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

Jennifer Lundelius Welch
Leslie Lamport
Nancy Lynch

June 1988
DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.
**A Lattice-Structured Proof Technique Applied to a Minimum Spanning Tree Algorithm**

**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.
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 Shira [FLS] to prove the correctness of Awerbuch's synchronizer [Al].

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
Section 1: Introduction

(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 \( Q \land Q_1 \land \cdots \land Q_i \) 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 au-
tomaton 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 di-
rectly. Assuming that $Q_1, \ldots, Q_j$ are invariants for the versions $B_1, \ldots, B_i$
directly above $A$ in the lattice, we verify that predicate $P = Q \land Q_1 \land \cdots \land Q_i$
is invariant for $A$, by demonstrating mappings that preserve $Q$ and take executions of $A$ to execu-
tions of $B_1, \ldots, B_i$ (thus preserve $Q_1 \land \cdots \land Q_i$). (Finding these mappings requires
Section 2: Foundations

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, \( A_1, A_2, \ldots \), 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 \( \varphi \) of \( B \), a definition of \( A \) being equitable for \( \varphi \) 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 \( \varphi \) 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 \( \varphi \) of \( B \) are described. The first method is to show that there is an action \( \rho \) of \( A \) that is enabled whenever \( \varphi \) is, and whose occurrence implies \( \varphi \)'s occurrence. (Cf. Lemma 5.)

The second method uses a definition of \( A \) being progressive for \( \varphi \). The intu-
Section 2.1: Safety

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 $\varphi$ of $B$, we demonstrate an action $\rho$ of $D$ that is "similar" to $\varphi$, such that $C$ is progressive for $\rho$ using a set $\Psi$ of helping actions, and $A$ is equitable for all the helping actions in $\Psi$. (Cf. Lemma 7.)

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (D) {$D$};
  \node (rho) [below of=D] {$\rho$};
  \node (varphi) [left of=rho] {$\varphi$};
  \node (B) [below of=varphi] {$B$};
  \node (C) [right of=B] {$C$};
  \node (Psi) [right of=C] {$\Psi$};
  \node (A) [below of=C] {$A$};
  \draw (D) -- (rho);
  \draw (varphi) -- (B);
  \draw (B) -- (C);
  \draw (C) -- (Psi);
  \draw (A) -- (B);
\end{tikzpicture}
\caption{Figure 1}
\end{figure}

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 $\text{alt-seq}(B)$ 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 $\text{states}(A)$ to $\text{states}(B)$ and $\mathcal{A}$ maps pairs $(s, \pi)$, of states $s$ of $A$ and actions $\pi$ of $A$ enabled in $s$, to $\text{alt-seq}(B)$.
Section 2.1: Safety

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 \( \text{states}(B) \), and \( S \) maps \( \text{states}(A) \) to \( \text{states}(B) \), then \( (Q \circ S) \), applied to state \( s \) of \( A \), is the predicate “\( Q \) is true in \( S(s) \),” and is also written \( (Q(S(s))) \).) 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 \( \mathcal{M} = (S, \mathcal{A}) \) be an abstraction mapping from \( A \) to \( B \). \( P \) be a predicate on \( \text{states}(A) \), 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 \( \text{start}(A) \), then
   a. \( P(s) \) is true, and
Section 2.1: Safety

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

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

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

The first lemma verifies that if $A$ simulates $B$ via $M = (S, A)$, $P$ and $Q$, then $M(e)$ is an execution of $B$ and a certain predicate is true of all states of $e$.

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

(1) $M(e)$ is an execution of $B$.

(2) $(Q \circ S) \land P$ 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) $M(e_i)$ is an execution of $B$ and $S(s_i) = t$, where $t$ is the final state in $M(e_i)$.

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

**Basis:** $i = 0$. (1) $M(e_0) = S(s_0)$. Since $e_0$ is an execution of $A$, $s_0$ is in $\text{start}(A)$. Condition (1b) implies that $S(s_0)$ is in $\text{start}(B)$, so $M(e_0)$ 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 $S(s_0)$ is in $\text{start}(B)$, it is a reachable state of $B$, and $Q(S(s_0))$ is true.

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

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

Section 2.1: Safety

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

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

(2) By the inductive hypothesis for (2), $(Q \circ S) \land P$ is true in every state of $\epsilon_i$, except (possibly) $s_i$. By condition (3a), $P(s_i)$ is true. The final state in $M(\epsilon_i)$ is $S(s_i)$. Since, by part (1), $M(\epsilon_i)$ is an execution of $B$ and $S(s_i)$ equals the final state of $M(\epsilon_i)$, $S(s_i)$ is a reachable state of $B$. By definition of $Q$, $Q(S(s_i))$ 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 $M_r = (S_r, A_r)$ 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 $\text{states}(A)$. We say $A$ simultaneously simulates $\{A_r : r \in I\}$ via $\{M_r : r \in I\}$, $P$, and $\{Q_r : r \in I\}$ if the following three conditions are true.

1. If $s$ is in $\text{start}(A)$, then
   a. $P(s)$ is true, and
   b. $S_r(s)$ is in $\text{start}(A_r)$ for all $r \in I$.

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

9
Section 2.2: Liveness

(3) Let \((s', \pi, s)\) be a step of \(A\) such that \(\bigwedge_{r \in I} Q_r(S_r(s'))\) and \(P(s')\) are true. Then

(a) \(P(s)\) is true,
(b) if \(A_r(s', \pi)\) is empty, then \(S_r(s) = S_r(s')\), for all \(r \in I\), and
(c) if \(A_r(s', \pi) = \varphi_1 t_1 \ldots t_{m-1} \varphi_m\), then \(S_r(s') \varphi_1 t_1 \ldots t_{m-1} \varphi_m S_r(s)\) is an execution fragment of \(A_r\), for all \(r \in I\).

The statement \("A\) simultaneously simulates \(\{A_1, A_2\}\) via \(\{M_1, M_2\}\), \(P\) and \(\{Q_1, Q_2\}\)" is weaker than the statement \("A\) simulates \(A_1\) via \(M_1\), \(P\) and \(Q_1\), and \(A\) simulates \(A_2\) via \(M_2\), \(P\) 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 \(\{M_r : r \in I\}\), \(P\) and \(\{Q_r : r \in I\}\), where \(M_r = (S_r, A_r)\) for all \(r \in I\), then the following are true of any execution \(e\) of \(A\).

1. \(M_r(e)\) is an execution of \(A_r\), for all \(r \in I\).
2. \(\bigwedge_{r \in I} (Q_r \circ S_r) \land P\) 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 \(e = s_0 \pi_1 s_1 \ldots\) of automaton \(A\) to satisfy \(S \hookrightarrow (T, X)\), where \(S\) and \(T\) are subsets of \(\text{states}(A)\) and \(X\) is a subset of \(\text{states}(A) \times \text{acts}(A)\), 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 \(M = (S, A)\) is an abstraction mapping from \(A\) to \(B\), then for each locally-controlled action \(\varphi\) of \(B\), we make the following definitions:

- \(E_\varphi\) is the set of all states \(s\) of \(A\) such that \(\varphi\) is enabled in \(S(s)\);
- \(D_\varphi\) is \(\text{states}(A) - E_\varphi\);
- \(D'_{\varphi}\) is the set of all states \(t\) of \(B\) such that \(\varphi\) is not enabled in \(t\);
- \(X_\varphi\) is the set of all pairs \((s, \pi)\) of states \(s\) of \(A\) and actions \(\pi\) of \(A\) such that \(\varphi\) is in \(A(s, \pi)\);
- \(X'_\varphi\) is \(\text{states}(B) \times \{\varphi\}\).

**Definition:** Suppose \(M\) is an abstraction mapping from \(A\) to \(B\). Let \(\varphi\) be a locally-controlled action of \(B\). If every fair execution of \(A\) satisfies \(\text{states}(A) \hookrightarrow (D_\varphi, X_\varphi)\), then \(A\) is \(\textit{equitable}\) for \(\varphi\) via \(M\). If \(A\) is equitable for \(\varphi\) via \(M\) for every locally-controlled action \(\varphi\) of \(B\), then \(A\) is \(\textit{equitable}\) for \(B\).
Section 2.2: Liveness

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

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

Proof: Let \( \mathcal{M} = (S, A) \). Let \( e = s_0 a_1 s_1 \ldots \) be a fair execution of A, and let \( \mathcal{M}(e) = t_0 \phi_1 t_1 \ldots \). For any \( i \geq 0 \), define \( \text{index}(i) \) to be \( j \) such that \( \mathcal{M}(s_0 \ldots s_i) = t_0 \ldots t_j \). Choose \( i \geq 0 \).

