and (c).
18. If $bestwt(q)=w\neq\infty$, then following *bestlinks* from q leads to the minimum-weight external link l of $subtree(q)$ and $wt(l)=w$, by Claims 4, 6, 7 and 14 and DC-K(b) and (c).

Claims 3 and 15 through 18 give the result, together with the fact that edge weights are distinct. □

- DC-Q: If a REPORT is at the head of $dcqueue(\langle q,p\rangle)$ and is not headed toward $mw\text{-}root(f)$, then $inbranch(p)=\langle p,q\rangle$.

Proof: If $(p,q)=core(f)$, then $inbranch(p)=\langle p,q\rangle$ by DC-A(a). Suppose $(p,q)\neq core(f)$, and, in contradiction, that $inbranch(p)\neq\langle p,q\rangle$. By DC-A(g), $dcstatus(p)=$ find, and by DC-I(a) p is up-to-date, i.e., following *inbranches* from p leads toward and over $core(f)$. Thus the REPORT in $dcqueue(\langle q,p\rangle$is headed toward both endpoints of $core(f)$, contradicting the hypothesis. □

- DC-R: If $dcstatus(p)=$ find, then no REPORT is in $dcqueue(inbranch(p))$.

Proof: Let $inbranch(p)=\langle p,q\rangle$.

1. $dcstatus(p) =$ find, by assumption.
2. p is up-to-date, by Claim 1 and DC-I(a).
3. Following *inbranches* from p leads toward and over $core(f)$, by Claim 2.
4. Either $(p, q) = core(f)$, or $inbranch(q) \neq \langle q, p\rangle$, or no REPORT is in $dcqueue(\langle p, q\rangle)$, by Claim 3 and DC-B(b).
5. If $(p, q) = core(f)$, then no REPORT is in $dcqueue(\langle p, q\rangle)$, by Claim 1 and DC-C(b).
6. If $inbranch(q) \neq \langle q, p\rangle$, then no REPORT is in $dcqueue(\langle p, q\rangle)$, by Claim 1 and DC-A(d).
7. No REPORT is in $dcqueue(\langle p, q\rangle)$, by Claims 4, 5 and 6. □

- DC-S: At most one FIND message is headed toward p.

Proof: Suppose a FIND message is headed toward p.
1. A FIND is in $dcqueue(\langle q, r\rangle)$, by assumption.
2. $p \in subtree(r)$, by assumption.
3. $dcstatus(r) =$ unfind, by Claim 1 and DC-D(b).
4. $dcstatus(t) =$ unfind for all $t \in subtree(r)$, by Claim 3 and DC-H(a).
5. No FIND message is in $dcqueue(\langle t, u\rangle)$, for any $(t, u) \in subtree(r)$, by Claim 4 and DC-D(a).

If $(q, r) = core(f)$, Claim 5 proves the result. Suppose $(q, r) \neq core(f)$.

6. $(q, r) \neq core(f)$, by assumption.
7. $dcstatus(q) =$ find, by Claims 1 and 6 and DC-D(a).
8. $dcstatus(t) =$ find for all t between q and the endpoint of $core(f)$ closest to q, by Claim 7 and DC-H(a).
9. No FIND message is in $dcqueue(\langle t, u\rangle)$ for any (t, u) between $core(f)$ and q, by Claim 8 and DC-D(b).

Claim 9 completes the proof. □

- DC-T: If $(p, q) = core(f)$, no FIND is in $dcqueue(\langle p, q\rangle)$, p is up-to-date, and $dcstatus(q) =$ unfind, then q is up-to-date.

Proof:
1. $(p, q) = core(f)$, by assumption.
2. No FIND is in $dcqueue(\langle p, q\rangle)$, by assumption.
3. p is up-to-date, by assumption.
4. $dcstatus(q) =$ unfind, by assumption.
5. No FIND is headed toward q, by Claims 1 and 2 and DC-D(a).

6. No FIND is in $dcqueue(\langle q,p\rangle)$, by Claim 3 and DC-D(b) and (c).
7. $AfterMerge(p,q)$ is not enabled, by Claim 6.
8. $inbranch(q) = \langle q,p\rangle$, by Claims 5 and 7 and DC-F.
9. $q \notin testset(f)$, by Claims 4, 5 and 7 and DC-E.
10. q is up-to-date, by Claims 1, 8 and 9. □

Lemma 19: *DC simulates GC via* $\mathcal{M}_4$, P_{DC}, *and* P'_{GC}.

Proof: By inspection, the types of DC, GC, $\mathcal{M}_4$, and P_{DC} are correct. By Corollary 16, P'_{GC} is a predicate true in every reachable state of GC.

(1) Let s be in $start(DC)$. Obviously, P_{DC} is true in s, and $\mathcal{S}_4(s)$ is in $start(GC)$.

(2) Obviously, $\mathcal{A}_4(s,\pi)|ext(GC) = \pi|ext(DC)$.

(3) Let (s',π,s) be a step of DC such that P'_{GC} is true of $\mathcal{S}_4(s')$ and P_{DC} is true of s'. For (3a) we verify below only those DC predicates whose truth in s is not obvious.

i) π **is Start(p), ChangeRoot(f), InTree(l), or NotInTree(l).** $\mathcal{A}_4(s',\pi) = \pi$. Obviously $\mathcal{S}_4(s')\pi\mathcal{S}_4(s)$ is an execution fragment of GC and P_{DC} is true in s.

ii) π **is ChannelSend(l,m) or ChannelRecv(l,m).** $\mathcal{A}_4(s',\pi)$ is empty. Obviously $\mathcal{S}_4(s) = \mathcal{S}_4(s')$ and P_{DC} is true in s.

iii) π **is TestNode(p).** Let $f = fragment(p)$ in s'.

(3c) $\mathcal{A}_4(s',\pi) = \pi$. Obviously, π is enabled in $\mathcal{S}_4(s')$. To show the effects are mirrored in $\mathcal{S}_4(s)$, we must show that $accmin(f)$ is updated properly (which is obvious) and that $minlink(f)$ is unchanged. Since $p \in testset(f)$ in s', $minlink(f) = nil$ in s' by GC-C. If $accmin(f) \neq nil$, or if p has an external link in s', then $accmin(f) \neq nil$ in s, and $minlink(f)$ is still nil in s. If some $q \neq p$ is in $testset(f)$ in s', then by DC-E either a FIND is in $subtree(f)$ or $dcstatus(q) =$ find; since the same is true in s, $minlink(f)$ is still nil in s. Finally, if $accmin(f) = nil$, p has no external link, and p is the sole element of $testset(f)$ in s', then f has no external link in s' or in s, and $minlink(f)$ is still nil in s.

(3a) Two cases are considered. First we prove some facts true in both cases.

Claims about s'*:*

1. $dcstatus(p) =$ find, by precondition.
2. $p \in testset(f)$, by precondition.
3. If $\langle p, u \rangle$, the minimum-weight external link of p, exists, then $level(f) \leq level(fragment(u))$, by precondition.
4. p is up-to-date, by Claim 1 and DC-I(a).
5. No FIND is headed toward p, by Claim 1 and DC-D(c).
6. If $(p, r) = core(f)$, then no REPORT is in $dcqueue(\langle p, r \rangle)$, for any r, by Claim 1 and DC-C(b).
7. If a REPORT is in $dcqueue(\langle p, r \rangle)$, then $inbranch(r) = \langle r, p \rangle$, for any r, by Claim 1 and DC-A(d).
8. $AfterMerge(r, t)$, where $p \in subtree(t)$, is not enabled, by Claim 1 and DC-H(a).
9. If $bestlink(p) = nil$, then $bestwt(p) = \infty$ and there is no external link of any node r, where r is in the subtree of any completed child of p, by Claims 2 and 4 and DC-K(b).
10. If $bestlink(p) \neq nil$, then following *bestlinks* from p leads to the minimum-weight external link l of all nodes r, where r is in the subtree of any completed child of p; $wt(l) = bestwt(p)$ and $level(f) \leq level(fragment(target(l)))$, by Claims 2 and 4 and DC-K(c).

Case 1: $findcount(p) \neq 0$ in s'.

More claims about s':

11. $findcount(p) \neq 0$, by assumption.
12. $findcount(p) > 0$, by Claim 11 and DC-M.
13. Some child r of p is not completed, by Claims 4 and 12 and DC-K(a).
14. There is a child r of p such that either some node in $subtree(r)$ is in $testset(f)$, or a REPORT is in $subtree(r)$ or $dcqueue(\langle r, p \rangle)$ headed toward p, by Claim 13.

DC-A(c): By Claim 7, changing $bestwt(p)$ and removing p from $testset(f)$ are OK.

DC-C: By Claim 6, changing $bestwt(p)$ is OK.

DC-D(c): By Claim 5, removing p from $testset(f)$ is OK.

DC-G: By Claim 8 and the fact that $dcstatus(p)$ is still find in s, removing p from $testset(f)$ is OK.

DC-I(b): By Claim 14, removing p from $testset(f)$ is OK.

DC-K: (b) By Claim 9 and code. (c) by Claims 3 and 10 and code.

DC-N: If p is $mw\text{-}minnode(f)$, then by Claim 4, removing p from $testset(p)$ is OK.

Case 2: $findcount(p) = 0$ in s'. Let $\langle p, q\rangle = inbranch(p)$.

More claims about s'*:*

15. $findcount(p) = 0$, by assumption.
16. If $(p, q) = core(f)$ and $inbranch(q) \neq \langle q, p\rangle$, then a FIND is in $dcqueue(\langle p, q\rangle)$, by Claim 5 and DC-F.
17. All children of p are completed, by Claims 3 and 15 and DC-K(a).
18. If $(p, q) \neq core(f)$, then $dcstatus(q) =$ find, by Claim 1 and DC-H(a).
19. If REPORT is in $dcqueue(\langle q, p\rangle)$, then $(p, q) = core(f)$, by Claim 4 and DC-B(a).
20. No REPORT is in $dcqueue(\langle p, q\rangle)$, by Claim 1 and DC-R.
21. If FIND is in $dcqueue(\langle p, q\rangle)$, then $(p, q) = core(f)$, by Claim 4 and DC-D(a).
22. Every node $r \neq p$ in $subtree(p)$ has $dcstatus(r) =$ unfind, by Claims 1 and 17 and DC-I(b).
23. Every node $r \neq p$ in $subtree(p)$ has $findcount(r) = 0$ by Claim 22 and DC-H(b).

DC-A: By Claim 7 and the fact that $inbranch(p) = \langle p, q\rangle$, we need only consider the REPORT added to $dcqueue(\langle p, q\rangle)$. (a) by Claim 16. (b), (c) and (d) by code. (e) by Claim 17. (f) by code. (g) by Claims 4 and 18.

DC-B for REPORT added to $dcqueue(\langle p, q\rangle)$: If $inbranch(q) = \langle q, p\rangle$, then $(p, q) = core(f)$, by Claim 4.

DC-B for REPORT that might be in $dcqueue(\langle q, p\rangle)$: by Claim 19.

DC-C: By Claim 4, $inbranch(p)$ is the only relevant link; by Claim 20, the new message is the only REPORT in that queue. (a) by Claim 4. (b) and (c) by code.

DC-D(a) and (c): By Claim 5, it is OK to change $dcstatus(p)$ to unfind and remove p from $testset(f)$.

DC-E: The addition of a REPORT to $dcqueue(\langle p, q\rangle)$ in s cannot cause *After-Merge*(q, p) to go from enabled in s' to disabled in s, by Claim 1.

DC-F: Cf. DC-E.

DC-G: By Claim 8 and the addition of REPORT to $dcqueue(\langle p,q\rangle)$, removing p from $testset(f)$ is OK.

DC-H: (a) By Claim 22 and code. (b) By Claim 23.

DC-I(b): Suppose $r \neq q$ is some node such that $p \in subtree(r)$ and $dcstatus(r) =$ find in s'. By Claim 4, removing p from $testset(f)$ is compensated for by adding REPORT to $dcqueue(\langle p,q\rangle)$.

DC-J: By Claim 4, the only link of p that can be part of $core(f)$ is $\langle p,q\rangle$. If $(p,q) = core(f)$ and $dcstatus(q) =$ find, then the fact that $dcstatus(p)$ becomes unfind in s is compensated for by the addition of REPORT to $dcqueue(\langle p,q\rangle)$.

DC-K(b) and (c): As in Case 1.

DC-N: As in Case 1.

DC-O: By Claims 20, 21 and code.

iv) π is ReceiveReport($\langle$q,p$\rangle$,w). Let $f = fragment(p)$ in s'.

(3b)/(3c) *Case 1:* $(p,q) = core(f)$ and $dcstatus(p) =$ unfind and $w > bestwt(p)$ in s'. $\mathcal{A}_4(s',\pi) = ComputeMin(f)$.

Let $\langle r,t\rangle$ be the minimum-weight external link of f in s'. (Below we show it exists.)

Claims about s'*:*

1. REPORT(w) is at the head of $dcqueue(\langle q,p\rangle)$, by precondition.
2. $(p,q) = core(f)$, by assumption.
3. $dcstatus(p) =$ unfind, by assumption.
4. $w > bestwt(p)$, by assumption.
5. No FIND is in $dcqueue(\langle q,p\rangle)$, by Claim 1 and DC-O.
6. q is up-to-date, by Claims 1 and 2 and DC-C(a).
7. p is up-to-date, by Claims 2, 3, 5 and 6 and DC-T.
8. $dcstatus(q) =$ unfind, by Claims 1 and 2 and DC-C(b).
9. $bestwt(q) = w$, by Claims 1 and 2 nad DC-C(c).
10. $p = mw\text{-}root(f)$ (so $\langle r,t\rangle$ exists), by Claims 1, 2, 3 and 4 and DC-P(b).
11. $minlink(f) = nil$, by Claims 1 and 10.

12. $findcount(p) = 0$, by Claim 3 and DC-H(b).
13. $findcount(q) = 0$, by Claim 8 and DC-H(b).
14. Every child of p is completed, by Claims 7 and 12 and DC-K(a).
15. Every child of q is completed, by Claims 6 and 13 and DC-K(a).
16. $p \notin testset(f)$, by Claims 3 and 7.
17. $q \notin testset(f)$, by Claims 6 and 8.
18. $testset(f) = \emptyset$, by Claims 14 through 17.
19. $accmin(f) = \langle r, t \rangle$, by Claims 11 and 18.

By Claims 11, 18 and 19, $ComputeMin(f)$ is enabled in s'.

Now we must show that the effects of $ComputeMin(f)$ are mirrored in s. All that must be shown is that $minlink(f)$ and $accmin(f)$ are updated properly.

More claims about s':

20. $dcstatus(u) =$ unfind, for all $u \in subtree(p)$, by Claim 3 and DC-H(a).
21. $dcstatus(u) =$ unfind, for all $u \in subtree(q)$, by Claim 8 and DC-H(a).
22. No REPORT is headed toward p in $subtree(p)$, by Claim 14.
23. No REPORT is headed toward q in $subtree(q)$, by Claim 15.
24. Only one REPORT is in $subtree(p)$, by DC-O.
25. No FIND is in $subtree(f)$, by Claim 18 and DC-D(c).
26. Following *bestlinks* from p leads to $\langle r, t \rangle$, by Claims 7, 10, 14 and 16 and DC-K(b) and (c).

By Claims 10 and 20 through 26, $minlink(f) = \langle r, t \rangle$ in s. By Claim 19, this is the correct value. Thus, $accmin(f) = nil$ in s.

Case 2: $(p, q) \neq core(f)$ or $dcstatus(p) =$ find or $w \leq bestwt(p)$ in s'. $\mathcal{A}_4(s', \pi)$ is empty. We just need to verify that $minlink(f)$ and $accmin(f)$ are unchanged in order to show that $\mathcal{S}_4(s') = \mathcal{S}_4(s)$.

Subcase 2a: $(p, q) \neq core(f)$ in s'.

Suppose $\langle p, q \rangle = inbranch(p)$ in s'. By DC-B(b), $dcstatus(p) =$ unfind, so the only effect is to remove the REPORT. By DC-B(a), $p \in subtree(q)$, so this REPORT message is not headed toward $mw\text{-}root(f)$ in s'. Thus $minlink(f)$ is unchanged, and $accmin(f)$ is also unchanged.

Suppose $\langle p, q \rangle \neq inbranch(p)$ in s'.

Claims about s':

1. REPORT(w) is at the head of $dcqueue(\langle q,p\rangle)$, by precondition.
2. $\langle p,q\rangle \neq inbranch(p)$, by assumption.
3. $(p,q) \neq core(f)$, by assumption.
4. $dcstatus(p) =$ find, by Claims 1, 2 and 3 and DC-A(g).
5. p is up-to-date, by Claim 4 and DC-I(a).
6. Following *inbranches* from p leads toward and over $core(f)$, by Claim 5.
7. A REPORT message is headed toward $mw\text{-}root(f)$, by Claims 1 and 6.
8. $minlink(f) = nil$, by Claim 7.
9. If $core(f) = (p,t)$ for some t, then FIND is in $dcqueue(\langle p,t\rangle)$, $dcstatus(t) =$ find, or REPORT is in $dcqueue(\langle t,p\rangle)$, by Claim 4 and DC-J.

Claims about s:

10. $subtree(f)$, $core(f)$, $nodes(f)$, and $testset(f)$ do not change, by code.
11. REPORT is in $inbranch(p)$, by code.
12. Following *inbranches* from p leads toward and over $core(f)$, by Claims 6 and 10 and code.
13. If $p \neq mw\text{-}root(f)$, then REPORT is headed toward $mw\text{-}root(f)$, by Claims 11 and 12.
14. If $p = mw\text{-}root(f)$, then FIND is in $dcqueue(\langle p,t\rangle)$, $dcstatus(t) =$ find, or REPORT is in $dcqueue(\langle t,p\rangle)$, where $(p,t) = core(f)$, by Claim 9 and code.
15. $minlink(f) = nil$, by Claims 13 and 14.
16. $accmin(f)$ does not change, by Claims 8, 10 and 15.

Claims 15 and 16 give the result.

Subcase 2b: $(p,q) = core(f)$ and $dcstatus(p) =$ find in s'. Since REPORT(w) is at the head of $dcqueue(\langle q,p\rangle)$, DC-A(a) implies that $inbranch(p) = \langle p,q\rangle$. The only change is that the REPORT message is requeued. Obviously $minlink(f)$ and $accmin(f)$ are unchanged.

Subcase 2c: $(p,q) = core(f)$ and $dcstatus(p) =$ unfind and $w \leq bestwt(p)$ in s'. As in Subcase 2b, $inbranch(p) = \langle p,q\rangle$. The only change is that the REPORT message is removed. If $w = bestwt(p)$, then by DC-P(c), there is no external link of f in s' or in s. Thus $minlink(f)$ and $accmin(f)$ are both nil in s' and s.

Suppose $w < bestwt(p)$. By DC-P(a), $q = mw\text{-}root(f)$. Thus the REPORT message in $dcqueue(\langle q,p\rangle)$ is not headed toward $mw\text{-}root(f)$ in s', and no criteria for $minlink(f)$, or $accmin(f)$ changes.

(3a) *Case 1:* $\langle p,q\rangle = inbranch(p)$ in s'.

Suppose $dcstatus(p) =$ find. By DC-D(b), no FIND is in $dcqueue(\langle q,p\rangle)$ in s', so by DC-O, $dcqueue(\langle q,p\rangle)$ contains just the one REPORT message in s'. Since the only effect is to requeue the message, the *DC* state is unchanged.

Suppose $dcstatus(p) =$ unfind. The only change is the removal of the REPORT message from $dcqueue(\langle q,p\rangle)$. By DC-B(a), either $(p,q) = core(f)$, or $p \in subtree(q)$ in s'. In both cases, the REPORT is not headed toward any node whose subtree it is in.

DC-I(b): By remark above.

DC-J: Even though REPORT is removed from $dcqueue(\langle q,p\rangle)$, $dcstatus(p) =$ unfind in s.

DC-K(a): By remark above, removing the REPORT does not affect the completeness of any node's child.

Case 2: $\langle p,q\rangle \neq inbranch(p)$. Let $\langle p,r\rangle = inbranch(p)$.

Claims about s':

1. REPORT(w) is at head of $dcqueue(\langle q,p\rangle)$, by precondition.
2. $\langle p,q\rangle \neq inbranch(p)$, by assumption.
3. $(p,q) \neq core(f)$, by Claims 1 and 2 and DC-A(a).
4. $\langle q,p\rangle = inbranch(q)$, by Claims 1 and 2 and DC-A(b).
5. $w = bestwt(q)$, by Claims 1 and 2 and DC-A(c).
6. $dcstatus(q) =$ unfind, by Claims 1 and 2 and DC-A(d).
7. Every child of q is completed, by Claims 1 and 2 and DC-A(e).
8. $q \notin testset(f)$, by Claims 1 and 2 and DC-A(f).
9. $dcstatus(p) =$ find, by Claim 3 and DC-A(g).
10. If REPORT is in $dcqueue(p,t)$, then $inbranch(t) = \langle t,p\rangle$, for any t, by Claim 9 and DC-A(d).

11. p is up-to-date, by Claim 9 and DC-I(a).
12. $inbranch(p)$ leads toward and over $core(f)$, by Claim 11.
13. q is an uncompleted child of p, by Claims 1, 2 and 12.
14. $findcount(p) \geq 1$, by Claims 11 and 13 and DC-K(a).
15. Only one REPORT is in $dcqueue(\langle q, p \rangle)$, by Claim 1 and DC-O.
16. q is up-to-date, by Claims 4, 8 and 12.
17. If REPORT is in $dcqueue(\langle p, t \rangle)$, then $(p, t) \neq core(f)$, for all t, by Claim 9 and DC-C(b).
18. If $bestwt(p) = \infty$, then there is no external link of p (if $p \notin testset(f)$) or of any node in the subtree of any completed child of p, by Claim 11 and DC-F(b) and (c).
19. If $bestwt(p) \neq \infty$, then following $bestlinks$ from p leads to the minimum-weight external link l of all nodes in C_p; $wt(l) = bestwt(p)$; and $level(f) \leq level(fragment(target(l)))$, by Claim 11 and DC-F(b) and (c).
20. If $w = \infty$, then there is no external link of $subtree(q)$, by Claims 5, 7, 8 and 16 and DC-K(b) and (c).
21. If $w \neq \infty$, then following $bestlinks$ from q leads to the minimum-weight external link l of $subtree(q)$; $wt(l) = w$, and $level(f) \leq level(fragment(target(l)))$, by Claims 5, 7, 8 and 16 and DC-F(b) and (c).

Subcase 2a: $p \in testset(f)$ or $findcount(p) \neq 1$ in s'.

More claims about s'*:*

22. $p \in testset(f)$ or $findcount(p) \neq 1$, by assumption.
23. If $findcount(p) \neq 1$, then $findcount(p) > 1$, by Claim 14.
24. If $findcount(p) \neq 1$, then some child $t \neq q$ of p is not completed, by Claims 11 and 23 and DC-K(a).
25. If $findcount(p) = 1$, then $p \in testset(f)$, by Claim 22.

DC-A(c): by Claim 10, any change to $bestwt(p)$ is OK.

DC-C: By Claim 17, changing $bestwt(p)$ is OK.

DC-F: Cf. DC-G.

DC-G: Removing REPORT from $dcqueue(\langle q, p \rangle)$ does not cause $AfterMerge(p, q)$ to become enabled, by Claim 3.

DC-I(b): Let t be some node such that $p \in subtree(t)$ and $dcstatus(t) =$ find in s'. By Claims 24 and 25, either a REPORT message is in $subtree(p)$ headed toward

p (and hence toward t), or some node in $subtree(p)$ (and hence in $subtree(t)$) is in $testset(f)$.

DC-J: The removal of the REPORT message is OK by Claim 3.

DC-K(a): Since $findcount(p)$ is decremented by 1, we just need to show that the number of uncompleted children of p decreases by 1: by Claim 1, q is not completed in s'. By Claims 7, 8 and 15 and code, q is completed in s.

DC-K(b) and (c): by Claims 18, 19, 20 and 21 and code.

DC-M: By Claim 14 and code.

Subcase 2b: $p \notin testset(f)$ and $findcount(p) = 1$.

26. $p \notin testset(f)$, by assumption.
27. $findcount(p) = 1$, by assumption.
28. No FIND is headed toward p, by Claim 9 and DC-D(b).
29. If $(p,r) = core(f)$ and $inbranch(r) \neq \langle r,p\rangle$,then FIND is in $dcqueue(\langle p,r\rangle)$, by Claim 28 and DC-F.
30. No REPORT is in $dcqueue(\langle p,r\rangle)$, by Claim 9 and DC-R.
31. Every child of p but q is completed, by Claims 11, 13, 27 and DC-K(a).
32. No FIND is in $dcqueue(\langle p,t\rangle)$, $t \neq r$, by Claims 7, 8 and 31 and DC-D(c).
33. If REPORT is in $dcqueue(\langle r,p\rangle)$, then $(p,r) = core(f)$, by Claim 9 and DC-B(a) and (b).
34. If $(p,r) \neq core(f)$, then $dcstatus(r) =$ find, by Claims 9 and 12 and DC-H(a).
35. If FIND is in $dcqueue(\langle p,r\rangle)$, then $(p,r) = core(f)$, by Claim 12 and DC-D(a).

DC-A: By Claim 10 and the fact that $inbranch(p) = \langle p,r\rangle)$, we need only consider the REPORT added to $dcqueue(\langle p,r\rangle)$. (a) by Claim 29. (b), (c) and (d) by code. (e) by Claim 31 for any child of p except q; by Claims 7, 8 and 15 and code for q. (f) by Claim 8. (g) by Claims 12 and 34.

DC-B for REPORT added to $dcqueue(\langle p,r\rangle)$: if $inbranch(r) = \langle r,p\rangle$, then by Claim 12, $core(f) = (p,r)$.

DC-B for REPORT in $dcqueue(\langle r,p\rangle)$: By Claim 33, $core(f) = (p,r)$.

DC-C: By Claim 12, $inbranch(p)$ is the only relevant link; by Claim 30, the new message is the only REPORT message in its queue. (a) by Claim 11. (b) and (c) by code.

DC-D(a): By Claims 32 and 35, changing $dcstatus(p)$ to unfind is OK.

DC-E: The addition of the REPORT to $dcqueue(\langle p, r\rangle)$ in s cannot cause $AfterMerge(r,p)$ to go from enabled in s' to disabled in s, because $dcstatus(p) =$ find in s' by Claim 9.

DC-F: Cf. DC-E.

DC-H(a): By Claims 7 and 8, no node in $subtree(q)$ is in $testset(f)$. By Claim 31, no node in $subtree(t)$, for any child $t \neq q$ of p, is in $testset(f)$. By Claim 23, $p \notin testset(f)$.

DC-H(b): By Claim 27 and code.

DC-I(b): Let $t \neq p$ be such that $p \in subtree(t)$ and $dcstatus(t) =$ find in s'. By Claim 12, removing the REPORT from $dcqueue(\langle q, p\rangle)$ is compensated for by adding the REPORT to $dcqueue(\langle p, r\rangle)$.

DC-J: By Claim 12, the only link of p that can be part of $core(f)$ is $\langle p, r\rangle$. If $(p, r) = core(f)$ and $dcstatus(q) =$ find in s', then changing $dcstatus(p)$ to unfind in s is compensated for by adding the REPORT to $dcqueue(\langle p, r\rangle)$.

DC-K: As in Subcase 2a.

DC-M: Claim 27 and code.

DC-O: by Claim 30 and DC-O and code.

v) π is ReceiveFind($\langle$q,p$\rangle$). Let $f = fragment(p)$.

(3b) $\mathcal{A}_4(s', \pi)$ is empty. To show that $\mathcal{S}_4(s') = \mathcal{S}_4(s)$, we just need to show that $minlink(f)$ and $accmin(f)$ are unchanged. Because of the FIND message, $minlink(f) = nil$ in s', and $minlink(f) = nil$ in s since $dcstatus(p) =$ find. Since there is no change to $minlink(f)$, $nodes(f)$, $testset(f)$, or $subtree(f)$, $accmin(f)$ is unchanged.

(3a) *Claims about s':*

1. FIND is at head of $dcqueue(\langle q, p\rangle)$, by precondition.
2. $AfterMerge(p, q)$ is not enabled, by precondition.
3. If $(p, q) \neq core(f)$, then p is a child of q, by Claim 1 and DC-D(a).
4. If $(p, q) \neq core(f)$, then $dcstatus(q) =$ find, by Claim 1 and DC-D(a).

5. $dcstatus(p) =$ unfind, by Claim 1 and DC-D(b).
6. Every node in $subtree(p)$ is in $testset(f)$, by Claim 1 and DC-D(c).
7. No REPORT is in $dcqueue(\langle p,r\rangle)$ with $inbranch(r) \neq \langle r,p\rangle$, for all r, by Claim 6 and DC-A(f).
8. If REPORT is in $dcqueue(\langle p,r\rangle)$, then $(p,r) \neq core(f)$, for all r, by Claim 6 and DC-C.
9. If REPORT is in $dcqueue(\langle q,p\rangle)$, then $(p,q) = core(f)$, by Claim 1 and DC-O.
10. If $(r,p) \in subtree(f)$, $r \neq q$, then r is a child of p, by Claim 3.
11. No REPORT is in $dcqueue(\langle r,p\rangle)$, $r \neq q$, with $inbranch(p) \neq \langle p,r\rangle)$, by Claims 6 and 10 and DC-A(f).
12. No REPORT is in $dcqueue(\langle r,p\rangle)$, $r \neq q$, with $inbranch(p) = \langle p,r\rangle$, by Claim 10 and DC-B(a).
13. If $\langle p,r\rangle \in S$, then r is a child of p, by Claim 10.
14. $dcstatus(r) =$ unfind for all $r \in subtree(p)$, by Claim 5 and DC-H(a).
15. If $(p,q) \neq core(f)$, then $dcstatus(r) =$ find, for all r such that $q \in subtree(r)$, by Claim 4 and DC-H(a).
16. $dcqueue(\langle p,r\rangle)$ is either empty or contains only a REPORT for all r such that $\langle p,r\rangle \in S$, by Claims 5 and 13 and DC-D(a) and DC-O.
17. If $(p,q) \neq core(f)$, then following inbranches from q leads toward and over $core(f)$, by Claim 4 and DC-I(a).

DC-A(a): By Claim 7, we need not consider any REPORT in a link leaving p. By Claim 11 we need not consider any REPORT in a link coming into p, except for $\langle q,p\rangle$. Since $inbranch(p)$ is set to $\langle p,q\rangle$ in s, removing FIND from $dcqueue(\langle q,p\rangle)$ is OK.

DC-B: By Claim 9 and 12, changing $dcstatus(p)$ is OK.

DC-C: By Claim 8, changing $dcstatus(p)$ and $bestwt(p)$ is OK.

DC-D: (a) by Claim 13 and code. (b) by Claim 14. (c) by Claim 6.

DC-E: By Claim 12 and code (adding FIND messages and setting $dcstatus(p)$ to find), removing FIND from $dcqueue(\langle q,p\rangle)$ is OK.

DC-F: As argued for DC-I(a), the only possible link of p that is part of $core(f)$ is $\langle p,q\rangle$. Since code sets $inbranch(p)$ to $\langle p,q\rangle$, removing the FIND is OK.

DC-H(a): If $(p,q) = core(f)$, then changing $dcstatus(p)$ to find is OK. If $(p,q) \neq core(f)$, then Claim 15 implies that it is OK to change $dcstatus(p)$ to find.

DC-I: (a) If $(p, q) = core(f)$, then code gives the result, since $inbranch(p)$ is set to $\langle p, q\rangle$ and $dcstatus(p)$ is set to find. If $(p, q) \neq core(f)$, then Claim 17, the fact that p is a child of q by DC-D(a), and code give the result. (b) by Claim 6.

DC-J: By Claims 1 and 2.

DC-K: (a) $findcount(p) = |S|$ = number of children of p. None is complete, by Claim 6. (b) and (c) are true by code, since no children are complete.

DC-L: by code and Claim 3.

DC-M: by code.

DC-O: Removing the FIND from $dcqueue(\langle q, p\rangle)$ is OK. Adding FIND to $dcqueue(\langle p, r\rangle)$, $\langle p, r\rangle \in S$, is OK by Claim 16.

vi) π is Merge(f,g).

(3c) $\mathcal{A}_4(s', \pi) = \pi$. Obviously π is enabled in $\mathcal{S}_4(s')$. Effects are mirrored in $\mathcal{S}_4(s)$ if we can show $accmin(h) = minlink(h) = nil$ in s. Inspecting the code reveals that in s, a FIND message is in $subtree(h)$, so $minlink(h) = nil$, and $nodes(h) = testset(h)$, so $accmin(h) = nil$.

(3a) *Claims about s':*

1. $f \neq g$, by precondition.
2. $rootchanged(f)$ = true, by precondition.
3. $rootchanged(g)$ = true, by precondition.
4. $minedge(f) = minedge(g)$, by precondition.
5. $minlink(f) \neq nil$, by Claim 2 and COM-B.
   Let $\langle p, q\rangle = minlink(f)$.
6. $minlink(g) = \langle q, p\rangle$, by Claims 1, 4 and 5.
7. No REPORT is headed toward $root(f)$, by Claim 5.
8. No REPORT is headed toward $root(g)$, by Claim 6.
9. No FIND is in $subtree(f)$, by Claim 5.
10. No FIND is in $subtree(g)$, by Claim 6.
11. $dcstatus(r)$ = unfind for all $r \in nodes(f)$, by Claim 5.
12. $dcstatus(r)$ = unfind for all $r \in nodes(g)$, by Claim 6.
13. $\langle p, q\rangle$ is the minimum-weight external link of f, by Claim 5 and COM-A.
14. $\langle q, p\rangle$ is the minimum-weight external link of g, by Claim 6 and COM-A.
15. $testset(f) = \emptyset$, by Claim 5 and GC-C.

16. $testset(g) = \emptyset$, by Claim 6 and GC-C.
17. If REPORT is in $dcqueue(\langle r, t\rangle)$, then $inbranch(t) = \langle t, r\rangle$, for all $(r, t) \in subtree(f)$, by Claims 9 and 11 and DC-A(a) and (f).
18. If REPORT is in $dcqueue(\langle r, t\rangle)$, then $inbranch(t) = \langle t, r\rangle$, for all $(r, t) \in subtree(f)$, by Claims 10 and 12 and DC-A(a) and (f).
19. If REPORT is in $dcqueue(\langle r, t\rangle)$ and $(r, t) = core(f)$, then $r = root(f)$, by Claim 7.
20. If REPORT is in $dcqueue(\langle r, t\rangle)$ and $(r, t) = core(g)$, then $r = root(g)$, by Claim 8.
21. If REPORT is in $dcqueue(\langle r, t\rangle)$ and $(r, t) \neq core(f)$, then t is a child of r, for all $(r, t) \in subtree(f)$, by Claim 17 and DC-B(a).
22. If REPORT is in $dcqueue(\langle r, t\rangle)$ and $(r, t) \neq core(g)$, then t is a child of r, for all $(r, t) \in subtree(g)$, by Claim 18 and DC-B(a).
23. If REPORT is in $dcqueue(\langle r, t\rangle)$, then (r, t) is not on the path between $root(f)$ and p, for all $(r, t) \in subtree(f)$, by Claims 5, 7, 13, 15 and 17 and DC-N.
24. If REPORT is in $dcqueue(\langle r, t\rangle)$, then (r, t) is not on the path between $root(g)$ and q, for all $(r, t) \in subtree(g)$, by Claims 6, 8, 14, 16 and 18 and DC-N.
25. $dcqueue(\langle p, q\rangle)$ is empty, by Claim 13 and DC-A(g), DC-B(a) and DC-D(a).
26. $dcqueue(\langle q, p\rangle)$ is empty, by Claim 14 and DC-A(g), DC-B(a) and DC-D(a).
27. $findcount(r) = 0$ for all $r \in nodes(f)$, by Claim 11 and DC-H(b).
28. $findcount(r) = 0$ for all $r \in nodes(g)$, by Claim 12 and DC-H(b).

Claims about s:

29. $subtree(h)$ is the old $subtree(f)$ and $subtree(g)$ and (p, q), by code.
30. $core(h) = (p, q)$, by code.
31. $testset(h) = nodes(h)$, by code.
32. $dcqueue(\langle p, q\rangle)$ contains only a FIND, by Claim 25 and code.
33. No FIND is in any other link of $subtree(h)$, by Claims 9, 10 and 29.
34. $dcstatus(r) =$ unfind for all $r \in nodes(h)$, by Claims 11, 12 and 29.
35. If REPORT is in $dcqueue(\langle r, t\rangle)$, then $inbranch(t) = \langle t, r\rangle$, for all $(r, t) \in subtree(h)$, by Claims 17, 18, 25, 26 and 29.
36. If REPORT is in $dcqueue(\langle r, t\rangle)$, then t is a child of r, for all $(r, t) \in subtree(h)$, by Claims 21 through 26 and 28.
37. $AfterMerge(q, p)$ is enabled, by Claims 30, 32, 33 and 34.
38. $dcqueue(\langle q, p\rangle)$ is empty, by Claim 26.
39. $findcount(r) = 0$ for all $r \in nodes(h)$, by Claims 27, 28 and 29.

DC-A: Vacuously true, by Claim 35.

DC-B: By Claims 34 and 36.

DC-C: By Claims 30, 32 and 38.

DC-D: The only FIND is in $dcqueue(\langle p,q\rangle)$, by Claims 32 and 33. (a) by Claim 30. (b) by Claim 34. (c) by Claim 31.

DC-E: By Claim 32 for $subtree(q)$; by Claim 37 for $subtree(p)$.

DC-F: By Claims 32 and 37.

DC-G: By Claim 31.

DC-H: (a) by Claim 34. (b): by Claim 39.

DC-I: Vacuously true by Claim 34.

DC-J: Vacuously true by Claim 34.

DC-K: By Claims 31 and 34, none is up-to-date.

DC-M: By Claim 39.

DC-N: Vacuously true by Claim 31.

DC-O: By Claim 30.

vii) π is AfterMerge(p,q). Let $f = fragment(p)$.

(3b) $\mathcal{A}_4(s', \pi)$ is empty. We just need to show that $accmin(f)$ and $minlink(f)$ do not change. The FIND message(s) imply that $minlink(f) = nil$ in both s' and s. Since there is no change to $minlink(f)$, $nodes(f)$, $testset(f)$, or $subtree(f)$, $accmin(f)$ does not change.

(3a) *Claims about* s'*:*

1. $(p,q) = core(f)$, by precondition.
2. FIND is in $dcqueue(\langle q,p\rangle)$, by precondition.
3. No FIND is in $dcqueue(\langle p,q\rangle)$, by precondition.
4. $dcstatus(q) =$ unfind, by precondition.
5. No REPORT is in $dcqueue(\langle q,p\rangle)$, by precondition.
6. Every node in $subtree(q)$ is in $testset(f)$, by Claims 1 through 5 and DC-G.
7. $p \in testset(f)$, by Claim 2 and DC-D(c).

8. No REPORT is in $dcqueue(\langle p,q\rangle)$, by Claim 7 and DC-C.
9. $dcqueue(\langle q,p\rangle)$ consists solely of a FIND, by Claims 2 and 5 and DC-O.
10. $dcqueue(\langle p,q\rangle)$ is empty, by Claims 3 and 8 and DC-O.
11. $(p,q) \in subtree(f)$, by Claim 1 and COM-F.

Claims about s:

12. $(p,q) = core(f)$, by Claim 1.
13. Every node in $subtree(q)$ is in $testset(f)$, by Claim 6.
14. $dcqueue(\langle q,p\rangle)$ consists solely of FIND, by Claim 9.
15. $dcqueue(\langle p,q\rangle)$ consists solely of FIND, by Claim 10 and code.
16. $dcstatus(q) =$ unfind, by Claim 4.
17. $AfterMerge(p,q)$ is not enabled, by Claim 15.
18. $AfterMerge(q,p)$ is not enabled, by Claim 14.