Case 1: \( t_i \) is not interpolated. Choose any \( l \) be such that \( \text{index}(l) = i \). Then \( t_i = S(s_i) \), as argued in the proof of Lemma 1. Suppose there is an \( m \geq l \) such that \( s_m \in D_{\phi} \). Then there is a \( j = \text{index}(m) \geq i \) such that \( t_j = S(s_m) \), and by definition of \( D_{\phi} \), \( t_j \) is in \( D'_{\phi} \). Suppose there is an \( m \geq l \) such that \( (s_m, \pi_{m+1}) \in X_{\phi} \). Then there is a \( j = \text{index}(m) \geq i \) such that \( \phi_j = \phi \), by definition of \( X_{\phi} \), and \( (t_j, \phi_{j+1}) \) is in \( X'_{\phi} \).

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 \( D'_{\phi} \) or \( \phi \) 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 \( D'_{\phi} \) or \( \phi \) occurs in \( \mathcal{M}(e) \).

The next lemma is the analog of Lemma 3 for simultaneously simulates. (\( D'_{\phi} \) and \( X'_{\phi} \) 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 \( \phi \) be a locally-controlled action of \( A_r \) for some \( r \). If A is equitable for \( \phi \) via \( \mathcal{M}_r \), then \( \mathcal{M}_r(e) \) satisfies \( \text{states}(B) \rightarrow (D'_{\phi}, X'_{\phi}) \), for every fair execution \( e \) of A.

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

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

(1) \( \rho \) is enabled in \( s \) if and only if \( \phi \) is enabled in state \( S(s) \) of B, and

(2) if \( \rho \) is enabled in \( s \), then \( \phi \) is included in \( A(s, \rho) \).
Section 2.2: Liveness

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

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

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

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

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

2. For any step $(s', \pi, s)$ of $A$, where $s'$ is reachable and in $E_\varphi$, $(s', \pi) \notin X_\varphi$, 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 $(s', \pi) \notin \Psi$, $\psi$ is enabled in $s'$, and $(s', \psi)$ is in $\Psi$, then $\psi$ is enabled in $s$ and $(s, \psi)$ is in $\Psi$. □

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

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

Let $\epsilon = s_0, \pi_1, s_1, \ldots$ be a fair execution of $A$. Choose $i \geq 0$. If $s_i \in D_\varphi$, we are done. Suppose $s_i \notin E_\varphi$. 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 $\psi$ enabled in $s_i$ such that $(s_i, \psi)$ is in $\Psi$. By condition (2c), as long as $(s_j, \pi_{j+1}) \notin \Psi$, $\psi$ is enabled in $s_{j+1}$ and $(s_{j+1}, \psi) \in \Psi$, for $j \geq i$. Since $\epsilon$ 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. □

12
Section 2.2: Liveness

The next lemma demonstrates a third technique for showing that $A$ is equitable for locally-controlled action $\varphi$ of $B$, in a situation when there are multiple higher-level algorithms. The main idea is to show that there is some action $\rho$ of $D$ that is "similar" to $\varphi$ (cf. conditions (2) and (3)) such that $C$ is progressive for $\rho$ using certain helping actions (cf. condition (4)), and $A$ is equitable for all the helping actions for $\rho$ (cf. condition (5)). By "similar", we mean that if $\varphi$ is enabled in the $B$-image of state $s$ of $A$, then $\rho$ is enabled in the $D$-image of the $C$-image of $s$; and if $\rho$ occurs in the $D$-image of the $C$-image of the pair $(s', \pi)$, then $\varphi$ occurs in the $B$-image of $(s', \pi)$. Condition (1) is needed for technical reasons. (For convenience, we define abstraction function $M$ applied to the empty sequence to be the empty sequence. To avoid ambiguity, we add the superscript $AB$ to $E_\varphi, D_\varphi, X_\varphi$ 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 $M_{AB} = (S_{AB}, A_{AB})$ is an abstraction function from $A$ to $B$, and similarly for $M_{AC}$ and $M_{CD}$. Let $\varphi$ be a locally-controlled action of $B$. Suppose the following conditions are true.

1. $M_{AC}(e)$ is an execution of $C$ for every execution $e$ of $A$.

2. There is a locally-controlled action $\rho$ of $D$ such that for any reachable state $s$ of $A$, if $s \in E^A_\varphi$, then $S_{AC}(s) \in E^C_\rho$.

3. If $(s', \pi, s)$ is a step of $A$, $s'$ is reachable, and $\rho$ is in $M_{CD}(M_{AC}(s'\pi s))$, then $\varphi$ is in $A_{AB}(s', \pi)$.

4. $C$ is progressive for $\rho$ via $M_{CD}$, using the set $\Psi_\rho$ and the function $v_\rho$.

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

Then $A$ is equitable for $\varphi$ via $M_{AB}$.

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

Choose $i \geq 0$. If $s_i \in D^A_\varphi$, we are done. Suppose $s_i \in E^A_\varphi$. Assume in contradiction that for all $j \geq i$, $(s_j, \pi_{j+1}) \notin X^A_\varphi$ and $s_j \in E^A_\varphi$. Let $m = \text{index}(i)$. By assumption (2), there is a locally-controlled action $\rho$ of $D$ such that $t_n \in E^C_\rho$ for all $n \geq m$. By assumption (3), $(t_n, \varphi_{n+1}) \notin X^C_\rho$ for all $n \geq m$. 13
Section 3: Problem Statement

By assumption (4), $C$ is progressive for $\rho$ via $\mathcal{M}_{CD}$, using set $\Psi_\rho$ and function $v_\rho$. Thus, there is a locally-controlled action $\psi$ of $C$ enabled in $S_{AC}(s_i) = t_m$ such that $(t_m, \psi) \in \Psi_\rho$. By assumption (5), $A$ is equitable for $\psi$ via $\mathcal{M}_{AC}$. Since $\epsilon$ 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 $m_1 = \text{index}(i_1)$.

Case 1: $(s_{i_1-1}, \pi_{i_1}) \in X_{\psi}^{AC}$. Then $A_{AC}(s_{i_1-1}, \pi_{i_1})$ includes $\psi$. 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 $\psi$ can go from enabled in $t_m$ to disabled in $t_{m_1}$ is for some action in $\Psi_\rho$ to occur between $\varphi_{m+1}$ and $\varphi_{m_1}$. 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 $v_\rho(S_{AC}(s_{i_2})) < v_\rho(S_{AC}(s_{i_1}))$. We can continue this indefinitely, contradicting the range of $v_\rho$ 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 $\epsilon$ of $A$, there is a fair execution $f$ of $B$ such that $\text{sched}(\epsilon)|\text{ext}(A) = \text{sched}(f)|\text{ext}(B)$. Given $\epsilon$, let $f$ be $\mathcal{M}(\epsilon)$. We verify that $\mathcal{M}(\epsilon)$ 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 $\varphi$ of $B$. By Lemma 3, if $\varphi$ is enabled in any state of $f$, then subsequently in $f$, either a state occurs in which $\varphi$ is not enabled, or $\varphi$ occurs. Thus, $f$ is fair. Finally, $\text{sched}(\epsilon)|\text{ext}(A) = \text{sched}(f)|\text{ext}(B)$ 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 $A_r$. 

14
Section 3: Problem Statement

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), $(p, q)$ and $(q, p)$. 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 $w(p, q)$; $w((p, q))$ is defined to be $w(p, q)$; and $w(nil)$ is defined to be $\infty$.

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, $w(e) < w(e')$. Thus replacing $e'$ with $e$ in $E(T_2)$ yields a spanning tree of $G$ with smaller weight than $T_2$, contradicting the assumption. \hfill $\Box$

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 $e$, $w(e) < w(e')$. 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. \hfill $\Box$

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

- no output action occurs unless an input action occurs;
Section 4: Proof of Correctness

- if an input action occurs, then exactly one output action occurs for each $l \in L(G)$;
- if $InTree((p, q))$ occurs, then $(p, q)$ is in $T(G)$; and
- if $NotInTree((p, q))$ 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](image)

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
Section 4: Proof of Correctness

[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.
Section 4.1: HI Solves $MST(G)$

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 $MST(G)$. 

18
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. $InTree((p,q))$ can only occur if $(p,q)$ is already in a subgraph, or is the minimum-weight external edge of a subgraph (i.e., is destined to be in a subgraph). $NotInTree((p,q))$ 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:

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

  Effects:
  
  $answered((p,q)) := true$

- $NotInTree((p,q)), (p,q) \in L(G)$
  
  Preconditions:
  
  $awake = true$
Section 4.1: HI Solves MST(G)

\[ p, q \in F \text{ and } (p, q) \notin F, \text{ for some } F \in FST \]
\[ \text{answered}(p, q) = \text{false} \]

Effects:
\[ \text{answered}(p, q) := \text{true} \]

Internal actions:

- \textit{Combine}(F, F', e), F, F' \in FST, e \in E(G)

  Preconditions:
  \[ \text{awake} = \text{true} \]
  \[ F \neq F' \]
  \[ \text{e is an external edge of } F \]
  \[ \text{e is the minimum-weight external edge of } F' \]

  Effects:
  \[ \text{FST} := \text{FST} - \{F, F'\} \cup \{F \cup F' \cup e\} \]

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 \( P_{HI} = \text{HI-A} \land \text{HI-B} \). 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).

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

\textbf{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 \((s', \pi, s)\) is a step of HI and \( P_{HI} \) is true in \( s' \). If \( \pi \neq \text{Combine}(F, F', e) \), then, since FST is unchanged, \( P_{HI} \) is obviously true in \( s \) as well.

Suppose \( \pi = \text{Combine}(F, F', e) \). 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.
Section 4.1: HI Solves \( \text{MST}(G) \)

We now show that \( \text{HI} \) solves \( \text{MST}(G) \). Let \( e \) be a fair execution of \( \text{HI} \). The use of the variable \( \text{awake} \) ensures that no output action occurs in \( e \) unless an input action occurs in \( e \). The use of the variables \( \text{answered}(l) \) ensures that at most one output action occurs in \( e \) for each link \( l \). Suppose \( \text{InTree}(p, q) \) 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 \( F \in \text{FST} \). By HI-B and Lemmas 10 and 11, \( (p, q) \) is in \( T(G) \). Suppose \( \text{NotInTree}(p, q) \) occurs in \( e \). Then in the preceding state, \( p \) and \( q \) are in \( F \) and \( (p, q) \) is not in \( F \), for some \( F \in \text{FST} \). 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, \( \pi_1 \) is an input action. Only a finite number of output actions can occur in \( e \). Choose \( m \) such that \( \pi_m \) 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 \( s_m \), \( \text{awake} = \text{true} \) in \( s_m \). \( |\text{FST}| = 1 \) in \( s_m \), because otherwise some \( \text{Combine}(F, F', e') \) action would be enabled in \( s_m \), contradicting \( e \) being fair. Let \( \text{FST} = \{ F \} \). By HI-A and HI-B, \( F = T(G) \) in \( s_m \). Furthermore, \( \text{answered}(l) \) 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 \( \text{answered}(l) \) can be true in \( s_m \) is if an output action for \( l \) occurs in \( e \). \( \square \)

4.2 Safety

Each algorithm in the lattice below \( \text{HI} \) 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: $\text{level}(f)$, $\text{core}(f)$, $\text{minlink}(f)$, and $\text{rootchanged}(f)$. Two milestones must be reached before a fragment can combine. First, the $\text{ComputeMin}(f)$ action causes the minimum-weight external link of fragment $f$ to be identified as $\text{minlink}(f)$, and second, the $\text{ChangeRoot}(f)$ action indicates that fragment $f$ is ready to combine, by setting the variable $\text{rootchanged}(f)$. This automaton distinguishes two ways that fragments (and hence, their associated subgraphs) can combine. The $\text{Merge}(f, g)$ 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 $\text{Absorb}(f, g)$ action causes a fragment $g$ to be engulfed by the fragment $f$ at the other end of $\text{minlink}(g)$, 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:

- $\text{subtree}(f)$, a subgraph of $G$;
- $\text{core}(f)$, an edge of $G$ or nil;
- $\text{level}(f)$, a nonnegative integer;
- $\text{minlink}(f)$, a link of $G$ or nil;
- $\text{rootchanged}(f)$, a Boolean.
Section 4.2.1: COM Simulates HI

The state also contains Boolean variables, answered(l) one for each l ∈ 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 (p, q) is an external link of p and of fragment(p) if fragment(p) ≠ fragment(q); otherwise the link is internal.
- If minlink(f) = (p, q), then minedge(f) is the edge (p, q), minn-de(f) = p, and root(f) is the endpoint of core(f) closest to p.
- If (p, q) 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 ∈ subtree(p) and (p, q) ∈ subtree(fragment(p)).

Input actions:

- Start(p), p ∈ V(G)
  Effects:
  awake := true

Output actions:

- InTree((p, q)), (p, q) ∈ L(G)
  Preconditions:
  awake = true
  (p, q) ∈ subtree(fragment(p)) or (p, q) = minlink(fragment(p))
Section 4.2.1: COM Simulates HI

\[ \text{answered}(p, q) = \text{false} \]
Effects:
\[ \text{answered}(p, q) := \text{true} \]

- \text{NotInTree}(p, q), \ (p, q) \in L(G)
  Preconditions:
  \[ \text{fragment}(p) = \text{fragment}(q) \text{ and } (p, q) \notin \text{subtree}(\text{fragment}(p)) \]
  Effects:
  \[ \text{answered}(p, q) = \text{false} \]

Internal actions:

- \text{ComputeMin}(f), f \in \text{fragments}
  Preconditions:
  \[ \text{minlink}(f) = \text{nil} \]
  \[ l \text{ is the minimum-weight external link of } f \]
  \[ \text{level}(f) \leq \text{level}(\text{fragment}(\text{target}(l))) \]
  Effects:
  \[ \text{minlink}(f) := l \]

- \text{ChangeRoot}(f), f \in \text{fragments}
  Preconditions:
  \[ \text{awake} = \text{true} \]
  \[ \text{rootchanged}(f) = \text{false} \]
  \[ \text{minlink}(f) \neq \text{nil} \]
  Effects:
  \[ \text{rootchanged}(f) := \text{true} \]

- \text{Merge}(f, g), f, g \in \text{fragments}
  Preconditions:
  \[ f \neq g \]
  \[ \text{rootchanged}(f) = \text{rootchanged}(g) = \text{true} \]
  \[ \text{minedge}(f) = \text{minedge}(g) \]
  Effects:
  add a new element \( h \) to \text{fragments}
  \[ \text{subtree}(h) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f) \]
  \[ \text{core}(h) := \text{minedge}(f) \]
  \[ \text{level}(h) := \text{level}(f) + 1 \]
  \[ \text{minlink}(h) := \text{nil} \]
Section 4.2.1: COM Simulates HI

\( \text{rootchanged}(h) := \text{false} \)
\( \text{delete } f \text{ and } g \text{ from } \text{fragments} \)

- Absorb\((f, g), f, g \in \text{fragments} \)
  
  ** Preconditions:**
  - rootchanged\((g) = \text{true} \)
  - level\((g) < \text{level}(f) \)
  - fragment\((\text{target}(\text{minlink}(g))) = f \)
  
  ** Effects:**
  - subtree\((f) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(g) \)
  - 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\((f) \leq \text{level}((\text{fragment}(\text{target}(l)))) \).
- COM-B: If rootchanged\((f) = \text{true} \), then minlink\((f) \neq \text{nil} \).
- COM-C: If awake = false, then minlink\((f) \neq \text{nil} \), rootchanged\((f) = \text{false} \), and subtree\((f) = \{p\} \) for some \( p \).
- COM-D: If \( f \neq g \), then subtree\((f) \neq \text{subtree}(g) \).
- COM-E: If subtree\((f) = \{p\} \) for some \( p \), then minlink\((f) \neq \text{nil} \).
- COM-F: If \(|\text{nodes}(f)| = 1\), then level\((f) = 0 \) and \( \text{core}(f) = \text{nil} \); if \(|\text{nodes}(f)| > 1\), then level\((f) > 0 \) and \( \text{core}(f) \in \text{subtree}(f) \).

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 \( \{\text{subtree}(f) : f \in \text{fragments}\} \).

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

- If \( \pi = \text{Start}(p), \text{InTree}(l), \text{or NotInTree}(l) \), then \( \mathcal{A}_1(s, \pi) = \pi \).
Section 4.2.1: COM Simulates HI

- If \( \pi = \text{ComputeMin}(f) \) or \( \text{ChangeRoot}(f) \), then \( A_1(s, \pi) \) is empty.

- If \( \pi = \text{Merge}(f, g) \) or \( \text{Absorb}(f, g) \), then \( A_1(s, \pi) = \text{Combine}(F, F', e) \) where \( F = \text{subtree}(f) \) in \( s \), \( F' = \text{subtree}(g) \) in \( s \), and \( e = \text{minedge}(g) \) in \( s \).

The following predicate is true in every state of \( \text{COM} \) satisfying \((P_{HI} \circ S_1) \land P_{COM}\). (I.e., it is deducible from \( P_{COM} \) and the HI predicates.)

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

**Proof:** Let \( s \) be a state of \( \text{COM} \) satisfying \((P_{HI} \circ S_1) \land P_{COM}\). In \( S_1(s), \) \( \text{FST} = \{ \text{subtree}(f) : f \in \text{fragments} \} \). By HI-B, \( \text{FST} \) forms a partition of \( V(G) \). By COM-D, the multiset \( \{ \text{subtree}(f) : f \in \text{fragments} \} = \text{FST} \), and thus it forms a partition of \( V(G) \). Consequently, \( \text{fragment}(p) \) is well-defined.