DC-D: (a) by Claim 12. (b) by Claim 16. (c) by Claim 13.

DC-E: By Claim 15 (FIND in $dcqueue(\langle p,q\rangle)$ replaces $AfterMerge(p,q)$ being enabled).

DC-F: By Claim 15 (FIND in $dcqueue(\langle p,q\rangle)$ replaces $AfterMerge(p,q)$ being enabled).

DC-G: vacuously true by Claims 17 and 18.

DC-O: By Claim 15.

viii) π is Absorb(f,g).

(3c) $\mathcal{A}_4(s',\pi) = \pi$. Obviously π is enabled in $\mathcal{S}_4(s')$. Effects are mirrored in $\mathcal{S}_4(s)$ if we can show that $accmin(f)$ and $minlink(f)$ do not change.

Case 1: $p \in testset(f)$ in s'. By GC-C, $minlink(f) = nil$ in s'. By inspecting the code, a FIND message is in $subtree(f)$ in s, so $minlink(f) = nil$ in s also.

Suppose $accmin(f) = nil$ in s'. Then there is no external link of any $q \in nodes(f) - testset(f)$ in s'. Since $testset(f)$ does not change and no formerly internal links become external, $accmin(f) = nil$ in s also.

Suppose $accmin(f) = \langle q,r\rangle$ in s'. By GC-A, $level(f) \leq level(fragment(r))$. So by precondition, $fragment(r) \neq g$. Since all of $nodes(g)$ is added to $testset(f)$, there is no change to $nodes(f) - testset(f)$. Thus $accmin(f)$ is unchanged.

Case 2: $p \notin testset(f)$ in s'.

Claims about s':

1. $rootchanged(g) =$ true, by precondition.
2. $level(g) < level(f)$, by precondition.
3. $minlink(g) = \langle q, p \rangle \neq nil$, by precondition.
4. $fragment(p) = f$, by precondition.
5. $dcstatus(r) =$ unfind for all $r \in nodes(g)$, by Claim 3.
6. No FIND message is in $subtree(g)$, by Claim 3.
7. No REPORT message is headed toward $mw\text{-}root(g)$, by Claim 3.
8. $root(g) = mw\text{-}root(g)$, by Claim 3 and COM-A.
9. $wt(l) > wt(q, p)$ for all external links l of g, by Claim 3 and COM-A.
10. If $minlink(f) = \langle r, t \rangle$, then $level(fragment(t)) \geq level(f)$, by COM-A.
11. If $minlink(f) = \langle r, t \rangle$, then $g \neq fragment(t)$, by Claims 2 and 10.
12. If $accmin(f) = \langle r, t \rangle$, then $level(fragment(t)) \geq level(f)$, by GC-A.
13. If $accmin(f) = \langle r, t \rangle$, then $g \neq fragment(t)$, by Claims 2 and 12.

If $minlink(f) = nil$ in s', then obviously it is still nil in s. Suppose $minlink(f) = \langle r, t \rangle$ in s'. By Claims 5, 6, 7, 8 and 11 (and code), $minlink(f) = \langle r, t \rangle$ in s as well.

If $accmin(f) = \langle r, t \rangle$ in s', then it is unchanged in s by Claims 9 and 13. Suppose $accmin(f) = nil$ in s'. If this is because $minlink(f) \neq nil$ in s', then, since we just showed that $minlink(f)$ does not change, $accmin(f)$ is still nil in s. Suppose $accmin(f) = nil$ not because $minlink(f) = nil$, but because no node in $nodes(f) - testset(f)$ has an external link. But by the assumption for this case, $p \notin testset(f)$, yet it is in $nodes(f)$ by Claim 4, and $\langle p, q \rangle$ is an external link of p by Claim 3 and COM-A.

(3a) We consider two cases. First we prove some facts true in both cases.

Claims about s':

1. $rootchanged(g) =$ true, by precondition.
2. $level(g) < level(f)$, by precondition.
3. $minlink(g) = \langle q, p \rangle$, by precondition.
4. $p \in nodes(f)$, by precondition.
5. No REPORT is headed toward $root(g)$, by Claim 3.
6. No FIND is in $subtree(g)$, by Claim 3.

7. $dcstatus(r) =$ unfind, for all $r \in nodes(g)$, by Claim 3.
8. $\langle q, p\rangle$ is the minimum-weight external link of g, by Claim 3 and COM-A.
9. $testset(g) = \emptyset$, by Claim 3 and GC-C.
10. q is up-to-date, by Claim 9 and DC-N.
11. Following *bestlinks* from q leads toward and over $core(g)$, by Claim 10.
12. If REPORT is in $dcqueue(\langle r,t\rangle)$, then $inbranch(t) = \langle t, r\rangle$, for all $(r,t) \in subtree(g)$, by Claims 6 and 7 and DC-A(a) and (f).
13. If REPORT is in $dcqueue(\langle r,t\rangle)$ and $(r,t) = core(f)$, then $r = root(g)$, for all $(r,t) \in subtree(g)$, by Claim 5.
14. If REPORT is in $dcqueue(\langle r,t\rangle)$ and $(r,t) \neq core(f)$, then t is a child of r, for all $(r,t) \in subtree(g)$, by Claim 9 and DC-B(a).
15. If REPORT is in $dcqueue(\langle r,t\rangle)$, then (r,t) is not on the path between $root(g)$ and q, for all $(r,t) \in subtree(g)$, by Claims 3, 5, 8, 9 and DC-N.
16. No REPORT is headed toward q, by Claims 5, 14 and 15.
17. $dcqueue(\langle p,q\rangle)$ and $dcqueue(\langle q,p\rangle)$ are empty, by Claim 8 and DC-A(g), DC-B(a) and DC-D(a).

Case 1: $p \notin testset(f)$.

More claims about s'*:*

18. $p \notin testset(f)$, by assumption.
19. $AfterMerge(r,t)$, where $p \in subtree(t)$, is not enabled, by Claim 18 and DC-G.
20. No FIND is headed toward p, by Claim 18 and DC-C(a).

DC-A: By Claim 12, vacuously true for any REPORT in old g. For a REPORT that could be in some $dcqueue(\langle r,t\rangle)$ with $p \in subtree(t)$: (e) by Claims 16 and 17.

DC-B: By Claim 16, change in location of core for nodes formerly in g is OK.

DC-D(a): by Claim 6, change in location of core for nodes formerly in g is OK. By Claim 20, it is OK not to add $nodes(g)$ to $testset(f)$.

DC-G: By Claim 19, vacuously true.

DC-H(a): By Claim 7.

DC-K: Choose any up-to-date node r in $nodes(f)$ in s. By Claims 7 and 11 and code, no node that is in $nodes(g)$ in s' is up-to-date in s. Thus r is in $nodes(f)$ in s', and is up-to-date.

(a) If $r = p$, then *findcount*(p) is changed (incremented by 1) if and only if the number of children of p that are not completed is changed (increased by 1). If $r \neq p$, then neither *findcount*(r) nor the number of children of r that are not completed is changed.

(b) Suppose $bestlink(r) = nil$ in s. Then the same is true in s'. By DC-K(b), $bestwt(r) = \infty$ and there is no external link of C_r in s'. In going to s, there is no change to $bestwt(r)$, and no internal links become external.

(c) Suppose $bestlink(r) \neq nil$ in s. Then the same is true in s'. Let l be the minimum-weight external link of C_r in s'. By DC-K(c), following *bestlinks* from r leads to l, $wt(l) = bestwt(r)$, and $level(h) \geq level(f)$, where $h = fragment(target(l))$, in s'. By the precondition on $level(g)$, $h \neq g$ in s', and thus l is still external in s. If $p \notin C_r$ in s', then C_r is unchanged in s, and the predicate is still true. Suppose $p \in C_r$ in s'. By COM-A, $wt(p,q)$ is less than the weight of any other external link of g, and thus $wt(l)$ is less than the weight of any external link of g in s'. Thus adding all the nodes of g to C_r in going to s does not falsify the predicate.

DC-O: By Claim 6, the former $core(g)$ is OK.

DC-N: Let l be the minimum-weight external link of f in s'. If $l \neq \langle p,q\rangle$, then $wt(l) < wt(p,q)$, and by Claim 8, $wt(l) < wt(l')$ for any external link l' of g. Thus, in s, l is still the minimum-weight external link of s, and DC-N is true in s.

Now suppose $l = \langle p,q\rangle$. By DC-N and Claim 18, p is up-to-date. But by DC-K(b) and (c), $bestlink(p) = \langle p,q\rangle$ and $level(f) \leq level(g)$, wich contradicts Claim 2.

Case 2: $p \in testset(f)$.

More claims about s'*:*

21. $p \in testset(f)$, by assumption.
22. For all $\langle r,t\rangle$ such that $p \in subtree(r)$ and $inbranch(t) = \langle t,r\rangle$, no REPORT is in $dcqueue(\langle r,t\rangle)$, by Claim 21 and DC-A(e).
23. A FIND is headed toward p, or $dcstatus(p) =$ find, or $AfterMerge(r,t)$ is enabled, where $p \in subtree(t)$, by Claim 21 and DC-E.

DC-A(e): by Claim 22, the addition of uncompleted child q to p is OK.

DC-B: As in Case 1.

DC-D: As in Case 1.

DC-E: By Claim 23.

DC-G: By code, since all of *nodes*(g) is added to *testset*(f).

DC-H: By Claim 7.

DC-K: As in Case 1.

DC-M: By code, since *findcount*(p) is incremented.

DC-N: By code, since all of *nodes*(g) is added to *testset*(f).

DC-O: By Claim 17 and code. □

Let $P'_{DC} = (P'_{GC} \circ \mathcal{S}_4) \wedge P_{DC}$.

Corollary 20: *P'_{DC} is true in every reachable state of DC.*

Proof: By Lemmas 1 and 19. □

4.2.5 NOT Simulates COM

This automaton refines on *COM* by implementing the level and core of a fragment with local variables *nlevel*(p) and *nfrag*(p) for each node p in the fragment, and with NOTIFY messages. When two fragments merge, a NOTIFY message is sent over one link of the new core, carrying the level and core of the newly created fragment. The action *AfterMerge*(p, q) adds such a NOTIFY message to the other link of the new core. A *ComputeMin*(f) action cannot occur until the source of *minlink*(f) has the correct *nlevel*, and the target of *minlink*(f) has an *nlevel* at least as big as the source's. The preconditions for *Absorb*(f, g) now include the fact that the level of fragment g must be less than the *nlevel* of the target of *minlink*(g). When an *Absorb*(f, g) occurs, a NOTIFY message is sent to the old fragment g, over the reverse link of *minlink*(g), with the *nlevel* and *nfrag* of the target of *minlink*(g).

Define automaton *NOT* (for "Notify") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- $subtree(f)$, a subgraph of G;
- $minlink(f)$, a link of G or nil; and
- $rootchanged(f)$, a Boolean.

For each node p, there are associated two variables:

- $nlevel(p)$, a nonnegative integer; and
- $nfrag(p)$, an edge of G or nil.

For each link $\langle p, q\rangle$, there are associated three variables:

- $nqueue_p(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at p to be sent;
- $nqueue_{pq}(\langle p, q\rangle)$, a FIFO queue of messages from p to q that are in the communication channel; and
- $nqueue_q(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at q to be processed.

The set of possible messages M is $\{\text{NOTIFY}(l, c) : l \geq 0, c \in E(G)\}$. The state also contains Boolean variables, $answered(l)$, one for each $l \in L(G)$, and Boolean variable $awake$.

In the start state of NOT, $fragments$ has one element for each node in $V(G)$; for fragment f corresponding to node p, $subtree(f) = \{p\}$, $minlink(f)$ is the minimum-weight link adjacent to p, and $rootchanged(f)$ is false. For each node p, $nlevel(p) = 0$ and $nfrag(p) = nil$. The message queues are empty. Each $answered(l)$ is false and $awake$ is false.

We say that a message m is *in subtree*(f) if m is in some $nqueue(\langle q, p\rangle)$ and $p \in nodes(f)$. A NOTIFY message is *headed toward* p if it is in $nqueue(\langle q, r\rangle)$ and $p \in subtree(r)$. The following are derived variables:

- For link $\langle p, q\rangle$, $nqueue(\langle p, q\rangle)$ is defined to be $nqueue_q(\langle p, q\rangle) \mid\mid nqueue_{pq}(\langle p, q\rangle) \mid\mid nqueue_p(\langle p, q\rangle)$.
- For fragment f, $level(f) = \max\{l : nlevel(p) = l$ for $p \in nodes(f)$, or a NOTIFY(l, c) message is in $subtree(f)$ for some $c\}$.

- For fragment f, $core(f) = nfrag(p)$ if $nlevel(p) = level(f)$ for some $p \in nodes(f)$, and $core(f) = c$, if a NOTIFY$(level(f), c)$ message is in $subtree(f)$.

As for the DC action $ReceiveFind$, $ReceiveNotify(\langle q, p\rangle, l, c)$ is only enabled if $AfterMerge(p, q)$ is not enabled, in order to make sure that q's side of the subtree is notified of the new information.

Input actions:

- $Start(p)$, $p \in V(G)$
  Effects:
  $awake :=$ true

Output actions:

- $InTree(\langle p, q\rangle)$, $\langle p, q\rangle \in L(G)$
  Preconditions:
  $awake =$ true
  $(p, q) \in subtree(fragment(p))$ or $\langle p, q\rangle = minlink(fragment(p))$
  $answered(\langle p, q\rangle) =$ false
  Effects:
  $answered(\langle p, q\rangle) :=$ true

- $NotInTree(\langle p, q\rangle)$, $\langle p, q\rangle \in L(G)$
  Preconditions:
  $fragment(p) = fragment(q)$ and $(p, q) \notin subtree(fragment(p))$
  $answered(\langle p, q\rangle) =$ false
  Effects:
  $answered(\langle p, q\rangle) :=$ true

Internal actions:

- $ChannelSend(\langle p, q\rangle, m)$, $\langle p, q\rangle \in L(G)$, $m \in M$
  Preconditions:
  m at head of $nqueue_p(\langle p, q\rangle)$
  Effects:
  dequeue($nqueue_p(\langle p, q\rangle)$)
  enqueue(m, $nqueue_{pq}(\langle p, q\rangle)$)

- $ChannelRecv(\langle p, q\rangle, m)$, $\langle p, q\rangle \in L(G)$, $m \in M$
  Preconditions:

m at head of $nqueue_{pq}(\langle p, q \rangle)$

Effects:

dequeue($nqueue_{pq}(\langle p, q \rangle)$)

enqueue($m, nqueue_q(\langle p, q \rangle)$)

- $ReceiveNotify(\langle q, p \rangle, l, c)$, $\langle q, p \rangle \in L(G)$, $l \geq 0$, $c \in E(G)$

  Preconditions:

  NOTIFY(l, c) at head of $nqueue_p(\langle q, p \rangle)$

  $AfterMerge(p, q)$ not enabled

  Effects:

  dequeue($nqueue_p(\langle q, p \rangle)$)

  $nlevel(p) := l$

  $nfrag(p) := c$

  — let $S = \{\langle p, r \rangle : (p, r) \in subtree(fragment(p)), r \neq q\}$ —

  enqueue(NOTIFY(l, c), $nqueue_p(k)$) for all $k \in S$

- $ComputeMin(f)$, $f \in fragments$

  Preconditions:

  $minlink(f) = nil$

  $\langle p, q \rangle$ is the minimum-weight external link of f

  $nlevel(p) = level(f)$

  $level(f) \leq nlevel(q)$

  Effects:

  $minlink(f) := l$

- $ChangeRoot(f)$, $f \in fragments$

  Preconditions:

  $awake =$ true

  $rootchanged(f) =$ false

  $minlink(f) \neq nil$

  Effects:

  $rootchanged(f) :=$ true

- $Merge(f, g)$, $f, g \in fragments$

  Preconditions:

  $f \neq g$

  $rootchanged(f) = rootchanged(g) =$ true

  $minedge(f) = minedge(g)$

  Effects:

  add a new element h to *fragments*

$subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$
$minlink(h) := nil$
$rootchanged(h) :=$ false
— let $(p, q) = minedge(f)$ —
enqueue(NOTIFY($nlevel(p) + 1, (p, q)$), $nqueue_p(\langle p, q \rangle)$)
delete f and g from $fragments$

- $AfterMerge(p, q)$, $p, q \in V(G)$
  Preconditions:
  $(p, q) = core(fragment(p))$
  NOTIFY($nlevel(p) + 1, (p, q)$) message in $nqueue(\langle q, p \rangle)$
  no NOTIFY($nlevel(p) + 1, (p, q)$) message in $nqueue(\langle p, q \rangle)$
  $nlevel(q) \neq nlevel(p) + 1$
  Effects:
  enqueue(NOTIFY($nlevel(p) + 1, (p, q)$), $nqueue_p(\langle p, q \rangle)$)

- $Absorb(f, g)$, $f, g \in fragments$
  Preconditions:
  $rootchanged(g) =$ true
  — let $\langle q, p \rangle = minlink(g)$ —
  $level(g) < nlevel(p)$
  $fragment(p) = f$
  Effects:
  $subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$
  enqueue(NOTIFY($nlevel(p), nfrag(p)$), $nqueue_p(\langle p, q \rangle)$)
  delete g from $fragments$

Define the following predicates on states of NOT. (All free variables are universally quantified.)

- NOT-A: $core(f)$ is well-defined. (I.e., the set of all c such that a NOTIFY($level(f), c$) is in $subtree(f)$ or some $p \in nodes(f)$ has $nlevel(p) = level(f)$ and $nfrag(p) = c$, has exactly one element.)

- NOT-B: If $q \in subtree(p)$, then $nlevel(q) \leq nlevel(p)$.

- NOT-C: If $(p, q) = core(f)$, then $nlevel(p) \geq level(f) - 1$.

- NOT-D: If $minlink(f) = \langle p, q \rangle$, then $nlevel(p) = level(f) \leq nlevel(q)$.

- NOT-E: If $nfrag(p) = core(fragment(p))$, then $nlevel(p) = level(fragment(p))$.

- NOT-F: Either $nlevel(p) = 0$ and $nfrag(p) = nil$, or else $nlevel(p) > 0$ and $nfrag(p) \in subtree(fragment(p))$.

- NOT-G: If $nlevel(p) < level(fragment(p))$, then either a NOTIFY($level$ ($fragment(p)$), $core(fragment(p))$) message is headed toward p, or else $AfterMerge$ (q, r) is enabled, where $p \in subtree(r)$.

- NOT-H: If a NOTIFY(l, c) message is in $nqueue(\langle q, p\rangle)$, then
  (a) $nlevel(p) < l$;
  (b) if $(p, q) \neq core(fragment(p))$, then $nlevel(q) \geq l$;
  (c) if $c = core(fragment(p)))$ then $l = level(fragment(p))$;
  (d) if NOTIFY(l', c') is ahead of the NOTIFY(l, c) in $nqueue(\langle q, p\rangle)$, then $l' < l$;
  (e) p is a child of q, or $(p, q) = core(fragment(p))$;
  (f) if $(p, q) = core(fragment(p))$, then $l = level(fragment(p))$;
  (g) $c \in subtree(fragment(p))$; and
  (h) $l > 0$.

Let P_{NOT} be the conjunction of NOT-A through NOT-H.

In order to show that NOT simulates COM, we define an abstraction mapping $\mathcal{M}_5 = (\mathcal{S}_5, \mathcal{A}_5)$ from NOT to COM. Define the function $\mathcal{S}_5$ from $states(NOT)$ to $states(COM)$ by simply ignoring the message queues, and mapping the derived variables $level(f)$ and $core(f)$ in the NOT state to the (non-derived) variables $level(f)$ and $core(f)$ in the COM state. Define the function $\mathcal{A}_5$ as follows. Let s be a state of NOT and π an action of NOT enabled in s.

- If $\pi = ChannelSend(k, m)$, $ChannelRecv(k, m)$, $ReceiveNotify(k, l, c)$, or $AfterMerge(p, q)$, then $\mathcal{A}_5(s, \pi)$ is empty.

- For all other values of π, $\mathcal{A}_5(s, \pi) = \pi$.

The following predicates are true in any state of NOT satisfying $(P'_{COM} \circ \mathcal{S}_5) \wedge P_{NOT}$. Recall that $P'_{COM} = (P_{S1} \circ \mathcal{S}_1) \wedge P_{COM}$. If $P'_{COM}(\mathcal{S}_5(s))$ is true, then the COM predicates are true in $\mathcal{S}_5(s)$, and the S1 predicates are true in $\mathcal{S}_1(\mathcal{S}_5(s))$. Thus, these predicates follow from P_{NOT}, together with the HI and COM predicates.

- NOT-I: If $p = minnode(f)$, then no NOTIFY message is headed toward p.

- NOT-J: For all p, at most one NOTIFY(l, c) message is headed toward p, for a fixed l.

Lemma 21: *NOT simulates COM via $\mathcal{M}_5$, P_{NOT}, and P'_{COM}.*

Proof: By inspection, the types of NOT, COM, $\mathcal{M}_5$, and P_{NOT} are correct. By Corollary 14, P'_{COM} is a predicate true in every reachable state of COM.

(1) Let s be in $start(NOT)$. Obviously P_{NOT} is true in s and $\mathcal{S}_5(s)$ is in $start(COM)$.

(2) Obviously, $\mathcal{A}_5(s,\pi)|ext(COM) = \pi|ext(NOT)$.

(3) Let (s',π,s) be a step of NOT such that P'_{COM} is true of $\mathcal{S}_5(s')$ and P_{NOT} is true of s'. Below, we only show (3a) for those predicates that are not obviously true in s.

i) π is Start(p), InTree(l), NotInTree(l), or ChangeRoot(f). $\mathcal{A}_5(s',\pi) = \pi$. Obviously, $\mathcal{S}_5(s')\pi\mathcal{S}_5(s)$ is an execution fragment of COM, and P_{NOT} is true in s.

ii) π is ChannelSend(l,m) or ChannelRecv(l,m). $\mathcal{A}_5(s',\pi)$ is empty. Obviously, $\mathcal{S}_5(s') = \mathcal{S}_5(s)$, and P_{NOT} is true in s.

iii) π is ReceiveNotify($\langle$q,p$\rangle$,l,c). Let $f = fragment(p)$.

(3b) $\mathcal{A}_5(s',\pi)$ is empty. To show that $\mathcal{S}_5(s) = \mathcal{S}_5(s')$, we only need to show that $level(f)$ and $core(f)$ don't change. By NOT-H(a), $nlevel(p) < l$ in s', and thus $nlevel(p) \neq level(f)$. So changing $nlevel(p)$ is OK. Also, since $nlevel(p)$ and $nfrag(p)$ are set to l and c, removing the NOTIFY(l,c) from $nqueue(\langle q,p\rangle)$ is OK.

(3a) NOT-A: By code.

NOT-B: By NOT-B, $nlevel(q) \leq nlevel(r)$ for all r such that $q \in subtree(r)$ in s'. By NOT-H(b), if $(p,q) \neq core(f)$, then $nlevel(q) \geq l$ in s'. Since $nlevel(p) = l$ in s, the predicate is true.

NOT-C: Since this predicate is true in s' and fact that $nlevel(p)$ increases.

NOT-D: As argued in (3b), $nlevel(p) < l \leq level(f)$. By NOT-D, $p \neq minnode(f)$ in s', or in s. Suppose $p = target(minlink(g))$ in s', for some g. Since $nlevel(p)$ increases in going from s' to s, the predicate is still true in s.

NOT-E: By NOT-H(c), $c = core(f)$ implies that $l = level(f)$ in s'. So in s, $c = nfrag(p) = core(f)$ implies that $l = nlevel(p) = level(f)$.

NOT-F: By NOT-H(g), $c \neq nil$, and by NOT-H(h), $l > 0$ in s'. Thus in s, $c = nfrag(p) \neq nil$ and $l = nlevel(p) \neq 0$.

NOT-G: The NOTIFY(l, c) message removed from $nqueue(\langle q, p\rangle)$ is replaced by the NOTIFY(l, c) messages added to $nqueue(\langle p, r\rangle)$, for all $\langle p, r\rangle \in S$.

NOT-H: Suppose NOTIFY(l, c) is added to $nqueue(p, r)$ in s. (I.e., $\langle p, r\rangle \in S$.)

Claims about s':

1. NOTIFY(l, c) is at head of $nqueue(\langle q, p\rangle)$, by precondition.
2. $p \in subtree(q)$ or $(p, q) = core(f)$, by Claim 1 and NOT-H(e).
3. $r \in subtree(p)$, by Claim 2 and definition of S.
4. $nlevel(r) \leq nlevel(p)$, by Claim 3 and NOT-B.
5. $nlevel(p) < l$, by Claim 1 and NOT-H(a).
6. If NOTIFY(l', c') is in $nqueue(\langle p, r\rangle)$, then $l' < l$, by Claims 3 and 5 and NOT-H(b).
7. $nlevel(r) < l$, by Claims 4 and 5.

(a) by Claim 7. (b) by Claim 3. (d) by Claim 7. (e) by Claim 3. (f) vacuously true by Claim 3. (c), (g) and (h) since the same is true for the NOTIFY(l, c) in $nqueue(\langle q, p\rangle)$ in s'.

iv) π is ComputeMin(f).

(3c) $\mathcal{A}_5(s', \pi) = \pi$. Obviously π is enabled in $\mathcal{S}_5(s')$, since by definition $nlevel(q) \leq level(fragment(q))$. The effects are obviously mirrored in $\mathcal{S}_5(s)$.

(3a) By the preconditions, NOT-D is true in s. No other predicate is affected.

v) π is Merge(f,g).

(3c) $\mathcal{A}_5(s', \pi) = \pi$. Obviously π is enabled in $\mathcal{S}_5(s')$. To show that its effects are mirrored in $\mathcal{S}_5(s)$, we show that $level(h)$ and $core(h)$ are correct. Let $minlink(f) = \langle p, q\rangle$ and $l = level(f)$ in s'.

Claims about s':

1. $minedge(f) = minedge(g)$, by precondition.
2. $level(g) = l$, by Claim 1 and COM-A.
3. $rootchanged(f) =$ true, by precondition.
4. $minlink(f) \neq nil$, by Claim 3 and COM-B.
5. $nlevel(p) = l$, by Claim 4 and NOT-D.
6. $nlevel(r) \leq l$ for all $r \in nodes(f)$, by definition of $level(f)$.
7. If NOTIFY(m, c) is in $subtree(f)$, then $m \leq l$, by definition of $level(f)$.
8. $rootchanged(g) =$ true, by precondition.

9. $minlink(g) \neq nil$, by Claim 8 and COM-B.
10. $nlevel(q) = l$, by Claims 2 and 9 and NOT-D.
11. $nlevel(r) \leq l$ for all $r \in nodes(g)$, by definition of $level(g)$.
12. If NOTIFY(m, c) is in $subtree(g)$, then $m \leq l$, by definition of $level(g)$.
13. $\langle p, q\rangle$ is an external link of f, by COM-A.
14. $nqueue(\langle p, q\rangle)$ and $nqueue(\langle q, p\rangle)$ are empty, by Claim 13 and NOT-H(e).

Claims about s:

15. $nlevel(r) < l + 1$, for all $r \in nodes(h)$, by Claims 6 and 11 and code.
16. The only NOTIFY message in $subtree(h)$ with level greater than l is the NOTIFY$(l+1, (p, q))$ message added to $nqueue(\langle p, q\rangle)$, by Claims 7, 12 and 14 and code.
17. $level(h) = l + 1$, by Claims 15 and 16.
18. $core(h) = (p, q)$, by Claims 15 and 16.

Claims 17 and 18 give the result.

(3a) Only fragment h needs to be checked.

NOT-A: By Claims 15 and 16.

NOT-B: As argued in the proof of NOT-I, $nlevel(r) = l$ for all r on the path from $core(f)$ to p, and all r on the path from $core(g)$ to q. Since these are the only nodes affected by the change of core, the predicate is still true in s.

NOT-C: By Claims 5, 10 and 17.

NOT-D: vacuously true since $minlink(h) = nil$ by code.

NOT-E: By NOT-F and Claim 13, $nfrag(r) \neq (p, q)$ for all r in $nodes(f)$ or $nodes(g)$. So the predicate is vacuously true.

NOT-F: No relevant change.

NOT-G: If r is in $nodes(g)$ in s', the predicate is true in s because of Claims 17 and 18 and the NOTIFY$(l + 1, (p, q))$ added to $nqueue(\langle p, q\rangle)$ in s. If r is in $nodes(f)$ in s', then $AfterMerge(q, p)$ is enabled in s, by code and Claims 5, 10, 14 and 18.

NOT-H for the NOTIFY$(l+1, (p, q))$ added to $nqueue(\langle p, q\rangle)$: (a) $nlevel(q) < l+1$, by Claim 15. (b) By Claim 18. (c) By Claim 17. (d) Vacuously true by Claim 14. (e) By Claim 18. (f) By Claims 17 and 18. (g) By code. (h) By COM-F, $l \geq 0$, so $l + 1 > 0$.

NOT-H for any NOTIFY(l', c') message in $subtree(f)$ in s' (similar argument for g): (a), (d), (g) and (h) No relevant change.

(b) Suppose the message is in a link of $core(f) = (r, t)$. Suppose $p \in subtree(t)$. By NOT-I, the message is not in $nqueue(\langle r, t\rangle)$. As argued in the proof of NOT-I, $nlevel(t) = l$. If the message is in $nqueue(\langle t, r\rangle)$, then, since $l' \leq l$, the predicate is true in s.

(c) By Claim 13 and NOT-H(g), $c' \neq (p, q)$, so the predicate is vacuously true in s.

(e) The only nodes for which the subtree relationship changes are those along the path from $core(f)$ to p. By NOT-I, there is no NOTIFY message in this path.

(f) Vacuously true, by Claim 18.

vi) π **is AfterMerge(p,q).** Let $f = fragment(p)$.

(3b) $\mathcal{A}_5(s')$ is empty. Obviously $\mathcal{S}_5(s') = \mathcal{S}_5(s)$.

(3a) Let $l = nlevel(p) + 1$ and $c = (p, q)$.

NOT-A: Obvious.

NOT-B, C, D, and E: No relevant changes.

NOT-G: The NOTIFY(l, c) message added to $nqueue(\langle p, q\rangle)$ in s compensates for the fact that $AfterMerge(p, q)$ goes from enabled in s' to disabled in s.

NOT-H: Let $c = (p, q)$ and $l = nlevel(p) + 1$. Consider the NOTIFY(l, c) added to $nqueue(\langle p, q\rangle)$.

1. $(p, q) = core(f)$, by precondition.
2. NOTIFY(l, c) is in $nqueue(\langle q, p\rangle)$, by precondition.
3. No NOTIFY(l, c) is in $nqueue(\langle p, q\rangle)$, by precondition.
4. $nlevel(q) \neq l$, by precondition.
5. $l = level(f)$, by Claims 1 and 2 and NOT-H(f).
6. $nlevel(q) < l$, by Claims 4 and 5.
7. If NOTIFY(l', c') is in $nqueue(\langle p, q\rangle)$, then $l' = l$, by Claims 1 and 5 and NOT-H(d).
8. If NOTIFY(l', c') is in $nqueue(\langle p, q\rangle)$, then $c' = c$, by Claim 7 and NOT-A.
9. No NOTIFY is in $nqueue(\langle p, q\rangle)$, by Claims 3, 7 and 8.
10. $nlevel(p) \geq 0$, by NOT-F.

(a) by Claim 6. (b) vacuously true, by Claim 1. (c) by Claim 5. (d) by Claim 9. (e) by Claim 1. (f) by Claim 5. (g) by Claim 1 and COM-F. (h) by Claim 10.

vii) π is Absorb(f,g).

(3c) $\mathcal{A}(s', \pi) = \pi$.

Claims about s':

1. $rootchanged(g) =$ true, by precondition.
2. $level(g) < nlevel(p)$, by precondition.
3. $fragment(p) = f$, by precondition.
4. $nlevel(p) \leq level(f)$, by Claim 3 and definition of level.
5. $nlevel(r) \leq level(g)$, for all $r \in nodes(g)$, by definition of level.
6. If NOTIFY(l, c) is in $subtree(g)$, then $l \leq level(g)$, by definition of level.
7. $\langle q, p \rangle$ is an external link of g, by COM-A.
8. $nqueue(\langle p, q \rangle)$ and $nqueue(\langle q, p \rangle)$ are empty, by Claim 7 and NOT-H(e).

By Claim 4, π is enabled in $\mathcal{S}_5(s')$. The effects of π are mirrored in $\mathcal{S}_5(s)$ if $core(f)$ and $level(f)$ are unchanged; by code and Claims 6, 7 and 8, they are unchanged.

(3a) Let $l = nlevel(p)$ and $c = nfrag(p)$ in s'.

More claims about s':

9. $f \neq g$, by Claims 7 and 3.
10. $level(f) > 0$, by Claims 2 and 3 and COM-F.
11. $core(f) \in subtree(f)$, by Claim 10 and COM-F.
12. $nfrag(r) \neq core(f)$, for all $r \in nodes(g)$, by Claim 11 and NOT-F.
13. $nlevel(q) \leq level(g)$, by definition.
14. $nfrag(p) \in subtree(f)$, by Claims 2 and 10 and NOT-F.

NOT-A: by code and Claims 6, 7 and 8.

NOT-B: Same argument as for $Merge(f, g)$.

NOT-D: No relevant changes.

NOT-E: By Claim 12, vacuously true for nodes formerly in $nodes(g)$.

NOT-F: No relevant changes.

NOT-G: Suppose *nlevel*(p) = *level*(f) in s'. By code, in s there is a NOTIFY(*level*$(f), c$) message headed toward every node formerly in *nodes*(g).

Suppose *nlevel*$(p) \neq$ *level*(f) in s'. By NOT-G, either a NOTIFY(*level*$(f), c$) message is headed toward p in s', and thus is headed toward all nodes formerly in *nodes*(g) in s, or *AfterMerge*(r, t) is enabled in s' with $p \in$ *subtree*(t), and thus in s, *AfterMerge*(r, t) is still enabled and every node formerly in *nodes*(g) is in *subtree*(t).

NOT-H for the NOTIFY(l, c) added to *nqueue*$(\langle p, q\rangle)$: (a) by Claims 2 and 12. (b) by code. (c) by NOT-E. (d) vacuously true by Claim 8. (e) q is a child of p, by Claim 11. (f) vacuously true, by Claim 11. (g) by Claim 14. (h) by Claims 2 and 10.

NOT-H for any NOTIFY(l', c') in *subtree*(g) in s': (a), (d), (g) and (h): no relevant change. (b) and (e) same argument as for *Merge*(f, g). (c) vacuously true, by Claim 11. (f) vacuously true, by code. □

Let $P'_{NOT} = (P_{COM} \circ \mathcal{S}_5) \wedge P_{NOT}$.

Corollary 22: *P'_{NOT} is true in every reachable state of NOT.*

Proof: By Lemmas 1 and 21. □

4.2.6 CON Simulates OM

This automaton concentrates on what happens after *minlink*(f) is identified, until fragment f merges or is absorbed, i.e., the *ChangeRoot*(f, g), *Merge*(f, g) and *Absorb*(g, f) actions are broken down into a series of actions, involving message-passsing. The variable *rootchanged*(f) is now derived. As soon as *ComputeMin*(f) occurs, the node adjacent to the core closest to *minlink*(f) sends a CHANGEROOT message on its outgoing link that leads to *minlink*(f). A chain of such messages makes its way to the source of *minlink*(f), which then sends a CONNECT(*level*(f)) message over *minlink*(f). The presence of a CONNECT message in *minlink*(f) means that *rootchanged*(f) is true. Thus, the *ChangeRoot*(f) action is only needed for fragments f consisting of a single node. Two fragments can merge when they have the same *minedge* and a CONNECT message is in both its links; the result is that one of the CONNECT messages is removed. The action *AfterMerge*(p, q) removes the other CONNECT message from the new core. (A delicate point is that *ComputeMin*(f) cannot occur until the appropriate *AfterMerge*(p, q) has, in order to make sure old CONNECT messages are not hanging around.) *Absorb*(f, g) can occur if there is a CONNECT(l) message in *minlink*(g), and *minlink*(g) points to a fragment whose level is greater than l.

Define automaton *CON* (for "Connect") as follows.

The state consists of a set *fragments*. Each element f of the set is called a *fragment*, and has the following components:

- *subtree*(f), a subgraph of G;
- *core*(f), an edge of G or *nil*;
- *level*(f), a nonnegative integer; and
- *minlink*(f), a link of G or *nil*.

For each link $\langle p, q\rangle$, there are associated three variables:

- $cqueue_p(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at p to be sent;
- $cqueue_{pq}(\langle p, q\rangle)$, a FIFO queue of messages from p to q that are in the communication channel; and
- $cqueue_q(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at q to be processed.

The set of possible messages M is $\{\text{CONNECT}(l) : l \geq 0\} \cup \{\text{CHANGEROOT}\}$. The state also contains Boolean variables, *answered*(l), one for each $l \in L(G)$, and Boolean variable *awake*.

In the start state of COM, *fragments* has one element for each node in $V(G)$; for fragment f corresponding to node p, $subtree(f) = \{p\}$, $core(f) = nil$, $level(f) = 0$, and $minlink(f)$ is the minimum-weight link adjacent to p. The message queues are empty. Each *answered*(l) is false and *awake* is false.

The derived variable $cqueue(\langle p,q\rangle)$ is $cqueue_q(\langle p,q\rangle) \mid\mid cqueue_{pq}(\langle p,q\rangle) \mid\mid c$-$queue_p(\langle p,q\rangle)$. For each fragment f, we define the derived Boolean variable $rootchanged(f)$ to be true if and only if a CONNECT message is in $cqueue(\langle p,q\rangle)$, for some external link $\langle p,q\rangle$ of f. Derived variable $tominlink(p)$ is defined to be the link $\langle p,q\rangle$ such that (p,q) is on the path in $subtree(fragment(p))$ from p to $minnode(fragment(p))$.

Message m is defined to be *in* $subtree(f)$ if m is in $cqueue(\langle q,p\rangle)$ and $p \in nodes(f)$.

Input actions:

- $Start(p)$, $p \in V(G)$
  Effects:
  $awake :=$ true

Output actions:

- $InTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  $awake =$ true
  $(p,q) \in subtree(fragment(p))$ or $\langle p,q\rangle = minlink(fragment(p))$
  $answered(\langle p,q\rangle) =$ false
  Effects:
  $answered(\langle p,q\rangle) :=$ true

- $NotInTree(\langle p,q\rangle)$, $\langle p,q\rangle \in L(G)$
  Preconditions:
  $fragment(p) = fragment(q)$ and $(p,q) \notin subtree(fragment(p))$
  $answered(\langle p,q\rangle) =$ false
  Effects:
  $answered(\langle p,q\rangle) :=$ true

Internal actions:

- $ChannelSend(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  - Preconditions:
    - m at head of $cqueue_p(\langle p,q\rangle)$
  - Effects:
    - dequeue($cqueue_p(\langle p,q\rangle)$)
    - enqueue(m, $cqueue_{pq}(\langle p,q\rangle)$)

- $ChannelRecv(\langle p,q\rangle, m)$, $\langle p,q\rangle \in L(G)$, $m \in M$
  - Preconditions:
    - m at head of $cqueue_{pq}(\langle p,q\rangle)$
  - Effects:
    - dequeue($cqueue_{pq}(\langle p,q\rangle)$)
    - enqueue(m, $cqueue_q(\langle p,q\rangle)$)

- $ComputeMin(f)$, $f \in fragments$
  - Preconditions:
    - $minlink(f) = nil$
    - l is the minimum-weight external link of $subtree(f)$
    - $level(f) \leq level(fragment(target(l)))$
    - no CONNECT message is in $cqueue(k)$, for any internal link k of f
  - Effects:
    - $minlink(f) := l$
    - — let $p = root(f)$ —
    - if $p \neq minnode(f)$ then enqueue(CHANGEROOT, $cqueue_p(tominlink(p))$)
    - else enqueue(CONNECT($level(f)$), $cqueue_p(minlink(f))$)

- $ReceiveChangeRoot(\langle q,p\rangle)$, $\langle q,p\rangle \in L(G)$
  - Preconditions:
    - CHANGEROOT at head of $cqueue_p(\langle q,p\rangle)$
  - Effects:
    - dequeue($cqueue_p(\langle q,p\rangle)$)
    - — let $f = fragment(p)$ —
    - if $p \neq minnode(f)$ then enqueue(CHANGEROOT, $cqueue_p(tominlink(p))$)
    - else enqueue(CONNECT($level(f)$), $cqueue_p(minlink(f))$)

- $ChangeRoot(f)$, $f \in fragments$
  - Preconditions:
    - $awake$ = true

$rootchanged(f) = \text{false}$
$subtree(f) = \{p\}$
Effects:
enqueue(CONNECT(0), $cqueue_p(minlink(f))$)

- $Merge(f, g)$, $f, g \in fragments$
  Preconditions:
  CONNECT(l) in $cqueue(\langle p, q\rangle)$, $\langle p, q\rangle$ external link of f
  CONNECT(l) at head of $cqueue_p(\langle q, p\rangle)$, $\langle q, p\rangle$ external link of g
  Effects:
  dequeue($cqueue_p(\langle q, p\rangle)$)
  add a new element h to $fragments$
  $subtree(h) := subtree(f) \cup subtree(g) \cup minedge(f)$
  $core(h) := minedge(f)$
  $level(h) := level(f) + 1$
  $minlink(h) := nil$
  delete f and g from $fragments$

- $AfterMerge(p, q)$, $p, q \in V(G)$
  Preconditions:
  $fragment(p) = fragment(q)$
  CONNECT(l) at head of $cqueue_p(\langle q, p\rangle)$
  Effects:
  dequeue($cqueue_p(\langle q, p\rangle)$)

- $Absorb(f, g)$, $f, g \in fragments$
  Preconditions:
  — let $p = target(minlink(g))$ —
  CONNECT(l) at head of $cqueue_p(minlink(g))$
  $l < level(f)$
  $f = fragment(p)$
  Effects:
  dequeue($cqueue_p(minlink(g))$)
  $subtree(f) := subtree(f) \cup subtree(g) \cup minedge(g)$
  delete g from $fragments$

Define the following predicates on states of CON. (All free variables are universally quantified.)

- CON-A: If $awake = \text{false}$, then $cqueue(\langle q, p\rangle)$ is empty.

- CON-B: If *rootchanged*(f) = false and *minlink*(f) $\neq$ *nil*, then either *subtree*(f) = $\{p\}$, or else *minnode*(f) $\neq$ *root*(f) and there is exactly one CHANGEROOT message in *subtree*(f).

- CON-C: If a CHANGEROOT message is in *cqueue*($\langle q, p\rangle$), then *minlink*(f) $\neq$ *nil*, *rootchanged*(f) = false, p is a child of q, and *minnode*(f) $\in$ *subtree*(p), where f = *fragment*(p).

- CON-D: If a CONNECT(l) message is in *cqueue*(k), where k is an external link of f, then k = *minlink*(f), l = *level*(f), and only one CONNECT message is in *cqueue*(k).

- CON-E: If a CONNECT(l) message is in *cqueue*($\langle p, q\rangle$), where $\langle p, q\rangle$ is an internal link of f, then (p, q) = *core*(f), $l <$ *level*(f), and only one CONNECT message is in *cqueue*($\langle p, q\rangle$).

- CON-F: If *minlink*(f) $\neq$ *nil*, then no CONNECT message is in *cqueue*(k), for any internal link k of f.

Let P_{CON} be the conjunction of CON-A through CON-F.

In order to show that *CON* simulates *COM*, we define an abstraction mapping $\mathcal{M}_6 = (\mathcal{S}_6, \mathcal{A}_6)$ from *CON* to *COM*.

Define the function $\mathcal{S}_6$ from *states*(*CON*) to *states*(*COM*) by simply ignoring the message queues, and mapping the derived variables *rootchanged*(f) in the CON state to the (non-derived) variables *rootchanged*(f) in the COM state.

Define the function $\mathcal{A}_6$ as follows. Let s be a state of *CON* and π an action of *CON* enabled in s. If the minimum-weight external link of f is adjacent to *core*(f), then *ComputeMin*(f) causes *ComputeMin*(f), immediately followed by *ChangeRoot*(f), to be simulated in *COM*. Otherwise, *ChangeRoot*(f) is simulated when the source of *minlink*(f) receives a CHANGEROOT message.

- If π = *ChannelSend*($\langle p, q\rangle, m$), *ChannelRecv*($\langle p, q\rangle, m$), or *AfterMerge*(p, q), then $\mathcal{A}_6(s, \pi)$ is empty.

- If π = *ComputeMin*(f) and *mw-root*(f) = *mw-minnode*(f) in s, then $\mathcal{A}_6(s, \pi)$ = *ComputeMin*(f) t *ChangeRoot*(f), where t is identical to $\mathcal{S}_6(s)$ except that *minlink*(f) equals the minimum-weight external link of f in t.

- If $\pi = ComputeMin(f)$ and $mw\text{-}root(f) \neq mw\text{-}minnode(f)$ in s, then $\mathcal{A}_6(s,\pi) = ComputeMin(f)$.
- If $\pi = ReceiveChangeRoot(\langle q,p\rangle)$ and $p = minnode(fragment(p))$ in s, then $\mathcal{A}_6(s,\pi) = ChangeRoot(fragment(p))$.
- If $\pi = ReceiveChangeRoot(\langle q,p\rangle)$ and $p \neq minnode(fragment(p))$ in s, then $\mathcal{A}_6(s,\pi)$ is empty.
- For all other values of π, $\mathcal{A}_6(s,\pi) = \pi$.

Recall that $P'_{COM} = (P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$. If $P'_{COM}(\mathcal{S}_6(s))$ is true, then the COM predicates are true in $\mathcal{S}_6(s)$, and the HI predicates are true in $\mathcal{S}_1(\mathcal{S}_6(s))$.

Lemma 23: *CON simulates COM via $\mathcal{M}_6$, P_{CON}, and P'_{COM}.*

Proof: By inspection, the types of CON, COM, $\mathcal{M}_6$, and P_{CON} are correct. By Corollary 14, P'_{COM} is a predicate true in every reachable state of COM.

(1) Let s be in $start(CON)$. Obviously P_{CON} is true in s and $\mathcal{S}_6(s)$ is in $start(COM)$.

(2) Obviously, $\mathcal{A}_6(s,\pi)|ext(COM) = \pi|ext(CON)$.

(3) Let (s',π,s) be a step of CON such that P'_{COM} is true of $\mathcal{S}_6(s')$ and P_{CON} is true of s'. Below we show (3a) only for those predicates that are not obviously true in s.

i) π is Start(p), InTree(l) or NotInTree(l). $\mathcal{A}_6(s',\pi) = \pi$. Obviously, $\mathcal{S}_6(s')\pi\mathcal{S}_6(s)$ is an execution fragment of COM, and P_{CON} is true in s.

ii) π is ChannelSend($\langle$q,p$\rangle$,m) or ChannelRecv($\langle$q,p$\rangle$,m). $\mathcal{A}_6(s',\pi)$ is empty. Obviously, $\mathcal{S}_6(s') = \mathcal{S}_6(s)$, and P_{CON} is true in s.

iii) π is ComputeMin(f).

Case 1: $mw\text{-}root(f) \neq mw\text{-}minnode(f)$ in s'.

(3b) $\mathcal{A}_6(s',\pi) = \pi$. Obviously $\mathcal{S}_6(s')\pi\mathcal{S}_6(s)$ is an execution fragment of COM.

(3a) *Claims about s':*

1. $minlink(f) = nil$, by precondition.

2. l is the minimum-weight external link of f, by precondition.
3. $level(f) \leq level(fragment(target(l)))$, by precondition.
4. No CONNECT message is in $cqueue(k)$, for any internal link k of f, by precondition.
5. $p = mw\text{-}root(f)$, by assumption.
6. $p \neq mw\text{-}minnode(f)$, by assumption.
7. $awake =$ true, by Claim 1 and COM-C.
8. No CHANGEROOT mesage is in $subtree(f)$, by Claim 1 and CON-C.
9. $mw\text{-}minnode(f) \in subtree(p)$, by Claim 5.
10. $rootchanged(f) =$ false, by Claim 1 and COM-B.

Claims about s:

11. $minlink(f) = l$, the minimum-weight external link of f, by Claim 2 and code.
12. $level(f) \leq level(fragment(target(l)))$, by Claim 3.
13. $p = root(f)$, by Claims 5 and 11.
14. $p \neq minnode(f)$, by Claims 6 and 11.
15. $awake =$ true, by Claim 7.
16. Exactly one CHANGEROOT message is in $subtree(f)$, by Claim 8 and code.
17. $minnode(f) \in subtree(p)$, by Claims 9 and 11.
18. $rootchanged(f) =$ false, by Claim 10.
19. No CONNECT message is in $cqueue(k)$, for any internal link k of f, by Claim 4.

CON-A is true by Claim 15. CON-B is true by Claims 13, 14, and 16. CON-C is true by definition of *tominlink*, Claims 17, 18 and 11. CON-D and CON-E are true since no relevant changes are made. CON-F is true by Claim 19.

Case 2: $mw\text{-}root(f) = mw\text{-}minnode(f)$ in s'.

(3b) $\mathcal{A}_6(s', \pi) = \pi\ t\ ChangeRoot(f)$, where t is identical to $\mathcal{S}_6(s')$ except that $minlink(f)$ equals the minimum-weight external link of f in t.

Claims about s':

1. $minlink(f) = nil$, by precondition.
2. l is the minimum-weight external link of f, by precondition.
3. $level(f) \leq level(fragment(target(l)))$, by precondition.
4. $awake =$ true, by Claim 1 and COM-C.
5. $rootchanged(f) =$ false, by Claim 1 and COM-B.

Claims about t:

6. *minlink*(f) is the minimum-weight external link of f, by definition of t.
7. *awake* = true, by Claim 4.
8. *rootchanged*(f) = false, by Claim 5.

Claims about s:

9. *minlink*(f) is the minimum-weight external link of f, by code.
10. A CONNECT message is in *cqueue*(*minlink*(f)), by code.
11. *rootchanged*(f) = true, by Claims 9 and 10.

By Claims 1, 2 and 3, π is enabled in $\mathcal{S}_6(s')$. By Claim 6 (and definition of t), the effects of π are mirrored in t. By Claims 6, 7, and 8, *ChangeRoot*(f) is enabled in t. By Claim 11 (and definition of t), the effects of *ChangeRoot*(f) are mirrored in $\mathcal{S}_6(s)$. Therefore, $\mathcal{S}_6(s')\pi\ t\ ChangeRoot(f)\mathcal{S}_6(s)$ is an execution fragment of *COM*.

(3a) *More claims about s':*

12. No CHANGEROOT message is in *subtree*(f), by Claim 1 and CON-C.
13. No CONNECT message is in any *cqueue*(k), where k is an external link of f, by Claim 1 and CON-D.
14. No CONNECT message is in any *cqueue*(k), where k is an internal link of f, by precondition.

More claims about s:

15. *awake* = true, by Claim 4.
16. No CHANGEROOT message is in *subtree*(f), by Claim 12.

CON-A is true by Claim 15. CON-B is true by Claim 11. CON-C is true by Claim 16. CON-D is true by Claims 9, 10, and 13 and code. CON-E is true because no relevant changes are made. CON-F is true by Claim 14.

iv) π is ReceiveChangeRoot($\langle$q,p$\rangle$). Let f = *fragment*(p).

Case 1: $p \neq$ *minnode*(f) in s'.

(3c) $\mathcal{A}_6(s',\pi)$ is empty. Below we show that *rootchanged*(f) is the same in s' and s, which implies that $\mathcal{S}_6(s) = \mathcal{S}_6(s')$.

Claims about s':

1. A CHANGEROOT message is in *cqueue*($\langle q,p\rangle$), by precondition.

2. $(p,q) \in subtree(f)$, by Claim 1 and CON-C.
3. $rootchanged(f) =$ false, by Claims 1 and 2 and CON-A.

Claims about s:

4. $rootchanged(f) =$ false, by Claim 1 and code.

Claims 2 and 4 give the result.

(3a) Let $\langle p,r \rangle = tominlink(p)$.

More claims about s′:

5. $awake =$ true, by Claim 1 and CON-A.
6. $minlink(f) \neq nil$, by Claims 1 and 2 and CON-C.
7. $minnode(f) \in subtree(p)$, by Claims 1 and 2 and CON-C.
8. There is exactly one CHANGEROOT message in $subtree(f)$, by Claims 2, 3 and 6 and CON-B.
9. r is a child of p and $minnode(f) \in subtree(r)$, by definition of $tominlink(p)$.

More claims about s:

10. $awake =$ true, by Claim 5.
11. There is exactly one CHANGEROOT message in $subtree(f)$, by Claim 8 and code.
12. r is a child of p, by Claim 9.
13. $minlink(f) \neq nil$, by Claim 6.
14. $(p,r) \neq core(f)$, by Claim 9.
15. $minnode(f) \in subtree(r)$, by Claims 7 and 9.

CON-A is true by Claim 10. CON-B is true by Claim 11 and assumption for Case 1. CON-C is true by Claims 4, 12, 13, 14 and 15. CON-D, CON-E and CON-F are true because no relevant changes are made.

Case 2: $p = minnode(f)$ in s'.

(3b) $\mathcal{A}_6(s',\pi) = ChangeRoot(f)$.

Claims about s′ :

1. A CHANGEROOT message is in $cqueue(\langle q,p \rangle)$, by precondition.

2. $p = minnode(f)$, by assumption.
3. $awake$ = true, by Claim 1 and CON-A.
4. $minlink(f) \neq nil$, by Claim 1 and CON-C.
5. $rootchanged(f)$ = false, by Claim 1 and CON-C.
6. $minlink(f)$ is an external link of f, by Claim 4 and COM-A.

By Claims 3, 4 and 5, $ChangeRoot(f)$ is enabled in $\mathcal{S}_6(s')$.

Claims about s:

7. A CONNECT message is in $cqueue(minlink(f))$, by code.
8. $minlink(f)$ is an external link of f, by Claim 6.
9. $rootchanged(f)$ = true, by Claims 7 and 8.

By Claim 9, the effects of $ChangeRoot(f)$ are mirrored in $\mathcal{S}_6(s)$.

So $\mathcal{S}_6(s')$ $ChangeRoot(f)$ $\mathcal{S}_6(s)$ is an execution fragment of COM.

(3a) *More claims about s':*

10. p is a child of q, by Claim 1 and CON-C.
11. Exactly one CHANGEROOT message is in $subtree(f)$, by Claims 5, 4, 10 and CON-B.
12. No CONNECT message is in any $cqueue(k)$, where k is an external link of f, by Claim 5.
13. No CONNECT message is in any $cqueue(k)$, where k is an internal link of f, by Claim 4 and CON-F.

More claims about s:

14. $awake$ = true, by Claim 3.
15. No CHANGEROOT message is in $subtree(f)$, by Claims 1, 10 and 11 and code.
16. No CONNECT message is in any $cqueue(k)$, where k is an internal link of f, by Claim 13.

CON-A is true by Claim 14. CON-B is true by Claim 9. CON-C is true by Claim 15. CON-D is true by Claims 7, 8, 12 and code. CON-E is true because no relevant changes are made. CON-F is true by Claim 16.

v) π is ChangeRoot(f).

(3b) $\mathcal{A}_6(s', \pi) = \pi$.

Claims about s':

1. *awake* = true, by precondition.
2. *rootchanged*(f) = false, by precondition.
3. *subtree*(f) = $\{p\}$, by precondition.
4. *minlink*(f) $\neq$ *nil*, by Claim 3 and COM-E.
5. *minlink*(f) is an external link of f, by Claim 4 and COM-A.

Claims 1, 2 and 4 imply that π is enabled in $\mathcal{S}_6(s')$.

Claims about s:

6. *minlink*(f) is an external link of f, by Claim 5.
7. A CONNECT message is in *cqueue*(*minlink*(f)), by code.
8. *rootchanged*(f) = true, by Claims 6 and 7.

Claim 8 implies that the effects of π are mirrored in $\mathcal{S}_6(s)$.

So $\mathcal{S}_6(s')\pi\mathcal{S}_6(s)$ is an execution fragment of *COM*.

(3a) *More claims about s':*

9. No CHANGEROOT message is in *cqueue*($\langle q, p \rangle$), for any q, by Claim 3 and CON-C.
10. No CONNECT message is in any *cqueue*(k), where k is an external link of f, by Claim 2.
11. No CONNECT message is in any *cqueue*(k), where k is an internal link of f, by Claim 3.

More claims about s:

12. *awake* = true, by Claim 1 and code.
13. No CHANGEROOT message is in *cqueue*($\langle q, p \rangle$), for any q, by Claim 9.
14. No CONNECT message is in any *cqueue*(n), where n is an internal link of f, by Claim 11.

CON-A is true by Claim 12. CON-B is true by Claim 8. CON-C is true by Claim 13. CON-D is true by Claims 6, 7 and 10 and code. CON-E is true because no relevant changes are made. CON-F is true, by Claims 6 and 14.

vi) π is Merge(f,g).

(3b) $\mathcal{A}_6(s', \pi) = \pi$.

Claims about s':

1. A CONNECT(l) message is in $cqueue(\langle p, q\rangle)$, by precondition.
2. $\langle p, q\rangle$ is an external link of f, by precondition.
3. A CONNECT(l) message is in $cqueue(\langle q, p\rangle)$, by precondition.
4. $\langle q, p\rangle$ is an external link of g, by precondition.
5. $f \neq g$, by Claims 2 and 4.
6. $rootchanged(f)$ = true, by Claims 1 and 2.
7. $rootchanged(g)$ = true, by Claims 3 and 4.
8. $\langle p, q\rangle = minlink(f)$, by Claims 1 and 2 and CON-D.
9. $\langle q, p\rangle = minlink(g)$, by Claims 3 and 4 and CON-D.
10. $minedge(f) = minedge(g)$, by Claims 8 and 9.
11. If $k \neq minlink(f)$ is an external link of f, then no CONNECT message is in $cqueue(k)$, by CON-D.
12. If $k \neq minlink(g)$ is an external link of g, then no CONNECT message is in $cqueue(k)$, by CON-D.

By Claims 5, 6, 7 and 10, π is enabled in $\mathcal{S}_6(s')$. By Claims 11 and 12 and definition of h, $rootchanged(h)$ = false in s, so the effects of π are mirrored in $\mathcal{S}_6(s)$. Thus, $\mathcal{S}_6(s')\pi\mathcal{S}_6(s)$ is an execution fragment of COM.

(3a) *More claims about s':*

13. $awake$ = true, by Claim 1 and COM-A.
14. No CHANGEROOT message is in $subtree(f)$, by Claim 6 and CON-C.
15. No CHANGEROOT message is in $subtree(g)$, by Claim 7 and CON-C.
16. No CONNECT message is in $cqueue(k)$, for any internal link k of f, by Claim 8 and CON-F.
17. No CONNECT message is in $cqueue(k)$, for any internal link k of g, by Claim 9 and CON-F.
18. Exactly one CONNECT message is in $cqueue(\langle p, q\rangle)$, by Claims 1 and 2 and CON-D.
19. Exactly one CONNECT message is in $cqueue(\langle q, p\rangle)$, by Claims 3 and 4 and CON-D.
20. $l = level(f)$, by Claims 1 and 2 and CON-D.

Claims about s:

21. $awake$ = true, by Claim 13 and code.
22. $minlink(h) = nil$, by code.
23. No CHANGEROOT message is in $subtree(h)$, by Claims 14 and 15 and code.
24. No CONNECT message is in $cqueue(k)$, for any external link k of h, by Claims 11 and 12 and code.
25. Exactly one CONNECT message is in $cqueue(\langle p, q\rangle)$ and $(p, q) = core(h)$, by Claim 18 and code.
26. $l < level(h)$, by Claim 20 and code.
27. No CONNECT message is in $cqueue(\langle q, p\rangle)$, by Claim 19 and code.
28. No CONNECT message is in any non-core internal link of h, by Claims 16 and 17 and code.

CON-A is true by Claim 21. CON-B is true by Claim 22. CON-C is true by Claim 23. CON-D is true by Claim 24. CON-E is true by Claims 25, 26, 27 and 28. CON-F is true by Claim 22.

vii) π is AfterMerge(p,q). $\mathcal{A}_6(s', \pi)$ is empty. Obviously, $\mathcal{S}_6(s) = \mathcal{S}_6(s')$, and P_{CON} is true in s.

viii) π is Absorb(f,g).

(3b) $\mathcal{A}_6(s', \pi) = \pi$.

Claims about s':

1. $\langle q, p\rangle = minlink(g)$, by assumption.
2. A CONNECT(l) message is in $cqueue(minlink(g))$, by precondition.
3. $l < level(f)$, by precondition.
4. $f = fragment(p)$, by precondition.
5. $minlink(g)$ is an external link of g, by Claim 1 and COM-A.
6. $rootchanged(g)$ = true, by Claims 2 and 5.
7. $l = level(g)$, by Claim 2 and CON-D.
8. $level(g) < level(f)$, by Claims 7 and 3.
9. If a CONNECT message is in $cqueue(\langle p, q\rangle)$, then $\langle p, q\rangle = minlink(f)$, by Claims 4 and 5 and CON-D.
10. If a CONNECT message is in $cqueue(\langle p, q\rangle)$, then $level(f) \leq level(g)$, by Claim 9 and COM-A.
11. No CONNECT message is in $cqueue(\langle p, q\rangle)$, by Claims 8 and 10.
12. No CONNECT message is in $cqueue(k)$, for any external link $k \neq minlink(g)$ of g, by CON-D.

By Claims 6, 8, 4 and 1, π is enabled in $\mathcal{S}_6(s')$. By Claims 11 and 12, *rootchanged*(f) remains unchanged, and the effects of π are mirrored in $\mathcal{S}_6(s)$. Thus, $\mathcal{S}_6(s')\pi\mathcal{S}_6(s)$ is an execution fragment of *COM*.

(3a) *More claims about s':*

13. *awake* = true, by Claim 2 and CON-A.
14. $l \geq 0$, by COM-F.
15. $level(f) > 0$, by Claims 7, 8 and 14.
16. $|nodes(f)| > 1$, by Claim 15 and COM-F.
17. No CHANGEROOT message is in *subtree*(g), by Claim 6 and CON-C.
18. No CONNECT message is in *cqueue*(k), where k is an internal link of g, by Claim 1 and CON-F.

Claim about s:

19. *awake* = true, by Claim 12 and code.

CON-A is true by Claim 19. CON-B is true since by Claims 16 and 17 no relevant changes are made. CON-C is true since by Claim 11, 12 and 17 no relevant changes are made. CON-D is true since by Claim 12 no relevant changes are made. CON-E is true since by Claims 11 and 18 no relevant changes are made. CON-F is true by Claim 18 and code. □

Let $P'_{CON} = (P'_{COM} \circ \mathcal{S}_6) \wedge P_{CON}$.

Corollary 24: *P'_{CON} is true in every reachable state of CON.*

Proof: By Lemmas 1 and 23. □

4.2.7 GHS Simultaneously Simulates TAR, DC, NOT and CON

This automaton is a fully distributed version of the original algorithm of [GHS]. (We have made some slight changes, which are discussed below.) The functions of *TAR*, *DC*, *NOT* and *CON* are united into one. All variables that are derived in one of these automata are also derived (in the same way) in *GHS*. In addition, there are the following derived variables. The variable *dcstatus*(p) of *DC* is refined by the variable *nstatus*(p), and has values sleeping, find, and found; initially, it is sleeping. The *awake* variable is now derived, and is true if and only if at least one node is not sleeping. The fragments are also derived, as follows. A subgraph of G is defined to have node set $V(G)$ and edge set equal to all edges of G, at least one of whose links is classified as branch and has no CONNECT message in it. A fragment is associated with each connected component of this graph. Also, *testset*(f) is defined to be all nodes p such that either *testlink*(p) $\neq$ *nil*, or a FIND message is headed toward p (or will be soon).

The bulk of the arguing done at this stage is showing that the derived variables (*subtree, level, core, minlink, testset, rootchanged*) have the proper values in the state mappings. In addition, a substantial argument is needed to show that the implementation of *level* and *core* by local variables interacts correctly with the test-accept-reject protocol. (See in particular the definition of the *TAR* action mapping for *ReceiveTest*, and the case for *ReceiveTest* in Lemma 25.) It would be ideal to do this argument in *NOT*, where the rest of the argument that *core* and *level* are implemented correctly is done, but reorganizing the lattice to allow this consolidation caused graver violations of modularity.

The messages sent in this automaton are all those sent in *TAR*, *DC*, *NOT* and *CON*, except that NOTIFY messages are replaced by INITIATE messages, which have a parameter that is either find or found, and FIND messages are replaced by INITIATE messages with the parameter equal to find.

Some minor changes were made to the algorithm as presented in [GHS]. First, our version initializes all variables to convenient values. (This change makes it easier to state the predicates.) Second, provision is made for the output actions *InTree*(l) and *NotInTree*(l). Third, when node p receives an INITIATE message, variables *inbranch*(p), *bestlink*(p) and *bestwt*(p) are only changed if the parameter of the INITIATE message is find. This change does not affect the performance or correctness of the algorithm. The values of these variables will not be relevant until p subsequently receives an INITIATE-find message, yet the receipt of this message will cause these variables to be reset. The advantage of the change is that it greatly simplifies the state mapping from *GHS* to *DC*.

Our version of the algorithm is slightly more general than that in [GHS]. There, each node p has a single queue for incoming messages, whereas in our description, p has a separate queue of incoming messages for each of its neighbors. A node p in our algorithm could happen to process messages in the order, taken over all the neighbors, in which they arrive (modulo the requeueing), which would be consistent with the original algorithm. But p could also handle the messages in some other order (although, of course, still in order for each individual link). Thus, the set of executions of our version is a proper superset of the set of executions of the original.

A small optimization to the original algorithm was also found. (It does not affect the worst-case performance.) When a CONNECT message is received by p under circumstances that cause fragment g to be absorbed into fragment f, an INITIATE message with parameter find is only sent if $testlink(p) \neq nil$ in our version, instead of whenever $nstatus(p) =$ find as in the original. As a result of this change, if $nstatus(p) =$ find and $testlink(p) = nil$, p need not wait for the entire (former) fragment g to find its new minimum-weight external link before p can report to its parent, since this link can only have a larger weight than the minimum-weight external link of p already found.

The automaton *GHS* is the result of composing an automaton $Node(p)$, for all $p \in V(G)$, and $Link(l)$, for all $l \in L(G)$, and then hiding actions appropriately to fit the $MST(G)$ problem specification.

First we describe the automaton $Node(p)$, for $p \in V(G)$. The state has the following components:

- $nstatus(p)$, either sleeping, find, or found;
- $nfrag(p)$, an edge of G or nil;
- $nlevel(p)$, a nonnegative integer;
- $bestlink(p)$, a link of G or nil;
- $bestwt(p)$, a weight or ∞;
- $testlink(p)$, a link of G or nil;
- $inbranch(p)$, a link of G or nil; and
- $findcount(p)$, a nonnegative integer.

For each link $\langle p, q\rangle \in L_p(G)$, there are the following variables:

- $lstatus(\langle p, q\rangle)$, either unknown, branch or rejected;
- $queue_p(\langle p, q\rangle)$, a FIFO queue of messages from p to q waiting at p to be sent;

- $queue_p(\langle q, p\rangle)$, a FIFO queue of messages from q to p waiting at p to be processed; and
- $answered(\langle p, q\rangle)$, a Boolean.

The set of possible messages M is $\{\text{CONNECT}(l) : l \geq 0\} \cup \{\text{INITIATE}(l, c, st) : l \geq 0, c \in E(G), st \text{ is find or found}\} \cup \{\text{TEST}(l, c) : l \geq 0, c \in E(G)\} \cup \{\text{REPORT}(w) : w \text{ is a weight or } \infty\} \cup \{\text{ACCEPT}, \text{REJECT}, \text{CHANGEROOT}\}$.

In the start state of $Node(p)$, $nstatus(p)$ = sleeping, $nfrag(p) = nil$, $nlevel(p) = 0$, $bestlink(p)$ is arbitrary, $bestwt(p)$ is arbitrary, $testlink(p) = nil$, $inbranch(p)$ is arbitrary, $findcount(p) = 0$, $lstatus(l)$ = unknown for all $l \in L_p(G)$, $answered(l)$ = false for all $l \in L_p(G)$, and both queues are empty.

Now we describe the actions of $Node(p)$.

Input actions:

- $Start(p)$
  Effects:
  if $nstatus(p)$ = sleeping then execute procedure $WakeUp(p)$
- $ChannelRecv(l)$, $l \in L_p(G)$, $m \in M$
  Effects:
  enqueue(m, $queue_p(l)$)

Output actions:

- $InTree(l)$, $l \in L_p(G)$
  Preconditions:
  $answered(l)$ = false
  $lstatus(l)$ = branch
  Effects:
  $answered(l)$:= true
- $NotInTree(l)$, $l \in L_p(G)$
  Preconditions:
  $answered(l)$ = false
  $lstatus(l)$ = rejected
  Effects:
  $answered(l)$:= true

- $ChannelSend(l, m)$, $l \in L_p(G)$, $m \in M$
  Preconditions:
  m at head of $queue_p(l)$
  Effects:
  dequeue($queue_p(l)$)

Internal actions:

- $ReceiveConnect(\langle q, p\rangle, l)$, $\langle p, q\rangle \in L_p(G)$
  Preconditions:
  CONNECT(l) at head of $queue_p(\langle q, p\rangle)$
  Effects:
  dequeue($queue_p(\langle q, p\rangle)$)
  if $nstatus(p)$ = sleeping then execute procedure $WakeUp(p)$
  if $l < nlevel(p)$ then [
  $lstatus(\langle p, q\rangle)$:= branch
  if $testlink(p) \neq nil$, then [
  enqueue(INITIATE($nlevel(p)$, $nfrag(p)$,find), $queue_p(\langle p, q\rangle)$)
  $findcount(p) := findcount(p) + 1$]
  else enqueue(INITIATE($nlevel(p)$, $nfrag(p)$,found), $queue_p(\langle p, q\rangle)$)]
  else
  if $lstatus(\langle p, q\rangle)$ = unknown then enqueue(CONNECT(l), $queue_p(\langle q, p\rangle)$)
  else enqueue(INITIATE($nlevel(p) + 1, \langle p, q\rangle$, find), $queue_p(\langle p, q\rangle)$)

- $ReceiveInitiate(\langle q, p\rangle, l, c, st)$, $\langle p, q\rangle \in L_p(G)$
  Preconditions:
  INITIATE(l, c, st) at head of $queue_p(\langle q, p\rangle)$
  Effects:
  dequeue($queue_p(\langle q, p\rangle)$)
  $nlevel(p) := l$
  $nfrag(p) := c$
  $nstatus(p) := st$
  — let $S = \{\langle p, r\rangle : lstatus(\langle p, r\rangle) = \text{branch}, r \neq q\}$ —
  enqueue(INITIATE(l, c, st), $queue_p(k)$) for all $k \in S$
  if st = find then [
  $inbranch(p) := \langle p, q\rangle$
  $bestlink(p) := nil$
  $bestwt(p) := \infty$
  execute procedure $Test(p)$

$findcount(p) := |S|$]

- $ReceiveTest(\langle q,p\rangle, l, c)$, $\langle p,q\rangle \in L_p(G)$
  Preconditions:
  TEST(l,c) at head of $queue_p(\langle q,p\rangle)$
  Effects:
  dequeue($queue_p(\langle q,p\rangle)$)
  if $nstatus(p)$ = sleeping then execute procedure $WakeUp(p)$
  if $l > nlevel(p)$ then enqueue(TEST(l,c), $queue_p(\langle q,p\rangle)$)
  else
  if $c \neq nfrag(p)$ then enqueue(ACCEPT, $queue_p(\langle p,q\rangle)$)
  else [
  if $lstatus(\langle p,q\rangle)$ = unknown then $lstatus(\langle p,q\rangle)$:= rejected
  if $testlink(p) \neq \langle p,q\rangle$ then enqueue(REJECT, $queue_p(\langle p,q\rangle)$)
  else execute procedure $Test(p)$]

- $ReceiveAccept(\langle q,p\rangle)$, $\langle p,q\rangle \in L_p(G)$
  Preconditions:
  ACCEPT at head of $queue_p(\langle q,p\rangle)$
  Effects:
  dequeue($queue_p(\langle q,p\rangle)$)
  $testlink(p) := nil$
  if $wt(p,q) < bestwt(p)$ then [
  $bestlink(p) := \langle p,q\rangle$
  $bestwt(p) := wt(p,q)$]
  execute procedure $Report(p)$

- $ReceiveReject(\langle q,p\rangle)$, $\langle p,q\rangle \in L_p(G)$
  Preconditions:
  REJECT at head of $queue_p(\langle q,p\rangle)$
  Effects:
  dequeue($queue_p(\langle q,p\rangle)$)
  if $lstatus(\langle p,q\rangle)$ = unknown then $lstatus(\langle p,q\rangle)$:= rejected
  execute procedure $Test(p)$

- $ReceiveReport(\langle q,p\rangle, w)$, $\langle p,q\rangle \in L_p(G)$
  Preconditions:
  REPORT(w) at head of $queue_p(\langle q,p\rangle)$
  Effects:
  dequeue($queue_p(\langle q,p\rangle)$)

if $\langle p,q\rangle \neq inbranch(p)$ then [
  $findcount(p) := findcount(p) - 1$
  if $w < bestwt(p)$ then [
    $bestwt(p) := w$
    $bestlink(p) := \langle p,q\rangle$]
  execute procedure $Report(p)$]
else
  if $nstatus(p) =$ find then $enqueue(\text{REPORT}(w), queue_p(\langle q,p\rangle))$
  else if $w > bestwt(p)$ then execute procedure $ChangeRoot(p)$

- $ReceiveChangeRoot(\langle q,p\rangle)$, $\langle p,q\rangle \in L_p(G)$
  Preconditions:
    CHANGEROOT at head of $queue_p(\langle q,p\rangle)$
  Effects:
    dequeue($queue_p(\langle q,p\rangle)$)
    execute procedure $ChangeRoot(p)$

Procedures

- $WakeUp(p)$
  — let $\langle p,q\rangle$ be the minimum-weight link of p —
  $lstatus(\langle p,q\rangle) :=$ branch
  $nstatus(p) :=$ found
  enqueue(CONNECT(0), $queue_p(\langle p,q\rangle)$)

- $Test(p)$
  if l, the minimum-weight link of p with $lstatus(l) =$ unknown, exists then [
    $testlink(p) := l$
    enqueue(TEST($nlevel(p), nfrag(p)$), $queue_p(l)$)]
  else [
    $testlink(p) := nil$
    execute procedure $Report(p)$]

- $Report(p)$
  if $findcount(p) = 0$ and $testlink(p) = nil$ then [
    $nstatus(p) :=$ found
    enqueue(REPORT($bestwt(p)$), $queue_p(inbranch(p))$)]

- $ChangeRoot(p)$
  if $lstatus(bestlink(p)) =$ branch then

enqueue(CHANGEROOT, $queue_p(bestlink(p))$)
else [
enqueue(CONNECT($nlevel(p)$), $queue_p(bestlink(p))$)
$lstatus(bestlink(p))$:= branch]

Now we describe the automaton $Link(\langle p, q\rangle)$, for each $\langle p, q\rangle \in L(G)$.

The state consists of the single variable $queue_{pq}(\langle p, q\rangle)$, a FIFO queue of messages. The set of messages, M, is the same as for $Node(p)$. The queue is empty in the start state.

Input Actions:

- $ChannelSend(\langle p, q\rangle, m)$, $m \in M$
  Effects:
  enqueue(m, $queue_{pq}(\langle p, q\rangle)$)

Output Actions:

- $ChannelRecv(\langle p, q\rangle, m)$, $m \in M$
  Preconditions:
  m at head of $queue_{pq}(\langle p, q\rangle)$
  Effects:
  dequeue($queue_{pq}(\langle p, q\rangle)$)

Now we can define the automaton that models the entire network. Define the automaton GHS to be the result of composing the automata $Node(p)$, for all $p \in V(G)$, and $Link(l)$, for all $l \in L(G)$, and then hiding all actions except for $Start(p)$, $p \in V(G)$, $InTree(l)$ and $NotInTree(l)$, $l \in L(G)$.

Given a FIFO queue q and a set M, define $q|M$ to be the FIFO queue obtained from q by deleting all elements of q that are not in M.

Derived Variables:

- $queue(\langle p, q\rangle)$ is $queue_p(\langle p, q\rangle) \mid\mid queue_{pq}(\langle p, q\rangle) \mid\mid queue_q(\langle p, q\rangle)$.
- $tarqueue_p(\langle p, q\rangle)$ is $queue_p(\langle p, q\rangle)|M_{TAR}$, where M_{TAR} is the set of all possible messages in TAR; similarly for $tarqueue_{pq}(\langle p, q\rangle)$ and $tarqueue_q(\langle p, q\rangle)$.

Similar definitions are made for the *dcqueue*'s, *nqueue*'s, and *cqueue*'s, except that for the *dcqueue*'s, each INITIATE(l, c,find) message is replaced with a FIND message, and for the *nqueue*'s, each INITIATE($l, c, *$) message is replaced with a NOTIFY(l, c) message.

- *awake* is false if and only if $nstatus(p)$ = sleeping for all $p \in V(G)$.
- For all $p \in V(G)$, $dcstatus(p)$ = unfind if $nstatus(p)$ = sleeping or found, and $dcstatus(p)$ = find if $nstatus(p)$ = find.
- MSF is the subgraph of G whose nodes are $V(G)$, and whose edges are all edges (p, q) of G such that either (1) $lstatus(\langle p, q\rangle)$ = branch and no CONNECT message is in $queue(\langle p, q\rangle)$, or (2) $lstatus(\langle q, p\rangle)$ = branch and no CONNECT message is in $queue(\langle p, q\rangle)$.
- *fragments* is a set of elements, called *fragments*, one for each connected component of MSF.

Each fragment f has the following components:

- $subtree(f)$, the corresponding connected component of MSF;
- $level(f)$, defined as in NOT;
- $core(f)$, defined as in NOT;
- $testset(f)$, the set of all $p \in nodes(f)$ such that one of the following is true: (1) a FIND message is headed toward p, (2) $testlink(p) \neq nil$, or (3) a CONNECT message is in $queue(\langle q, r\rangle)$, where $(q, r) = core(f)$ and $p \in subtree(q)$;
- $minlink(f)$, defined as in DC;
- $rootchanged(f)$, defined as in CON; and
- $accmin(f)$, defined as in TAR and DC.

Define the following predicates on $states(GHS)$. (All free variables are universally quantified.)

- GHS-A: If $nstatus(p)$ = sleeping, then
  (a) there is a fragment f such that $subtree(f) = \{p\}$,
  (b) $queue(\langle p, q\rangle)$ is empty for all q, and
  (c) $lstatus(\langle p, q\rangle)$ = unknown for all q.

- GHS-B: If CONNECT(l) is in $queue(\langle q,p\rangle)$, $lstatus(\langle p,q\rangle) \neq$ unknown, and no CONNECT is in $queue(\langle p,q\rangle)$, then

  (a) the state of $queue(\langle q,p\rangle)$ is CONNECT(l) followed by INITIATE($l+1,(p,q)$, find);

  (b) $queue(\langle p,q\rangle)$ is empty;

  (c) $nstatus(q) \neq$ find; and

  (d) $nlevel(p) = nlevel(q) = l$.

- GHS-C: If a CONNECT message is in $queue(l)$, then no FIND message precedes the CONNECT in $queue(l)$, and no TEST or REJECT message is in $queue(l)$.

- GHS-D: If INITIATE(l,c,find) is in $subtree(f)$, then $l = level(f)$.