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

**Proof:** By inspection, the types of \( \text{COM}, \) 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 \( \text{start}((COM) \). Obviously, \( P_{COM} \) is true in \( s \), and \( S_1(s) \) is in \( \text{start}(HI) \).

2. Obviously, \( A_1(s, \pi)|_{\text{ext}(HI)} = \pi|_{\text{ext}(COM)} \) for any state \( s \) of \( A \).

3. Let \((s', \pi, s)\) be a step of \( \text{COM} \) such that \( P_{HI} \) is true of \( S_1(s') \) and \( P_{COM} \) is true of \( s' \). We consider each possible value of \( \pi \).

   i) \( \pi \) is \( \text{Start}(p) \), \( \text{InTree}(l) \), or \( \text{NotInTree}(l) \). \( A_1(s', \pi) = \pi \). Obviously, \( P_{COM} \) is true in \( s \), and \( S_1(s') \pi S_1(s) \) is an execution fragment of HI.

   ii) \( \pi \) is \( \text{ComputeMin}(f) \) or \( \text{ChangeRoot}(f) \). \( A_1(s', \pi) \) is empty. Obviously, \( S_1(s') = S_1(s) \). Obviously, COM-A, COM-B, COM-D and COM-F are true in \( s \). By COM-C for \( \text{ComputeMin}(f) \) and by precondition for \( \text{ChangeRoot}(f) \), \( \text{awake} = \text{true in} \ s' \), 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 \( \pi = \text{ComputeMin}(f) \), then \( \text{minlink}(f) \neq \text{nil} \) in \( s \), and COM-E is vacuously true in \( s \) for \( f \). If \( \pi = \text{ChangeRoot}(f) \), then by COM-B, \( \text{minlink}(f) \neq \text{nil} \) in \( s' \) and also in \( s \), so COM-E is vacuously true in \( s \) for \( f \).
Section 4.2.1: COM Simulates HI

iii) \( \pi \) is Merge(f,g).

(3c) \( A_1(s', \pi) = \text{Combine}(F, F', e) \), where \( F = \text{subtree}(f) \) in \( s' \), \( F' = \text{subtree}(g) \) in \( s' \), and \( e = \text{minedge}(g) \) in \( s' \), for some fragments \( f \) and \( g \).

Claims about \( s' \):

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

Let \( F = \text{subtree}(f), F' = \text{subtree}(g) \) and \( e = \text{minedge}(g) \).

Claims about \( S_1(s') \): (All depend on the definition of \( S_1 \).)

8. \( \text{awake} = \text{true} \), 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, \( \text{Combine}(F, F', e) \) is enabled in \( S_1(s') \). Obviously, its effects are mirrored in \( S_1(s) \).

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

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

Claims about \( s \):

14. \( \text{subtree}(h) = \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f) \), by code.
15. \( \text{core}(h) = \text{minedge}(f) \), by code.
16. \( \text{level}(h) = \text{level}(f) + 1 \), by code.
17. \( \text{minlink}(h) = \text{nil} \), by code.
18. \( \text{rootchanged}(h) = \text{false} \), by code.
19. \( f \) and \( g \) are removed from fragments, by code.
20. \( \text{awake} = \text{true} \), by Claim 4.
21. \( \text{subtree}(f') \) and \( \text{subtree}(g') \) are disjoint, for all \( f' \neq g' \), by Claims 13, 14 and 19.
Section 4.2.1: COM Simulates HI

22. $|\text{nodes}(h)| > 1$, by Claim 14.
23. $\text{level}(h) > 1$, by Claims 12 and 16.
24. $\text{core}(h) \in \text{subtree}(h)$, 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) $\pi$ is Absorb($f,g$).

(3c) $A_1(s', \pi) = \text{Combine}(F, F', e)$, where $F = \text{subtree}(f)$ in $s'$, $F' = \text{subtree}(g)$ in $s'$, and $e = \text{minedge}(g)$ in $s'$, for some fragments $f$ and $g$.

Claims about $s'$:

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

Let $F = \text{subtree}(f)$, $F' = \text{subtree}(g)$ and $e = \text{minedge}(g)$.

Claims about $S_1(s')$: (All depend on the definition of $S_1$.)

9. $\text{awake} = \text{true}$, 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, $\text{Combine}(F, F', e)$ is enabled in $S_1(s')$. Obviously, its effects are mirrored in $S_1(s)$.

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

More claims about $s'$:
Section 4.2.2: GC Simulates COM

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

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

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

More claims about \( s' \):

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

Claims about \( s \):

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

Let \( P'_\text{COM} = (P_{HI} \circ S_1) \wedge P_\text{COM} \).

Corollary 14: \( P'_\text{COM} \) is true in every reachable state of COM.

Proof: By Lemmas 1 and 13. \( \square \)
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 \( \text{testset}(f) \) of nodes that are participating in the search. Once a node in \( f \) has found its minimum-weight external link, it is removed from \( \text{testset}(f) \). A new action, \( \text{TestNode}(p) \), 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 \( \text{accmin}(f) \) (for "accumulated minlink") stores the link with the minimum weight over all links external to nodes of \( f \) no longer in \( \text{testset}(f) \). \( \text{ComputeMin}(f) \) cannot occur until \( \text{testset}(f) \) is empty. When an \( \text{Absorb}(f,g) \) action occurs, all the nodes formerly in \( g \) are added to \( \text{testset}(f) \) if and only if the target of \( \text{minlink}(g) \) is in \( \text{testset}(f) \). 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 \( \text{fragments} \). Each element \( f \) of the set is called a fragment, and has the following components:

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

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

In the start state of \( COM \), \( \text{fragments} \) has one element for each node in \( V(G) \): for fragment \( f \) corresponding to node \( p \), \( \text{subtree}(f) = \{p\} \), \( \text{core}(f) = \text{nil} \), \( \text{level}(f) = 0 \), \( \text{minlink}(f) \) is the minimum-weight link adjacent to \( p \), \( \text{rootchanged}(f) \) is false, \( \text{testset}(f) \) is empty, and \( \text{accmin}(f) \) is \( \text{nil} \). Each \( \text{answered}(l) \) is false and \( \text{awake} \) is false.
Section 4.2.2: GC Simulates COM

Input actions:

- \textit{Start}(p), p \in V(G)
  
  Effects:
  
  \texttt{awake} := \texttt{true}

Output actions:

- \textit{InTree}((p, q)), (p, q) \in L(G)
  
  Preconditions:
  
  \texttt{awake} = \texttt{true}
  
  \( (p, q) \in \text{subtree}(\text{fragment}(p)) \) or \((p, q) = \text{minlink}(\text{fragment}(p)) \)
  
  \texttt{answered}(p, q) = \texttt{false}
  
  Effects:
  
  \texttt{answered}((p, q)) := \texttt{true}

- \textit{NotInTree}((p, q)), (p, q) \in L(G)
  
  Preconditions:
  
  \texttt{fragment}(p) = \texttt{fragment}(q) \) and \((p, q) \notin \text{subtree}(\text{fragment}(p)) \)
  
  \texttt{answered}(p, q) = \texttt{false}
  
  Effects:
  
  \texttt{answered}((p, q)) := \texttt{true}

Internal actions:

- \textit{TestNode}(p), p \in V(G)
  
  Preconditions:
  
  \begin{itemize}
  \item -- let \( f = \text{fragment}(p) \) --
  \item \( p \in \text{testset}(f) \)
  \item if \((p, q)\), the minimum-weight external link of \( p \), exists
  \item then \( \text{level}(f) \leq \text{level}(\text{fragment}(q)) \)
  \end{itemize}

  Effects:
  
  \begin{itemize}
  \item \text{testset}(f) := \text{testset}(f) - \{p\}
  \item if \((p, q)\), the minimum-weight external link of \( p \), exists
  \item and \( \text{wt}(p, q) < \text{wt}(\text{accmin}(f)) \)
  \item then \text{accmin}(f) := (p, q)
  \end{itemize}

- \textit{ComputeMin}(f), f \in \text{fragments}
  
  Preconditions:
  
  \texttt{minlink}(f) = \texttt{nil}
Section 4.2.2: GC Simulates COM

\[
\text{accmin}(f) \neq \text{nil} \\
\text{testset}(f) = \emptyset
\]

Effects:
\[
\begin{align*}
\text{minlink}(f) & := \text{accmin}(f) \\
\text{accmin}(f) & := \text{nil}
\end{align*}
\]

- **ChangeRoot**\( (f) \), \( f \in \text{fragments} \)
  
  Preconditions:
  \[
  \begin{align*}
  \text{awake} & = \text{true} \\
  \text{rootchanged}(f) & = \text{false} \\
  \text{minlink}(f) & \neq \text{nil}
  \end{align*}
  \]
  
  Effects:
  \[
  \text{rootchanged}(f) := \text{true}
  \]

- **Merge**\( (f, g) \), \( f, g \in \text{fragments} \)
  