- GHS-E: If INITIATE(l,c,st) is in $queue(\langle p,q\rangle)$ and $(p,q) = core(fragment(p))$, then $st =$ find.

- GHS-F: If TEST(l,c) is in $queue(\langle q,p\rangle)$, then $nlevel(q) \geq l$.

- GHS-G: If ACCEPT is in $queue(\langle q,p\rangle)$, then $nlevel(p) \leq nlevel(q)$.

- GHS-H: If $testlink(p) \neq nil$, then $nstatus(p) =$ find.

- GHS-I: If p is up-to-date, then $nlevel(p) = level(fragment(p))$.

- GHS-J: If p is up-to-date, $p \notin testset(fragment(p))$, and $\langle p,q\rangle$ is the minimum-weight external link of p, then $nlevel(p) \leq nlevel(q)$.

- GHS-K: If $subtree(f) = \{p\}$ and $nstatus(p) \neq$ sleeping, then $rootchanged(f) =$ true.

Let P_{GHS} be the conjunction of GHS-A through GHS-K.

We now define $\mathcal{M}_x = (\mathcal{S}_x, \mathcal{A}_x)$, an abstraction mapping from GHS to x, for $x = TAR$, DC, NOT and CON. $\mathcal{S}_x$ should be obvious for all x, given the above derived functions. We now define $\mathcal{A}_x(s,\pi)$ for all x, states s of GHS, and actions π of GHS enabled in s.

- $\pi = InTree(l)$ or $NotInTree(l)$. $\mathcal{A}_x(s,\pi) = \pi$ for all x.

- $\pi = Start(p)$. Let $f = fragment(p)$.

Case 1: $nstatus(p) =$ sleeping in s. For all x, $\mathcal{A}_x(s,\pi) = Start(p)\ t_x\ ChangeRoot(f)$, where t_x is the same as $\mathcal{S}_x(s)$ except that $awake =$ true in t_x.

Case 2: $nstatus(p) \neq$ sleeping in s. $\mathcal{A}_x(s, \pi) = \pi$ for all x.

- $\pi = ChannelRecv(k, m)$. For all x, $\mathcal{A}_x(s, \pi)$ is empty, with the following exceptions: If $m =$ CONNECT(l) or CHANGEROOT, then $\mathcal{A}_{CON}(s, \pi) = \pi$. If $m =$ INITIATE(l, c, st), then $\mathcal{A}_{NOT}(s, \pi) = ChannelRecv(k,$ NOTIFY$(l, c))$, and if $st =$ find, then $\mathcal{A}_{DC}(s, \pi) = ChannelRecv(k,$ FIND$)$. If $m =$ TEST, ACCEPT or REJECT, then $\mathcal{A}_{TAR}(s, \pi) = \pi$. If $m =$ REPORT(w), then $\mathcal{A}_{DC}(s, \pi) = \pi$.

- $\pi = ChannelSend(k, m)$. Analogous to $ChannelRecv(k, m)$.

- $\pi = ReceiveConnect(\langle q, p\rangle, l)$. Let $f = fragment(p)$ and $g = fragment(q)$. (Later we will show that the following four cases are exhaustive.)

Case 1: $nstatus(p) =$ sleeping in s. If $\langle p, q\rangle$ is not the minimum-weight external link of p in s, then $\mathcal{A}_x(s, \pi) = ChangeRoot(f)$ for all x. If $\langle p, q\rangle$ is the minimum-weight external link of p in s, then, for all x, $\mathcal{A}_x(s, \pi) = ChangeRoot(f)$ t_x $Merge(f, g)$, where t_x is the state of x resulting from applying $ChangeRoot(f)$ to $\mathcal{S}_x(s)$.

Case 2: $nstatus(p) \neq$ sleeping, $l = nlevel(p)$, and no CONNECT message is in $queue(\langle p, q\rangle)$ in s. If $lstatus(\langle p, q\rangle) =$ unknown in s, then $\mathcal{A}_x(s, \pi)$ is empty for all x. If $lstatus(\langle p, q\rangle) \neq$ unknown in s, then $\mathcal{A}_{TAR}(s, \pi)$ is empty, and $\mathcal{A}_x(s, \pi) = AfterMerge(p, q)$ for all other x.

Case 3: $nstatus(p) \neq$ sleeping, $l = nlevel(p)$, and a CONNECT message is in $queue(\langle p, q\rangle)$ in s. $\mathcal{A}_x(s, \pi) = Merge(f, g)$ for all x.

Case 4: $nstatus(p) \neq$ sleeping, and $l < nlevel(p)$ in s. $\mathcal{A}_x(s, \pi) = Absorb(f, g)$ for all x.

- $\pi = ReceiveInitiate(\langle q, p\rangle, l, c, st)$.

$\mathcal{A}_{TAR}(s, \pi) = SendTest(p)$ if $st =$ find, and is empty otherwise.

If $st \neq$ find, then $\mathcal{A}_{DC}(s, \pi)$ is empty; if $st =$ find and there is a link $\langle p, r\rangle$ such that $lstatus(\langle p, r\rangle) =$ unknown in s, then $\mathcal{A}_{DC}(s, \pi) = ReceiveFind(\langle q, p\rangle)$; if $st =$ find and there is no link $\langle p, r\rangle$ such that $lstatus(\langle p, r\rangle) =$ unknown in s, then $\mathcal{A}_{DC}(s, \pi) = ReceiveFind(\langle q, p\rangle)$ t $TestNode(p)$, where t is the state of DC resulting from applying $ReceiveFind(\langle q, p\rangle)$ to $\mathcal{S}_{DC}(s)$.

$\mathcal{A}_{NOT}(s, \pi) = ReceiveNotify(\langle q, p\rangle, l, c)$.

$\mathcal{A}_{CON}(s,\pi)$ is empty.

- $\pi = ReceiveTest(\langle q,p\rangle, l, c)$. Let $f = fragment(p)$.

Case 1: $nstatus(p) =$ sleeping in s.

$\mathcal{A}_{TAR}(s,\pi) = ChangeRoot(f)\ t\ \pi$, where t is the same as $\mathcal{S}_{TAR}(s)$ except that $rootchanged(f) =$ true and $lstatus(minlink(f)) =$ branch in t.

$\mathcal{A}_x(s,\pi) = ChangeRoot(f)$ for all other x.

Case 2: $nstatus(p) \neq$ sleeping in s.

$\mathcal{A}_{TAR}(s,\pi) = \pi$ if $l \leq nlevel(p)$ or $nlevel(p) = level(f)$ in s, and is empty otherwise.

$\mathcal{A}_{DC}(s,\pi) = TestNode(p)$ if $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p,q\rangle$, and $lstatus(\langle p,r\rangle) \neq$ unknown for all $r \neq q$, in s, and is empty otherwise.

$\mathcal{A}_x(s,\pi)$ is empty for all other x.

- $\pi = ReceiveAccept(\langle q,p\rangle)$.

$\mathcal{A}_{TAR}(s,\pi) = \pi$.

$\mathcal{A}_{DC}(s,\pi) = TestNode(p)$.

$\mathcal{A}_x(s,\pi)$ is empty for all other x.

- $\pi = ReceiveReject(\langle q,p\rangle)$.

$\mathcal{A}_{TAR}(s,\pi) = \pi$.

$\mathcal{A}_{DC}(s,\pi) = TestNode(p)$ if there is no $r \neq q$ such that $lstatus(\langle p,r\rangle) =$ unknown in s, and is empty otherwise.

$\mathcal{A}_x(s,\pi)$ is empty for all other x.

- $\pi = ReceiveReport(\langle q,p\rangle, w)$. Let $f = fragment(p)$.

Case 1: $(p,q) = core(f)$, $nstatus(p) \neq$ find, $w > bestwt(p)$, and $lstatus(bestlink(p)) =$ branch in s.

$\mathcal{A}_{DC}(s,\pi) = \pi$.

$\mathcal{A}_x(s,\pi) = ComputeMin(f)$ for all other x.

Case 2: $(p,q) = core(f)$, $nstatus(p) \neq$ find, $w > bestwt(p)$, and $lstatus(bestlink(p)) \neq$ branch in s.

$\mathcal{A}_{DC}(s,\pi) = \pi\ t_{DC}\ ChangeRoot(f)$, where t_{DC} is the state of DC resulting from applying π to $\mathcal{S}_{DC}(s)$.

$\mathcal{A}_{CON}(s,\pi) = ComputeMin(f)$.

$\mathcal{A}_x(s,\pi) = ComputeMin(f)\ t_x\ ChangeRoot(f)$ for all other x, where t_x is the state of x resulting from applying $ComputeMin(f)$ to $\mathcal{S}_x(s)$.

Case 3: $(p,q) \neq core(f)$ or $nstatus(p) =$ find or $w \leq bestwt(p)$ in s.

$\mathcal{A}_{DC}(s,\pi) = \pi$.

$\mathcal{A}_x(s,\pi)$ is empty for all other x.

- $\pi = ReceiveChangeRoot(\langle q,p\rangle)$. Let $f = fragment(p)$.

$\mathcal{A}_{CON}(s,\pi) = \pi$.

For all other x, $\mathcal{A}_x(s,\pi) = ChangeRoot(f)$ if $lstatus(bestlink(p)) \neq$ branch in s, and is empty otherwise.

For the rest of this chapter, let I be the set of names $\{TAR, DC, NOT, CON\}$. The following predicates are true in any state of GHS satisfying $\bigwedge_{x\in I}(P'_x \circ \mathcal{S}_x) \wedge P_{GHS}$. I.e., they are derivable from P_{GHS}, together with the TAR, DC, NOT, CON, GC, COM and HI predicates.

- GHS-L: If $AfterMerge(p,q)$ is enabled for DC or NOT, then a CONNECT message is at the head of $queue(\langle q,p\rangle)$.

Proof: First we show the predicate for DC. Let $f = fragment(p)$.
1. $(p,q) = core(f)$, by precondition.
2. FIND is in $dcqueue(\langle q,p\rangle)$, by precondition.
3. No FIND is in $dcqueue(\langle p,q\rangle)$, by precondition.
4. $dcstatus(q) =$ unfind, by precondition.
5. No REPORT is in $dcqueue(\langle q,p\rangle)$, by precondition.
6. $q \in testset(f)$, by Claims 1 through 5 and DC-G.
7. $testlink(p) = nil$, by Claim 4 and GHS-H.

8. A CONNECT is in $queue(\langle q,p\rangle)$, by Claims 1, 3, 6 and 7.
9. $(p,q) \in subtree(f)$, by Claim 1 and COM-F.
10. No INITIATE($*,*$,found) is in $queue(\langle q,p\rangle)$, by Claim 1 and GHS-E.
11. No CHANGEROOT is in $queue(\langle q,p\rangle)$, by Claim 1.
12. No ACCEPT is in $queue(\langle q,p\rangle)$, by Claim 9 and TAR-F.
13. CONNECT precedes any FIND, TEST, or REJECT in $queue(\langle q,p\rangle)$, by Claim GHS-C.

Claims 5, 8, 10, 11, 12 and 13 give the result.

For NOT, we show that if *AfterMerge*(p,q) for NOT is enabled, then *AfterMerge*(p,q) for DC is enabled.

1. $(p,q) = core(f)$, by precondition.
2. NOTIFY$(nlevel(p)+1,(p,q))$ is in $nqueue(\langle q,p\rangle)$, by precondition.
3. No NOTIFY$(nlevel(p)+1,(p,q))$ is in $nqueue(\langle p,q\rangle)$, by precondition.
4. $nlevel(q) \neq nlevel(p)+1$, by precondition.
5. INITIATE$(nlevel(p)+1,(p,q)$,find) is in $queue(\langle q,p\rangle)$, by Claims 1 and 2 and GHS-E.
6. $nlevel(p)+1 = level(f)$, by Claim 5 and GHS-D.
7. No INITIATE($*,*$,find) is in $queue(\langle p,q\rangle)$, by Claims 3 and 6 and GHS-D.
8. q is not up-to-date, by Claims 4 and 6 and GHS-I.
9. $dcstatus(q) \neq$ find, by Claim 8 and DC-I(a).
10. No REPORT is in $queue(\langle q,p\rangle)$, by Claims 1 and 8 and DC-C(a).

By Claims 1, 5, 7, 9 and 10, *AfterMerge*(p,q) for DC is enabled. □

- GHS-M: If $testlink(p) \neq nil$ or $findcount(p) > 0$, then no FIND message is headed toward p, and no CONNECT message is in $queue(\langle q,r\rangle)$, where $(q,r) = core(fragment(p))$ and $p \in subtree(q)$.

Proof:

1. $testlink(p) \neq nil$ or $findcount(p) > 0$, by assumption.
2. $nstatus(p) =$ find, by Claim 1 and either GHS-H or DC-H(b).
3. $dcstatus(t) =$ find for all t between q and p inclusive, by Claim 2 and DC-H(a).
4. No FIND message is headed toward p, by Claim 4 and DC-D(b).
5. No CONNECT is in $queue(\langle q,r\rangle)$, or $lstatus(\langle r,q\rangle) =$ unknown, or CONNECT is in $queue(\langle r,q\rangle)$, by Claim 3 and GHS-B(c).
6. $(q,r) \in subtree(fragment(p))$, by COM-F.
7. $lstatus(\langle r,q\rangle) \neq$ unknown, by Claim 6 and TAR-A(b).
8. If CONNECT is in $queue(\langle r,q\rangle)$ then no CONNECT is in $queue(\langle q,r\rangle)$, by Claim 6.

9. If no CONNECT is in $queue(\langle r,q\rangle)$ then no CONNECT is in $queue(\langle q,r\rangle)$, by Claims 5 and 7.

Claims 4, 8 and 9 give the result. □

Lemma 25: *GHS simultaneously simulates the set of automata* $\{TAR, DC, NOT, CON\}$ via $\{\mathcal{M}_x : x \in I\}$, P_{GHS}, and $\{P'_x : x \in I\}$.

Proof: By inspection, the types are correct. By Corollaries 18, 20, 22 and 24, P'_x is a predicate true in every reachable state of x, for all x.

(1) Let s be in $start(GHS)$. Obviously P_{GHS} is true in s and $\mathcal{S}_x(s)$ is in $start(x)$ for all x.

(2) Obviously, $\mathcal{A}_x(s,\pi)|ext(x) = \pi|ext(GHS)$ for all x.

(3) Let (s', π, s) be a step of GHS such that $\bigwedge_{x\in I} P'_x(\mathcal{S}_x(s'))$ and $P_{GHS}(s')$ are true. By Corollaries 18, 20, 22 and 24, we can assume the HI, COM, GC, TAR, DC, NOT and CON predicates are true in s', as well as the GHS predicates. Below, we show (3a), that P_{GHS} is true in s (only for those predicates whose truth in s is not obvious), and either (3b) or (3c), as appropriate, that the step simulations for TAR, DC, NOT, and CON are correct.

i) π is InTree($\langle$p,q$\rangle$). Let $f = fragment(p)$ in s'.

(3a) Obviously, P_{GHS} is true in s.

(3b)/(3c) $\mathcal{A}_x(s',\pi) = \pi$ for all x.

Claims about s':

1. $answered(\langle p,q\rangle)$ = false, by precondition.
2. $lstatus(\langle p,q\rangle)$ = branch, by precondition.
3. $nstatus(p) \neq$ sleeping, by Claim 2 and GHS-A(c).
4. $awake$ = true, by Claim 3.
5. $\langle p,q\rangle \in subtree(f)$ or $\langle p,q\rangle = minlink(f)$, by Claim 2 and TAR-A(a).

π is enabled in $\mathcal{S}_x(s')$ by Claims 1 and 2 for $x = TAR$, and by Claims 1, 4 and 5 for all other x. Obviously, its effects are mirrored in $\mathcal{S}_x(s)$ for all x.

ii) π is NotInTree($\langle$p,q$\rangle$). Let $f = fragment(p)$ in s'.

(3a) Obviously, P_{GHS} is true in s.

(3b)/(3c) $\mathcal{A}_x(s, \pi) = \pi$ for all x.

Claims about s':

1. *answered*($\langle p, q\rangle$) = false, by precondition.
2. *lstatus*($\langle p, q\rangle$) = rejected, by precondition.
3. *nstatus*(p) $\neq$ sleeping, by Claim 2 and GHS-A(c).
4. *awake* = true, by Claim 3.
5. *fragment*(p) = *fragment*(q) and $(p, q) \neq$ *subtree*(f), by Claim 2 and TAR-B.

π is enabled in $\mathcal{S}_x(s')$ by Claims 1 and 2 for $x = TAR$, and by Claims 1, 4 and 5 for all other x. Obviously its effects are mirrored in $\mathcal{S}_x(s)$ for all x.

iii) π **is Start(p).** Let f = *fragment*(p).

Case 1: *nstatus*(p) $\neq$ sleeping in s'. $\mathcal{A}_x(s', \pi) = \pi$ for all x. Obviously $\mathcal{S}_x(s')\pi\mathcal{S}_x(s)$ is an execution fragment of x for all x, and P_{GHS} is true in s.

Case 2: *nstatus*(p) = sleeping in s'.

(3b)/(3c) For all x, $\mathcal{A}_x(s', \pi) = \pi\ t_x$ *ChangeRoot*(f), where t_x is the same as $\mathcal{S}_x(s')$ except that *awake* = true in t_x. For all x, we must show that π is enabled in $\mathcal{S}_x(s')$ (which is true because π is an input action), that its effects are mirrored in t_x (which is true by definition of t_x), that *ChangeRoot*(f) is enabled in t_x, and that its effects are mirrored in $\mathcal{S}_x(s)$.

Let l be the minimum-weight external link of p. (It exists by GHS-A(a) and the assumption that $|V(G)| > 1$.)

Claims about s':

1. *nstatus*(p) = sleeping, by assumption.
2. *subtree*(f) = $\{p\}$, by Claim 1 and GHS-A.
3. *minlink*(f) = l, by Claim 2 and definition.
4. *lstatus*($\langle p, q\rangle$) = unknown, for all q, by Claim 1 and GHS-A(c).
5. *rootchanged*(f) = false, by Claim 4 and TAR-H.

Claims about t_x, for all x:

6. *awake* = true, by definition.
7. *subtree*(f) = $\{p\}$, by Claim 2.
8. *rootchanged*(f) = false, by Claim 5.

9. $minlink(f) = l$, by Claim 3.

ChangeRoot(f) is enabled in t_{CON} by Claims 6, 7 and 8. For all other x, *ChangeRoot*(f) is enabled in t_x by Claims 6, 8 and 9.

Claims about s:

10. CONNECT(0) is in $queue(l)$, by code.
11. $lstatus(l)$ = branch, by code.
12. $rootchanged(f)$ = true, by Claims 10 and 11 and choice of l.

For most of the other derived variables, it is obvious that they are the same in s' and s. Although $nstatus(p)$ changes, $dcstatus(p)$ remains unchanged. Even though $lstatus(l)$ changes to branch, MSF does not change, since a CONNECT message is in $queue(l)$.

For $x = TAR$, the effects of *ChangeRoot*(f) are mirrored in $\mathcal{S}_x(s)$ by Claims 11 and 12. For $x = CON$, the effects of *ChangeRoot*(f) are mirrored in $\mathcal{S}_x(s)$ by Claim 10. For all other x, the effects of *ChangeRoot*(f) are mirrored in $\mathcal{S}_x(s)$ by Claim 12.

(3a) *More Claims about s':*

13. $lstatus(\langle q,p\rangle) \neq$ rejected, for all q, by Claim 2 and TAR-B.
14. If $lstatus(\langle q,p\rangle)$ = branch, then a CONNECT is in $queue(\langle q,p\rangle)$, for all q, by Claim 2.
15. $testset(f) = \emptyset$, by Claim 3 and GC-C.
16. $testlink(p) = nil$, by Claim 15.
17. $queue(l)$ is empty, by Claim 1 and GHS-A(b).

GHS-A is vacuously true since $nstatus(p)$ = found in s.

GHS-B: vacuously true for CONNECT added to $queue(l)$ by Claims 13 and 14; vacuously true for any CONNECT already in $queue(reverse(l))$ by Claim 10; vacuously true for any CONNECT already in $queue(\langle q,p\rangle)$, for any q such that $\langle p,q\rangle \neq l$, by Claim 4.

GHS-C is true by Claim 17 and code.

GHS-H is vacuously true by Claim 16.

No change affects the others.

iv) π is ChannelRecv(k,m) or ChannelSend(k,m). Obviously $P_{GHS}(s)$ is true, and the step simulations are correct.

v) π is ReceiveConnect($\langle$q,p$\rangle$,l). Let $f = fragment(p)$, and $g = fragment(q)$ in s'. We consider four cases. We now show that they are exhaustive, i.e., that $l > nlevel(p)$ is impossible. First, suppose $\langle q,p \rangle$ is an external link of g. By CON-D, $l = level(g)$ and $\langle q,p \rangle = minlink(g)$. By NOT-D, $level(g) \leq nlevel(p)$. Second, suppose $\langle q,p \rangle$ is an internal link of $g = f$. By CON-E, $(p,q) = core(f)$, and $l < level(f)$. But by NOT-C, $nlevel(p) \geq level(f) - 1$.

Case 1: $nstatus(p) =$ sleeping. This case is divided into two subcases. First we prove some claims true in both subcases. Let k be the minimum-weight external link of p.

Claims about s'*:*

1. CONNECT(l) is at head of $queue_p(\langle q,p \rangle)$, by precondition.
2. $nstatus(p) =$ sleeping, by assumption.
3. $subtree(f) = \{p\}$, by Claim 2 and GHS-A.
4. $rootchanged(f) =$ false, by Claim 2, GHS-A(c) and TAR-H.
5. $minlink(f) = k$, by Claim 3 and definition.
6. $awake =$ true, by Claim 1 and CON-A.
7. No FIND is in $queue(\langle q,p \rangle)$, by Claim 3 and DC-D(a).
8. $f \neq g$, by Claim 3.
9. $\langle q,p \rangle$ is an external link of g, by Claim 8.
10. $minlink(g) = \langle q,p \rangle$, by Claims 1 and 9 and CON-D
11. $level(g) \leq level(f)$, by Claim 10 and COM-A.
12. $l = level(g)$, by Claims 1 and 9 and CON-D.
13. $level(f) = 0$, by Claim 3 and COM-F.
14. $l \leq 0$, by Claims 11, 12 and 13.
15. $l = 0$, by Claim 14 and COM-F.
16. $nlevel(p) = 0$, by Claims 3 and 13.

Subcase 1a: $\langle p,q \rangle \neq k$. By Claim 2 and GHS-A(c), $lstatus(\langle p,q \rangle) =$ unknown in s', and the same is true in s. This fact, together with Claims 15 and 16, shows that the only change is that the CONNECT(l) message is requeued.

(3a) P_{GHS} can be shown to be true in s by an argument very similar to that for $\pi = Start(p)$, Case 2, since the only change is that the CONNECT(l) message is requeued. Claim 7 verifies that GHS-C is true in s.

(3b)/(3c) For all x, $\mathcal{A}_x(s', \pi) = ChangeRoot(f)$. For $x = CON$, $ChangeRoot(f)$ is enabled in $\mathcal{S}_x(s')$ by Claims 6, 4 and 3; for all other x, it is enabled by Claims 6, 4 and 5.

Claims about s:

17. $lstatus(k) =$ branch, by code.
18. CONNECT(0) is added to the end of $queue(k)$, by code.
19. $rootchanged(f) =$ true, by Claims 17 and 18 and choice of k.

For most of the other derived variables, it is obvious that they are the same in s' and s. Although $nstatus(p)$ changes, $dcstatus(p)$ remains unchanged. Even though $lstatus(k)$ changes to branch, MSF does not change, since a CONNECT message is in $queue(k)$.

The effects of $ChangeRoot(f)$ are mirrored in $\mathcal{S}_x(s)$ by Claims 17 and 19 for $x = TAR$, by Claim 18 for $x = CON$, and by Claim 19 for all other x.

Subcase 1b: $\langle p, q \rangle = k$.

(3b)/(3c) For all x, $\mathcal{A}_x(s', \pi) = ChangeRoot(f)\ t_x\ Merge(f, g)$, where t_x is the result of applying $ChangeRoot(f)$ to $\mathcal{S}_x(s')$. $ChangeRoot(f)$ is enabled in $\mathcal{S}_x(s')$ by Claims 6, 4 and 3 for $x = CON$, and by Claims 6, 4 and 5 for all other x. Its effects are obviously mirrored in t_x.

More claims about s':

20. $k = \langle p, q \rangle$, by assumption.
21. $\langle p, q \rangle$ is an external link of f, by Claim 8.
22. $rootchanged(g) =$ true, by Claim 1 and Claim 9.
23. Only one CONNECT message is in $queue(\langle q, p \rangle)$, by Claims 1 and 9 and CON-D.
24. $lstatus(\langle q, p \rangle) =$ branch, by Claims 10 and 22 and TAR-H.
25. $level(g) = 0$, by Claims 12 and 15.
26. $subtree(g) = \{q\}$, by Claim 25 and COM-F.
27. $nlevel(q) = 0$, by Claims 25 and 26.

28. No INITIATE message is in $queue(\langle p,q \rangle)$ or $queue(\langle q,p \rangle)$, by Claims 9 and 21 and NOT-H(e).
29. No CONNECT message is in $queue(\langle p,r \rangle)$ for any $r \neq q$, by Claims 3 and 20 and CON-D.
30. No CONNECT message is in $queue(\langle q,r \rangle)$ for any $r \neq p$, by Claims 10 and 26 and CON-D.

Claims about t_x:

31. $f \neq g$, by Claim 8.
32. $rootchanged(f) =$ true, by definition of t_x.
33. $rootchanged(g) =$ true, by Claim 22.
34. $minedge(f) = minedge(g) = (p,q)$, by Claims 5, 10 and 20.
35. If $x = CON$, then CONNECT(0) is in $cqueue(\langle p,q \rangle)$, by definition of t_x.
36. If $x = CON$, then CONNECT(0) is at the head of $cqueue(\langle q,p \rangle)$, by Claims 1 and 15.

$Merge(f,g)$ is enabled in t_x by Claims 34, 35 and 36 for $x = CON$, and by Claims 31, 32, 33 and 34 for all other x.

As we shall shortly show, MSF has changed — the connected components corresponding to f and g have combined. Let h be the fragment corresponding to this new connected component.

Claims about s:

37. No CONNECT is in $queue(\langle q,p \rangle)$, by Claim 23 and code.
38. $lstatus(\langle q,p \rangle) =$ branch, by Claim 24 and code.
39. $(p,q) \in MSF$, by Claims 37 and 38.
40. $subtree(h)$ is nodes p and q and the edge between them, by Claims 3, 26 and 39.
41. INITIATE$(1,(p,q),$find$)$ is in $queue(\langle p,q \rangle)$, by code.
42. $level(h) = 1$, by Claims 16, 27, 28, 40 and 41.
43. $core(h) = (p,q)$, by Claims 16, 27, 28, 40 and 41.
44. CONNECT(0) is in $queue(\langle p,q \rangle)$, by code.
45. $testset(h) = \{p,q\}$, by Claims 41 and 44.
46. $minlink(h) = nil$, by Claim 45.
47. $rootchanged(h) =$ false, by Claims 29, 30 and 40.
48. f and g are no longer in *fragments*, by Claims 3, 26, 40 and 43.

The effects of $Merge(f,g)$ are mirrored in $\mathcal{S}_x(s)$ by Claims 40, 42, 43, 45, 46, 47 and 48 for $x = TAR$; by Claims 40, 41, 42, 43, 45, 47 and 48 for $x = DC$; by

Claims 40, 41, 46, 47 and 48 for $x = NOT$; and by Claims 40, 42, 43, 46 and 48 for $x = CON$.

(3a) GHS-A: vacuously true for p by code. By Claim 1 and GHS-A(c), $nstatus(q) \neq$ sleeping in s'; since the same is true in s, changing q's subtree does not invalidate GHS-A(a).

GHS-B: Obviously, the only situation affected is the CONNECT added to $queue(\langle p, q \rangle)$.

(a) $queue(\langle p, q \rangle)$ has the correct contents in s because of the code and the fact that $queue(\langle p, q \rangle)$ is empty in s' by Claim 2 and GHS-A(b).

(b) To show that $queue(\langle q, p \rangle)$ is empty in s, we must show that it contains only the CONNECT in s'. By Claim 1 and GHS-C, there is no TEST or REJECT in $queue(\langle q, p \rangle)$. By Claim 2 and GHS-H, $testlink(p) = nil$; thus, by TAR-D, no ACCEPT is in $queue(\langle q, p \rangle)$. By Claim 3, DC-A(g) and DC-B(a), there is no REPORT in $queue(\langle q, p \rangle)$. By Claim 3 and NOT-H(e), there is no NOTIFY in $queue(\langle q, p \rangle)$. By Claim 3 and CON-C, there is no CHANGEROOT in $queue(\langle q, p \rangle)$. By Claim 1, CON-D and CON-E, there is only one CONNECT in $queue(\langle q, p \rangle)$.

(c) $nstatus(p) \neq$ find in s by code.

(d) By Claims 16 and 27, $nlevel(p) = nlevel(q) = 0$.

GHS-C: No FIND is in $queue(\langle p, q \rangle)$ in s' by Claim 3 and DC-D(a). No REJECT is in $queue(\langle p, q \rangle)$ in s' by Claim 3 and TAR-G. No TEST(l, c), for any l and c, is in $queue(\langle p, q \rangle)$ in s', because by Claims 25 and 13 and TAR-E(b) and TAR-E(c), $l = 0$; yet by TAR-M, $l \geq 1$.

GHS-D: By Claim 42.

GHS-E: By code for the INITIATE added to $queue(\langle p, q \rangle)$. By Claim 28, this is the only relevant message affected.

GHS-H is true in s since $nstatus(p)$ goes from sleeping to found, and $testlink(p)$ is unchanged.

GHS-I: By Claim 45, p and q are both in $testset(h)$ in s. We now show that $nstatus(p) \neq$ find and $nstatus(q) \neq$ find. Then by Claim 40, no node in $subtree(h)$ is up-to-date, so the predicate is vacuously true (for h). By code, $dcstatus(p) =$ found. By Claim 10 and GC-C, $testset(g) = \emptyset$ in s'; by Claim 26, no REPORT message is in $subtree(g)$ in s'. Thus, by DC-I(b), $dcstatus(q) \neq$ find in s'.

GHS-J: vacuously true by Claims 40 and 45 for p and q. No relevant change for any other node.

No change affects the rest.

Case 2: $nstatus(p) \neq$ sleeping, $l = nlevel(p)$, and no CONNECT message is in $queue(\langle p, q\rangle)$ in s'.

Subcase 2a: $lstatus(\langle p, q\rangle) =$ unknown in s'. The only change in going from s' to s is that the CONNECT message is requeued.

(3a) The only GHS predicates affected are GHS-B(a) and GHS-C. By TAR-A(b), $(p, q) \neq subtree(f)$. Thus, by DC-D(a), no FIND is in $queue(\langle q, p\rangle)$ in s', and the predicates are still true in s.

(3b)/(3c) $\mathcal{A}_x(s', \pi)$ is empty for all x. We now show that $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all x, by showing that $cqueue(\langle q, p\rangle)$ contains only the one CONNECT message in s'. By TAR-A(b), (p, q) is not in MSF. Thus, by CON-C, no CHANGEROOT is in $cqueue(\langle q, p\rangle)$. By CON-D and CON-E, only one CONNECT message is in $cqueue(\langle q, p\rangle)$.

Subcase 2b: $lstatus(\langle p, q\rangle) \neq$ unknown in s'.

(3b)/(3c) $\mathcal{A}_{TAR}(s', \pi)$ is empty, and $\mathcal{A}_x(s', \pi) = AfterMerge(p, q)$ for all other x.

Claims about s'*:*

1. CONNECT is at head of $queue_p(\langle q, p\rangle)$, by precondition.
2. $nstatus(p) \neq$ sleeping, by assumption.
3. $nlevel(p) = l$, by assumption.
4. No CONNECT is in $queue(\langle p, q\rangle)$, by assumption.
5. $lstatus(\langle p, q\rangle) \neq$ unknown, by assumption.
6. If $lstatus(\langle p, q\rangle) =$ rejected, then $fragment(p) = fragment(q)$, by TAR-B.

7. If $lstatus(\langle p,q\rangle) =$ branch, then $(p,q) \in subtree(f)$, by Claim 4 and definition of *MSF*.
8. $\langle p,q\rangle$ is an internal link of f, by Claims 5, 6 and 7.
9. $(p,q) = core(f)$, by Claims 1 and 8 and CON-E.
10. INITIATE($nlevel(p)+1,(p,q)$,find) is in $queue(\langle q,p\rangle)$, by Claims 1, 3, 4 and 5 and GHS-B(a).
11. No INITIATE($nlevel(p)+1,(p,q),*$) is in $queue(\langle p,q\rangle)$, by Claims 1, 3, 4 and 5 and GHS-B(b).
12. $dcstatus(q) \neq$ find, by Claims 1, 4 and 5 and GHS-B(c).
13. No REPORT is in $queue(\langle q,p\rangle)$, by Claims 1, 4 and 5 and GHS-B(a).
14. $nlevel(q) = l$, by Claims 1, 4 and 5 and GHS-B(d).

$AfterMerge(p,q)$ is enabled in $\mathcal{S}_x(s')$ by Claims 9, 10, 11, 12 and 13 for $x = DC$; by Claims 3, 9, 10, 11 and 14 for $x = NOT$; and by Claims 1 and 9 for $x = CON$.

Claims about s:

15. CONNECT(l) is dequeued from $queue_p(\langle q,p\rangle)$, by code.
16. FIND is in $queue(\langle p,q\rangle)$, by code.
17. INITIATE($nlevel(p)+1,(p,q)$,find) is in $queue(\langle p,q\rangle)$, by code.

The only derived variables that are not obviously unchanged are $testset(f)$, $level(f)$ and $core(f)$. Claims 15 and 16 show that $testset(f)$ is unchanged. Claims 10 and 17 show that $level(f)$ and $core(f)$ are unchanged.

The effects of $AfterMerge(p,q)$ are mirrored in $\mathcal{S}_x(s)$ by Claim 16 for $x = DC$; by Claim 17 for $x = NOT$; and by Claim 15 for $x = CON$. It is easy to see that $\mathcal{S}_{TAR}(s') = \mathcal{S}_{TAR}(s)$.

(3a) GHS-A: By Claim 2, adding a message to a queue of p does not invalidate GHS-A(b).

GHS-B: By Claim 8 and CON-E, there is only one CONNECT message in $queue(\langle q,p\rangle)$ in s'. Since it is removed in s, the predicate is vacuously true for a CONNECT in $queue(\langle q,p\rangle)$. By Claim 4, the predicate is vacuously true for a CONNECT in $queue(\langle p,q\rangle)$.

GHS-C: By Claim 4, vacuously true for $queue(\langle p,q\rangle)$.

GHS-D: By Claim 10 and GHS-D, $nlevel(p) + 1 = level(f)$. This together with Claim 9 gives the result.

GHS-E is true by code.

No change affects the rest.

Case 3: $nstatus(p) \neq$ sleeping, $l = nlevel(p)$, and a CONNECT message is in $queue(\langle p, q \rangle)$ in s'.

(3b)/(3c) $\mathcal{A}_x(s', \pi) = Merge(f, g)$ for all x.

Claims about s':

1. CONNECT(l) is at head of $queue(\langle q, p \rangle)$, by precondition.
2. $l = nlevel(p)$, by assumption.
3. CONNECT(m) is in $queue(\langle p, q \rangle)$, by assumption.
4. $\langle p, q \rangle$ is an external link of p, by Claims 1 and 3.
5. $\langle q, p \rangle$ is an external link of q, by Claims 1 and 3.
6. $f \neq g$, by Claim 4.
7. $rootchanged(f) =$ true, by Claims 1 and 4.
8. $rootchanged(g) =$ true, by Claims 3 and 5.
9. $\langle q, p \rangle = minlink(g)$, by Claims 1 and 5 and CON-D.
10. $\langle p, q \rangle = minlink(f)$, by Claims 3 and 4 and CON-D.
11. $minedge(f) = minedge(g)$, by Claims 9 and 10.
12. $m = level(f)$, by Claims 3 and 4 and CON-D.
13. $nlevel(p) = level(f)$, by Claim 10 and NOT-D.
14. $m = l$, by Claims 2, 12 and 13.

$Merge(f, g)$ is enabled in $\mathcal{S}_{CON}(s')$ by Claims 1, 3, 4, 5 and 14, and for all other x by Claims 6, 7, 8 and 11.

15. Only one CONNECT message is in $queue(\langle q, p \rangle)$, by Claim 1 and CON-D.
16. $lstatus(\langle q, p \rangle) =$ branch, by Claims 8 and 9 and TAR-H.
17. $lstatus(\langle p, q \rangle) =$ branch, by Claims 7 and 10 and TAR-H.
18. $level(g) = l$, by Claims 1 and 5 and CON-D.
19. If INITIATE($l', c, *$) is in $subtree(f)$, then $l' \leq l$, by Claims 12 and 14.
20. If INITIATE($l', c, *$) is in $subtree(g)$, then $l' \leq l$, by Claim 18.
21. $nlevel(r) \leq l$ for all $r \in nodes(f)$, by Claims 12 and 14.

22. $nlevel(r) \leq l$ for all $r \in nodes(g)$, by Claim 18.
23. No INITIATE message is in $queue(\langle q,p \rangle)$ or $queue(\langle p,q \rangle)$, by Claims 4 and 5 and NOT-H(e).
24. No CONNECT is in $queue(\langle r,t \rangle)$, where $r \in nodes(f)$, and $\langle r,t \rangle \neq \langle p,q \rangle$, by Claim 10 and CON-D and CON-F.
25. No CONNECT is in $queue(\langle r,t \rangle)$, where $r \in nodes(g)$ and $\langle r,t \rangle \neq \langle q,p \rangle$, by Claim 9 and CON-D and CON-F.
26. $(p,q) \neq core(f)$, by Claim 4 and COM-F.
27. $(p,q) \neq core(g)$, by Claim 5 and COM-F.

As we shall shortly show, MSF has changed — the connected components corresponding to f and g have combined. Let h be the fragment corresponding to this new connected component.

Claims about s:

28. No CONNECT is in $queue(\langle q,p \rangle)$, by Claim 15 and code.
29. $lstatus(\langle q,p \rangle) =$ branch, by Claim 16.
30. $(p,q) \in MSF$, by Claims 28 and 29.
31. $subtree(h)$ is the union of the old $subtree(f)$ and $subtree(g)$ and (p,q), by Claim 30.
32. INITIATE$(l+1,(p,q)$,find) is in $queue(\langle p,q \rangle)$, by Claim 2 and 17 and code.
33. if INITIATE$(l',c,*)$ is in $subtree(h)$, then $l' \leq l+1$, by Claims 19, 20, 23, 31 and 32.
34. $nlevel(r) \leq l$ for all $r \in nodes(h)$, by Claims 21, 22 and 31.
35. $level(h) = l+1$, by Claims 33 and 34.
36. $core(h) = (p,q)$, by Claims 19, 20, 23, 31, 32, and 34.
37. CONNECT(l) is in $queue(\langle p,q \rangle)$, by Claims 3 and 14
38. $testset(h) = nodes(h)$, by Claims 31, 32 and 37.
39. $minlink(h) = nil$, by Claim 38.
40. $rootchanged(h) =$ false, by Claims 24, 25 and 31.
41. f and g are no longer in *fragments*, by Claims 26, 27, 31 and 36.

The effects of $Merge(f,g)$ are mirrored in $\mathcal{S}_x(s)$ by Claims 31, 35, 36, 38, 39, 40 and 41 for TAR; by Claims 31, 35, 36, 38, 40 and 41 for DC; by Claims 31, 39, 40 and 41 for NOT; and by Claims 28, 31, 35, 36, 39, and 41 for CON.

(3a) GHS-A: Vacuously true for p by assumption. Vacuously true for q by Claim 1 and GHS-A(b).

GHS-B: Obviously, the only situation affected is the CONNECT in $queue(\langle p,q\rangle)$.