  Preconditions:
  \[
  f \neq g \\
  \text{rootchanged}(f) = \text{rootchanged}(g) = \text{true} \\
  \text{minedge}(f) = \text{minedge}(g) \neq \text{nil}
  \]
  
  Effects:
  \[
  \begin{align*}
  \text{add a new element } h \text{ to } \text{fragments} \\
  \text{subtree}(h) & := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f) \\
  \text{core}(h) & := \text{minedge}(f) \\
  \text{level}(h) & := \text{level}(f) + 1 \\
  \text{minlink}(h) & := \text{nil} \\
  \text{rootchanged}(h) & := \text{false} \\
  \text{testset}(h) & := \text{nodes}(h) \\
  \text{accmin}(h) & := \text{nil} \\
  \text{delete } f \text{ and } g \text{ from } \text{fragments}
  \end{align*}
  \]

- **Absorb**\( (f, g) \), \( f, g \in \text{fragments} \)
  
  Preconditions:
  \[
  \begin{align*}
  \text{rootchanged}(g) & = \text{true} \\
  \text{level}(g) & < \text{level}(f) \\
  \text{let } p & = \text{target}(\text{minlink}(g)) \\
  \text{fragment}(p) & = f
  \end{align*}
  \]
  
  Effects:
  \[
  \begin{align*}
  \text{subtree}(f) & := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(g) \\
  \text{if } p \in \text{testset}(f) \text{ then } \text{testset}(f) & := \text{testset}(f) \cup \text{testset}(g)
  \end{align*}
  \]
Section 4.2.2: GC Simulates COM

delete g from fragments

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

- GC-A: If \( \text{accmin}(f) = (p,q) \), then \( (p,q) \) is the minimum-weight external link of any node in \( \text{nodes}(f) - \text{testset}(f) \), and \( \text{level}(f) \leq \text{level}((\text{fragment}(q))) \).

- GC-B: If there is an external link of \( f \), if \( \text{minlink}(f) = \text{nil} \), and if \( \text{testset}(f) = \emptyset \), then \( \text{accmin}(f) \neq \text{nil} \).

- GC-C: If \( \text{testset}(f) \neq \emptyset \), then \( \text{minlink}(f) = \text{nil} \).

Let \( P_{GC} = \text{GC-A} \land \text{GC-B} \land \text{GC-C} \).

In order to show that GC simulates COM, we define an abstraction mapping \( M_2 = (S_2, A_2) \) from GC to COM. Define the function \( S_2 \) from states(GC) to states(COM) by simply ignoring the variables \( \text{accmin}(f) \) and \( \text{testset}(f) \) for all fragments \( f \) when going from a state of GC to a state of COM.

Define the function \( A_2 \) as follows. Let \( s \) be a state of GC and \( \pi \) an action of GC enabled in \( s \). If \( \pi = \text{TestNode}(p) \), then \( A_2(s, \pi) \) is empty. Otherwise, \( A_2(s, \pi) = \pi \).

Recall that \( P'_{COM} = (P_{HI} \circ S_1) \land P_{COM} \). If \( P'_{COM}(S_2(s)) \) is true, then the COM predicates are true in \( S_2(s) \), and the HI predicates are true in \( S_1(S_2(s)) \).

**Lemma 15:** GC simulates COM via \( M_2, P_{GC}, \) and \( P'_{COM} \).

**Proof:** By inspection, the types of GC, COM, \( M_2 \), 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 \( \text{start}(GC) \). Obviously, \( P_{GC} \) is true in \( s \), and \( S_2(s) \) is in \( \text{start}(COM) \).

2. Obviously, \( A_2(s, \pi) \mid \text{ext}(COM) = \pi \mid \text{ext}(GC) \).

3. Let \( (s', \pi, s) \) be a step of GC such that \( P'_{COM} \) is true of \( S_2(s') \) and \( P_{GC} \) is true of \( s' \).

   i) \( \pi \) is \( \text{Start}(p) \), \( \text{InTree}(l) \), \( \text{NotInTree}(l) \), or \( \text{ChangeRoot}(f) \). Obviously, \( S_2(s') \) is an execution fragment of COM, and \( P_{GC} \) is true in \( s \).

   ii) \( \pi \) is \( \text{ComputeMin}(f) \).
Section 4.2.2: GC Simulates COM

(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 $S_2(s')$ 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) $A_2(s', \pi) = \pi$.

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 $S_2(s') \pi S_2(s)$ is an execution fragment of $COM$.

iii) $\pi$ 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) $A_2(s', \pi)$ is empty, and obviously $S_2(s') = S_2(s)$.

iv) $\pi$ 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) $A_2(s', \pi) = \pi$. Obviously, $S_2(s') \pi S_2(s)$ is an execution fragment of $COM$.

v) $\pi$ 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
Section 4.2.3: TAR Simulates GC

\( f \) 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 \( \text{accmin}(f) = \text{nil} \) in \( s' \), then the same is true in \( s \), so GC-A is vacuously true for \( f \) in \( s \).

Assume \( \text{accmin}(f) = (r, t) \). Let \( \text{minlink}(g) = (q, p) \).

Claims about \( s' \):

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

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

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

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

\( 3c) \ A_2(s, \pi) = \pi \). Obviously \( S_2(s') \pi S_2(s) \) is an execution fragment of \( \text{COM} \).

Let \( P_{GC}' = (P_{COM} \circ S_2) \land P_{GC} \).

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 \((p, q)\): 
- \(\text{tarqueue}_p((p, q))\), the outgoing queue local to \(p\); 
- \(\text{tarqueue}_{pq}((p, q))\), modelling the communication channel; and 
- \(\text{tarqueue}_q((p, q))\), the incoming queue local to \(q\). The action \(\text{ChannelSend}(l, m)\) transfers a message \(m\) from the outgoing local queue of link \(l\) to the communication channel of \(l\); and the action \(\text{ChannelRecv}(l, m)\) 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 \(\text{status}(l)\) 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 \(\text{SendTest}(p)\), which causes \(p\) to identify its minimum-weight unknown link as \(\text{testlink}(p)\), 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 \(\text{ChangeRoot}(f)\) occurs, \(\text{minlink}(f)\) is marked as branch; when an \(\text{Absorb}(f, g)\) occurs, the reverse link of \(\text{minlink}(g)\) is marked as branch. As soon as a link \(l\) is classified as branch, the \(\text{InTree}(l)\) output action can occur; as soon as a link \(l\) is classified as rejected, the \(\text{NotInTree}(l)\) 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 (q,p) is the minimum-weight external link of g. First, q
determines that (q,p) 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 (q, p). Then ChangeRoot(g) occurs, and (q, p)
is marked as branch. Then Absorb(f, g) occurs, and (p, q) is marked as branch. The
next time that q tries to process p’s test(8, d) message, it succeeds, determines that
(q, p) 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 (q, p) from branch to rejected. Similarly,
when p receives q’s reject message, it had better not change the classification of
(p, q) 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 subgrapt 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).
Section 4.2.3: TAR Simulates GC

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

For each link $(p,q)$, there are associated four variables:

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

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

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 $p$, $\text{subtree}(f) = \{p\}$, $\text{core}(f) = \text{nil}$, $\text{level}(f) = 0$. $\text{minlink}(f)$ is the minimum-weight link adjacent to $p$, $\text{rootchanged}(f)$ is false, and $\text{testset}(f)$ is empty. For all $p$, $testlink(p)$ is $\text{nil}$. For each link $l$, $lstatus(l) = \text{unknown}$. The message queues are empty. Each $answered(l)$ is false and $awake$ is false.