(a) We must show that in s', $queue(\langle p,q\rangle)$ consists only of a CONNECT(l) message. (The code adds the appropriate INITIATE message.) By Claim 3 and GHS-C, no TEST or REJECT is in $queue(\langle p,q\rangle)$. By Claim 4, DC-A(g) and DC-B(a), no REPORT is in $queue(\langle p,q\rangle)$. By Claim 23, no NOTIFY is in $queue(\langle p,q\rangle)$. By Claim 4 and CON-C, no CHANGEROOT is in $queue(\langle p,q\rangle)$. By Claims 3 and 14, a CONNECT(l) message is in $queue(\langle p,q\rangle)$, and by CON-E and CON-F, it is the only CONNECT message in that queue.

(b) A very similar argument to that in (a) shows that in s', $queue(\langle q,p\rangle)$ consists only of a CONNECT(l) message. (Since it is removed in s, the queue is then empty.)

(c) If $|nodes(f)| > 1$, then $dcstatus(p) \neq$ find by Claim 10. Suppose $subtree(f) = \{p\}$. Obviously, no REPORT message is headed toward p in s'. By Claim 10 and GC-C, $testset(f) = \emptyset$ in s'. Thus, by DC-I(b), $dcstatus(p) \neq$ find in s'. In both cases, $nstatus(p)$ does not change in s.

(d) $nlevel(p) = l$ in s' by assumption. $nlevel(q) = l$ in s' by Claims 9 and 18 and NOT-D. These values are unchanged in s.

GHS-C: By the same argument as in GHS-B(a), adding the INITIATE message is OK.

GHS-D: by Claim 35.

GHS-E: By code, for the INITIATE added. By Claim 23, there are no leftover INITIATE messages affected by the change of core.

GHS-I: We show no $r \in nodes(h)$ in s is up-to-date. By Claim 38, r is in $testset(h)$. By the same argument as in GHS-B(c), $dcstatus(r) \neq$ find.

GHS-J: Vacuously true by Claim 38.

No change affects the rest.

Case 4: $nstatus(p) \neq$ sleeping, and $l < nlevel(p)$ in s'.

(3b)/(3c) $\mathcal{A}_x(s', \pi) = Absorb(f,g)$ for all x.

Claims about s':

1. CONNECT(l) is at head of $queue(\langle q,p\rangle)$, by precondition.
2. $l < nlevel(p)$, by assumption.
3. $lstatus(\langle p,q\rangle) =$ unknown, or a CONNECT is in $queue(\langle p,q\rangle)$, by Claims 1 and 2 and GHS-B(d).
4. $\langle q,p\rangle$ is an external link of g, by Claims 1 and 3.
5. $minlink(g) = \langle q,p\rangle$, by Claims 1 and 4 and CON-D.
6. $l = level(g)$, by Claims 1 and 4 and CON-D.
7. $rootchanged(g) =$ true, by Claims 1 and 4.
8. $nlevel(p) \leq level(f)$, by definition of $level(f)$.
9. $level(g) < level(f)$, by Claims 2, 6 and 8.
10. $lstatus(\langle q,p\rangle) =$ branch, by Claims 5 and 7 and TAR-H.
11. If INITIATE($l', c, *$) is in $subtree(g)$, then $l' < level(f)$, by Claims 6 and 9.
12. If INITIATE($l', c, *$) is in $subtree(f)$, then $l' \leq level(f)$, by definition of $level(f)$.
13. $nlevel(r) < level(f)$, for all $r \in nodes(g)$, by Claims 6 and 9.
14. $nlevel(r) \leq level(f)$, for all $r \in nodes(f)$, by definition of $level(f)$.
15. No INITIATE message is in $queue(\langle q,p\rangle)$ or $queue(\langle p,q\rangle)$, by Claim 4 and NOT-H(e).
16. No CONNECT message is in $queue(\langle r,t\rangle)$, where $r \in nodes(g)$, $\langle r,t\rangle \neq \langle q,p\rangle$, by Claim 5 and CON-D and CON-F.
17. $f \neq g$, by Claim 4.
18. $l \geq 0$, by Claim 6 and COM-F.
19. $level(f) > 0$, by Claims 18 and 9.
20. $core(f) \neq nil$, by Claim 19 and COM-F.
21. $core(f) \in subtree(f)$, by Claim 20 and COM-F.
22. If $subtree(g) = \{q\}$, then $core(g) = nil$, by COM-F.
23. if $subtree(g) \neq \{q\}$, then $core(g) \in subtree(g)$, by COM-F.
24. Only one CONNECT message is in $queue(\langle q,p\rangle)$, by Claims 1 and 4 and CON-D.
25. $testset(g) = \emptyset$, by Claim 5 and GC-C.
26. $testlink(r) = nil$, for all $r \in nodes(g)$, by Claim 25.
27. If $testlink(p) \neq nil$, then $p \in testset(f)$, by definition.
28. If $testlink(p) \neq nil$, then $nstatus(p) =$ find, by GHS-H.
29. If $nstatus(p) =$ find, then no FIND message is headed toward p, by DC-D(b) and DC-H(a).
30. $lstatus(\langle r,t\rangle) \neq$ unknown, where $(r,t) = core(f)$, by Claim 21 and TAR-A(b).
31. If CONNECT is in $queue(\langle r,t\rangle)$, then no CONNECT is in $queue(\langle t,r\rangle)$, where $(r,t) = core(f)$, by Claim 21.
32. If $nstatus(p) =$ find and $p \in subtree(r)$, then $nstatus(r) =$ find, for all r, by DC-H(a).

33. If $nstatus(p) =$ find, then no CONNECT is in $queue(\langle r,t\rangle)$, where $(r,t) = core(f)$ and $p \in subtree(r)$, by Claims 30, 31 and 32 and GHS-B(c).
34. If $nstatus(p) =$ find and $p \in testset(f)$, then $testlink(p) \neq nil$, by Claims 29 and 33.

$Absorb(f,g)$ is enabled in $S_x(s')$ by Claims 7, 9 and 5 for TAR and DC; by Claims 7, 6 and 2, and 5 for NOT; and by Claims 1, 6 and 9, and 5 for CON.

As we shall shortly show, MSF has changed — the connected components corresponding to f and g have combined. Let h be the fragment corresponding to this new connected component. We shall show that $h = f$, i.e., that the core of h in s is non-nil, and is the same as the core of f in s'.

Claims about s:

35. No CONNECT message is in $queue(\langle q,p\rangle)$, by Claim 24 and code.
36. $lstatus(\langle q,p\rangle) =$ branch, by Claim 10.
37. $(p,q) \in MSF$, by Claims 35 and 36.
38. $subtree(h)$ is the union of the old $subtree(f)$ and $subtree(g)$ and (p,q), by Claim 37.
39. INITIATE$(nlevel(p), nfrag(p), nstatus(p))$ is in $queue(\langle p,q\rangle)$, by code.
40. $level(h) =$ old $level(f)$, by Claims 11, 12, 13, 14, 15 and 38.
41. $core(h) =$ old $core(f)$, by Claims 11, 12, 13, 14, 15 and 38.
42. $h = f$, by Claim 41.
43. $g \notin fragments$, by Claims 38 and 41.
44. NOTIFY$(nlevel(p), nfrag(p))$ is added to $queue_p(\langle p,q\rangle)$, by code.

First, we discuss how $testset(f)$ changes. If $p \in testset(f)$ in s' because of a FIND or CONNECT message, then every node in $nodes(g)$ in s' is in $testset(f)$ in s because of the same FIND or CONNECT message. If $p \in testset(f)$ in s' because $testlink(p) \neq nil$, then a FIND message is added to $queue(\langle p,q\rangle)$ in s, causing every node formerly in $nodes(g)$ to be in $testset(f)$. If p is not in $testset(f)$ in s', then no FIND message is headed toward p, and no CONNECT message is in $queue(\langle r,t\rangle)$, with $p \in subtree(r)$; thus, Claim 25 implies that in s, no node formerly in $nodes(g)$ is in $testset(f)$.

By the previous paragraph, and inspection, the effects of $Absorb(f,g)$ are mirrored in $S_x(s)$ by Claims 36, 38, 42 and 43 for $x = TAR$; by Claims 27, 28, 34, 38, 42 and 43 for $x = DC$; by Claims 38, 42, 43 and 44 for $x = NOT$; and by Claims 35, 38, 42 and 43 for $x = CON$.

(3a) GHS-A is vacuously true in s by assumption that $nstatus(p) \neq$ sleeping in s'.

GHS-B: vacuously true for a CONNECT in $queue(\langle q,p\rangle)$ by Claim 35. By Claim 4 and CON-D, if CONNECT is in $queue(\langle p,q\rangle)$, then $minlink(f) = \langle p,q\rangle$. But by Claim 9 and COM-A, this cannot be. Thus the predicate is vacuously true for a CONNECT in $queue(\langle p,q\rangle)$.

GHS-D: Suppose $nstatus(p) =$ find in s'. By DC-I(a), p is up-to-date, and by GHS-I, $nlevel(p) = level(f)$.

GHS-E: Vacuously true by Claims 4, 21 and 41.

GHS-I: As argued in GHS-J, no node formerly in $nodes(g)$ is up-to-date in s. No change affects nodes formerly in $nodes(f)$.

GHS-J: Let r be any node in $nodes(f)$ in s'. If r is up-to-date, $r \notin testset(f)$, and $\langle r,t\rangle$ is the minimum-weight external link of r, then $nlevel(r) \leq nlevel(t)$ by GHS-J. By Claim 9, $fragment(t) \neq g$. Thus in s, $\langle r,t\rangle$ is still external. By DC-L, $inbranch(r)$ is in $subtree(g)$ (or nil) for all $r \in nodes(g)$ in s'. By Claim 21, $core(f) \in subtree(f)$ in s', and by Claim 41, $core(f)$ is unchanged in s. Thus following *inbranches* in s from any r formerly in $nodes(g)$ does not lead to $core(f)$, so no r formerly in $nodes(g)$ is up-to-date in s.

No change affects the rest.

vi) π **is ReceiveInitiate(⟨q,p⟩,l,c,st).** Let $f = fragment(p)$.

(3b)/(3c) *Case 1:* $st =$ find. $\mathcal{A}_{TAR}(s',\pi) = SendTest(p)$.

If there is a link $\langle p,r\rangle$ such that $lstatus(\langle p,r\rangle) =$ unknown in s', then $\mathcal{A}_{DC}(s',\pi) = ReceiveFind(\langle q,p\rangle)$; otherwise $\mathcal{A}_{DC}(s',\pi) = ReceiveFind(\langle q,p\rangle)\ t\ TestNode(p)$, where t is the state resulting from applying $ReceiveFind(\langle q,p\rangle)$ to $\mathcal{S}_{DC}(s')$.

$\mathcal{A}_{NOT}(s',\pi) = ReceiveNotify(\langle q,p\rangle,l,c)$.

$\mathcal{A}_{CON}(s',\pi)$ is empty.

Claims about s':

1. INITIATE(l, c,find) is at the head of $queue_p(\langle q,p\rangle)$, by precondition.

2. $(p,q) \in subtree(f)$, by Claim 1 and DC-D(a).
3. $minlink(f) = nil$, by Claims 1 and 2.
4. If $lstatus(\langle p,r\rangle) =$ rejected then $fragment(p) = fragment(r)$, for all r, by TAR-B.
5. If $lstatus(\langle p,r\rangle) =$ branch, then $(p,r) \in subtree(f)$, for all r, by Claim 3 and TAR-A(a).
6. If $(p,r) \in subtree(f)$, then $lstatus(\langle p,r\rangle) =$ branch for all r, by TAR-A(b).
7. If $|S| = 0$ and no $lstatus(\langle p,r\rangle)$ is unknown, then $p \neq mw\text{-}root(f)$, by definition of *mw-root* and Claims 4, 5 and 6.
8. $p \in testset(f)$, by Claims 1 and 2.
9. $dcstatus(p) =$ unfind, by Claim 1 and DC-D(b).
10. $testlink(p) = nil$, by Claim 9 and GHS-H.
11. $l = level(f)$, by Claims 1 and 2 and GHS-D.
12. $c = core(f)$, by Claims 1 and 11 and NOT-A.
13. No other FIND message is headed toward p, by Claims 1 and 2 and DC-S.
14. $core(f) \neq nil$, by Claim 2 and COM-F.

Let $(r,t) = core(f)$.

15. $(r,t) \in subtree(f)$, by Claim 14 and COM-F.

Let p be in $subtree(r)$.

16. If $(p,q) \neq (r,t)$ then $dcstatus(q) =$ find, by Claim 1 and DC-D(a).
17. If $(p,q) \neq (r,t)$ then $dcstatus(r) =$ find, by Claim 16 and DC-H(a).
18. If $(p,q) \neq (r,t)$ then either no CONNECT is in $queue(\langle r,t\rangle)$, or $lstatus(\langle t,r\rangle) =$ unknown, or a CONNECT is in $queue(\langle t,r\rangle)$, by Claim 17 and GHS-B(c).
19. If $(p,q) = (r,t)$ then either no CONNECT is in $queue(\langle r,t\rangle)$, or $lstatus(\langle t,r\rangle) =$ unknown, or a CONNECT is in $queue(\langle t,r\rangle)$, by Claim 1 and GHS-B(b).
20. Either no CONNECT is in $queue(\langle r,t\rangle)$, or $lstatus(\langle t,r\rangle) =$ unknown, or a CONNECT is in $queue(\langle t,r\rangle)$, by Claims 18 and 19.
21. $lstatus(\langle t,r\rangle) \neq$ unknown, by Claim 15 and TAR-A(b).
22. If CONNECT is in $queue(\langle t,r\rangle)$ then no CONNECT is in $queue(\langle r,t\rangle)$, by Claim 15.
23. If no CONNECT is in $queue(\langle t,r\rangle)$ then no CONNECT is in $queue(\langle r,t\rangle)$, by Claims 20, 21 and 22.
24. No CONNECT is in $queue(\langle r,t\rangle)$, by Claims 22 and 23.
25. If $(p,q) \neq (r,t)$ then $AfterMerge(p,q)$ is not enabled (for *DC* or *NOT*), since $(r,t) = core(f)$.
26. If $(p,q) = (r,t)$ then $AfterMerge(p,q)$ is not enabled (for *DC* or *NOT*), by Claim 24 and GHS-L.

27. If there is no unknown link of p, then there is no external link of p, by Claims 4 and 5.
28. If $(p, q) \neq (r,)$, then q is up-to-date, by Claim 16 and DC-I(a).

SendTest(p) is enabled in $\mathcal{S}_{TAR}(s')$ by Claims 8 and 10. *ReceiveFind*($\langle q, p\rangle$) is enabled in $\mathcal{S}_{DC}(s')$ by Claims 1, 25 and 26. *ReceiveNotify*($\langle q, p\rangle, l, c$) is enabled in $\mathcal{S}_{NOT}(s')$ by Claims 1, 25 and 26.

Claims about t: (only defined when there are no unknown links of p in s')

29. $p \in testset(f)$, by Claim 8.
30. There is no external link of p, by Claim 27.
31. $dcstatus(p) =$ find, by definition of t.

TestNode(p) is enabled in t by Claims 29, 30 and 31.

Claims about s:

32. $level(f) = l$, by Claim 11 and code.
33. $core(f) = c$, by Claim 12 and code.
34. No FIND message is headed toward p, by Claim 13 and code.
35. No CONNECT is in $queue(\langle t, r\rangle)$, by Claim 24 and code.
36. There is no unknown link of p (in s') if and only if $testlink(p) = nil$ (in s), by Claim 10 and code.
37. There is no unknown link of p (in s') if and only if $p \notin testset(f)$ (in s), by Claims 34, 35 and 36.
38. If $|S| > 0$ (in s') then a FIND message is in $subtree(f)$, by Claim 5 and code.
39. If $|S| = 0$ and there is no unknown link of p (in s'), then $p \neq mw\text{-}root(f)$ (in s), by Claim 7 and code.
40. If $|S| = 0$ and there is no unknown link of p (in s'), then either a REPORT message is headed toward $mw\text{-}root(f)$, or there is no external link of f (in s), by Claims 28 and 39 and code.
41. If there is an unknown link of p (in s'), then $nstatus(p) =$ find (in s), by code.
42. $minlink(f) = nil$, by Claims 38, 40 and 41.

The changes (or lack of changes) to the remaining derived variables are obvious.

The effects of *SendTest*(p) are mirrored in $\mathcal{S}_{TAR}(s)$ by Claims 11, 12, and 37 for the changes, and Claims 32, 33, 3 and 42 for the lack of changes. If there is an unknown link of p in s', then the effects of *ReceiveFind*($\langle q, p\rangle$) are mirrored in $\mathcal{S}_{DC}(s)$ by Claims 5, 6, 36 and 37 for changes, and Claims 3, 11, 12, 32, 33, 37 and

42 for lack of changes. If there is no unknown link of p in s', then the effects of *ReceiveFind*($\langle q,p\rangle$) followed by *TestNode*(p) are mirrored in $\mathcal{S}_{DC}(s)$ by Claims 5, 6, 36 and 37 for changes, and Claims 3, 11, 12, 32, 33 and 42 for lack of changes. The effects of *ReceiveNotify*($\langle q,p\rangle, l, c$) are mirrored in $\mathcal{S}_{NOT}(s)$ by Claims 3, 4 and 42. $\mathcal{S}_{CON}(s') = \mathcal{S}_{CON}(s)$ by Claims 3, 11, 12, 32, 33, and 42.

Case 2: $st \neq$ find.

$\mathcal{A}_{NOT}(s', \pi) = ReceiveNotify(\langle q,p\rangle, l, c)$. $\mathcal{A}_x(s', \pi)$ is empty for all other x.

Claims about s':

1. INITIATE(l, c,found) is at the head of $queue_p(\langle q,p\rangle)$, by precondition.
2. $(p,q) \in subtree(f)$, by Claim 1 and NOT-H(e).
3. $nlevel(p) < l$, by Claim 1 and NOT-H(a).
4. $nlevel(p) < level(f)$, by Claims 1, 2 and 3.
5. $p \neq minnode(f)$, by Claims 1 and 2 and NOT-I.
6. If $lstatus(\langle p,r\rangle) =$ branch, then $(p,r) \in subtree(f)$, for all $r \neq q$, by Claim 5 and TAR-A(a).
7. If $(p,r) \in subtree(f)$, then $lstatus(\langle p,r\rangle) =$ branch, for all $r \neq q$, by TAR-A(b).
8. p is not up-to-date, by Claim 4 and GHS-I.
9. $nstatus(p) \neq$ find, by Claim 8 and DC-I(a).
10. $(p,q) \neq core(f)$, by Claim 1 and GHS-E.
11. *AfterMerge*(p,q) for *NOT* is not enabled, by Claim 10.

By Claim 9, $dcstatus(p) =$ unfind in both s' and s, and thus $minlink(f)$ is unchanged. The changes, or lack of changes, to the remaining derived variables are obvious.

By Claims 1 and 11, *ReceiveNotify*($\langle q,p\rangle, l, c$) is enabled in $\mathcal{S}_{NOT}(s')$. Its effects are mirrored in $\mathcal{S}_{NOT}(s)$ by Claims 6 and 7.

It is easy to see that $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all other x.

(3a) GHS-A: By DC-D(a), $(p,q) \in subtree(f)$. So by GHS-A(a), $nstatus(p) \neq$ sleeping in s'. Since the same is true in s, the predicate is vacuously true.

GHS-B: Vacuously true for a CONNECT in $queue(\langle q,p\rangle)$ by GHS-B(a) and the fact that INITIATE is first in the queue. Vacuously true for a CONNECT in $queue(\langle p,q\rangle)$ by GHS-B(b) and the presence of INITIATE in $queue(\langle q,p\rangle)$. The only other situation to consider is the addition of an INITIATE message to $queue(\langle p,r\rangle)$, $r \neq q$, with $lstatus(\langle p,r\rangle) =$ branch. As shown in (b)/(c), $(p,r) \in subtree(f)$. By NOT-H(e), either $(p,q) = core(f)$ or p is a child of q, so $(p,r) \neq core(f)$. Thus by CON-E, no CONNECT is in $queue(\langle p,r\rangle)$, or in $queue(\langle r,p\rangle)$.

GHS-C: Adding a FIND message does not falsify the predicate. Suppose a TEST message is added to $queue(\langle p,r\rangle)$. Then in s', $st =$ find.

Case 1: $\langle p,r\rangle$ is an internal link of f. By TAR-A(b), $(p,r) \neq subtree(f)$. By COM-F, $(p,r) \neq core(f)$. By CON-E, no CONNECT is in $queue(\langle p,r\rangle)$.

Case 2: $\langle p,r\rangle$ is an external link of f. Since there is a FIND message in $subtree(f)$ in s', $minlink(f) = nil$. By CON-D, no CONNECT is in $queue(\langle p,r\rangle)$.

GHS-D: Since it is true for the INITIATE in $queue(\langle q,p\rangle)$ in s', it is true for any INITIATE added in s.

GHS-E: As shown in GHS-B, $(p,r) \neq core(f)$.

GHS-F: By NOT-H(a), $nlevel(p)$ increases, so the predicate is still true for any leftover TEST messages. The predicate is true by code for the TEST message added.

GHS-G: *Case 1:* An ACCEPT is in $queue(\langle p,r\rangle)$. By NOT-H(a), $nlevel(p)$ increases, so the predicate is still true.

Case 2: An ACCEPT is in $queue(\langle r,p\rangle)$. By TAR-D, $testlink(p) = \langle p,r\rangle$. By GHS-H, $nstatus(p) =$ find. But by Claim 9 (for both Case 1 and Case 2 of (3b)/(3c)), $nstatus(p) \neq$ find. So there is no ACCEPT in $queue(\langle r,p\rangle)$, and the predicate is vacuously true.

GHS-H is true by code.

GHS-I: *Case 1:* $st =$ find. By code $nlevel(p) = l$, and by Claim 32 in Case 1 of (3b)/(3c), $l = level(f)$.

Case 2: $st \neq$ found. By NOT-H(a), $nlevel(p) < l$. Thus $nlevel(p) < level(f)$, so by GHS-I, p is not up-to-date in s'. Since all *inbranches* remain the same in s and $nstatus(p) \neq$ find in s, p is still not up-to-date.

GHS-J: *Case 1:* $st =$ find. By Claim 37 in Case 1 of (3b)/(3c), $p \notin testset(f)$ in s if and only if there is no external link of p, so the predicate is vacuously true.

Case 2: $st \neq$ find. As in GHS-I, Case 2, p is not up-to-date, so the predicate is vacuously true.

vii) π is ReceiveTest($\langle$q,p$\rangle$,l,c). Let $f = fragment(p)$.

Case 1: $nstatus(p) =$ sleeping in s'.

(3b)/(3c) $\mathcal{A}_{TAR}(s', \pi) = ChangeRoot(f)\ t\ \pi$, where t is the same as $\mathcal{S}_{TAR}(s')$ except that $rootchanged(f) =$ true and $lstatus(minlink(f)) =$ branch in t.

$\mathcal{A}_x(s', \pi) = ChangeRoot(f)$ for all other x.

Claims about s':

1. TEST(l, c) is at the head of $queue_p(\langle q, p\rangle)$, by precondition.
2. $nstatus(p) =$ sleeping, by assumption.
3. $subtree(f) = \{p\}$, by Claim 2 and GHS-A.
4. $minlink(f) \neq nil$, by Claim 3 and definition.
5. $rootchanged(f) =$ false, by Claim 2, GHS-A(c) and TAR-H.
6. $level(f) = 0$, by Claim 3 and COM-F.
7. $nlevel(p) = 0$, by Claims 3 and 6.
8. $l \geq 1$, by TAR-M.
9. $l > nlevel(p)$, by Claims 7 and 8.
10. $l > level(f)$, by Claims 6 and 8.
11. $awake =$ true, by Claim 1 and GHS-A(b).

Claims about s:

12. The TEST message is requeued, by Claim 9.
13. $lstatus(minlink(f)) =$ branch, by code.
14. CONNECT(0) is in $queue(minlink(f))$, by code.
15. $minlink(f)$ does not change (i.e., is still external), by Claims 13 and 14.
16. $rootchanged(f) =$ true, by Claims 14 and 15.

$ChangeRoot(f)$ is enabled in $\mathcal{S}_x(s')$ by Claims 11, 3 and 5 for $x = CON$, and by Claims 11, 4 and 5 for all other x.

TAR: Effects of $ChangeRoot(f)$ are mirrored in t by its definition. π is enabled in t by definition. Its effects are mirrored in $\mathcal{S}_{TAR}(s)$ by Claim 12.

For all other x, the effects of $ChangeRoot(f)$ are mirrored in $\mathcal{S}_x(s)$ by Claim 16 for DC and NOT, and by Claim 14 for CON.

(3a) P_{GHS} is true in s by essentially the same argument as in $\pi = Start(p)$, Case 2.

Case 2: $nstatus(p) \neq$ sleeping in s'.

(3b)/(3c) $\mathcal{A}_{TAR}(s', \pi) = \pi$ if $l \leq nlevel(p)$ or $nlevel(p) = level(f)$ in s', and is empty otherwise.

$\mathcal{A}_{DC}(s', \pi) = TestNode(p)$ if $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p, q\rangle$ and $lstatus(\langle p, r\rangle) \neq$ unknown for all $r \neq q$, in s', and is empty otherwise.

$\mathcal{A}_x(s', \pi)$ is empty for all other x.

First we discuss what happens to $testset(f)$ and $minlink(f)$.

We show $testset(f)$ is unchanged, except that p is removed from $testset(f)$ if and only if $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p, q\rangle$, and there is no link $\langle p, r\rangle$, $r \neq q$, with $lstatus(\langle p, r\rangle) =$ unknown. If $testlink(p)$ does not change from non-nil to nil (or vice versa), then obviously $testset(f)$ is unchanged. The only place $testlink(p)$ is changed in this way is in procedure $Test(p)$, exactly if there are no more unknown links of p; $Test(p)$ is executed if and only if $l \leq nlevel(p)$, $c = nfrag(p)$, and $testlink(p) = \langle p, q\rangle$ in s'. Suppose $testlink(p)$ is changed from non-nil to nil. Since $testlink(p) \neq nil$ in s', GHS-M implies that no FIND message is headed toward p, and no CONNECT message is in $queue(\langle r, t\rangle)$, where $(r, t) = core(f)$ and $p \in subtree(r)$. Thus in s, since $testlink(p) = nil$, p is not in $testset(f)$.

Now we show that $minlink(f)$ does not change. If $dcstatus(p)$ does not change, and no REPORT message is added to any queue, then obviously $minlink(f)$ does not change. Suppose $dcstatus(p)$ changes, and a REPORT message is added to a queue (in procedure $Report(p)$). Then $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p, q\rangle$, there are no more unknown links of p (so $testlink(p)$ is set to nil), and $findcount(p) = 0$.

Claims about s'*:*

1. $testlink(p) = \langle p, q\rangle$, by assumption.

2. $nstatus(p) =$ find, by Claim 1 and GHS-H.
3. $minlink(f) = nil$, by Claim 2.
4. If $(p, r) = core(f)$, then a FIND message is in $queue(\langle p, r\rangle)$, or $dcstatus(r) =$ find, or a REPORT message is in $queue(\langle r, p\rangle)$, by Claim 2 and DC-J.
5. p is up-to-date, by Claim 2 and DC-I(a).

Claims about s:

6. If $p \neq mw\text{-}root(f)$, then either there is no external link of f, or a REPORT is headed toward $mw\text{-}root(f)$, by Claim 5 and code.
7. If $p = mw\text{-}root(f)$, then either a FIND is in $queue(\langle p, r\rangle)$, or $dcstatus(r) =$ find, or a REPORT is in $queue(\langle r, p\rangle)$, where $core(f) = (p, r)$, by Claim 4 and code.
8. $minlink(f) = nil$, by Claims 6 and 7.

Claims 3 and 8 give the result.

TAR: First, suppose $l > nlevel(p)$ and $nlevel(p) \neq level(f)$.

Claims about s′:

1. $l > nlevel(p)$, by assumption.
2. $nlevel(p) \neq level(f)$, by assumption.
3. p is not up-to-date, by Claim 2 and GHS-I.
4. $nstatus(p) \neq$ find, by Claim 3 and DC-I(a).
5. $testlink(p) = nil$, by Claim 4 and GHS-H.
6. There is no protocol message for $\langle p, q\rangle$, by Claim 5 and TAR-D.
7. The TEST message in $queue(\langle q, p\rangle)$ is a protocol message for $\langle q, p\rangle$, by Claim 6.
8. $testlink(q) = \langle q, p\rangle$, by Claim 7 and TAR-D.
9. There is exactly one protocol message for $\langle q, p\rangle$, by Claim 8 and TAR-C(c).
10. There is only one TEST message in $tarqueue(\langle q, p\rangle)$, by Claim 9.

By Claims 6 and 10, the TEST is the only *TAR* message in $tarqueue(\langle q, p\rangle)$. Since the TEST message is requeued in *GHS*, $tarqueue(\langle q, p\rangle)$ is unchanged. By earlier remarks about $testset(f)$ and $minlink(f)$, and by inspection, the other derived variables (for *TAR*) are unchanged. Thus, $\mathcal{S}_{TAR}(s') = \mathcal{S}_{TAR}(s)$,

Second, suppose $l > level(p)$ and $nlevel(p) = level(f)$. Then the TEST message is requeued in *GHS* and in *TAR*. By earlier remarks about $testlink(f)$ and $minlink(f)$, and by inspection, $\mathcal{S}_{TAR}(s')\pi\mathcal{S}_{TAR}(s)$ is an execution fragment of *TAR*.

Third, suppose $l \leq nlevel(p)$. Let $g = fragment(q)$.

Claims about s':

1. TEST(l, c) is at the head of $queue_p(\langle q, p\rangle)$, by precondition.
2. $l \leq nlevel(p)$, by assumption.
3. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $c = core(g)$ and $l = level(g)$, by Claim 1 and TAR-E(b).
4. If $lstatus(\langle q, p\rangle) =$ rejected, then $c = core(f)$ and $l = level(f)$, by Claim 1 and TAR-E(c).
5. $c \neq nil$, by Claim 1 and TAR-M.

Next we show that $c = core(f)$ if and only if $c = nfrag(p)$. First, suppose $c = core(f)$.

6. $c = core(f)$, by assumption.
7. If $lstatus(\langle q, p\rangle) =$ rejected, then $nlevel(p) = level(f)$, by Claims 2 and 4 and definition of $level(f)$.
8. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $core(g) = core(f)$, by Claims 3 and 6.
9. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $c \in subtree(g)$ and $c \in subtree(f)$, by Claims 5, 6 and 8 and COM-F.
10. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $f = g$, by Claim 9 and COM-G.
11. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $l = level(f)$, by Claims 3 and 10.
12. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $nlevel(p) = level(f)$, by Claims 2 and 11 and definition of $level(f)$.
13. $nlevel(p) = level(f)$, by Claims 8 and 12.
14. $nfrag(p) = core(f)$, by Claim 13 and NOT-A.
15. $nfrag(p) = c$, by Claims 6 and 14.

Now suppose $c = nfrag(p)$.

16. $c = nfrag(p)$, by assumption.
17. $c \in subtree(f)$, by Claims 5 and 16 and NOT-F.
18. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $c \in subtree(g)$, by Claims 5 and 3 and COM-F.
19. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $f = g$, by Claims 17 and 18 and COM-G.
20. If $lstatus(\langle q, p\rangle) \neq$ rejected, then $c = core(f)$, by Claims 3 and 19.
21. $c = core(f)$, by Claims 4 and 20.

π is enabled in $\mathcal{S}_{TAR}(s')$ by Claim 1. We now verify that the effects are mirrored in $\mathcal{S}_{TAR}(s)$. By the above argument, $c \neq frag(p)$ if and only if $c \neq core(f)$. Thus, the body of *ReceiveTest* for TAR is simulated correctly. Consider procedure $Test(p)$. If it is executed, then $c = nfrag(p)$ in s'. By Claim 21, $nfrag(p) = core(f)$, and by

NOT-E, $nlevel(p) = level(f)$. Thus the TEST messages sent in procedure $Test(p)$ in *GHS* correspond to those sent in *TAR*. By the discussion at the beginning of Case 2, $testset(f)$ is updated correctly, and $minlink(f)$ is unchanged. The changes or lack of changes to the other derived variables are obvious.

DC: First, suppose $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p, q \rangle$, and $lstatus(\langle p, r \rangle) \neq$ unknown for all $r \neq q$, in s'.

Claims about s':

1. TEST(l, c) is at the head of $queue_p(\langle q, p \rangle)$, by precondition.
2. $l \leq nlevel(p)$, by assumption.
3. $c = nfrag(p)$, by assumption.
4. $testlink(p) = \langle p, q \rangle$, by assumption.
5. $lstatus(\langle p, r \rangle) \neq$ unknown, for all $r \neq q$, by assumption.
6. $p \in testset(f)$, by Claim 4 and TAR-C(b).
7. $minlink(f) = nil$, by Claim 6 and GC-C.
8. If $lstatus(\langle p, r \rangle) =$ branch, then $(p, r) \in subtree(f)$, for all $r \neq q$, by Claim 7 and TAR-A(a).
9. If $lstatus(\langle p, q \rangle) =$ rejected, then $fragment(r) = f$, for all $r \neq q$, by TAR-B.
10. $c = core(f)$, by Claims 1, 2 and 3 and the argument just given for *TAR*.
11. $fragment(q) = f$, by Claims 1 and 10 and TAR-N.
12. There is no external link of p, by Claims 8, 9, 11 and 5.
13. $nstatus(p) =$ find, by Claim 4 and GHS-H.

$TestNode(p)$ is enabled in $\mathcal{S}_{DC}(s')$ by Claims 6, 12 and 13. Its effects are mirrored in $\mathcal{S}_{DC}(s)$ by the earlier discussion about $testset(f)$ and $minlink(f)$ and by Claim 12. (The disposition of the rest of the derived variables should be obvious.)

Now suppose $l > nlevel(p)$ or $c \neq nfrag(p)$ or $testlink(p) \neq \langle p, q \rangle$ or there is a link $\langle p, r \rangle$ with $lstatus(\langle p, r \rangle) =$ unknown and $r \neq q$. Then $\mathcal{S}_{DC}(s') = \mathcal{S}_{DC}(s)$ by inspection and earlier discussion of $testset(f)$ and $minlink(f)$.

NOT and *CON*: We want to show$\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for $x = NOT$ and CON. The only derived variables for these two that are not obviously unchanged are $minlink(f)$ and $rootchanged(f)$. (Because of the presence of the TEST message in $queue(\langle q, p \rangle$, GHS-A(b) implies that $awake =$ true in s', so changes to $nstatus(p)$ do not change $awake$.) Since we already showed $minlink(f)$ is unchanged, it is obvious that $rootchanged(f)$ is unchanged.

(3a) GHS-A is vacuously true by the assumption that $nstatus(p) \neq$ sleeping.

GHS-B: First we show that if the hypotheses of this predicate are false for a link in s', then they are still false in s. The only way they could go from false to true is by $lstatus(\langle p,q\rangle)$ going from unknown to rejected. But since TEST is in $queue(\langle q,p\rangle)$ in s', by GHS-C no CONNECT is in $queue(\langle q,p\rangle)$ in s', or in s.

Now we show that the state changes do not invalidate (a) through (d) for a link, assuming that the hypotheses are true for that link in s'.

Case A: TEST is requeued. No change affects the predicate.

Case B: ACCEPT or REJECT is added to $queue(\langle p,q\rangle)$. We already showed that no CONNECT is in $queue(\langle q,p\rangle)$. Because of the TEST in $queue(\langle q,p\rangle)$, the preconditions of the predicate are not true for a CONNECT in $queue(\langle p,q\rangle)$ in s'.

Case C: TEST is added to some $queue(\langle p,r\rangle)$. Since $lstatus(\langle p,r\rangle) =$ unknown, the preconditions are not true in s' for a CONNECT in $queue(\langle r,p\rangle)$. Since the TEST is added, $testlink(p) = \langle p,q\rangle$ in s'. By GHS-H, $nstatus(p) =$ find in s'. So by GHS-B(c), the preconditions are not true in s' for a CONNECT in $queue(\langle p,r\rangle)$.

Case D: REPORT is added to $queue(inbranch(p))$. Let $\langle p,r\rangle = inbranch(p)$ in s'. As in Case 3, the predicate is vacuously true for a CONNECT in $queue(\langle p,r\rangle)$. As in Case 3, $nstatus(p) =$ find in s', so p is up-to-date by DC-I(a). By GHS-I, $nlevel(p) = level(f)$. Since by DC-L, $(p,r) \in subtree(f)$, there cannot be an INITIATE$(nlevel(p)+1,*,*)$ message in $queue(\langle r,p\rangle)$. By GHS-B(a), the preconditions are not true for a CONNECT in $queue(\langle r,p\rangle)$.

GHS-H: By code.

GHS-J: If p is removed from $testset(f)$, then as in Claim 12 of (3b)/(3c) for DC, there is no external link of p.

GHS-C: *Case 1:* REJECT is added to $queue(\langle p,q\rangle)$. Then $l \leq nlevel(p)$, $c = nfrag(p)$, and $testlink(p) \neq \langle p,q\rangle)$ in s'. As argued in Lemma 17, verifying (3c) of Case 1 for $\pi = ReceiveTest$, $\langle p,q\rangle$ is an internal link of f. By TAR-E(a), $(p,q) \neq core(f)$, so by CON-E, no CONNECT is in $queue(\langle p,q\rangle)$.

Case 2: TEST is added to $queue(\langle p,r\rangle)$. Then in s', $l \leq nlevel(p)$, $c = nfrag(p)$, $testlink(p) = \langle p,q\rangle$, and $lstatus(\langle p,r\rangle) =$ unknown.

Case 2a: $\langle p,r\rangle$ is an internal link of f. By TAR-A(b), $(p,r) \notin subtree(f)$. By COM-F, $(p,r) \neq core(f)$. By CON-E, no CONNECT is in $queue(\langle p,r\rangle)$.

Case 2b: $\langle p,r\rangle$ is an external link of f. By GHS-H, $nstatus(p) =$ find. Thus $minlink(f) = nil$. By CON-D, no CONNECT is in $queue(\langle p,r\rangle)$.

GHS-G: Suppose ACCEPT is added to $queue(\langle p,q\rangle)$. Then $l \leq nlevel(p)$ in s'. As argued in Lemma 17, verifying TAR-F for $\pi = ReceiveTest$, $l = level(fragment(q))$. By GHS-F, $l \leq nlevel(q)$. So $l = nlevel(q)$.

No changes affect the rest.

viii) π is ReceiveAccept($\langle$q,p$\rangle$). Let $f = fragment(p)$.

(3b)/(3c) $\mathcal{A}_{TAR}(s',\pi) = \pi$. $\mathcal{A}_{DC}(s',\pi) = TestNode(p)$. $\mathcal{A}_x(s',\pi)$ is empty for all other x.

An argument similar to that used in $\pi = ReceiveTest(\langle q,p\rangle, l, c)$, Case 2, shows that $minlink(f)$ is unchanged.

TAR: *Claims about* s'*:*

1. ACCEPT is at the head of $queue_p(\langle q,p\rangle)$, by precondition.
2. There is a protocol message for $\langle p,q\rangle$, by Claim 1.
3. $testlink(p) = \langle p,q\rangle$, by Claim 2 and TAR-D.
4. No FIND message is headed toward p, by Claim 3 and GHS-M.
5. No CONNECT message is in $queue(\langle r,t\rangle)$, where $(r,t) = core(f)$ and $p \in subtree(r)$, by Claim 3 and GHS-M.

Claims about s:

6. $testlink(p) = nil$, by code.
7. No FIND message is headed toward p, by Claim 4.
8. No CONNECT message is in $queue(\langle r,t\rangle)$, where $(r,t) = core(f)$ and $p \in subtree(r)$, by Claim 5 and code.
9. $p \notin testset(f)$, by Claims 6, 7 and 8.