The derived variable $\text{tarqueue}((p,q))$ is defined to be $\text{tarqueue}_p((p,q)) \parallel \text{tarqueue}_{pq}((p,q)) \parallel \text{tarqueue}_q((p,q))$.\footnote{Given two FIFO queues $q_1$ and $q_2$, define $q_1 \parallel q_2$ to be the FIFO queue obtained by appending $q_2$ to the end of $q_1$. Obviously this operation is associative.}

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

Input actions:

- $\text{Start}(p)$, $p \in V(G)$

  Effects:
Section 4.2.3: TAR Simulates GC

\[ \text{awake} := \text{true} \]

Output actions:

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

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

Internal actions (and a procedure):

- \( \text{ChannelSend}((p,q),m), (p,q) \in L(G), m \in M \)
  
  Preconditions:
  
  \( m \) at head of \( \text{tarqueue}_p((p,q)) \)
  
  Effects:
  
  dequeue(\text{tarqueue}_p((p,q)))
  
  enqueue(\text{tarqueue}_p((p,q)))

- \( \text{ChannelRecv}((p,q),m), (p,q) \in L(G), m \in M \)
  
  Preconditions:
  
  \( m \) at head of \( \text{tarqueue}_p((p,q)) \)
  
  Effects:
  
  dequeue(\text{tarqueue}_p((p,q)))
  
  enqueue(\text{tarqueue}_p((p,q)))

- \( \text{SendTest}(p), p \in V(G) \)
  
  Preconditions:
  
  \( p \in \text{testset(fragment}(p)) \)
  
  \( \text{testlink}(p) = \text{nil} \)
  
  Effects:
  
  execute procedure Test(p)
Section 4.2.3: TAR Simulates GC

- Procedure \texttt{Test}(p), \( p \in V(G) \)
  
  - let \( f = \text{fragment}(p) \)
  
  if \( l \), the minimum-weight link of \( p \) with \( lstatus(l) = \text{unknown} \), exists then
    
    \begin{align*}
    \text{testlink}(p) & := l \\
    \text{enqueue}(\text{Test}(\text{level}(f), \text{core}(f)), \text{tarqueue}_p(l))
    \end{align*}

  else
    
    remove \( p \) from \( \text{testset}(f) \)
    
    \text{testlink}(p) := \text{nil}

- \texttt{ReceiveTest}((q,p),l,c), \((p,q) \in L(G)\)
  
  Preconditions:
  
  \( \text{TEST}(l,c) \) at head of \( \text{tarqueue}_p((q,p)) \)

  Effects:
  
  \begin{align*}
  & \text{enqueue}(\text{tarqueue}_p((q,p))) \\
  & \text{if } l > \text{level}(\text{fragment}(p)) \text{ then} \\
  & \text{enqueue}(\text{TEST}(l,c), \text{tarqueue}_p((q,p)))
  \end{align*}

  else
    
    if \( c \neq \text{core}(\text{fragment}(p)) \) then
      
      \text{enqueue}(\text{ACCEPT}, \text{tarqueue}_p((p,q)))
    
    else
      
      if \( lstatus((p,q)) = \text{unknown} \) then
        \( lstatus((p,q)) := \text{rejected} \)
      
      if \( \text{testlink}(p) \neq (p,q) \) then
        
        \text{enqueue}(\text{REJECT}, \text{tarqueue}_p((p,q)))
      
    else execute procedure \text{Test}(p) \}

- \texttt{ReceiveAccept}((q,p)), \((q,p) \in L(G)\)
  
  Preconditions:
  
  \( \text{ACCEPT} \) at head of \( \text{tarqueue}_p((q,p)) \)

  Effects:
  
  \begin{align*}
  & \text{enqueue}(\text{tarqueue}_p((q,p))) \\
  & \text{testlink}(p) := \text{nil} \\
  & \text{remove } p \text{ from } \text{testset}(\text{fragment}(p))
  \end{align*}

- \texttt{ReceiveReject}((q,p)), \((q,p) \in L(G)\)
  
  Preconditions:
  
  \( \text{REJECT} \) at head of \( \text{tarqueue}_p((q,p)) \)

  Effects:
  
  \begin{align*}
  & \text{enqueue}(\text{tarqueue}_p((q,p))) \\
  & \text{if } lstatus((p,q)) = \text{unknown} \text{ then } lstatus((p,q)) := \text{rejected}
  \end{align*}
Section 4.2.3: TAR Simulates GC

execute procedure Test(p)

- ComputeMin(f), f ∈ fragments
  Preconditions:
  \[
  \text{minlink}(f) = \text{nil} \\
  \text{accmin}(f) \neq \text{nil} \\
  \text{testset}(f) = \emptyset
  \]
  Effects:
  \[
  \text{minlink}(f) := \text{accmin}(f)
  \]

- ChangeRoot(f), f ∈ fragments
  Preconditions:
  \[
  \text{awake} = \text{true} \\
  \text{rootchanged}(f) = \text{false} \\
  \text{minlink}(f) \neq \text{nil}
  \]
  Effects:
  \[
  \text{rootchanged}(f) := \text{true} \\
  \text{lstatus}(\text{minlink}(f)) := \text{branch}
  \]

- Merge(f, g), f, g ∈ fragments
  Preconditions:
  \[
  f \neq g \\
  \text{rootchanged}(f) = \text{rootchanged}(g) = \text{true} \\
  \text{minedge}(f) = \text{minedge}(g)
  \]
  Effects:
  add a new element h to fragments
  \[
  \text{subtree}(h) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f)
  \]
  \[
  \text{core}(h) := \text{minedge}(f)
  \]
  \[
  \text{level}(h) := \text{level}(f) + 1
  \]
  \[
  \text{minlink}(h) := \text{nil}
  \]
  \[
  \text{rootchanged}(h) := \text{false}
  \]
  \[
  \text{testset}(h) := \text{nodes}(h)
  \]
  delete f and g from fragments

- Absorb(f, g), f, g ∈ fragments
  Preconditions:
  \[
  \text{rootchanged}(g) = \text{true} \\
  \text{level}(g) < \text{level}(f)
  \]
  \[
  \text{let } (q, p) = \text{minlink}(g)
  \]
  \[
  \text{fragment}(p) = f
  \]
Section 4.2.3: TAR Simulates GC

Effects:

\[ \text{subtree}(f) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(g) \]

if \( p \in \text{testset}(f) \) then \( \text{testset}(f) := \text{testset}(f) \cup \text{nodes}(g) \)

\( lstatus((p,q)) := \text{branch} \)

delete \text{g} from fragments

A message \( m \) is defined to be a protocol message for link \( (p,q) \) in a state if \( m \) is one of the following:

(a) a TEST message in \( \text{tarqueue}((p,q)) \) with \( lstatus((p,q)) \neq \text{rejected} \).

(b) an ACCEPT message in \( \text{tarqueue}((q,p)) \)

(c) a REJECT message in \( \text{tarqueue}((q,p)) \)

(d) a TEST message in \( \text{tarqueue}((q,p)) \) with \( lstatus((q,p)) = \text{rejected} \).

A protocol message for \( (p,q) \) can be considered a message that is actively helping \( p \) to discover whether \( (p,q) \) is external.

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

- **TAR-A:**
  (a) If \( lstatus((p,q)) = \text{branch} \), then either \( (p,q) \in \text{subtree(fragment}(p)) \) or min-link \( (\text{fragment}(p)) = (p,q) \).
  (b) If \( (p,q) \in \text{subtree(fragment(p))} \), then \( lstatus((p,q)) = lstatus((q,p)) = \text{branch} \).

- **TAR-B:** If \( lstatus((p,q)) = \text{rejected} \), then \( \text{fragment}(p) = \text{fragment}(q) \) and \( (p,q) \notin \text{subtree(fragment}(p)) \).

- **TAR-C:** If \( \text{testlink}(p) \not= \text{nil} \), then
  (a) \( \text{testlink}(p) = (p,q) \) for some \( q \);
  (b) \( p \in \text{testset(fragment}(p)) \);
  (c) there is exactly one protocol message for \( (p,q) \);
  (d) if \( lstatus((p,q)) \neq \text{branch} \), then \( (p,q) \) is the minimum-weight link of \( p \) with \( lstatus \) unknown;
  (e) if \( lstatus((p,q)) = \text{branch} \), then \( lstatus((q,p)) = \text{branch} \) and \( \text{testlink}(q) \not= (q,p) \).

- **TAR-D:** If there is a protocol message for \( (p,q) \), then \( \text{testlink}(p) = (p,q) \).

- **TAR-E:** If \( \text{TEST(l,c)} \) is in \( \text{tarqueue}((p,q)) \) then
  (a) \( (p,q) \not= \text{core(fragment}(p)) \);
Section 4.2.3: TAR Simulates GC

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

- TAR-F: If ACCEPT is in tarqueue((p,q)), then $fragment(p) \neq fragment(q)$ and $level(fragment(p)) \geq level(fragment(q))$.
- TAR-G: If REJECT is in tarqueue((p,q)), then $fragment(p) = fragment(q)$ and $lstatus((p,q)) \neq$ 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 (p,q) has $lstatus((p,q)) = unknown$, or else there is an external link (r,t) of $fragment(p)$ with $level(fragment(p)) \leq level(fragment(t))$.
- TAR-J: If awake = false, then $lstatus((p,q)) = unknown$.

Let $PTAR$ be the conjunction of TAR-A through TAR-J.

In order to show that TAR simulates GC, we define an abstraction mapping $M_3 = (S_3, A_3)$ from TAR to GC. Define the function $S_3$ 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 $A_3$ as follows. Let $s$ be a state of TAR and $\pi$ 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 $\pi = ReceiveAccept((q,p))$, then $A_3(s, \pi) = TestNode(p)$.
- If $\pi = SendTest(p)$ or $ReceiveReject((q,p))$, then $A_3(s, \pi) = TestNode(p)$ if there is no link (p,r), r $\neq q$, with $lstatus((p,r)) = unknown$ in $s$; otherwise, $A_3(s, \pi)$ is empty.
- If $\pi = ReceiveTest((q,p), l, c)$, then $A_3(s, \pi) = TestNode(p)$ if $l \leq level(fragment(p))$, $c = core(fragment(p))$, $testlink(p) = (p,q)$, and there is no link (p,r), r $\neq q$, with $lstatus((p,r)) = unknown$ in $s$; otherwise, $A_3(s, \pi)$ is empty.
- If $\pi = ChannelSend((p,q), m)$ or $ChannelRecv((p,q), m)$, then $A_3(s, \pi)$ is empty.
Section 4.2.3: TAR Simulates GC

- For all other values of $\tau$, $A_3(s, \tau) = \tau$.