π is enabled in $\mathcal{S}_{TAR}(s')$ by Claim 1; its effects are mirrored in $\mathcal{S}_{TAR}(s)$ by Claims 6 and 9, and discussion of $minlink(f)$. (The disposition of the remaining derived variables should be obvious.)

DC: *More Claims about* s'*:*

10. $p \in testset(f)$, by Claim 3.
11. $minlink(f) = nil$, by Claim 10.
12. $fragment(q) \neq f$, by Claim 1 and TAR-F.
13. $level(f) \leq level(fragment(q))$, by Claim 1 and TAR-F.
14. $lstatus(\langle p, q\rangle) \neq$ branch, by Claims 11 and 12 and TAR-A(a).
15. $\langle p, q\rangle$ is the minimum-weight external link of p with *lstatus* unknown, by Claims 3 and 14 and TAR-C(d).
16. If $lstatus(\langle p, r\rangle) =$ rejected, then $\langle p, r\rangle$ is not external, for all r, by TAR-B.
17. If $lstatus(\langle p, r\rangle) =$ branch, then $\langle p, r\rangle$ is not external, for all r, by Claim 11 and TAR-A(a).
18. If $\langle p, r\rangle$ is external, then $lstatus(\langle p, r\rangle) =$ unknown, for all r, by Claims 16 and 17.
19. $\langle p, q\rangle$ is the minimum-weight external link of p, by Claims 15 and 18.
20. $nstatus(p) =$ find, by Claim 3 and GHS-H.

$TestNode(p)$ is enabled in $\mathcal{S}_{DC}(s')$ by Claims 10, 19 and 13, and 20. Its effects are mirrored in $\mathcal{S}_{DC}(s)$ by Claims 9, 19 and 6.

NOT and *CON*: It is easy to verify that $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for $x = NOT$ and *CON*.

(3a) GHS-A: By Claim 20, vacuously true in s.

GHS-B: Suppose a REPORT message is added to $queue(\langle p, r\rangle)$ in s. Let $\langle p, r\rangle = inbranch(p)$. By Claim 20 and DC-I(a), p is up-to-date in s'. By GHS-I, $nlevel(p) = level(f)$. By DC-L, $(p, r) \in subtree(f)$, so no INITIATE$(nlevel(p) + 1, *, *)$ can be in $queue(\langle p, r\rangle)$ or $queue(\langle r, p\rangle)$. By GHS-B(a), the preconditions for a CONNECT in $queue(\langle p, r\rangle)$ or $queue(\langle r, p\rangle)$ are not true in s', or in s.

GHS-H: By code, $testlink(p) = nil$.

GHS-J: By Claim 19 and GHS-G.

No changes affect the rest.

ix) π is ReceiveReject($\langle$q,p$\rangle$). Let $f = fragment(p)$.

(3b)/(3c) $\mathcal{A}_{TAR}(s', \pi) = \pi$.

$\mathcal{A}_{DC}(s', \pi) = TestNode(p)$ if there is no $r \neq q$ such that $lstatus(\langle p, r\rangle) =$ unknown in s, and is empty otherwise.

$\mathcal{A}_x(s', \pi)$ is empty for all other x.

An argument similar to that in $\pi = ReceiveTest(\langle q, p\rangle, l, c)$, Case 2, shows that $minlink(f)$ is unchanged.

TAR: *Claims about* s'*:*

1. REJECT is at the head of $queue_p(\langle q, p\rangle)$, by precondition.
2. There is a protocol message for $\langle p, q\rangle$, by Claim 1.
3. $testlink(p) = \langle p, q\rangle$, by Claim 2 and TAR-D.
4. No FIND message is headed toward p, by Claim 3 and GHS-M.
5. No CONNECT message is in $queue(\langle r, t\rangle)$, where $(r, t) = core(f)$ and $p \in subtree(r)$, by Claim 3 and GHS-M.
6. $nstatus(p) =$ find, by Claim 3 and GHS-H.
7. $nlevel(p) = level(f)$, by Claim 6, DC-I(a) and GHS-I.
8. $nfrag(p) = core(f)$, by Claim 7 and NOT-A.

Claims about s*:*

9. If there is no link $\langle p, r\rangle$ with $lstatus(\langle p, r\rangle) =$ unknown (in s'), then $testlink(p) = nil$ (in s), by code.
10. No FIND message is headed toward p, by Claim 4.
11. No CONNECT message is in $queue(\langle r, t\rangle)$, by Claim 5.
12. If there is no link $\langle p, r\rangle$ with $lstatus(\langle p, r\rangle) =$ unknown (in s'), then $p \notin testset(f)$ (in s), by Claims 9, 10 and 11.

π is enabled in $\mathcal{S}_{TAR}(s')$ by Claim 1. Its effects are mirrored in $\mathcal{S}_{TAR}(s)$ by Claims 9, 12, 7 and 8, and earlier discussion of $minlink(f)$.

DC: If there is a link $\langle p, r\rangle$ such that $lstatus(\langle p, r\rangle) =$ unknown and $r \neq q$, then it is easy to check that $\mathcal{S}_{DC}(s') = \mathcal{S}_{DC}(s)$. Suppose there is no unknown link (other than $\langle p, q\rangle$).

More claims about s'*:*

13. $lstatus(\langle p, r\rangle) \neq$ unknown, for all $r \neq q$, by assumption.
14. $minlink(f) = nil$, by Claim 6.
15. If $lstatus(\langle p, r\rangle) =$ branch, then $(p, r) \in subtree(f)$, for all $r \neq q$, by Claim 14 and TAR-A(a).

16. If $lstatus(\langle p, r\rangle)$ = rejected, then $fragment(r) = f$, for all $r \neq q$, by TAR-B.
17. $fragment(q) = f$, by Claim 1 and TAR-G.
18. There are no external links of p, by Claims 13, 15, 16 and 17.
19. $p \in testset(f)$, by Claim 3 and TAR-C(b).

$TestNode(p)$ is enabled in $\mathcal{S}_{DC}(s')$ by Claims 19, 18 and 6. Its effects are mirrored in $\mathcal{S}_{DC}(s)$ by Claims 9 and 12.

NOT and *CON*: It is easy to show that $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for $x = NOT$ and *CON*.

(3a) GHS-A: Vacuously true by Claim 6.

GHS-B: Either a TEST or a REPORT message is added. The argument is very similar to that in $\pi = ReceiveTest(\langle q, p\rangle, l, c)$, Case 2 of (a).

GHS-C: Only affected if a TEST is added. The argument is very similar to that in $\pi = ReceiveTest(\langle q, p\rangle, l, c)$, Case 2 of (a).

GHS-H: The argument is very similar to that in $\pi = ReceiveTest(\langle q, p\rangle, l, c)$, Case 2 of (a).

GHS-I: Suppose p is removed from $testset(f)$. By Claim 12, this only happens when there are no more unknown links. By Claim 18, p has no external links if there are no more unknown links.

No changes affect the rest.

x) π is ReceiveReport($\langle$q,p$\rangle$,w). Let $f = fragment(p)$.

(3b)/(3c) *Case 1:* $(p, q) = core(f)$, $nstatus(p) \neq$ find and $w > bestwt(p)$ in s'. This case is divided into two subcases; first we prove some claims true in both subcases. Let $\langle r, t\rangle$ be the minimum-weight external link of f in s'. (Below, we show it exists.)

Claims about s':

1. REPORT(w) is at the head of $queue(\langle q, p\rangle)$, by assumption.
2. $(p, q) = core(f)$, by assumption.
3. $nstatus(p) \neq$ find, by assumption.

4. $w > bestwt(p)$, by assumption.
5. $ReceiveReport(\langle q,p\rangle, w)$ is enabled in $\mathcal{S}_{DC}(s')$, by Claim 1.
6. $ComputeMin(f)$ (for GC) is enabled in $\mathcal{S}_4(\mathcal{S}_{DC}(s'))$, by Claims 2, 3, 4 and 5 and argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = ReceiveReport$.
7. $minlink(f) = nil$, by Claim 6.
8. $accmin(f) \neq nil$, by Claim 6.
9. $testset(f) = \emptyset$, by Claim 6.
10. $ComputeMin(f)$ (for COM) is enabled in $\mathcal{S}_2(\mathcal{S}_4(\mathcal{S}_{DC}(s')))$, by Claim 6 and argument in proof of Lemma 15, verifying (3c) for $\pi = ComputeMin$.
11. $level(f) \leq level(fragment(t))$, by Claim 10.
12. $accmin(f) = \langle r,t\rangle$, by Claims 8 and 9 and GC-A.
13. r is up-to-date, by Claim 9, DC-N, and choice of $\langle r,t\rangle$.
14. $nlevel(r) = level(f)$, by Claim 13 and GHS-I.
15. $nlevel(f) \leq nlevel(t)$, by Claims 9 and 13 and GHS-J.
16. No CONNECT message is in either queue of $core(f)$, by Claim 9.
17. No CONNECT message is in any internal queue of f, by Claim 16 and CON-E.
18. $inbranch(p) = \langle p,q\rangle$, by Claims 1 and 2 and DC-A(a).
19. p is up-to-date, by Claims 2, 9 and 18.
20. $findcount(p) = 0$, by Claim 3 and DC-H(b).
21. All children of p are completed, by Claims 19 and 20 and DC-K(a).
22. $r \in subtree(p)$, by Claims 1, 2, 3 and 4 and DC-P(b).
23. Following *bestlinks* from p leads along edges of $subtree(f)$ to $\langle r,t\rangle$, by Claims 9, 19, 21 and 22, choice of $\langle r,t\rangle$, and DC-K(b) and (c).

The following remarks apply to both Subcase 1a and Subcase 1b: $ComputeMin(f)$ is enabled in $\mathcal{S}_x(s')$ by Claims 7, 8 and 9 for $x = TAR$; by Claims 7, 14 and 15 (and definition of $\langle r,t\rangle$) for $x = NOT$; and by Claims 7, 11 and 17 for $x = CON$. π is obviously enabled in $\mathcal{S}_{DC}(s')$.

Subcase 1a: $lstatus(bestlink(p)) =$ branch. $\mathcal{A}_{DC}(s',\pi) = \pi$. $\mathcal{A}_x(s',\pi) = ComputeMin(f)$ for all other x.

More Claims about s':

24. $lstatus(bestlink(p)) =$ branch, by assumption.
25. $bestlink(p) \in subtree(f)$, by Claims 7 and 24 and TAR-A(a).
26. $p \neq r = mw\text{-}minnode(f)$, by Claims 23 and 25.

Claims about s:

27. The effects of π are reflected in $\mathcal{S}_{DC}(s)$, by code.
28. The effects of $ComputeMin(f)$ are reflected in $\mathcal{S}_4(\mathcal{S}_{DC}(s))$, by Claim 27 and argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = ReceiveReport$.
29. $minlink(f) = \langle r, t\rangle$, by Claims 28 and 12.
30. Following *bestlinks* from p leads to $\langle r, t\rangle$, by Claim 23.
31. $tominlink(p) = bestlink(p)$, by Claims 30 and 24.
32. $p \neq minnode(f)$, by Claims 29 and 24.
33. $p = root(f)$, by Claims 2, 22 and 29.

By Claims 3, 4 and 17, procedure $ChangeRoot(p)$ is executed in GHS. The effects of $ComputeMin(f)$ are reflected in $\mathcal{S}_x(s)$ by Claims 29 and 12 for $x = TAR$; by Claim 29 and choice of $\langle r, t\rangle$ for $x = NOT$; and by Claims 29, 31, 32, 33 and choice of $\langle r, t\rangle$ for $x = CON$. The effects of π are reflected in $\mathcal{S}_{DC}(s)$ by Claim 27.

Subcase 1b: $lstatus(bestlink(p)) \neq$ branch.

$\mathcal{A}_{DC}(s', \pi) = \pi\ t_{DC}\ ChangeRoot(f)$, where t_{DC} is the result of applying π to $\mathcal{S}_{DC}(s')$.

$\mathcal{A}_{CON}(s', \pi) = ComputeMin(f)$.

For all other x, $\mathcal{A}_x(s', \pi) = ComputeMin(f)\ t_x\ ChangeRoot(f)$, where t_x is the result of applying $ComputeMin(f)$ to $\mathcal{S}_x(s')$.

More claims about s':

34. $lstatus(bestlink(p)) \neq$ branch.
35. $bestlink(p) = \langle r, t\rangle$, by Claims 23, 34 and 7 and TAR-A(b).
36. $p = r = mw\text{-}minnode(f)$, by Claim 35.
37. $nstatus(q) \neq$ sleeping, by Claim 1 and GHS-A.
38. $awake =$ true, by Claim 37.
39. $rootchanged(f) =$ false, by Claim 7 and COM-B.

Claims about t_x, $x \neq CON$:

40. If $x = TAR$, then $minlink(f) = \langle r, t\rangle$, by Claim 12.
41. If $x = NOT$, then $minlink(f) = \langle r, t\rangle$, by choice of $\langle r, t\rangle$.
42. If $x = DC$, then $minlink(f) = \langle r, t\rangle$, by Claims 6 and 12 and argument in proof of Lemma 15, verifying (3c) for $\pi = ComputeMin$.

43. $awake$ = true, by Claim 38.
44. $rootchanged(f)$ = false, by Claim 39.

The effects of π are mirrored in t_{DC} and of $ComputeMin(f)$ in t_{TAR} and t_{NOT} by definition. $ChangeRoot(f)$ is enabled in t_x by Claims 40, 43 and 44 for $x = TAR$; by Claims 41, 43 and 44 for $x = NOT$; and by Claims 42, 43 and 44 for $x = DC$.

Claims about s:

45. $minlink(f) = \langle r, t\rangle$, by argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = ReceiveReport$.
46. $lstatus(bestlink(p))$ = branch, by code.
47. $lstatus(minlink(p))$ = branch, by Claims 35 and 45.
48. CONNECT is added to $queue(bestlink(p))$, by code.
49. $rootchanged(f)$ = true, by Claims 45 and 48.

The effects of $ChangeRoot(f)$ are mirrored in $\mathcal{S}_x(s)$ by Claims 47 and 49 for $x = TAR$; by Claim 49 for $x = DC$ and NOT. The effects of $ComputeMin(f)$ are mirrored in $\mathcal{S}_{CON}(s)$ by Claims 36, 14 and 45.

Case 2: $(p, q) \neq core(f)$ or $nstatus(p)$ = find or $w \leq bestwt(p)$ in s'.

$\mathcal{A}_{DC}(s', \pi) = \pi$. $\mathcal{A}_x(s', \pi)$ is empty for all other x.

Subcase 2a: $(p, q) \neq core(f)$ in s'. Suppose $\langle p, q\rangle = inbranch(p)$ in s'. By DC-B(b), $dcstatus(p)$ = unfind. Thus, the only effect is to remove the REPORT message. Thus $\mathcal{S}_{DC}(s')\pi\mathcal{S}_{DC}(s)$ is an execution fragment of DC. As proved in Lemma 19, Case 2a of verifying (3b) for $\pi = ReceiveReport$, $minlink(f)$ is unchanged. Thus $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all $x \neq DC$.

Now suppose $\langle p, q\rangle \neq inbranch(p)$.

Claims about s':

1. REPORT is at head of $queue(\langle q, p\rangle)$, by precondition.
2. $(p, q) \neq core(f)$, by assumption.
3. $\langle p, q\rangle \neq inbranch(p)$, by assumption.
4. $dcstatus(p)$ = find, by Claims 1, 2 and 3 and DC-A(g).
5. p is up-to-date, by Claim 4 and DC-I(a).
6. q is a child of p, by Claims 3 and 5.

7. $findcount(p) > 0$, by Claims 1, 5 and 6 and DC-K(a).
8. No FIND message is headed toward p, by Claim 7 and GHS-M.
9. No CONNECT is in $queue(\langle r,t\rangle)$, where $(r,t) = core(f)$ and $p \in subtree(r)$, by Claim 7 and GHS-M.
10. $p \in testset(f)$ if and only if $testlink(p) \neq nil$, by Claims 8 and 9.

Obviously, π is enabled in $\mathcal{S}_{DC}(s')$. By Claim 10 and inspection, the effects of π are mirrored in $\mathcal{S}_{DC}(s)$. Since the proof of Lemma 19, Case 2a of verifying (3b) for $\pi = ReceiveReport$, shows $minlink(f)$ is unchanged, $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all $x \neq DC$.

Subcase 2b: $(p,q) = core(f)$ and $nstatus(p) =$ find in s'. Since REPORT(w) is at the head of $queue(\langle q,p\rangle)$, DC-A(a) implies that $inbranch(p) = \langle p,q\rangle$. Thus, the only change is that the REPORT message is requeued. Obviously $\mathcal{S}_{DC}(s')\pi\mathcal{S}_{DC}(s)$ is an execution fragment of DC, and $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all $x \neq DC$.

Subcase 2c: $(p,q) = core(f)$, $nstatus(p) =$ find and $w \leq bestwt(p)$ in s'. As in Subcase 2b, $inbranch(p) = \langle p,q\rangle$. The only change is that the REPORT message is removed. Thus $\mathcal{S}_{DC}(s')\pi\mathcal{S}_{DC}(s)$ is an execution fragment of DC. As proved in Lemma 19, Case 2c of verifying (3b) for $\pi = ReceiveReport$, $minlink(f)$ is unchanged in s. Thus $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all $x \neq DC$.

(3a) *Case 1:* $inbranch(p) \neq \langle p,q\rangle$.

GHS-A: By DC-A(a), $(p,q) \neq core(f)$. By DC-A(g), $dcstatus(p) =$ find. The predicate is vacuously true.

GHS-B: Only the addition of a REPORT message affects this predicate. The argument is very similar to that in $\pi = ReceiveTest(\langle q,p\rangle, l, c)$, Case 2, of (3a).

GHS-H: By code (in procedure $Report(p)$).

No change affects the rest.

Case 2: $inbranch(p) = \langle p,q\rangle$. If $nstatus(p) =$ find or $w \leq bestwt(p)$, then no change affects any predicate. Suppose $nstatus(p) \neq$ find and $w > bestwt(p)$.

GHS-A: By DC-B(a), $subtree(p) \neq \{p\}$. By GHS-A(a), $nstatus(p) \neq$ sleeping, so the predicate is vacuously true.

GHS-B: Let $\langle p, r\rangle = bestlink(p)$ in s'. If $lstatus(\langle p, r\rangle) =$ branch, then no change affects this predicate. Suppose $lstatus(\langle p, r\rangle) \neq$ branch. As shown in (3b)/(3c), Claim 35 of Case 1b, $bestlink(p)$ is the minimum-weight external link of f. Thus $lstatus(\langle r, p\rangle) \neq$ rejected by TAR-B, and if $lstatus(\langle r, p\rangle) =$ branch, then there is a CONNECT in $queue(\langle r, p\rangle)$. So the predicate is vacuously true for the CONNECT added to $queue(\langle p, r\rangle)$. If there is a leftover CONNECT in $queue(\langle r, p\rangle)$, then the predicate is vacuously true because of the new CONNECT in $queue(\langle p, r\rangle)$.

GHS-C: Let $\langle p, r\rangle = bestlink(p)$ in s'. Since $bestlink(p)$ is external (as shown in (3b)/(3c)), no REJECT is in $queue(\langle p, r\rangle)$ by TAR-G. Also since it is external, $lstatus(\langle p, r\rangle) \neq$ rejected by TAR-B. Suppose a TEST is in $queue(\langle p, r\rangle)$. By TAR-D, $testlink(p) = \langle p, r\rangle$, and by GHS-H, $nstatus(p) =$ find, which contradicts the assumption for this case. Also since the link is external, no FIND is in $queue(\langle p, r\rangle)$ by DC-D(a).

No change affects the rest.

xi) π is ReceiveChangeRoot($\langle$q,p$\rangle$).

(3b)/(3c) There are two cases. First we prove some facts true in both cases.

Claims about s':

1. CHANGEROOT is at the head of $queue(\langle q, p\rangle)$, by precondition.
2. $minlink(f) \neq nil$, by Claim 1 and CON-C.
3. $rootchanged(f) =$ false, by Claim 1 and CON-C.
4. $p \in subtree(q)$, by Claim 1 and CON-C.
5. $minnode(f) \in subtree(p)$, by Claim 1 and CON-C.
6. $nlevel(minnode(f)) = level(f)$, by NOT-D.
7. $testset(f) = \emptyset$, by Claim 2 and GC-C
8. $minlink(f)$ is the minimum-weight external link of f, by Claim 2 and COM-A.
9. $minnode(f)$ is up-to-date, by Claims 7 and 8 and DC-N.
10. p is up-to-date, by Claims 5, 7 and 9.
11. No REPORT message is headed toward $mw\text{-}root(f)$, by Claim 2.
12. No REPORT message is headed toward p, by Claims 4 and 11.
13. $dcstatus(p) =$ unfind, by Claims 7 and 12 and DC-I(b).
14. $findcount(p) = 0$, by Claim 13 and DC-H(b).

15. All children of p are completed, by Claims 10 and 14 and DC-K(a).
16. Following *bestlinks* from p leads along edges in *subtree*(f) to the minimum-weight external link of *subtree*(p), by Claims 7, 10 and 15 and DC-K(b) and (c).

Case 1: *lstatus*(*bestlink*$(p)) \neq$ branch in s'.

$\mathcal{A}_{CON}(s', \pi) = \pi$. $\mathcal{A}_x(s', \pi) =$ *ChangeRoot*(f) for all other x.

More claims about s':

17. *lstatus*(*bestlink*$(p)) \neq$ branch, by assumption.
18. *bestlink*(p) is not in *subtree*(f), by Claim 17 and TAR-A(b).
19. *bestlink*$(p) =$ *minlink*(f), by Claims 5, 8, 16 and 18.
20. *nstatus*$(q) \neq$ sleeping, by Claim 1 and GHS-A(b).
21. *awake* = true, by Claim 20.

Claims about s:

22. *lstatus*(*bestlink*$(p))$ = branch, by code.
23. CONNECT is in *queue*(*bestlink*(p)), by code.
24. *MSF* does not change, Claims 22 and 23.
25. *bestlink*$(p) =$ *minlink*(f), by Claims 19 and 24.
26. *rootchanged*(f) = true, by Claims 23 and 25.

ChangeRoot(f) is enabled in $\mathcal{S}_x(s')$ by Claims 2, 3 and 21, for all $x \neq CON$. The effects of *ChangeRoot*(f) are mirrored in $\mathcal{S}_x(s)$ by Claims 22, 25 and 26 for $x = TAR$; and by Claim 26 for $x = DC$ and NOT. π is enabled in $\mathcal{S}_{CON}(s')$ by Claim 1; its effects are mirrored in $\mathcal{S}_{CON}(s)$ by Claims 6 and 19.

Case 2: *lstatus*(*bestlink*$(p))$ = branch in s'.

$\mathcal{A}_{CON}(s', \pi) = \pi$. $\mathcal{A}_x(s', \pi)$ is empty for all other x.

More Claims about s':

27. *lstatus*(*bestlink*$(p))$ = branch, by assumption.
28. *lstatus*(*minlink*$(f)) \neq$ branch, by Claim 3 and TAR-H.
29. *bestlink*(p) is in *subtree*(f), by Claims 27 and 28 and TAR-A(a).

30. $p \neq minnode(f)$, by Claims 16 and 29.
31. $bestlink(p) = tominlink(f)$, by Claims 8, 16 and 29.
32. $nlevel(p) = level(f)$, by Claim 10 and GHS-I.

Obviously, all derived (and non-derived) variables are unchanged, except *cqueues*. Thus, $\mathcal{S}_x(s') = \mathcal{S}_x(s)$ for all $x \neq CON$. π is enabled in $\mathcal{S}_{CON}(s')$ by Claim 1; its effects are mirrored in $\mathcal{S}_x(s)$ by Claims 30, 31 and 32.

(3a) GHS-A: By CON-C, $(p, q) \in subtree(f)$. By GHS-A(a), $nstatus(p) \neq$ sleeping in s', so the predicate is vacuously true in s.

GHS-B: Essentially the same argument as in $\pi = ReceiveReport(\langle q, p\rangle, w)$, Case 2 of (3a).

GHS-C: Essentially the same argument as in $\pi = ReceiveReport(\langle q, p\rangle, w)$, Case 2 of (3a).

No change affects the rest. □

Let $P'_{GHS} = \bigwedge_{x \in I}(P'_x \circ \mathcal{S}_x) \wedge P_{GHS}$.

Corollary 26: *P'_{GHS} is true in every reachable state of GHS.*

Proof: By Lemmas 1 and 25. □

4.3 Liveness

We show a path in the lattice along which liveness properties are preserved. The path is *HI*, *COM*, *GC*, *TAR*, *GHS*. In showing the edge from *GHS* to *TAR*, it is useful to know some liveness relationships between *GC* and *DC*, and between *COM* and *CON*.

The reason for considering liveness relationships in other parts of the lattice is to take advantage of the more abstract forms of the algorithm. For instance, the essence of showing that the *GHS* algorithm will take steps leading to the simulation of *ComputeMin*(f) in *TAR* is the same as showing that *DC* takes steps leading to the simulation of *ComputeMin*(f) in *GC*. (These steps are the convergecast of REPORT messages back to the core.) *DC* is not cluttered with variables and actions that are not relevant to this argument, unlike *GHS*. Thus, we make the argument for *DC* to *GC*, and then apply Lemma 7 for the *GHS* to *TAR* situation.

For the same reason, we show that the progression of CHANGEROOT messages in *CON* leads to the simulation of *ChangeRoot*(f) in *COM*, and that the movement of CONNECT messages over links in *CON* leads to *Absorb* and *Merge* in *COM*, and then apply Lemma 7.

4.3.1 COM is Equitable for HI

The main idea here is to show that as long as there exist two distinct subgraphs, progress is made; the heart of the argument is showing that some fragment at the lowest level can always take a step. This requires a global argument that considers all the fragments.

Lemma 27: *COM is equitable for HI via* $\mathcal{M}_1$.

Proof: By Corollary 14, $(P_{HI} \circ \mathcal{S}_1) \wedge P_{COM}$ is true in every reachable state of P_{COM}. Thus, in the sequel we will use the HI and COM predicates.

For each locally-controlled action φ of *HI*, we must show that *COM* is equitable for φ via $\mathcal{M}_1$.

i) φ **is Start(p) or NotInTree(l).** Since φ is enabled in $\mathcal{S}_1(s)$ if and only if it is also enabled in s, and since $\mathcal{A}_1(s, \varphi)$ includes φ, for any state s, Lemma 5 shows that *COM* is equitable for φ via $\mathcal{M}_1$.

ii) φ **is Combine(F,F',e).** We show *COM* is progressive for φ via $\mathcal{M}_1$; Lemma 6 implies *COM* is equitable for φ via $\mathcal{M}_1$.

Section 4.3.1: *COM* is Equitable for *HI*

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of *COM* and internal actions ψ of *COM* enabled in s. For reachable state s, let $v_\varphi(s) = (x, y, z)$, where x is the number of fragments in s, y is the number of fragments f with $rootchanged(f)$ = false in s, and z is the number of fragments f with $minlink(f) = nil$ in s. (Two triples are compared lexicographically.)

(1) Let s be a reachable state of *COM* in E_φ. We now demonstrate that some action ψ is enabled in s with $(s, \psi) \in \Psi_\varphi$.

Claims:

1. $awake$ = true in $\mathcal{S}_1(s)$, by precondition.
2. $F \neq F'$ in $\mathcal{S}_1(s)$, by precondition.
3. $awake$ = true in s, by Claim 1 and definition of $\mathcal{S}_1$.
4. There exist f and g in $fragments$ such that $subtree(f) = F$ and $subtree(g) = F'$ in s, by Claim 2 and definition of $\mathcal{S}_1$.
5. $f \neq g$ in s, by Claims 2 and 4.

Let $l = \min\{level(f') : f' \in fragments\}$ in s. (By Claim 4, $fragments$ is not empty in s, so l is defined.) Let $L = \{f' \in fragments : level(f') = l\}$.

Case 1: There exists $f' \in L$ with $minlink(f') = nil$. Let $\psi = ComputeMin(f')$. We now show ψ is enabled in s. By Claim 5, the minimum-weight external link $\langle p, q\rangle$ of f' exists. By choice of l, $level(f') \leq level(fragment(q))$. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2: For all $f' \in L$, $minlink(f') \neq nil$.

Case 2.1: There exists $f' \in L$ with $rootchanged(f')$ = false. Let $\psi = ChangeRoot(f')$. ψ is enabled in s by Claim 3 and the assumption for Case 2. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2.2: For all $f' \in L$, $rootchanged(f')$ = true.

Case 2.2.1: There exists fragment $g' \in L$ with $level(f') > l$, where $f' = fragment(target(minlink(g')))$. (By COM-G, f' is uniquely defined.) Let $\psi = Absorb(f', g')$. Obviously ψ is enabled in s, and $(s, \psi) \in \Psi_\varphi$.

Case 2.2.2: There is no fragment $g' \in L$ such that $level(f') > l$, where $f' = fragment(target(minlink(g')))$. Pick any fragment f_1 such that $level(f_1) = l$. For $i > 1$, define f_i to be $fragment(target(minlink(f_{i-1})))$.

More claims about s':

6. f_i is uniquely defined, for all $i \geq 1$. *Proof:* If $i = 1$, by definition. Suppose it is true for $i - 1 \geq 1$. Then it is true for i by COM-G, since $minlink(f_i)$ is well-defined and non-nil.
7. $minlink(f_i)$ is the minimum-weight external link of f_i, for all $i \geq 1$, by COM-A.
8. $f_i \neq f_{i-1}$, for all $i > 1$, by Claims 6 and 7 and definition of f_i.
9. If $minedge(f_i) \neq minedge(f_{i-1})$ for some $i > 1$, then f_{i+1} is not among $f_1, \ldots, f_i$, by Claims 7 and 8, and since the edge-weights are totally ordered.
10. There are only a finite number of fragments, by COM-D and the fact that $V(G)$ is finite.

By Claims 9 and 10, there is an $i > 1$ such that $minedge(f_i) = minedge(f_{i-1})$. Let $\psi = Merge(f_i, f_{i-1})$. Obviously ψ is enabled in s, and $(s, \psi) \in \Psi_\varphi$.

---

(2) Consider a step (s', π, s) of COM, where s' is reachable and in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) $v_\varphi(s) \leq v_\varphi(s')$, because there is no action of COM that increases the number of fragments; only a *Merge* action increases the number of fragments with *minlink* equal to *nil* or *rootchanged* equal to false, and it simultaneously causes the number of fragments to decrease.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. Then $v_\varphi(s) < v_\varphi(s')$, since *Absorb* and *Merge* decrease the number of fragments, *ComputeMin* maintains the number of fragments and the number of fragments with *rootchanged* = false and decreases the number with *minlink* = *nil*, and *ChangeRoot* maintains the number of fragments and decreases the number with *rootchanged* = false.

(c) Suppose $(s', \pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s', \psi) \in \Psi_\varphi$. Then ψ is still enabled in s, since the only possible values of π are *Start*(p), *InTree*(l) and *NotInTree*(l), none of which disables ψ. By definition, $(s, \psi) \in \Psi_\varphi$.

iii) φ is InTree($\langle$p,q$\rangle$). We show COM is progressive for φ via $\mathcal{M}_1$; Lemma 6 implies that COM is equitable for φ via $\mathcal{M}_1$.

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of COM and actions ψ of COM enabled in s such that ψ is either an internal action or is φ.

For reachable state s, let $v_\varphi(s) = v_{Combine(F,F',e)}(s)$.

(1) Let s be a reachable state of COM in E_φ. We now demonstrate that some action ψ is enabled in s with $(s, \psi) \in \Psi_\varphi$.

If $(p, q) \in F$ for some F in $\mathcal{S}_1(s)$, then $(p, q) \in subtree(fragment(p))$ in s. Let $\psi = InTree(\langle p, q \rangle)$.

Suppose $\langle p, q \rangle$ is the minimum-weight external link of some F in $\mathcal{S}_1(s)$. Then there is more than one fragment. Essentially the same argument as in $\varphi = Combine(F, F', e)$ shows that some $Absorb(f', g')$, or $Merge(f_i, f_{i+1})$, or $ChangeRoot(f')$, or $ComputeMin(f')$ is enabled in s.

(2) As in $\varphi = Combine(F, F', e)$, after noting that $\pi \neq InTree(\langle p, q \rangle)$. □

4.3.2 GC is Equitable for COM

The main part of the proof is showing that eventually every node is removed from $testset(f)$, so that eventually $ComputeMin(f)$ can occur. As in Section 4.3.1, a global argument is required, because a node might have to wait for many other fragments to merge or absorb until the level of the fragment at the other end of p's local minimum-weight external link is high enough.

Lemma 28: *GC is equitable for COM via $\mathcal{M}_2$.*

Proof: By Corollary 16, $(P'_{COM} \circ \mathcal{S}_2) \wedge P_{GC}$ is true in every reachable state of GC. Thus, in the sequel we will use the HI, COM, and GC predicates.

For each locally-controlled action φ of COM, we must show that GC is equitable for φ via $\mathcal{M}_2$.

i) φ is not ComputeMin(f) for any f. Since φ is enabled in s if and only if φ is enabled in $\mathcal{S}_2(s)$, and since $\mathcal{A}_2(s, \varphi)$ includes φ, for all s, Lemma 5 shows that GC is equitable for φ via $\mathcal{M}_2$.

ii) φ is ComputeMin(f). We show GC is progressive for φ via $\mathcal{M}_2$; Lemma 6 implies that GC is equitable for φ via $\mathcal{M}_2$.

Let Ψ_φ be the set of all pairs (s, π) of reachable states s of GC and internal actions π of GC enabled in s. For reachable state s, let $v_\varphi(s)$ be a quadruple with the following components:

1. the number of fragments;
2. the number of fragments with $rootchanged$ = false;

3. the number of fragments with $minlink = nil$; and
4. the sum of the number of nodes in each fragment's *testset*.

(1) Let s be a reachable state of GC in E_φ. So $ComputeMin(f)$ is enabled in $\mathcal{S}_2(s)$. We now show that some ψ is enabled in s with $(s, \psi) \in \Psi_\varphi$.

Let $\mathcal{G}$ be the directed graph defined as follows. There is one vertex of $\mathcal{G}$ for each element of *fragments* in s. We now specify the directed edges of $\mathcal{G}$. Let v and w be two vertices of $\mathcal{G}$, corresponding to fragments f' and g'. There is a directed edge from v to w in $\mathcal{G}$ if and only if there is a node p in $testset(f')$ whose minimum-weight external link is $\langle p, q\rangle$, $fragment(q) = g'$, and $level(f') > level(g')$. We will call fragment f' a *sink* if its corresponding vertex in $\mathcal{G}$ is a sink. (It should be obvious that there is at least one sink.)

Case 1: There is a sink f' such that $testset(f') \neq \emptyset$. Let $\psi = TestNode(p)$ for some $p \in testset(f')$. Since f' is a sink, ψ is enabled in s. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2: For all sinks f', $testset(f') = \emptyset$.

Case 2.1: There is a sink f' such that $minlink(f') = nil$. Let $\psi = ComputeMin(f')$. Since $ComputeMin(f)$ is enabled in $\mathcal{S}_2(s)$, there are at least two fragments, so there is an external link of f'. By GC-B, $accmin(f') \neq nil$. Thus ψ is enabled in s. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2.2: For all sinks f', $minlink(f') \neq nil$.

Case 2.2.1: There is a sink f' such that $rootchanged(f') =$ false. Let $\psi = ChangeRoot(f')$. Since $ComputeMin(f)$ is enabled in $\mathcal{S}_2(s)$, $minlink(f) = nil$. By COM-C then, $awake =$ true. Thus ψ is enabled in s. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2.2.2: For all sinks f', $rootchanged(f') =$ true. By COM-A, the following two cases are exhaustive.

Case 2.2.2.1: There is a sink f' such that $level(g') > level(f')$, where $g' = fragment\ (target(minlink(f')))$. Let $\psi = Absorb(g', f')$. Since f' is a sink, ψ is enabled in s. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2.2.2.2: For all sinks f', $level(g') = level(f')$, where $g' = fragment(target\ (minlink(f')))$. Let $m = \min\{level(f') : f' \text{ is a sink}\}$. Let f' be a sink with $level(f') = m$, and let $g' = fragment(target(minlink(f')))$. If g' is not a sink, then from the vertex in $\mathcal{G}$ corresponding to g' a sink is reachable (along the directed edges) whose corresponding fragment is a sink with level less than m, contradicting our choice of m. Thus g' is a sink. Since the edge weights are totally ordered, by COM-A there are two sinks f' and g' at level m such that $minedge(f') = minedge(g')$. Let $\psi = Merge(f', g')$. Obviously ψ is enabled in s, and $(s, \psi) \in \Psi_\varphi$.

---

(2) Consider step (s', π, s) of GC, where s' is reachable and in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) Obviously the external actions of GC do not change v_φ. This fact, together with (b) below, shows that $v_\varphi(s) \leq v_\varphi(s')$.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. If $\pi = TestNode(p)$, then component 4 of v_φ decreases and the rest stay the same. If $\pi = ComputeMin(f')$, then component 3 of v_φ decreases and the rest stay the same. If $\pi = ChangeRoot(f')$, then component 2 of v_φ decreases and the rest stay the same. If $\pi = Merge(f', g')$ or $Absorb(f', g')$, then component 1 of v_φ decreases.

(c) Suppose $(s', \pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s', \psi) \in \Psi_\varphi$. Since the only choice for π is an external action of GC, obviously ψ is enabled in s and $(s, \psi) \in \Psi_\varphi$. □

4.3.3 TAR is Equitable for GC

The substantial argument here is that a node p's local test-accept-reject protocol eventually finishes, thus simulating $TestNode(p)$ in GC. Again, we need a global argument: to show that the recipient of p' TEST message eventually responds to it, we must show that the level of the recipient's fragment eventually is large enough. This proof is where the state component of the set Ψ in the definition of progressive is used. The receipt of a TEST message will generally make progress, but if it is requeued and the state is unchanged, no function on states can decrease; thus, we exclude such a state-action pair from Ψ.

Lemma 29: *TAR is equitable for GC via $\mathcal{M}_3$.*

Proof: By Corollary 18, $(P'_{GC} \circ \mathcal{S}_3) \wedge P_{TAR}$ is true in every reachable state of TAR. Thus, in the sequel we will use the HI, COM, GC, and TAR predicates.

For each locally-controlled action φ of GC, we must show that TAR is equitable for φ via $\mathcal{M}_3$.

i) φ is not TestNode(p) for any p, or InTree(l) or NotInTree(l) for any l. Since φ is enabled in s if and only if φ is enabled in $\mathcal{S}_3(s)$, and since $\mathcal{A}_3(s, \varphi)$ includes φ, for all s, Lemma 5 implies that TAR is equitable for φ via $\mathcal{M}_3$.

ii) φ is TestNode(p). We show TAR is progressive for φ via $\mathcal{M}_3$; Lemma 6 implies that TAR is equitable for φ via $\mathcal{M}_3$. In the worst case, we have to wait for the levels to have the correct relationship. This requires a "global" argument.

Let Ψ_φ be the set of all pairs (s, π) of reachable states s of TAR and internal actions π of TAR enabled in s, such that if $\pi = \mathit{ReceiveTest}(\langle q, r\rangle, l, c)$, then in s either $\mathit{level}(\mathit{fragment}(r)) \geq l$, or there is more than one message in $\mathit{tarqueue}_r(\langle q, r\rangle)$.