The following predicates are true in every state of TAR satisfying $(P'_G \circ S_3) \land P_{TAR}$. Recall that $P'_G = (P'_{COM} \circ S_2) \land P_{GC}$. If $P'_{GC}(S_3(s))$ is true, then the GC predicates are true in $S_3(s)$, the COM predicates are true in $S_2(S_3(s))$, and the HI predicates are true in $S(S_2(S_3(s)))$. Thus, these predicates are derivable from $P_{TAR}$, together with the HI, COM and GC predicates.

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

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

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

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

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

Proof: Let $f = \text{fragment}(p)$ and $g = \text{fragment}(q)$.

1. $test(l, c)$ is in $\text{tarqueue}(p, q)$, by assumption.

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

   Case 2: $lstatus((p, q)) = rejected$.
   11. $lstatus((p, q)) = rejected$, by assumption.
   12. $c = \text{core}(g)$ and $l = \text{level}(g)$, by Claim 11 and TAR-E(c).
   13. $testlink(g) = (q, p)$, by Claims 1 and 11 and TAR-D.
   14. $q \in \text{testset}(g)$, by Claim 13 and TAR-C(b).
   15. $minlink(g) = \text{nil}$, by Claim 14 and GC-C.
   16. $\text{subtree}(g) \neq \{q\}$, by Claim 15 and COM-E.
Section 4.2.3: \textit{TAR} Simulates \textit{GC}

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

- \textbf{TAR-N}: If \textit{test}(l, c) is in \textit{tarqueue}((q, p)) and \textit{c} = \textit{core}(fragment(p)), then \textit{fragment}(p) = \textit{fragment}(q).

\textbf{Proof}:
1. \textit{test}(l, c) is in \textit{tarqueue}((q, p)), by assumption.
2. \textit{c} = \textit{core}(fragment(p)), by assumption.
3. \textit{c} \neq \textit{nil}, by Claim 1 and \textit{TAR-M}.
4. If \textit{lstatus}((q, p)) \neq \textit{rejected}, then \textit{c} = \textit{core}(fragment(q)), by \textit{TAR-E(b)}.
5. If \textit{lstatus}((q, p)) \neq \textit{rejected}, then \textit{fragment}((q, p)) = \textit{fragment}(p), by Claims 2, 3 and 4, and \textit{COM-F}.
6. If \textit{lstatus}((q, p)) = \textit{rejected}, then \textit{fragment}((q, p)) = \textit{fragment}(p), by \textit{TAR-B}.

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

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

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

\textbf{Proof}:
1. \textit{test}(l, c) is in \textit{tarqueue}((q, p)), by assumption.
2. \textit{c} = \textit{core}(fragment(p)), by assumption.
3. \textit{testlink}(p) = \textit{(p, q)}, by assumption.
4. \textit{lstatus}((q, p)) \neq \textit{rejected}, by assumption.
5. \textit{fragment}(p) = \textit{fragment}(q), by Claims 1 and 2 and \textit{TAR-N}.
6. No \textit{ACCEPT} message is in \textit{tarqueue}((q, p)), by Claim 5 and \textit{TAR-F}.
7. The \textit{test}((l, c)) message in \textit{tarqueue}((q, p)) is a protocol message for \textit{(q, p)}, by Claim 4.
8. \textit{testlink}(q) = \textit{(q, p)}, by Claim 7 and \textit{TAR-D}.
Section 4.2.3: TAR Simulates GC

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

Claims 13 and 15 give the result.

Lemma 17: TAR simulates GC via \( M_3 \), \( P_{TAR} \), and \( P'_{GC} \).

Proof: By inspection, the types of TAR, GC, \( M_3 \), 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 \( \text{start}(TAR) \). Obviously, \( P_{TAR} \) is true in \( s \), and \( S_3(s) \) is in \( \text{start}(GC) \).

(2) Obviously, \( A_3(s, \pi)|ext(GC) = \pi|ext(TAR) \).

(3) Let \( (s', \pi, s) \) be a step of TAR such that \( P'_{GC} \) is true of \( S_3(s') \) and \( P_{TAR} \) is true of \( s' \). Condition (3a) is only shown below for those predicates that are not obviously true in \( s \).

i) \( \pi \) is \( \text{ChannelSend}(p, q, m) \) or \( \text{ChannelRecv}(p, q, m) \). \( A_3(s', \pi) \) is empty. (3a) and (3b) are obviously true.

ii) \( \pi \) is \( \text{Start}(p) \) or \( \text{InTree}(l) \) or \( \text{NotInTree}(l) \).

\( (3c) A_3(s', \pi) = \pi \). If \( \pi = \text{InTree}(l) \), then by TAR-J and TAR-A(a), \( \pi \) is enabled in \( S_3(s') \). If \( \pi = \text{NotInTree}(l) \), then by TAR-J and TAR-B, \( \pi \) is enabled in \( S_3(s') \). Thus, \( S_3(s')\pi S_3(s) \) is an execution fragment of GC.

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

iii) \( \pi \) is \( \text{SendTest}(p) \). Let \( f = \text{fragment}(p) \) in \( s' \).

Case 1: There is a link \( (p, q) \) with \( lstatus((p, q)) = \text{unknown} \) in \( s' \).

(3b) \( A_3(s', \pi) \) is empty. It is easy to see that \( S_3(s') = S_3(s) \).
Section 4.2.3: TAR Simulates GC

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

TAR-C(c): In \textit{s}, there is exactly one protocol message for \textit{(p, q)}, namely the \textit{test} message in \textit{tarqueue}((p, q)).

TAR-D: The \textit{test} message added in \textit{s} is a protocol message for \textit{(p, q)}, and is not a protocol message for any other link. By the code, \textit{testlink}(p) = \textit{(p, q)}.

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

\begin{quote}
\textit{Case 2}: There is no link \textit{(p, q)} with \textit{lstatus)((p, q))} = \textit{unknown} in \textit{s'}.
\end{quote}

(3c) \textit{A}_{3}(s', \pi) = \textit{TestNode}(p).

\textit{Claims about s'}:

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

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

\textit{Claims about s}:

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

By Claims 4, 5, and 6, the effects of \textit{TestNode}(p) are mirrored in \textit{S}_{3}(s).

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

\textbf{iv)} \pi is \textit{ReceiveTest}((\textit{q}, \textit{p}), \textit{l}, \textit{c}). Let \textit{f} = \textit{fragment}(p) in \textit{s'}.

\begin{quote}
\textit{Case 1}: \textit{l} \leq level(f), \textit{c} = \textit{core}(f), \textit{testlink}(p) = \textit{(p, q)}, and there is no link \textit{(p, r)}, \textit{r} \neq \textit{q}, with \textit{lstatus}((p, r)) = \textit{unknown} in \textit{s'}.
\end{quote}

(3c) \textit{A}_{3}(s', \pi) = \textit{TestNode}(p).

\textit{Claims about s'}:
Section 4.2.3: TAR Simulates GC

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

By Claims 5, 7 and 8, \( \text{TestNode}(p) \) is enabled in \( s' \).

Claims about \( s' \):

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

By Claims 9, 10 and 11, the effects of \( \text{TestNode}(p) \) are mirrored in \( s \).

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

TAR-D:

Claims about \( s' \):

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

Case A: \( \text{Istatus}(p, q) = \text{rejected} \). The \( \text{TEST}(l, c) \) message in \( \text{tarqueue}(q, p) \) is the protocol message for \( (p, q) \) 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, \( \text{testlink}(q) \neq (q, p) \); thus, the predicate is still true for \( q \) in \( s \), even if \( \text{Istatus}(p, q) \) is changed to rejected.

Case B: \( \text{Istatus}(p, q) \neq \text{rejected} \).