For reachable state s, let $v_\varphi(s)$ be a 10-tuple of:

1. the number of fragments in s,
2. the number of fragments f with $\mathit{rootchanged}(f) =$ false in s,
3. the number of fragments f with $\mathit{minlink}(f) = \mathit{nil}$ in s,
4. the number of nodes q such that $q \in \mathit{testset}(\mathit{fragment}(q))$ in s,
5. the number of links l such that either $\mathit{lstatus}(l) =$ unknown, or else $\mathit{lstatus}(l) =$ branch and there is a protocol message for l, in s,
6. the number of links l such that no ACCEPT or REJECT message is in $\mathit{tarqueue}(l)$ in s,
7. the number of links l such that no TEST message is in $\mathit{tarqueue}(l)$ in s,
8. the number of messages in $\mathit{tarqueue}_q(\langle q, r\rangle)$, for all $\langle q, r\rangle \in L(G)$, in s,
9. the number of messages in $\mathit{tarqueue}_{qr}(\langle q, r\rangle)$, for all $\langle q, r\rangle \in L(G)$, in s,
10. the number of messages in $\mathit{tarqueue}_r(\langle q, r\rangle)$, for all $\langle q, r\rangle \in L(G)$, that are behind a TEST message in s.

(1) Let s be a reachable state of TAR in E_φ. We show that there exists an action ψ enabled in s such that $(s, \psi) \in \Psi_\varphi$.

Let $l = \min\{\mathit{level}(f) : f \in \mathit{fragments}\}$.

Case 1: All fragments f at level l have $\mathit{rootchanged}(f) =$ true. Then some $\mathit{Absorb}(f, g)$ or $\mathit{Merge}(f, g)$ is enabled in s, as argued in Lemma 27, Case 2.2.1 for $\varphi = \mathit{Combine}$. Let ψ be one of these enabled actions.

Case 2: $\mathit{level}(f) = l$ and $\mathit{rootchanged}(f) \neq$ true, for some $f \in \mathit{fragments}$.

Claims about s:

1. $p \in \mathit{testset}(\mathit{fragment}(q))$, by precondition of φ.

2. $awake$ = true, by Claim 1 and GC-C and COM-C.

Case 2.1: $minlink(f) \neq nil$. Let $\psi = ChangeRoot(f)$. By Claim 2 and assumption for Case 2.1, ψ is enabled in s.

Case 2.2: $minlink(f) = nil$.

Case 2.2.1: $testset(f) = \emptyset$.

3. Either there is no external link of f, or $accmin(f) \neq nil$, by GC-B and assumption for Case 2.2.1.
4. $fragment(p) \neq f$, by Claim 1 and assumption for Case 2.2.1.
5. $accmin(f) \neq nil$, by Claims 3 and 4.

Let $\psi = ComputeMin(f)$. It is enabled in s by Claim 5 and assumption for Case 2.2.1.

Case 2.2.2: $testset(f) \neq \emptyset$. Let q be some element of $testset(f)$.

Case 2.2.2.1: $testlink(q) = nil$. Let $\psi = SendTest(q)$. It is enabled in s by assumptions for Case 2.2.2.1.

Case 2.2.2.2: $testlink(q) \neq nil$. By TAR-C(a), $testlink(q) = \langle q, r\rangle$, for some r. There is a protocol message for $\langle q, r\rangle$, by TAR-C(c). So there is some message at the head of at least one of the six queues comprising $tarqueue(\langle q, r\rangle)$ and $tarqueue(\langle r, q\rangle)$. At least one of the following is enabled in s: $ReceiveTest(k, l', c')$, $ReceiveAccept(k)$, $ReceiveReject(k)$, $ChannelSend(k, m)$, and $ChannelRecv(k, m)$, where k is either $\langle q, r\rangle$ or $\langle r, q\rangle$, and $m \in M$.

Suppose in contradiction that there is no ψ enabled in s such that $(s, \psi) \in \Psi_\varphi$. That is, by definition of Ψ_φ, the only message in $tarqueue(\langle q, r\rangle)$ (if any) is a $\text{TEST}(l', c')$ in $tarqueue_r(\langle q, r\rangle)$ with $l' > level(fragment(r))$; and the only message in $tarqueue(\langle r, q\rangle)$ (if any) is a $\text{TEST}(l'', c'')$ in $tarqueue_q(\langle r, q\rangle)$ with $l'' > fragment(q))$.

Suppose the protocol message for $\langle q, r\rangle$ is a $\text{TEST}(l', c')$ in $tarqueue(\langle q, r\rangle)$, with $lstatus(\langle q, r\rangle) \neq$ rejected. By TAR-E(b), $l' = level(fragment(q))$. Since $fragment(q) = f$, $l' = l$ by choice of f. But $l' > level(fragment(r))$, by definition of Ψ_φ, which contradicts the definition of l.

Suppose the protocol message for $\langle q, r\rangle$ is a $\text{TEST}(l'', c'')$ in $tarqueue(\langle r, q\rangle)$, with $lstatus(\langle r, q\rangle)$ = rejected. By TAR-E(c), $l'' = level(fragment(q))$. But by definition of Ψ_φ, $l'' > level(fragment(q))$.

(2) Let (s', π, s) be a step of TAR, where s' is reachable and is in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) If $(s', \pi) \notin \Psi_\varphi$, then π is either $InTree(l)$, $NotInTree(l)$, or $Start(p)$, or else π is $ReceiveTest(\langle q, r\rangle, l, c)$ and in s, $l > level(fragment(r))$ and there is only one message in $tarqueue_r(\langle q, r\rangle)$. In all cases, no component of v_φ is changed, so $v_\varphi(s) = v_\varphi(s')$.

Part (b) below finishes the proof that $v_\varphi(s) \leq v_\varphi(s')$.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. We show $v_\varphi(s) < v_\varphi(s')$.

- Suppose $\pi = ChannelSend(l, m)$. Component 8 of v_φ decreases and components 1 through 7 do not change.

- Suppose $\pi = ChannelRecv(l, m)$. Component 9 of v_φ decreases and components 1 through 8 do not change.

- Suppose $\pi = SendTest(q)$. Let $\langle q, r\rangle$ be the minimum-weight link of q with $lstatus$ unknown in s'. By precondition, $testlink(q) = nil$ in s'. By TAR-D, there is no protocol message for $\langle q, r\rangle$ in s', so there is no TEST message in $tarqueue(\langle q, r\rangle)$ in s'. One is added in s. Thus component 7 of v_φ decreases and components 1 through 6 do not change. If there is no link of q with $lstatus$ unknown, then q is removed from $testset(fragment(q))$. Thus component 4 of v_φ decreases and components 1 through 3 do not change.

- Suppose $\pi = ReceiveTest(\langle q, r\rangle, l, c)$ and in s' either $l \leq level(fragment(r))$ or there is more than one message in $tarqueue_r(\langle q, r\rangle)$.

Case 1: $l \leq level(fragment(r))$ and either $c \neq core(fragment(r))$ or $testlink(r) \neq \langle r, q\rangle$ in s'.

Claims about s':

1. TEST(l, c) message is in $tarqueue(\langle q, r\rangle)$, by precondition.
2. $c \neq core(fragment(r))$ or $testlink(r) \neq \langle r, q\rangle$, by assumption.
3. If $c \neq core(fragment(r))$, then $lstatus(\langle q, r\rangle) \neq$ rejected, by TAR-E(c).
4. If $testlink(r) \neq \langle r, q\rangle$, then there is no protocol message for $\langle r, q\rangle$, by TAR-D.
5. If $testlink(r) \neq \langle r, q\rangle$, then $lstatus(\langle q, r\rangle) \neq$ rejected, by Claim 4 and definition.

6. The TEST(l, c) message in $tarqueue(\langle q, r\rangle)$ is a protocol message for $\langle q, r\rangle$, by Claims 2, 3 and 5.
7. $testlink(q) = \langle q, r\rangle$, by Claim 6 and TAR-D.
8. There is no ACCEPT or REJECT message in $tarqueue(\langle r, q\rangle)$, by Claims 6 and 7 and TAR-C(c).

If $lstatus(\langle q, r\rangle)$ is changed from unknown to rejected, then component 5 of v_φ decreases and components 1 through 4 are unchanged. Otherwise, an ACCEPT or REJECT message is added to $tarqueue(\langle r, q\rangle)$ in s, causing component 6 of v_φ to decrease by Claim 8, while components 1 through 5 stay the same.

Case 2: $l \leq level(fragment(r))$ and $c = core(fragment(r))$ and $testlink(r) = \langle r, q\rangle$ in s'.

Claims about s':

1. TEST(l, c) is in $tarqueue(\langle q, r\rangle)$, by precondition.
2. $c = core(fragment(r))$, by assumption.
3. $testlink(r) = \langle r, q\rangle$, by assumption.

Case 2.1: There is no link $\langle r, t\rangle$, $t \neq q$, with $lstatus$ unknown in s'. Then q is removed from $testset(fragment(q))$ in s, causing component 4 of v_φ to decrease while components 1 through 3 do not change.

Case 2.2: There is a link $\langle r, t\rangle$, $t \neq q$, with $lstatus(\langle r, t\rangle) =$ unknown in s'.

4. $lstatus(\langle r, q\rangle) \neq$ rejected, by Claim 3 and TAR-K.

By Claim 4, Cases 2.2.1 and 2.2.2 are exhaustive.

Case 2.2.1: $lstatus(\langle r, q\rangle) =$ unknown in s'. It is changed to rejected in s, causing component 5 of v_φ to decrease and components 1 through 4 to stay the same.

Case 2.2.2: $lstatus(\langle r, q\rangle) =$ branch.

Case 2.2.2.1: The TEST(l, c) message in $tarqueue(\langle q, r\rangle)$ is a protocol message for $\langle r, q\rangle$.

5. The TEST(l, c) message in $tarqueue(\langle q, r\rangle)$ is the only protocol message for $\langle r, q\rangle$, by TAR-C(c).

Since the only protocol message for $\langle r, q\rangle$ is removed in s, component 5 of v_φ decreases and components 1 through 4 stay the same.

Case 2.2.2.2: The TEST(l, c) message in *tarqueue*$(\langle q, r\rangle)$ is not a protocol message for $\langle r, q\rangle$.

6. *lstatus*$(\langle q, r\rangle) \neq$ rejected, by assumptions for Case 2.2.2.2.
7. There is a TEST(l', c') message in *tarqueue*$(\langle r, q\rangle)$ and *lstatus*$(\langle r, q\rangle) =$ unknown, by Claims 1, 2, 3, 6 and TAR-P.

But Claim 7 contradicts the assumption for Case 2.2.2.

Case 3: $l > level(fragment(r))$ and there is more than one message in $tarqueue_r(\langle q, r\rangle)$ in s'. All TEST messages in $tarqueue_r(\langle q, r\rangle)$ are protocol messages for the same link, either $\langle q, r\rangle$ or $\langle r, q\rangle$. Since by TAR-D and TAR-C(c) there is never more than one protocol message for any link, this TEST(l, c) message is the only one. The TEST(l, c) message is put at the end of $tarqueue_r(\langle q, r\rangle)$ in s, decreasing component 10 and not changing components 1 through 9.

- Suppose π = *ReceiveAccept*$(\langle q, r\rangle)$. Since r is removed from *testset*(*fragment*(r)), component 4 of v_φ decreases while components 1 through 3 stay the same.

- Suppose π = *ReceiveReject*$(\langle q, r\rangle)$. If there are no more unknown links, then r is removed from *testset*(*fragment*(r)), decreasing component 4 of v_φ and not changing components 1 through 3. Suppose there is another unknown link.

Claims about s':

1. REJECT is in *tarqueue*$(\langle q, r\rangle)$, by precondition.
2. There is a link $\langle r, t\rangle$, $t \neq q$, with *lstatus*$(\langle r, t\rangle) =$ unknown, by assumption.
3. *testlink*$(r) = \langle r, q\rangle$, by Claim 1 and TAR-D.
4. The REJECT in *tarqueue*$(\langle q, r\rangle)$ is the only protocol message for $\langle q, r\rangle$, by Claim 3 and TAR-C(c).
5. *lstatus*$(\langle r, q\rangle) \neq$ rejected, by Claim 3 and TAR-K.

By Claim 5, *lstatus*$(\langle r, q\rangle) \neq$ rejected. If *lstatus*$(\langle r, q\rangle) =$ unknown in s', it is changed to rejected in s. If *lstatus*$(\langle r, q\rangle) =$ branch in s', then it stays branch in s, but there are no more protocol messages for $\langle r, q\rangle$ in s, by Claim 4. Thus component 5 of v_φ decreases while components 1 through 4 stay the same.

- Suppose $\pi = ComputeMin(f)$. Component 3 of v_φ decreases and components 1 and 2 are unchanged.
- Suppose $\pi = ChangeRoot(f)$. Component 2 of v_φ decreases and component 1 is unchanged.
- Suppose $\pi = Merge(f,g)$ or $Absorb(f,g)$. Component 1 of v_φ decreases.

(c) Suppose $(s',\pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s',\psi) \in \Psi_\varphi$. Then ψ is still enabled in s and $(s,\psi) \in \Psi_\varphi$, since the only possibilities are: $\pi = InTree(l)$, $NotInTree(l)$, or $Start(p)$, or else $\pi = ReceiveTest(\langle q,r\rangle,l,c)$ and in s', $l > level(fragment(r))$ and there is only one message in $tarqueue_r(\langle q,r\rangle)$.

iii) φ is InTree($\langle$p,q$\rangle$). We show TAR is progressive for φ via $\mathcal{M}_3$; Lemma 6 implies that TAR is equitable for φ via $\mathcal{M}_3$. We simply show that if $\langle p,q\rangle = minlink(f)$, but $lstatus(\langle p,q\rangle)$ is not yet branch, then eventually $ChangeRoot(f)$ will occur.

Let Ψ_φ be all pairs (s,ψ) of reachable states s and actions ψ enabled in s such that one of the following is true: (Let $f = fragment(p)$ in s.)

- $\psi = InTree(\langle p,q\rangle)$, or
- $\langle p,q\rangle = minlink(f)$ in s, and $\psi = ChangeRoot(f)$.

For reachable state s, let $v_\varphi(s)$ be 1 if $\langle p,q\rangle = minlink(f)$ and $ChangeRoot(f)$ is enabled in s, and 0 otherwise.

(1) Let s be a reachable state of TAR in E_φ. We show that there exists an action ψ enabled in s such that $(s,\psi) \in \Psi_\varphi$. Let $f = fragment(p)$ in s.

Claims about s:

1. $awake$ = true, by precondition of φ.
2. $(p,q) \in subtree(f)$ or $\langle p,q\rangle = minlink(f)$, by precondition of φ.
3. $answered(\langle p,q\rangle)$ = false, by precondition of φ.
4. $lstatus(\langle p,q\rangle) \neq$ rejected, by Claim 2 and TAR-B.

By Claim 4, the following two cases are exhaustive.

Case 1: $lstatus(\langle p,q\rangle) =$ branch. Let $\psi = InTree(\langle p,q\rangle)$. It is enabled in s by Claims 1 and 3 and assumption for this case, and $(s,\psi) \in \Psi_\varphi$.

Case 2: $lstatus(\langle p,q\rangle) =$ unknown.

5. $minlink(f) = \langle p,q\rangle$, by Claim 2 and TAR-A(a).
6. $rootchanged(f) =$ false, by Claim 5 and TAR-H.

Let $\psi = ChangeRoot(f)$. It is enabled in s by Claims 1, 5 and 6, and $(s,\psi) \in \Psi_\varphi$.

---

(2) Let (s',π,s) be a step of TAR, where s' is reachable and is in E_φ, $(s',\pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) Suppose $(s',\pi) \notin \Psi_\varphi$. We show that no possibility for π can affect whether or not $ChangeRoot(f)$ is enabled, i.e., $v_\varphi(s) = v_\varphi(s')$. This together with (b) below shows that $v_\varphi(s) \leq v_\varphi(s')$.

Case 1: $ChangeRoot(f)$ is enabled in s'. No action sets *awake* to false. No action (other than $ChangeRoot(f)$) sets $rootchanged(f)$ to false. No action sets $minlink(f)$ to nil. f remains in *fragments* because π is not $Absorb(g,f)$, $Merge(f,g)$ or $Merge(g,f)$, for any g, since $rootchanged(f) =$ false.

Case 2: $rootchanged(f)$ is not enabled in s'. By precondition of φ, *awake* is true in s'. If $rootchanged(f) =$ true in s', then the same is true in s, because the only action that sets it to false is the *Merge* that created f. If $minlink(f) = nil$ in s', then $\langle p,q\rangle \neq minlink(f)$, so even if $minlink(f)$ becomes nonnil (by $ComputeMin(f)$), v_φ remains 0.

(b) Suppose $(s',\pi) \in \Psi_\varphi$. Since $(s',\pi) \notin X_\varphi$, $\pi \neq InTree(\langle p,q\rangle)$. Thus $minlink(f) = \langle p,q\rangle$ in s' and $\pi = ChangeRoot(f)$. Obviously v_φ goes from 1 to 0.

(c) Suppose $(s',\pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s',\psi) \in \Psi_\varphi$. The same argument as in (2a), Case 1, applies.

iv) φ is NotInTree($\langle$p,q$\rangle$). We show that TAR is progressive for φ via $\mathcal{M}_3$; Lemma 6 implies that TAR is equitable for φ via $\mathcal{M}_3$. The goal is to show that if $q \in nodes(fragment(p))$ and $(p,q) \notin subtree(fragment(p))$, then eventually $lstatus(\langle p,q\rangle) =$ rejected. This requires a global argument, as for $TestNode(p)$, because it could be that some unknown link will never be tested until only one fragment remains.

Let Ψ_φ be $\Psi_{TestNode(p)} \cup \{(s, NotInTree(\langle p,q\rangle)) : s \text{ reachable}, NotInTree(\langle p,q\rangle)$ enabled in $s\}$.

Let $v_\varphi(s) = v_{TestNode(p)}(s)$ for all reachable states s.

Let v_φ be the same as for $TestNode(p)$.

(1) Let s be a reachable state of TAR in E_φ. We show that there exists an action ψ enabled in s such that $(s,\psi) \in \Psi_\varphi$.

$lstatus(\langle p,q\rangle) \neq$ branch, by TAR-A(a). If $lstatus(\langle p,q\rangle) =$ rejected, then let $\psi = NotInTree(\langle p,q\rangle)$.

Suppose $lstatus(\langle p,q\rangle) =$ unknown in s. The rest of the argument is just like that for $TestNode(p)$, except for the following cases.

Case 2.1: $ChangeRoot(f)$ is enabled in s because $awake =$ true by the precondition of φ.

Case 2.2.1: We show that $ComputeMin(f)$ is enabled in s by showing that there are at least two fragments, as follows. If there is only one fragment, then $f = fragment(p)$, and $p \notin testset(f)$ (since we assume $testset(f) = \emptyset$). But since we also assume $lstatus(\langle p,q\rangle) =$ unknown, TAR-I gives as contradiction. Thus, there is an external link of f, and by GC-B, $accmin(f') \neq nil$.

(2) Like $TestNode(p)$, after noting that π cannot be $NotInTree(\langle p,q\rangle)$. □

4.3.4 DC is Progressive for an Action of GC

The main idea is to show that REPORT messages converge on the core. This argument is local to one fragment.

Lemma 30: *DC is progressive for $ComputeMin(f)$ via $\mathcal{M}_4$.*

Proof: By Corollary 20, $(P'_{GC} \circ \mathcal{S}_4) \wedge P_{DC}$ is true in every reachable state of DC. Thus, in the sequel we will use the HI, COM, GC and DC predicates.

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of DC and actions ψ of DC such that in s, a REPORT(w) is in some $dcqueue(\langle q,p\rangle)$ and either q is a child of p, or else $dcstatus(p) =$ unfind and $p = mw\text{-}root(f)$; and $\psi \in \{ChannelSend(\langle q,p\rangle,$ REPORT(w)), $ChannelRecv(\langle q,p\rangle,$ REPORT(w)), $ReceiveReport(\langle q,p\rangle, w)\}$.

For reachable state s, let $v_\varphi(s)$ be a quadruple with the following components:

1. The number of nodes $p \in nodes(f)$ with $dcstatus(p) =$ find.
2. The number of REPORT messages in $dcqueue_q(\langle q,p\rangle)$, for all $(p,q) \in subtree(f)$ such that either q is a child of p or else $p = mw\text{-}root(f)$ and $dcstatus(p) =$ unfind.
3. The number of REPORT messages in $dcqueue_{qp}(\langle q,p\rangle)$ for all $(p,q) \in subtree(f)$ such that either q is a child of p or else $p = mw\text{-}root(f)$ and $dcstatus(p) =$ unfind.
4. The number of REPORT messages in $dcqueue_p(\langle q,p\rangle)$ for all $(p,q) \in subtree(f)$ such that either q is a child of p or else $p = mw\text{-}root(f)$ and $dcstatus(p) =$ unfind.

(1) Let s be a reachable state of DC in E_φ. We show that there exists an action ψ enabled in s such that $(s,\psi) \in \Psi_\varphi$.

Claims about s:

1. $minlink(f) = nil$, by precondition.
2. $accmin(f) \neq nil$, by precondition.
3. $testset(f) = \emptyset$, by precondition.
4. There is an external link of f, by Claim 2 and GC-A.
5. No FIND message is in $subtree(f)$, by Claim 3 and DC-D(c).
6. If $dcstatus(p) =$ find, then a REPORT message is in $subtree(p)$ headed toward p, for any $p \in nodes(f)$, by Claim 3 and DC-I(b).

Suppose a REPORT(w) is in some $dcqueue(\langle q,p\rangle)$ and q is a child of p. By DC-B(a), $inbranch(p) \neq \langle p,q\rangle$. Obviously, $(p,q) \neq core(f)$, so by DC-A(g), $dcstatus(p) =$ find. By Claim 5 and DC-O, the REPORT(w) is the only message in $dcqueue(\langle q,p\rangle)$. If it is in $dcqueue_q(\langle q,p\rangle)$, let $\psi = ChannelSend(\langle q,p\rangle,$ REPORT(w)); if it is in $dcqueue_{qp}(\langle q,p\rangle)$, let $\psi = ChannelRecv(\langle q,p\rangle,$ REPORT(w)); if it is in $dcqueue_p(\langle q,p\rangle)$, let $\psi = ReceiveReport(w)$. Obviously, ψ is enabled in s, and $(s,\psi) \in \Psi_\varphi$.

Suppose no REPORT is in any $dcqueue(\langle q,p\rangle)$ with q a child of p. By Claim 6, $dcstatus(p) =$ unfind for all $p \in nodes(f)$. Then by Claims 1, 4 and 5, a REPORT(w) is in $dcqueue(\langle q,p\rangle)$, where $(p,q) = core(f)$ and $p = mw\text{-}root(f)$. By Claim 5 and DC-O, the REPORT(w) is the only message in $dcqueue(\langle q,p\rangle)$. If it is in $dcqueue_q(\langle q,p\rangle)$, let $\psi = ChannelSend(\langle q,p\rangle, \text{REPORT}(w))$; if it is in $dcqueue_{qp}(\langle q,p\rangle)$, let $\psi = ChannelRecv(\langle q,p\rangle, \text{REPORT}(w))$; if it is in $dcqueue_p(\langle q,p\rangle)$, let $\psi = ReceiveReport(w)$. Obviously, ψ is enabled in s, and $(s,\psi) \in \Psi_\varphi$.

(2) Let (s',π,s) be a step of DC, where s' is reachable and is in E_φ, $(s',\pi) \notin X_\varphi$, and $s \in E_\varphi$. We note the following claims about s'.

1. $testset(f) = \emptyset$, by precondition.
2. $minlink(f) = nil$, by precondition.
3. No FIND is in $subtree(f)$, by Claim 1 and DC-D(c).

(a) To show $v_\varphi(s) \leq v_\varphi(s')$, we show that $v_\varphi(s) = v_\varphi(s')$ if $(s',\pi) \notin \Psi_\varphi$; this together with part (b) below gives the result. Suppose $(s',\pi) \notin \Psi_\varphi$.

$TestNode(p)$ is not enabled, for $p \in nodes(f)$, by Claim 1. $ChangeRoot(f)$, $Merge(f,g)$, $Merge(g,f)$, and $Absorb(g,f)$ are not enabled, for $g \in fragments$, by Claim 2. $ReceiveFind(\langle p,q\rangle)$, $AfterMerge(p,q)$, $ChannelSend(\langle p,q\rangle, \text{FIND})$, and $ChannelRecv(\langle p,q\rangle, \text{FIND})$ are not enabled, for $p \in nodes(f)$, by Claim 3. Thus π is none of the above actions.

If $\pi = ChannelSend(\langle q,p\rangle, \text{REPORT}(w))$ or $ChannelRecv(\langle q,p\rangle, \text{REPORT}(w))$, for $(q,p) \in subtree(f)$, then v_φ is unchanged, since $(s',\pi) \notin \Psi_\varphi$.

Suppose $\pi = ReceiveReport(\langle q,p\rangle, w)$.

Case 1: p is a child of q. By DC-A(a), $inbranch(p) = \langle p,q\rangle$. By DC-B(b), $dcstatus(p) =$ unfind. So the only change is the removal of the message. Since p is a child of q, $p \neq mw\text{-}root(f)$, so v_φ is unchanged.

Case 2: $(p,q) = core(f)$ and $p \neq mw\text{-}root(f)$. By DC-A(a), $inbranch(p) = \langle p,q\rangle$. The only effect is that either the message is requeued (if $dcstatus(p) =$ find), or the message is removed (if $dcstatus(p) =$ unfind); in both cases, v_φ is unchanged.

Case 3: $(p,q) = core(f)$, $p = mw\text{-}root(f)$, and $dcstatus(p) =$ find. The only effect is that the message is requeued, so v_φ is unchanged.

Suppose $\pi = Merge(g,h)$. By precondition, $minlink(g) = minlink(h) \neq nil$ in s'. So $f \neq g$ and $f \neq h$. Obviously v_φ is unchanged.

Suppose $\pi = Absorb(g, h)$. By precondition, $minlink(h) \neq nil$ in s', so $f \neq h$ by Claim 2. If $f \neq g$, then obviously v_φ is unchanged. Suppose $f = g$. As in the proof of condition (3a) in Lemma 19 for *viii)* $\pi = Absorb$, Case 2, no REPORT message is headed toward $minnode(h)$ and $dcstatus(r) =$ unfind for all $r \in nodes(h)$ in s'. Thus v_φ does not change.

The remaining actions (not mentioned above) obviously do not affect v_φ.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. We show $v_\varphi(s) < v_\varphi(s')$. If $\psi = ChannelSend(l, m)$, component 2 of v_φ decreases and component 1 is unchanged. If $\psi = ChannelRecv(l, m)$, component 3 of v_φ decreases and components 1 and 2 are unchanged.

Suppose $\psi = ReceiveReport(\langle q, p \rangle, w)$.

Case 1: q is a child of p. By DC-B(a), $inbranch(p) \neq \langle p, q \rangle$. By DC-A(g), $dcstatus(p) =$ find. If $findcount(p) = 1$ in s', then component 1 of v_φ decreases. Otherwise, component 4 decreases and components 1 through 3 are unchanged.

Case 2: q is not a child of p, $p = mw\text{-}root(f)$, and $dcstatus(p) =$ unfind. So $(p, q) = core(f)$. By DC-P, $w > bestwt(p)$. But this contradicts $(s', \pi) \notin X_\varphi$.

(c) Suppose $(s', \pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s', \psi) \in \Psi_\varphi$. We show that ψ is still enabled in s and $(s, \psi) \in \Psi_\varphi$. Since the queues are FIFO, there is no way to disable ψ.

It remains to show that (s, ψ) is still in Ψ_φ.

One possible way (s, ψ) could no longer be in Ψ_φ is if the position of $mw\text{-}root(f)$ changes, i.e., if π is $Merge(f, g)$, $Merge(g, f)$, $Absorb(f, g)$, or $Absorb(g, f)$, for some fragment g. But by Claim 2, $minlink(f) = nil$. Thus π cannot be $Merge(f, g)$, $Merge(g, f)$, or $Absorb(g, f)$. Suppose $\pi = Absorb(f, g)$. Let $core(f) = (p, q)$, $p = mw\text{-}root(f)$, and q be the endpoint of $core(f)$ closest to $target(minlink(g))$ in s'. The minimum-weight external link of f has smaller weight than $minlink(g)$, which by COM-A is the minimum-weight external link of g. Thus $mw\text{-}root(f)$ does not change after $Absorb(f, g)$.

Another way is if the position of $core(f)$ changes. This only happens if π is $Merge(f, g)$, $Merge(g, f)$ or $Absorb(g, f)$, which we showed is impossible.

The third way is if $dcstatus(p)$ changes from unfind to find, where $p = mw\text{-}root(f)$. This only happens if $\pi = ReceiveFind(\langle q, p \rangle)$ for some q. But by Claim 3, no FIND is in $subtree(f)$, and by DC-D(a), no FIND can be in an external link. □

4.3.5 CON is Progressive for Some Actions of COM

To show that CON is progressive for *Merge* and *Absorb*, we just show that the CONNECT message on the *minlink* makes it across. For *ChangeRoot*, we show that the chain of CHANGEROOT messages eventually reaches the *minnode*. These arguments are all local to one fragment.

Lemma 31: *CON is progressive for $Merge(f,g)$, $Absorb(f,g)$ and $ChangeRoot(f)$ via $\mathcal{M}_6$.*

Proof: By Corollary 24, $(P'_{COM} \circ \mathcal{S}_6) \wedge P_{CON}$ is true in every reachable state of CON. Thus, in the sequel we will use the HI, COM, and CON predicates.

i) φ is Merge(f,g). Let $(p,q) = minedge(f)$. Let Ψ_φ be the set of all pairs (s,ψ) of reachable states s of CON and actions ψ of CON enabled in s, such that $\psi \in \{ChannelSend(\langle q,p\rangle, \text{CONNECT}(l)), ChannelRecv(\langle q,p\rangle, \text{CONNECT}(l)), Merge\ (f,g)\}$.

For reachable state s of CON, let $v_\varphi(s) = (x,y)$, where x is the number of messages in $cqueue_q(\langle q,p\rangle)$ in s, and y is the number of messages in $cqueue_{qp}(\langle q,p\rangle)$ in s.

(1) Suppose s is a reachable state of CON in E_φ. We show that there is a ψ enabled in s such that $(s,\psi) \in \Psi_\varphi$.

Claims about s:

1. $f \neq g$, by precondition.
2. $minedge(f) = minedge(g) = (p,q)$, by precondition.
3. $rootchanged(f) = \text{true}$, by precondition.
4. $rootchanged(g) = \text{true}$, by precondition.
5. A CONNECT(l) message is in $cqueue(k)$, for some external link k of f, by Claim 3.
6. A CONNECT(l) message is in $cqueue(\langle p,q\rangle)$, by Claims 2, 5 and CON-D.
7. A CONNECT(m) message is in $cqueue(k)$, for some external link k of g, by Claim 4.
8. A CONNECT(m) message is in $cqueue(\langle q,p\rangle)$, by Claims 2, 6 and CON-D.
9. $l = level(f)$, by Claim 5 and CON-D.
10. $m = level(g)$, by Claim 7 and CON-D.
11. $level(f) \leq level(g)$, by Claim 2 and COM-A.
12. $level(g) \leq level(f)$, by Claim 2 and COM-A.

13. $level(f) = level(g)$, by Claims 11 and 12.
14. $l = m$, by Claims 9, 10 and 13.
15. No CHANGEROOT message is in $cqueue(\langle q, p \rangle)$, by Claim 1 and CON-C.
16. Exactly one CONNECT message is in $cqueue(\langle q, p \rangle)$, by Claims 7, 8 and CON-D.

If CONNECT(l) is in $cqueue_q(\langle q, p \rangle)$, then let $\psi = ChannelSend(\langle q, p \rangle, \text{CONNECT}(l))$. If CONNECT($l$) is in $cqueue_{qp}(\langle q, p \rangle)$, then let $\psi = ChannelRecv(\langle q, p \rangle, \text{CONNECT}(l))$. If CONNECT($l$) is in $cqueue_p(\langle q, p \rangle)$, then let $\psi = Merge(f, g)$. It is easy to see in all cases that ψ is enabled in s and $(s, \psi) \in \Psi_\varphi$.

(2) Suppose (s', π, s) is a step of CON, s' is reachable and in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) The only actions that can increase v_φ are $ComputeMin(g)$, and $ChangeRoot(g)$. (Even though $ChannelSend(\langle q, p \rangle, m)$ would increase y, it would simultaneously decrease x.) By Claim 2, $ComputeMin(g)$ is not enabled in s'. By Claim 4, $ChangeRoot(g)$ is not enabled in s'.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. Since $(s', \pi) \notin X_\varphi$, $\pi \neq Merge(f, g)$. Obviously, the other two choices for ψ decrease v_φ.

(c) Suppose $(s', \pi) \notin \Psi_\varphi$, ψ is enabled in s' and $(s', \psi) \in \Psi_\varphi$. We show ψ is enabled in s and $(s, \psi) \in \Psi_\varphi$. If $\psi = ChannelSend$ or $ChannelRecv$, then it can only be disabled by occurring. If $\psi = Merge(f, g)$, then since $s \in E_\varphi$, ψ is still enabled in s (by the argument in part (1)). In all cases, $(s, \psi) \in \Psi_\varphi$.

ii) φ is Absorb(f,g). Let $\langle q, p \rangle = minlink(g)$. Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of CON and actions ψ of CON enabled in s, such that $\psi \in \{ChannelSend(\langle q, p \rangle, \text{CONNECT}(l)),\ ChannelRecv(\langle q, p \rangle, \text{CONNECT}(l)),\ Absorb\ (f, g)\}$.

For reachable state s of CON, let $v_\varphi(s) = (x, y)$, where x is the number of messages in $cqueue_q(\langle q, p \rangle)$ in s, and y is the number of messages in $cqueue_{qp}(\langle q, p \rangle)$ in s.

(1) Suppose s is a reachable state of CON in E_φ. We show that there is a ψ enabled in s such that $(s, \psi) \in \Psi_\varphi$.

Claims about s:

1. $level(g) < level(f)$, by precondition.

2. $\langle q,p\rangle = minlink(g)$, by assumption.
3. $f = fragment(p)$, by precondition.
4. $rootchanged(g) =$ true, by precondition.
5. A CONNECT(l) message is in $cqueue(k)$, where k is an external link of g, by Claim 4.
6. A CONNECT(l) message is in $cqueue(\langle q,p\rangle)$, by Claims 2, 5 and CON-D.
7. No CHANGEROOT message is in $cqueue(\langle q,p\rangle)$, by Claims 5 and 6 and CON-C.

If CONNECT(l) is in $cqueue_q(\langle q,p\rangle)$, then let $\psi = ChannelSend(\langle q,p\rangle,$ CONNECT$(l))$. If CONNECT(l) is in $cqueue_{qp}(\langle q,p\rangle)$, then let $\psi = ChannelRecv(\langle q,p\rangle,$ CONNECT$(l))$. If CONNECT(l) is in $cqueue_p(\langle q,p\rangle)$, then let $\psi = Absorb(f,g)$. In all cases, it is easy to see that ψ is enabled in s and $(s,\psi) \in \Psi_\varphi$.

(2) Suppose (s',π,s) is a step of CON, s' is reachable and in E_φ, $(s',\pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) The only actions that can increase v_φ are $ComputeMin(g)$, and $ChangeRoot(g)$. (Even though $ChannelSend(\langle q,p\rangle,m)$ would increase y, it would simultaneously decrease x.) By Claim 2, $ComputeMin(g)$ is not enabled in s'. By Claim 4, $ChangeRoot(g)$ is not enabled in s'.

(b) Suppose $(s',\pi) \in \Psi_\varphi$. Since $(s',\pi) \notin X_\varphi$, $\pi \neq Absorb(f,g)$. Obviously, the other two choices for ψ decrease v_φ.

(c) Suppose $(s',\pi) \notin \Psi_\varphi$, ψ is enabled in s' and $(s',\psi) \in \Psi_\varphi$. We show ψ is enabled in s and $(s,\psi) \in \Psi_\varphi$. If $\psi = ChannelSend$ or $ChannelRecv$, then it can only be disabled by occurring. If $\psi = Absorb(f,g)$, then since $s \in E_\varphi$, ψ is still enabled in s (by the argument in part (1)). In all cases, $(s,\psi) \in \Psi_\varphi$ by definition.

iii) φ is ChangeRoot(f). Let Ψ_φ be the set of all pairs (s,ψ) of reachable states s of CON and actions ψ of CON enabled in s, such that $\psi \in \{ReceiveChangeRoot(\langle q,p\rangle), ChannelSend(\langle q,p\rangle,$ CHANGEROOT$), ChannelRecv(\langle q,p\rangle,$ CHANGEROOT$) : p \in nodes(f)\} \cup \{ChangeRoot(f)\}$.

For reachable state s of CON, let $v_\varphi(s)$ be a triple defined as follows. If there is no CHANGEROOT message in $subtree(f)$ in s, then $v_\varphi(s)$ is $(0,0,0)$. Suppose, in s, there is a CHANGEROOT message in $cqueue(\langle q,p\rangle)$, where $p \in nodes(f)$. Then $v_\varphi(s)$ is:

1. the number of nodes in the path in $subtree(f)$ from p to $minnode(f)$ in s (counting the endpoints p and $minnode(f)$);

2. the number of CHANGEROOT messages in $cqueue_r(\langle r,t\rangle)$, for all $t \in nodes(f)$ in s; and
3. the number of CHANGEROOT messages in $cqueue_{rt}(\langle r,t\rangle)$, for all $t \in nodes(f)$ in s.

(By CON-B and CON-C, there is only one CHANGEROOT message in $subtree(f)$. By COM-G, HI-A and HI-B, there is a unique path in $subtree(f)$ from p to $minnode(f)$. Thus, $v_\varphi(s)$ is well-defined.)

(1) We show that if s is a reachable state of CON in E_φ, then there is a ψ enabled in s such that $(s,\psi) \in \Psi_\varphi$.

Claims about s:

1. $rootchanged(f) =$ false, by precondition of φ.
2. $minlink(f) \neq nil$, by precondition of φ.

If $|nodes(f)| = 1$ (i.e., $subtree(f) = \{p\}$, for some p), then let $\psi = ChangeRoot(f)$. Obviously, ψ is enabled in s and $(s,\psi) \in \Psi_\varphi$. Now suppose $|nodes(f)| > 1$.

3. $minnode(f) \neq root(f)$, by Claims 1 and 2 and CON-B.
4. Exactly one CHANGEROOT message is in $cqueue(\langle q,p\rangle)$, for some $(p,q) \in subtree(f)$, by Claims 1 and 2 and CON-B.
5. $(q,p) \neq core(f)$, by Claim 4 and CON-C.
6. No CONNECT message is in $cqueue(\langle q,p\rangle)$, by Claim 5 and CON-E.

If the CHANGEROOT message is in $cqueue_q(\langle q,p\rangle)$, then let $\psi = ChannelSend(\langle q,p\rangle$, CHANGEROOT$)$. If the CHANGEROOT message is in $cqueue_{qp}(\langle q,p\rangle)$, then let $\psi = ChannelRecv(\langle q,p\rangle$, CHANGEROOT$)$. If the CHANGEROOT message is in $cqueue_p(\langle q,p\rangle)$, then let $\psi = ReceiveChangeRoot(\langle q,p\rangle)$. In all three cases, ψ is enabled in s because of Claims 4 and 6. By definition, $(s,\psi) \in \Psi_\varphi$.

(2) Suppose (s',π,s) is a step of CON such that s' is reachable and in E_φ, $(s',\pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) We show that if $(s',\pi) \notin \Psi_\varphi$, then $v_\varphi(s) = v_\varphi(s')$. Together with (b) below, it implies that $v_\varphi(s) \leq v_\varphi(s')$.

Since $minlink(f) \neq nil$ in s', $\pi \neq ComputeMin(f)$. Since $rootchanged(f) =$ false in s', $\pi \neq Merge(f,g)$, $Merge(g,f)$, or $Absorb(g,f)$ for any g.

Suppose $\pi = Absorb(f, g)$. First we show that $minnode(f)$ is unchanged. By COM-A, $level(h) \geq level(f)$, where $h = fragment(target(minlink(f)))$; by precondition of $Absorb(f, g)$, $h \neq g$, and thus $wt(minlink(f)) < wt(minlink(g))$. Also by COM-A, $minlink(g)$ is the minimum-weight external link of g. Thus $minlink(f)$ does not change. Second, we show that no CHANGEROOT message is in $subtree(g)$. By precondition of $Absorb(f, g)$, $rootchanged(g) =$ true. Then by CON-C, no CHANGEROOT message is in $subtree(g)$.

No other value of π, such that $(s', \pi) \notin \Psi_\varphi$, affects v_φ.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. We show $v_\varphi(s) < v_\varphi(s')$.

If $\pi = ChannelSend(\langle q, p\rangle, \text{CHANGEROOT})$, then the second component of v_φ decreases while the first remains the same. If $\pi = ChannelRecv(\langle q, p\rangle, \text{CHANGEROOT})$, then the third component of v_φ decreases while the first two remain the same.

Suppose $\pi = ReceiveChangeRoot(\langle q, p\rangle)$. By CON-C and CON-B there is exactly one CHANGEROOT message in $subtree(f)$. Since $(s, \pi) \notin X_\varphi$, $p \neq minnode(f)$. Thus, the first component of $v_\varphi(s')$ is at least 1. The first component of v_φ decreases by 1 in s, by definition of $tominlink(p)$. Thus $v_\varphi(s) < v_\varphi(s')$.

(c) Suppose $(s', \pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s', \psi) \in \Psi_\varphi$. We show ψ is enabled in s, and $(s, \psi) \in \Psi_\varphi$.

Suppose $\psi = ChangeRoot(f)$.

Claims about s':

1. $rootchanged(f) =$ false, by precondition of ψ.
2. $minlink(f) \neq nil$, by precondition of ψ.
3. $subtree(f) = \{p\}$, by precondition of ψ.
4. No CHANGEROOT message is in $cqueue(\langle q, p\rangle)$ for any q, by Claim 3 and CON-C.
5. $ComputeMin(f)$ is not enabled, by Claim 2.
6. $Merge(f, g)$, $Merge(g, f)$, and $Absorb(g, f)$ are not enabled for any g, by Claim 1.
7. $ReceiveChangeRoot(\langle q, p\rangle)$ is not enabled for any q, by Claim 4.

By Claims 5, 6 and 7, π is no action that can disable ψ; hence, ψ is enabled in s. By definition, $(s, \psi) \in \Psi_\varphi$.

Suppose $\psi = ReceiveChangeRoot(\langle q,p\rangle)$, $ChannelSend(\langle q,p\rangle, \text{CHANGEROOT})$, or $ChannelRecv(\langle q,p\rangle, \text{CHANGEROOT})$. The only action that can disable ψ is ψ itself. Thus, ψ is enabled in s and $(s,\psi) \in \Psi_\varphi$.

4.3.6 GHS is Equitable for TAR

The interesting arguments are for showing GHS is equitable for $SendTest(p)$, and for $ChangeRoot(f)$ when $subtree(f)$ is a singleton node. For $SendTest(p)$, we show that an INITIATE-find message eventually reaches p. The big effort is for the $ChangeRoot(f)$. We must show that eventually every node will be awakened, either by a *Start* action, or by the receipt of a CONNECT or TEST message. This requires a global argument about the entire graph. This is another place in which the state component of Ψ in the definition of progressive is needed, since it is possible for a message to be requeued, leaving the state unchanged.

Lemma 32: *GHS is equitable for TAR via $\mathcal{M}_{TAR}$.*

Proof: We show that GHS is equitable for each locally-controlled action φ of TAR via $\mathcal{M}_{TAR}$. First, a point of notation: let $Receive(\langle q,p\rangle, m)$ be a synonym for $ReceiveConnect(\langle q,p\rangle, l)$ if $m = \text{CONNECT}(l)$, a synonym for $ReceiveInitiate(\langle q,p\rangle, l, c, st)$ if $m = \text{INITIATE}(l, c, st)$, etc.

By Corollary 26, P'_{GHS} is true in every reachable state of GHS. Thus, in the sequel we will use the HI, COM, GC, TAR, DC, NOT, CON and GHS predicates.

i) φ is InTree(l) or NotInTree(l). By Lemma 5, we are done.

ii) φ is ChannelSend($\langle$q,p$\rangle$,m). We show that GHS is progressive for φ via $\mathcal{M}_{TAR}$. Lemma 6 gives the result.

Let Ψ_φ be the set of all pairs (s,ψ) of reachable states s of GHS and actions ψ of GHS enabled in s such that m' is the message at the head of $queue_q(\langle q,p\rangle)$ in s, and $\psi = ChannelSend(\langle q,p\rangle, m')$.

For reachable state s, let $v_\varphi(s)$ be the number of messages in $queue_q(\langle q,p\rangle)$ ahead of the message at the head of $tarqueue_q(\langle q,p\rangle)$.

Verifying the progressive conditions is straightforward.

iii) φ is ChannelRecv($\langle$q,p$\rangle$,m). We show that GHS is progressive for φ via $\mathcal{M}_{TAR}$. Lemma 6 gives the result.

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of GHS and actions ψ of GHS enabled in s such that m' is the message at the head of $queue_{qp}(\langle q,p\rangle)$ in s, and $\psi = ChannelRecv(\langle q,p\rangle, m')$.

For reachable state s, let $v_\varphi(s)$ be the number of messages in $queue_{qp}(\langle q,p\rangle)$ ahead of the message at the head of $tarqueue_{qp}(\langle q,p\rangle)$.

Verifying the progressive conditions is straightforward.

iv) φ is ReceiveTest($\langle$q,p$\rangle$,l,c), ReceiveAccept($\langle$q,p$\rangle$), or Receive-Reject($\langle$q,p$\rangle$). We show that GHS is progressive for φ via $\mathcal{M}_{TAR}$. Lemma 6 gives the result.

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of GHS and actions ψ of GHS enabled in s such that m' is the message at the head of $queue_p(\langle q,p\rangle)$ in s, and $\psi = Receive(\langle q,p\rangle, m)$.

For reachable state s, let $v_\varphi(s)$ be the number of messages in $queue_p(\langle q,p\rangle)$ ahead of the message at the head of $tarqueue_p(\langle q,p\rangle)$.

Verifying the progressive conditions is straightforward.

v) φ is SendTest(p). We show that GHS is progressive for φ via $\mathcal{M}_{TAR}$. Lemma 6 gives the result.

Let Ψ_φ be the set of all pairs (s, π) of reachable states s of GHS and actions ψ of GHS enabled in s such that one of the following is true: (Let $f = fragment(p)$.)

- CONNECT(l) is in $queue(\langle q,r\rangle)$, where $(q,r) = core(f)$ and $p \in subtree(q)$, m is any message in $queue(\langle q,r\rangle)$ that is not behind the CONNECT(l) in s, and $\psi \in \{ChannelSend(\langle q,r\rangle, m),\ ChannelRecv(\langle q,r\rangle, m),\ Receive(\langle q,r\rangle, m)\}$.

- An INITIATE(l, c,find) message in $queue(\langle t,u\rangle)$ is headed toward p and m is any message in $queue(\langle t,u\rangle)$ that is not behind the INITIATE(l, c,find) in s, and $\psi \in \{ChannelSend(\langle t,u\rangle, m),\ ChannelRecv(\langle t,u\rangle, m),\ Receive(\langle t,u\rangle, m)\}$.

For reachable state s, $v_\varphi(s)$ is a 7-tuple with the following components.

If no CONNECT is in $queue(\langle q,r\rangle)$, where $(q,r) = core(f)$ and $p \in subtree(q)$ in s, then components 1 through 3 are 0. Suppose otherwise. By CON-D and CON-E, there is only one CONNECT message in $queue(\langle q,r\rangle)$.

1. The number of messages in $queue_q(\langle q, r\rangle)$ that are not behind the CONNECT.
2. The number of messages in $queue_{qr}(\langle q, r\rangle)$ that are not behind the CONNECT.
3. The number of messages in $queue_r(\langle q, r\rangle)$ that are not behind the CONNECT.

If no INITIATE(l, c,find) is headed toward p, then components 4 through 6 are 0. By DC-S, there is at most one such message. Suppose such a message is in $queue(\langle t, u\rangle)$.

4. The number of nodes on the path in $subtree(f)$ from u to p, including the endpoints.
5. The number of messages in $queue_t(\langle t, u\rangle)$ that are not behind the INITIATE(l, c, find).
6. The number of messages in $queue_{tu}(\langle t, u\rangle)$ that are not behind the INITIATE(l, c, find).
7. The number of messages in $queue_u(\langle t, u\rangle)$ that are not behind the INITIATE(l, c, find).

(1) Let s be a reachable state of GHS in E_φ. Thus, $p \in testset(f)$ and $testlink(p) = nil$. By the definition of $testset(f)$, either a FIND message is headed toward p in some $queue(\langle q, r\rangle)$, or a CONNECT message is in $queue(\langle q, r\rangle)$, where $(q, r) = core(f)$ and $p \in subtree(q)$. In either case, let m be the message at the head of $queue(\langle t, u\rangle)$. Let ψ be $ChannelSend(\langle q, r\rangle, m)$ if m is in $queue_q(\langle q, r\rangle)$; let ψ be $ChannelRecv(\langle q, r\rangle, m)$ if m is in $queue_{qr}(\langle q, r\rangle)$; let ψ be $Receive(\langle q, r\rangle, m)$ if m is in $queue_r(\langle q, r\rangle)$. Obviously, ψ is enabled in s and $(s, \psi) \in \Psi_\varphi$.

(2) Let (s', π, s) be a step of GHS, s' be reachable and in E_φ, $(s', \pi) \not\in X_\varphi$, and $s \in E_\varphi$.

(a) We show that if $(s', \pi) \not\in \Psi_\varphi$, then $v_\varphi(s') = v_\varphi(s)$; together with (b) below, this is enough. We consider all the ways that v_φ could change.

Can a CONNECT be added to $queue(\langle q, r\rangle)$, with $(q, r) = core(f)$ by π? By COM-F, $(p, q) \in subtree(f)$, so by TAR-A(b), $lstatus(\langle q, r\rangle) =$ branch. Yet by inspecting the code, we see that CONNECT is only added to a queue if its $lstatus$ is not branch, or if the source node is sleeping, in which case GHS-A(c) implies that the $lstatus$ is not branch.

Since we've assumed $(s', \pi) \not\in \Psi_\varphi$, no CONNECT can be removed from the relevant queue.

For a given fragment f, $core(f)$ never changes.

Can the identity of *fragment*(p) change? Since $p \in testset(f)$ by the precondition of φ, $minlink(f) = nil$ in s' by GC-C. Thus no $Absorb(g, f)$, $Merge(f, g)$ or $Merge(g, f)$ is enabled in s'.

The number of messages in the same queue as the relevant CONNECT message but not behind it cannot change, because the queues are FIFO (and $(s', \pi) \notin \Psi_\varphi$).

Can a relevant INITIATE message be added? The only way it can is if either a CONNECT message in $queue(\langle q, r \rangle)$ with $(q, r) = core(f)$ and $p \in subtree(q)$ is received, or if the same INITIATE message headed toward p is received. Since $(s', \pi) \notin \Psi_\varphi$, π is neither of these actions.

Can the path from u to p change, where an INITIATE$(l, c,$find$)$ is in $queue(\langle t, u \rangle)$ headed toward p? By definition of headed toward and HI-A and HI-B, there is a unique path from u to p in s'. Since HI-A and HI-B are also true in s and since the minimum spanning tree is unique (by Lemma 10), the same unique path from u to p exists in s.

The number of messages in the same queue as the relevant INITIATE message but not behind it cannot change, because the queues are FIFO (and $(s', \pi) \notin \Psi_\varphi$).

(b) It is easy to check that $v_\varphi(s) < v_\varphi(s')$ if $(s', \pi) \in \Psi_\varphi$.

(c) No action ψ such that $(s', \psi) \in \Psi_\varphi$ can become disabled in s without occurring, since the queues are FIFO.

vi) φ is ComputeMin(f). We show that the hypotheses of Lemma 7 are satisfied to get the result.

Let $A = GHS$, $B = TAR$, $C = DC$, $D = GC$, and $\rho = ComputeMin(f)$ in the hypotheses of Lemma 7.

(1) If e is an execution of GHS, then by Lemmas 1 and 25, $\mathcal{M}_{DC}(e)$ is an execution of DC.

(2) Let s be a reachable state of TAR. If φ is enabled in $\mathcal{S}_{TAR}(s)$, then as argued in Section 4.2.3 (TAR to GC), φ is enabled in $\mathcal{S}_3(\mathcal{S}_{TAR}(s))$. By the way the $\mathcal{S}$'s are defined, $\mathcal{S}_3(\mathcal{S}_{TAR}(s)) = \mathcal{S}_4(\mathcal{S}_{DC}(s))$, so $\rho = \varphi$ is enabled in $\mathcal{S}_4(\mathcal{S}_{DC}(s))$.

(3) Suppose (s', π, s) is a step of GHS and s' is reachable. If φ is not in $\mathcal{A}_{TAR}(s', \pi)$, then ρ is not in $\mathcal{M}_4(\mathcal{M}_{DC}(s'\pi s))$ by inspection.

(4) DC is progressive for ρ via $\mathcal{M}_4$, using Ψ_ρ and v_ρ, by Lemma 30.

(5) Let ψ be such that $(t, \psi) \in \Psi_\rho$ for some t. Possible values of ψ are $ChannelSend(l, \text{REPORT}(w))$, $ChannelRecv(l, \text{REPORT}(w))$, and $ReceiveReport(l, w)$. Essentially the same arguments as in *ii*), *iii*) and *iv*) show that GHS is progressive for ψ.

vii) φ is ChangeRoot(f) and subtree(f) is not {p} for any p. We show that the hypotheses of Lemma 7 are satisfied to get the result.

Let $A = GHS$, $B = TAR$, $C = CON$, $D = COM$, and $\rho = ChangeRoot(f)$ in the hypotheses of Lemma 7.

(1) If e is an execution of GHS, then by Lemmas 1 and 25, $\mathcal{M}_{CON}(e)$ is an execution of DC.

(2) Let s be a reachable state of TAR. Suppose φ is enabled in $\mathcal{S}_{TAR}(s)$. As argued in Section 4.2.3 (TAR to GC), φ is enabled in $\mathcal{S}_3(\mathcal{S}_{TAR}(s))$. As argued in Section 4.2.2 (GC to COM), φ is enabled in $\mathcal{S}_2(\mathcal{S}_3(\mathcal{S}_{TAR}(s)))$. By the way the $\mathcal{S}$'s are defined, $\mathcal{S}_2(\mathcal{S}_3(\mathcal{S}_{TAR}(s))) = \mathcal{S}_6(\mathcal{S}_{CON}(s))$, so $\rho = \varphi$ is enabled in $\mathcal{S}_6(\mathcal{S}_{CON}(s))$.

(3) Suppose (s', π, s) is a step of GHS and s' is reachable. If φ is not in $\mathcal{A}_{TAR}(s', \pi)$, then ρ is not in $\mathcal{M}_6(\mathcal{M}_{CON}(s'\pi s))$ by inspection.

(4) CON is progressive for ρ via $\mathcal{M}_6$, using Ψ_ρ and v_ρ, by Lemma 31.

(5) Let ψ be such that $(t, \psi) \in \Psi_\rho$ for some t. Possible values of ψ are $ChannelSend(l, \text{CHANGEROOT})$, $ChannelRecv(l, \text{CHANGEROOT})$, and *Receive-ChangeRoot*(l). Essentially the same arguments as in *ii*), *iii*) and *iv*) show that GHS is progressive for ψ.

viii) φ is ChangeRoot(f), subtree(f) is {p} for some p. We show that GHS is progressive for φ via $\mathcal{M}_{TAR}$. Lemma 6 gives the result.

Let Ψ_φ be the set of all pairs (s, ψ) of reachable states s of GHS and internal actions ψ of GHS enabled in s such that none of the following is true:

- $\psi = ReceiveConnect(\langle q, r \rangle, l)$ for some q, r and l, and in s, $nstatus(r) \neq$ sleeping, $l \geq nlevel(r)$, $lstatus(\langle r, q \rangle) =$ unknown, and only one message is in $queue_r(\langle q, r \rangle)$.
- $\psi = ReceiveTest(\langle q, r \rangle, l, c)$ for some q, r, l and c, and in s, $nstatus(r) \neq$ sleeping, $l > nlevel(r)$, and only one message is in $queue_r(\langle q, r \rangle)$.

- $\psi = ReceiveReport(\langle q,r\rangle, w)$ for some q, r and w, and in s, $inbranch(r) = \langle q,r\rangle$, $nstatus(r) =$ find, and only one message is in $queue_r(\langle q,r\rangle)$.

For reachable state s, let $v_\varphi(s)$ be the following tuple:

1. The number of fragments in s.
2. The number of fragments g with $rootchanged(g) =$ false in s.
3. The number of fragments g with $minlink(g) = nil$ in s.
4. The number of nodes $q \in V(G)$ such that $q \in testset(fragment(q))$.
5. The summation over all $q \in V(G)$ of $level(fragment(q)) - nlevel(q)$.
6. The summation over all $q \in V(G)$ of $findcount(q)$.
7. The number of links $\langle q,r\rangle$ such that either $lstatus(\langle q,r\rangle) =$ unknown, or else $lstatus(\langle q,r\rangle) =$ branch and there is a protocol message for $\langle q,r\rangle$.
8. The number of links $\langle q,r\rangle$ such that no ACCEPT or REJECT is in $queue(\langle q,r\rangle)$.
9. The summation over all fragments g such that a CHANGEROOT is in some $queue(\langle q,r\rangle)$ of $subtree(g)$ of the number of nodes in the path in $subtree(g)$ from r to $minnode(g)$.
10. The number of fragments g such that $AfterMerge(q,r)$ for DC is enabled for some $q \in nodes(g)$.
11. The number of messages in $queue_q(\langle q,r\rangle)$, for all $\langle q,r\rangle \in L(G)$.
12. The number of messages in $queue_{qr}(\langle q,r\rangle)$, for all $\langle q,r\rangle \in L(G)$.
13. The number of messages in $queue_r(\langle q,r\rangle)$, for all $\langle q,r\rangle \in L(G)$.
14. The number of messages in $queue_r(\langle q,r\rangle)$ that are behind a CONNECT or TEST, for all $\langle q,r\rangle \in L(G)$.

(1) Let s be a reachable state of GHS in E_φ. We now demonstrate that some action ψ is enabled in s with $(s,\psi) \in \Psi_\varphi$.

By preconditions of φ, $awake =$ true, $minlink(f) \neq nil$ and $rootchanged(f) =$ false in s. By GHS-K, $nstatus(p) =$ true in s. But since $awake =$ true, there is some node q such that $nstatus(q) \neq$ sleeping. Thus A, the set of all fragments g such that $nstatus(q) \neq$ sleeping for some $q \in nodes(g)$, is non-empty. Let l be the minimum level of all fragments in A, and let $A_l = \{g \in A : level(g) = l\}$.

The strategy is to use a case analysis as follows. For each case, we show that there is some $queue(\langle q,r\rangle)$ with some message m in it in s. Let ψ be chosen as follows. If some message m' is at the head of $queue_q(\langle q,r\rangle)$, let $\psi = ChannelSend(\langle q,r\rangle, m')$. If no message is in $queue_q(\langle q,r\rangle)$ and some message m' is at the head of $queue_{qr}(\langle q,r\rangle)$, let $\psi = ChannelSend(\langle q,r\rangle, m')$. If no message is in $queue_q(\langle q,r\rangle)$ or $queue_{qr}(\langle q,r\rangle)$, then at least one message, namely m, is in $queue_r(\langle q,r\rangle)$; let $\psi = Receive(\langle q,p\rangle, m')$, where m' is the message at the head of $queue_r(\langle q,r\rangle)$.

For each choice, ψ is obviously enabled in s. There are two methods to verify that $(s,\psi) \in \Psi_\varphi$. Method 1 is to show that m is not CONNECT, TEST or REPORT. Then, if $\psi = Receive(\langle q,r\rangle, m')$ and m' is CONNECT, TEST or REPORT, there is more than one message in $queue_r(\langle q,r\rangle)$. Method 2 is to show that some variable in s has a value such that even if $\psi = Receive(\langle q,r\rangle, m')$, where m' is CONNECT, TEST or REPORT, we have that $(s,\psi) \in \Psi_\varphi$.

Case 1: There is a fragment $g \in A_l$ with $testset(g) \neq \emptyset$. Let q be some element of $testset(g)$. By definition of $testset(g)$, Cases 1.1, 1.2 and 1.3 are exhaustive.

Case 1.1: A CONNECT(l) message is in $queue(r,t)$, where $(r,t) = core(g)$ and $q \in subtree(r)$ in s. We use Method 2. By COM-F, $(r,t) \in subtree(g)$, so by TAR-A(b), $lstatus(\langle t,r\rangle) =$ branch.

Case 1.2: An INITIATE(l, c,find) message is in some $queue(\langle r,t\rangle)$ headed toward q in s. By Method 1, we are done.

Case 1.3: $testlink(q) \neq nil$ in s. By TAR-C(a), $testlink(q) = \langle q,r\rangle$ for some r. By TAR-C(c), there is a protocol message for $\langle q,r\rangle$.

Case 1.3.1: The protocol message is an ACCEPT or REJECT in $queue(\langle r,q\rangle)$. By Method 1, we are done.

Case 1.3.2: The protocol message is TEST(l', c) in $queue(\langle q,r\rangle)$. Thus $lstatus$ $(\langle q,r\rangle) \neq$ rejected. By TAR-E(b), $l' = l$. If $nstatus(r) =$ sleeping or $l \leq nlevel(r)$, we are done, by Method 2. Suppose $nstatus(r) \neq$ sleeping and $l > nlevel(r)$. By definition of A_l, $l \leq level(fragment(r))$, and thus $nlevel(r) < level(fragment(r))$. By NOT-G, either a NOTIFY($level(fragment(r))$) message is in some $queue(\langle t,u\rangle)$ headed toward r, in which case we are done by Method 1, or $AfterMerge(t,u)$ is enabled for NOT, with $r \in subtree(u)$. In the latter case, by GHS-L, a CONNECT is at the head of $queue(\langle u,t\rangle)$; the same argument as in Case 1.1 gives the result.

Case 2: $testset(g) = \emptyset$ for all $g \in A_l$.

Case 2.1: There is a fragment g in A_l with $minlink(g) = nil$. Since $g \neq f$ and G is connected, there is an external link of g. Since $testset(g) = \emptyset$, by DC-D(c) no FIND message is in $subtree(g)$.

Suppose $dcstatus(q)$ = unfind for all $q \in nodes(g)$. By definition of $minlink(g)$, a REPORT message is in some $queue(\langle q, r\rangle)$ headed toward $mw\text{-}root(g)$. We are done by Method 2.

Suppose $dcstatus(q)$ = find for some $q \in nodes(g)$. By DC-I(b), since $testset(g) = \emptyset$, a REPORT message is in some $queue(\langle r, t\rangle)$ in $subtree(q)$ headed toward q. By DC-B(a), $inbranch(t) \neq \langle t, r\rangle$. We are done by Method 2.

Case 2.2: $minlink(g) \neq nil$ for all $g \in A_l$.

Case 2.2.1: There is a fragment g in A_l with $rootchanged(g)$ = false. By GHS-K, if $subtree(g) = \{q\}$ for some q, then $nstatus(q)$ = sleeping. By definition of A_l, $subtree(g) \neq \{q\}$ for any q. By CON-B, a CHANGEROOT message is in some $queue(\langle q, r\rangle)$ in $subtree(g)$. We are done by Method 1.

Case 2.2.2: $rootchanged(g)$ = true for all $g \in A_l$. By CON-D, a CONNECT message is in $queue(minlink(g))$ for all $g \in A_l$.

Case 2.2.2.1: There is a fragment g in A_l with $minlink(g) = \langle q, r\rangle$ and $level(fragment(r)) > l$.

If $nlevel(r) > l$, we are done by Method 2. Suppose $nlevel(r) \leq l$. Essentially the same argument as in Case 1.3(b) gives the result.

Case 2.2.2.2: For all fragments g in A_l, $level(fragment(target(minlink(g)))) \leq l$. By COM-A, $level(fragment(target(minlink(g)))) = l$ for all $g \in A_l$.

Case 2.2.2.2.1: There is a fragment g in A_l such that $minlink(g) = \langle q, r\rangle$, and $fragment(r) \notin A_l$. By definition of A_l, $nstatus(r)$ = sleeping, and we are done be Method 2.

Case 2.2.2.2.2: For all fragments g in A_l, $fragment(target(minlink(g))) \in A_l$. As argued in Lemma 27, Case 2.2.2 of verifying (1) for $\varphi = Combine$, there are two fragments g and h in A_l such that $minedge(g) = minedge(h) = (q, r)$. By TAR-H, $lstatus(\langle r, q\rangle) = lstatus(\langle q, r\rangle)$ = branch. By Method 2, we are done.

---

(2) Let (s', π, s) be a step of GHS, where s' is reachable and in E_φ, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.

(a) We show that if $(s', \pi) \notin \Psi_\varphi$, then $v_\varphi(s) = v_\varphi(s')$; together with part (b) below, this gives the result. Ψ_φ is defined to include all the state-action pairs that change the state. Thus, if $(s', \pi) \notin \Psi_\varphi$, then $s = s'$, and obviously $v_\varphi(s) = v_\varphi(s')$.

(b) Suppose $(s, \pi) \in \Psi_\varphi$. The breakdown of cases in this argument is essentially the same as in the proof of the safety step simulations in Lemma 25. The notation "Component 12" in a case means that component 12 of v_φ decreases in going from s' to s, and components 1 through 11 are unchanged.

- $\pi = \mathit{ChannelSend}(\langle q, r\rangle, m)$. Component 11.
- $\pi = \mathit{ChannelRecv}(\langle q, r\rangle, m)$. Component 12.
- $\pi = \mathit{ReceiveConnect}(\langle q, r\rangle, l)$.

Case 1: $\mathit{nstatus}(r) =$ sleeping in s'. If $\langle q, r\rangle$ is not the minimum-weight external link of r, then: component 2. Otherwise, component 1.

Case 2: $\mathit{nstatus}(r) \neq$ sleeping, $l = \mathit{nlevel}(r)$ and no CONNECT is in $\mathit{queue}(\langle r, q\rangle)$ in s'.

Suppose $\mathit{lstatus}(\langle r, q\rangle) =$ unknown. Since $(s', \pi) \in \Psi_\varphi$, another message is in $\mathit{queue}(\langle q, r\rangle)$. By CON-D, CON-E and GHS-C, the other message is not a CONNECT or TEST. Component 14.

Suppose $\mathit{lstatus}(\langle r, q\rangle) \neq$ unknown. Since DC simulates $\mathit{AfterMerge}(r, q)$, neither $\mathit{AfterMerge}(r, q)$ nor $\mathit{AfterMerge}(q, r)$ is enabled in s. Component 10.

Case 3: $\mathit{nstatus}(r) \neq$ sleeping, $l = \mathit{nlevel}(r)$, and CONNECT is in $\mathit{queue}(\langle r, q\rangle)$ in s'. Component 1.

Case 4: $\mathit{nstatus}(r) \neq$ sleeping and $l < \mathit{nlevel}(r)$ in s'. Component 1.

- $\pi = \mathit{ReceiveInitiate}(\langle q, r\rangle, l, c, st)$. By NOT-H(a), $l > \mathit{nlevel}(r)$. Component 5.
- $\pi = \mathit{ReceiveTest}(\langle q, r\rangle, l, c)$. Let $g = \mathit{fragment}(r)$.

Case 1: $\mathit{nstatus}(r) =$ sleeping in s'. Component 2.

Case 2: $\mathit{nstatus}(r) \neq$ sleeping in s'.

Case 2.1: $l \leq level(g)$, and either $c \neq core(g)$ or $testlink(r) \neq \langle r,q \rangle$ in s'. If an ACCEPT is added, then component 8. If a REJECT is added, then either component 7 or component 8.

Case 2.2: $l \leq level(g)$, $c = core(g)$, and $testlink(r) = \langle r,q \rangle$ in s'. If there is no link $\langle r,t \rangle$, $t \neq q$, with $lstatus(\langle r,t \rangle) =$ unknown, then component 4. If there is such a link, then component 7.

Case 2.3: $l > level(g)$ in s'. Since $(s,\pi) \in \Psi_\varphi$, there is another message in $queue_r(\langle q,r \rangle)$. By TAR-C(c) and GHS-C, the other message is not CONNECT or TEST. Component 14.

- $\pi = ReceiveAccept(langleq, r\rangle)$. Component 4.
- $\pi = ReceiveReject(\langle q,r \rangle)$. If there is no link $\langle r,t \rangle$, $t \neq q$, with $lstatus(\langle r,t \rangle) =$ unknown, then component 4. If there is such a link, then component 7.
- $\pi = ReceiveReport(\langle q,r \rangle, w)$.

Case 1: $(q,r) = core(g)$, $nstatus(r) \neq$ find and $w > bestwt(r)$ in s'. If $lstatus(bestlink(r)) =$ branch, then component 3. Otherwise, component 2.

Case 2a: $(q,r) \neq core(g)$ in s'. If $inbranch(r) = \langle r,q \rangle$, then component 13. Otherwise, component 6.

Case 2b: $(q,r) = core(g)$ and $nstatus(r) =$ find in s'. The only change is that the REPORT message is requeued. We show that there is no other message in $queue(\langle q,r \rangle)$, and thus $(s',\pi) \notin \Psi_\varphi$. First note that by COM-F, $(q,r) \in subtree(g)$. By GHS-B, no CONNECT is in the queue. By DC-O, no INITIATE($*,*$,found) is in the queue. By GHS-E, no INITIATE($*,*$,find) is in the queue. By TAR-E(a), no TEST or REJECT is in the queue. By DC-O, no other REPORT is in the queue. By TAR-F, no ACCEPT is in the queue. By CON-C, no CHANGEROOT is in the queue.

Case 2c: $(q,r) = core()$, $nstatus(r) =$ unfind, and $w \leq bestwt(p)$. Component 13.

- $\pi = ReceiveChangeRoot(\langle q,r \rangle)$. If $lstatus(bestlink(r)) \neq$ branch, then component 2. Otherwise, component 9.

(c) Suppose $(s',\pi) \notin \Psi_\varphi$, ψ is enabled in s', and $(s',\psi) \in \Psi_\varphi$. Since $(s',\pi) \notin \Psi_\varphi$, $s = s'$. Obviously, ψ is enabled in s and $(s,\psi) \in \Psi_\varphi$.

ix) φ is Merge(f,g). We use Lemma 7. The same argument as in *vii*), with $\rho = Merge(f, g)$ and (3) as below, gives the result.

(3) Let ψ be such that $(t, \psi) \in \Psi_\rho$ for some t. Possible values of ψ are $ChannelSend(k$, CONNECT$(l))$, $ChannelRecv(k$, CONNECT$(l))$, and $Merge(f, g)$. Essentially the same arguments as in *ii*), *iii*) and *iv*) show that GHS is progressive for ψ.

x) φ is Absorb(f,g). We use Lemma 7. The same argument as in *vii*), with $\rho = Absorb(f, g)$ and (3) as below, gives the result.

(3) Let ψ be such that $(t, \psi) \in \Psi_\rho$ for some t. Possible values of ψ are $ChannelSend(k$, CONNECT$(l))$, $ChannelRecv(k$, CONNECT$(l))$, and $Absorb(f, g)$. Essentially the same arguments as in *ii*), *iii*) and *iv*) show that GHS is progressive for ψ. □

4.4 Satisfaction

Theorem 33: *GHS solves $MST(G)$.*

Proof: By Theorem 12, HI solves $MST(G)$. By Lemmas 13 and 27 and Theorem 8, COM satisfies HI. By Lemmas 15 and 28 and Theorem 8, GC satisfies COM. By Lemmas 17 and 29 and Theorem 8, TAR satisfies GC. By Lemmas 25 and 32 and Theorem 9, GHS satisfies TAR. Thus, since "satisfies" and "solves" are defined using subsets of schedules, GHS solves $MST(G)$. □

Acknowledgments

We thank Yehuda Afek, Steve Garland, Michael Merritt, Liuba Shrira and members of the Theory of Distributed Systems research group at MIT for valuable discussions.

References

[A1] B. Awerbuch, "Complexity of Network Synchronization," *JACM* vol. 32, no. 4, pp. 804–823, 1985.

[A2] B. Awerbuch, "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and Related Problems," *Proc. 19^{th} Ann. ACM Symp. on Theory of Computing*, pp. 230–240, 1987.

[AG] B. Awerbuch and R. Gallager, "Distributed BFS Algorithms," *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 250–256, 1985.

[AS] B. Alpern and F. Schneider, "Proving Bool-ean Combinations of Deterministic Properties," *Proc. 2nd Ann. Symp. on Logic in Computer Science*, pp. 131–137, 1987.

[CT] F. Chin and H. F. Ting, "An Almost Linear Time and $O(n \log n + e)$ Messages Distributed Algorithm for Minimum-Weight Spanning Trees," *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 257–266, 1985.

[EF] T. Elrad and N. Francez, "Decomposition of Distributed Programs into Communication-Closed Layers," *Science of Computer Programming*, vol. 2, no. 3, pp. 155–173, December 1982.

[F] N. Francez, *Fairness*, Springer-Verlag, New York, 1986, Chapter 2.

[FLS] A. Fekete, N. Lynch, L. Shrira, "A Modular Proof of Correctness for a Network Synchronizer," *Proc. 2nd International Workshop on Distributed Algorithms*, 1987.

[G] E. Gafni, "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms," *Proc. 4th Ann. ACM Symp. on Principles of Distributed Computing*, pp. 175–185, 1985.

[GHS] R. Gallager, P. Humblet and P. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Trans. on Programming Languages and Systems*, vol. 5, no. 1, pp. 66–77, 1983.

[H] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987.

[K] R. Kurshan, "Reducibility in Analysis of Coordination," *Proc. IIASA Workshop on Discrete Event Systems*, 1987.

[L] L. Lamport, "Specifying Concurrent Program Modules," *ACM Trans. on Programming Languages and Systems*, vol. 5, no. 2, pp. 190–222, April 1983.

[LM] N. Lynch and M. Merritt, "Introduction to the Theory of Nested Transactions," to appear in *Theoretical Computer Science*. (Also available as technical report MIT/LCS/TR-367, Laboratory for Computer Science, Massachusetts Institute of Technology, 1986.)

[LPS] D. Lehmann, A. Pnueli, and J. Stavi, "Impartiality, Justice and Fairness: The Ethics of Concurrent Termination," *Proc. 8^{th} International Colloquium on Automata, Languages and Programming*, pp. 264–277, July 1981.

[LSc] L. Lamport and F. Schneider, "The 'Hoare Logic' and All That," *ACM Trans. on Programming Languages and Systems*, vol. 6, no. 2, pp. 281–296, April 1984.

[LSh] S. Lam and U. Shankar, "Protocol Verification via Projections," *IEEE Trans. on Software Engineering*, vol. SE-10, no. 4, pp. 325–342, July 1984.

[LT] N. Lynch and M. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," *Proc. 6^{th} Ann. ACM Symp. on Principles of Distributed Computing*, pp. 137–151, 1987. (Also available as technical report MIT/LCS/ TR-387, Laboratory for Computer Science, Massachusetts Institute of Technology, 1987.)

[MP] Z. Manna and A. Pnueli, "Verification of Concurrent Programs: Temporal Proof Principles," in D. Kozen, editor, *Logic of Programs, Lecture Notes in Computer Science* 131, pp. 200–252, Springer-Verlag, Berlin, 1981.

[OG] S. Owicki and D. Gries, "An Axiomatic Proof Technique for Parallel Programs I," *Acta Informatica*, vol. 6, no. 4, pp. 319–340, August 1976.

[S] E. Stark, "Foundations of a Theory of Specification for Distributed Systems," Ph.D. thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, August 1984. (Available as technical report MIT/LCS/TR-342.)

[SR] F. Stomp and W. de Roever, "A Correctness Proof of a Distributed Minimum-Weight Spanning Tree Algorithm," *Proc. 7^{th} International Conference on Distributed Computing Systems*, pp. 440–447, 1987.

[W] J. Welch, "Topics in Distributed Computing: The Impact of Partial Synchrony, and Modular Decomposition of Algorithms," Ph.D. thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, March 1988. (To appear as MIT/LCS technical report.)

Appendix

In this Appendix, we review the aspects of the model from [LT] that are relevant to this paper.

An *input-output automaton* A is defined by the following four components. (1) There is a (possibly infinite) set of *states* with a subset of *start states*. (2) There is a set of *actions*, associated with the state transitions. The actions are divided into three classes, *input*, *output*, and *internal*. Input actions are presumed to originate in the automaton's environment; consequently the automaton must be able to react to them no matter what state it is in. Output and internal actions (or, *locally-controlled* actions) are under the local control of the automaton; internal actions model events not observable by the environment. The input and output actions are the *external* actions of A, denoted $ext(A)$. (3) The transition relation is a set of (state, action, state) triples, such that for any state s' and input action π, there is a transition (s', π, s) for some state s. (4) There is an equivalence relation $part(A)$ partitioning the output and internal actions of A. The partition is meant to reflect separate pieces of the system being modeled by the automaton. Action π is *enabled* in state s' if there is a transition (s', π, s) for some state s.

An *execution* e of A is a finite or infinite sequence $s_0\pi_1 s_1 \ldots$ of alternating states and actions such that s_0 is a start state, (s_{i-1}, π_i, s_i) is a transition of A for all i, and if e is finite then e ends with a state. The *schedule* of an execution e is the subsequence of actions appearing in e.

We often want to specify a desired behavior using a set of schedules. Thus we define an *external schedule module* S to consist of input and output actions, and a set of schedules $scheds(S)$. Each schedule of S is a finite or infinite sequence of the actions of S. Internal actions are excluded in order to focus on the behavior visible to the outside world. External schedule module S' is a *sub-schedule module* of external schedule module S if S and S' have the same actions and $scheds(S') \subseteq scheds(S)$.

Automata can be composed to form another automaton, presumably modeling a system made of smaller components. Automata communicate by synchronizing on shared actions; the only allowed situations are for the output from one automaton to be the input to others, and for several automata to share an input. Thus, automata to be composed must have no output actions in common, and the internal actions of each must be disjoint from all the actions of the others. A state of the composite automaton is a tuple of states, one for each component. A start state of the composition has a start state in each component of the state. Any output action of a component becomes an output action of the composition, and similarly for an internal action. An input action of the composition is an action that is input for every component for which it is an action. In a transition of the composition on action π, each component of the state changes as it would in the component automaton if π occurred; if π is not an action of some component automaton, then the corresponding state component does not change. The partition of the composition is the union of the partitions of the component automata.

Given an automaton A and a subset Π of its actions, we define the automaton $Hide_{\Pi}(A)$ to be the automaton A' differing from A only in that each action in Π becomes an internal action. This operation is useful for hiding actions that model interprocess communication in a composite automaton, so that they are no longer visible to the environment of the composition.

An execution of a system is fair if each component is given a chance to make progress infinitely often. Of course, a process might not be able to take a step every time it is given a chance. Formally stated, execution e of automaton A is *fair* if for each class C of $part(A)$, the following two conditions hold. (1) If e is finite, then no action of C is enabled in the final state of e. (2) If e is infinite, then either actions from C appear infinitely often in e, or states in which no action of C is enabled appear infinitely often in e. Note that any finite execution of A is a prefix of some fair execution of A.

The *fair behavior* of automaton A, denoted $Fairbehs(A)$, is the external schedule module with the input and output actions of A, and with the set of schedules $\{\alpha | ext(A) : \alpha \text{ is the schedule of a fair execution of } A\}$.[1] A *problem* is (specified by) an external schedule module. Automaton A *solves* the problem P if $Fairbehs(A)$ is a sub-schedule module of P, i.e., the behavior of A visible to the outside world is consistent with the behavior required in the problem specification. Automaton A *satisfies* automaton B if $Fairbehs(A)$ is a sub-schedule module of $Fairbehs(B)$.

---

[1] If α is a sequence from a set S and T is a subset of S, then $\alpha | T$ is defined to be the subsequence of α consisting of elements in T.