6. A \( \text{TEST}(l', c') \) is in \( \text{tarqueue}(p, q) \) and \( \text{Istatus}(p, q) = \text{unknown} \), by Claims 1, 2, 3, assumptions for Case B and TAR-P.
Section 4.2.3: TAR Simulates GC

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

In \( s \), the \( \text{TEST}(i', c') \) message in \( \text{tarqueue}(\langle p, q \rangle) \), which exists by Claim 6, becomes a protocol message for \( \langle q, p \rangle \), since \( \text{Istatus}(\langle p, q \rangle) \) is changed to rejected. By Claim 7, \( \text{testlink}(q) \) 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 \( \text{Istatus}(\langle p, q \rangle) \) goes from unknown in \( s' \) to rejected in \( s \), while there is a \( \text{TEST}(l', c') \) message in \( \text{tarqueue}(\langle p, q \rangle) \). By TAR-E(b), \( c' = \text{core}(f) \) and \( l' = \text{level}(f) \) in \( s' \). By TAR-N, \( \text{fragment}(q) = f \). Thus \( c' = \text{core}(\text{fragment}(q)) \) and \( l' = \text{level}(\text{fragment}(q)) \).

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

TAR-J: The \( \text{test} \) message in \( \text{tarqueue}(\langle q, p \rangle) \) 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, \( \text{testlink}(p) = \langle p, q \rangle \), and by TAR-C(b), \( p \in \text{testset}(f) \). Thus, by GC-C, \( \text{minlink}(f) = \text{nil} \), and by COM-C \( \text{awake} = \text{true} \).

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

(3b) \( A_3(s', \pi) \) is empty. The only variables that are possibly changed are \( \text{Istatus}(\langle p, q \rangle) \), \( \text{tarqueue}'s \), and \( \text{testlink}(p) \), none of which is reflected (directly) in the state of \( \text{GC} \). Thus \( \text{accmin}(f) \) does not change and \( S_3(s') = S_3(s) \).

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

TAR-C(b): If \( \text{testlink}(p) \neq \text{nil} \) 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 \( l > \text{level}(f) \) in \( s' \), nothing affecting the predicate changes in going from \( s' \) to \( s \). Suppose \( l \leq \text{level}(f) \) in \( s' \).

Claims about \( s' \):

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

   Case A: \( c \neq \text{core}(f) \).

2. \( \text{Istatus}(\langle q, p \rangle) \neq \text{rejected} \), by TAR-E(c).
Section 4.2.3: TAR Simulates GC

3. The test(l, c) message in tarqueue((q, p)) is a protocol message for (q, p), by Claim 2.

The accept message added in s is a protocol message for (q, p). There is no change that affects the truth of the predicate for p.

Case B: c = core(f).

Case B.1: testlink(p) ≠ (p, q).

4. There is no protocol message for (p, q), by TAR-D.
5. The test(l, c) message in tarqueue((q, p)) is a protocol message for (q, p), by Claim 4.

The reject message added in s is a protocol message for (q, p). No change affects the truth of the predicate for p.

Case B.2: testlink(p) = (p, q).

6. There is a link (p, r), r ≠ q, with Istatus((p, r)) = unknown, by assumption for Case B.2.
7. There is no protocol message for (p, r), by Claim 6 and TAR-D.

Case B.2.1: Istatus((q, p)) ≠ rejected.
8. There is a test(l', c') message in tarqueue((p, q)) and Istatus((p, q)) = unknown, by assumptions for Case B.2.1 and TAR-P.
9. The test(l, c) message in tarqueue((q, p)) is a protocol message for (q, p), by assumptions for Case B.2.1.

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

Case B.2.2 Istatus((q, p)) = rejected.
10. The test(l, c) message in tarqueue((q, p)) is the protocol message for (p, q), by assumptions for Case B.2.2.
11. testlink(q) ≠ (q, p), 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 (p, q) by Claim 10, is removed in s; testlink(p) is now (p, r).
Section 4.2.3: TAR Simulates GC

and \((p, r)\) 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 \(l > \text{level}(f)\) in \(s'\), nothing affecting the predicate changes in going from \(s'\) to \(s\). Suppose \(l \leq \text{level}(f)\) in \(s'\).

Claims about \(s'\):

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

   Case A: \(c \neq \text{core}(f)\).

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

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

   Case B: \(c = \text{core}(f)\).

   Case B.1: \(\text{testlink}(p) \neq \langle p, q \rangle\).

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

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

   Case B.2: \(\text{testlink}(p) = \langle p, q \rangle\).

6. There is a link \(\langle p, r \rangle, r \neq q\), with \(\text{lstatus}(\langle p, r \rangle) = \text{unknown}\), 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: \(\text{lstatus}(\langle q, p \rangle)\) = rejected.

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

   No changes affect the truth of the predicate for \(q\). For \(p\): The \(\text{TEST}(l, c)\) message in \(\text{tarqueue}(\langle q, p \rangle)\) is the protocol message for \(\langle p, q \rangle\). It is removed in \(s\). A \text{TEST} message is added to \(\text{tarqueue}(\langle p, r \rangle)\) in \(s\), where \(\text{lstatus}(\langle p, r \rangle) = \text{unknown}\), and \(\text{testlink}(p) = \langle p, r \rangle\) by code.
Section 4.2.3: TAR Simulates GC

**Case B.2.2:** \textit{lstatus}((q,p)) \neq \text{rejected}.

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

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

For \(p\): The \textit{test}(l',c') message of Claim 9 is the protocol message for \(p,q\). The rest of the argument is as in Case B.2.1.

**TAR-E:** (a) Suppose a \textit{test} message is added to \textit{tarqueue}(p,r). As in \(\pi = \text{SendTest}(p)\), Case 1. (c) As in Case 1.

**TAR-F:** The only case of interest is when an \textit{accept} message is added to \textit{tarqueue}(p,q) in \(s\).

\textbf{Claims about }s':

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

Claims 7 and 8 are still true in \(s\).

**TAR-G:** The only case of interest is when a \textit{reject} message is added to \textit{tarqueue}(p,q).

\textbf{Claims about }s':

1. \textit{test}(l,c) is in \textit{tarqueue}(q,p), by precondition.
2. \(c = \text{core}(f)\), by assumption.
3. \textit{testlink}(p) \neq (p,q), by assumption.
4. If \textit{lstatus}(q,p) \neq \text{rejected}, then \(c = \text{core}(\text{fragment}(q))\), by Claim 1 and TAR-E(b).
Section 4.2.3: TAR Simulates GC

5. If \( lstatus((q,p)) \neq \text{rejected} \), then \( f = \text{fragment}(q) \), by Claim 4 and COM-F.
6. If \( lstatus((q,p)) = \text{rejected} \), then \( f = \text{fragment}(q) \), by TAR-B.
7. \( f = \text{fragment}(q) \), 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 \( \text{testset}(f) \). But when that happens, there are no unknown links of \( p \).

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

v) \( \pi \) is \( \text{ReceiveAccept}((q,p)) \). Let \( f = \text{fragment}(p) \) in \( s' \).

\( 3c) A_3(s', \pi) = \text{TestNode}(p) \).

Claims about \( s' \):

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

By Claims 6, 10, and 3, \( \text{TestNode}(p) \) is enabled in \( s' \).

Claims about \( s \):

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

By Claims 11 and 13, the effects of \( \text{TestNode}(p) \) are mirrored in \( s \).
Section 4.2.3: TAR Simulates GC

(3a) TAR-D: In \( s' \), ACCEPT in \( \text{tarqueue}(q,p) \) is a protocol message for \( (p,q) \). 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 \( \text{testset}(f) \).

vi) \( \pi \) is \( \text{ReceiveReject}((q,p)) \). Let \( f = \text{fragment}(p) \) in \( s' \).

Case 1: There is a link \( (p,r), r \neq q \), with \( \text{status}(p,r) = \text{unknown} \).

(3b) \( A_3(s', \pi) \) is empty. Obviously \( S_3(s') = S_3(s) \).

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

1. \( \text{REJECT} \) is in \( \text{tarqueue}(q,p) \), by assumption.
2. The \( \text{REJECT} \) in \( \text{tarqueue}(q,p) \) is a protocol message for \( (p,q) \), by Claim 1.
3. \( \text{testlink}(p) = (p,q) \), by Claim 2 and TAR-D.
4. There is only one protocol message for \( (p,q) \) 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. \( p \in \text{testset}(f) \), by Claim 3 and TAR-C(b).

TAR-B: Suppose \( \text{status}(p,q) \) goes from unknown in \( s' \) to rejected in \( s \). By TAR-G, \( f = \text{fragment}(q) \) in \( s' \). By TAR-A(b), \( (p,q) \not\in \text{subtree}(f) \) in \( s' \). Both facts are still true in \( s \).

TAR-C(b): By Claim 6.

TAR-C(c): In \( s \), \( \text{testlink}(p) = (p,r) \), and the TEST message is the sole protocol message for \( (p,r) \) by Claim 5.

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

TAR-E(a): Suppose a TEST message is added to some \( \text{tarqueue}(p,r) \). As in \( \pi = \text{SendTest}(p) \), Case 1.

TAR-E(c): The only case of interest is when \( \text{status}(p,q) \) goes from unknown in \( s' \) to rejected in \( s \). But by Claims 2 and 4, there is no TEST message in \( \text{tarqueue}(p,q) \) in \( s' \) if \( \text{status}(p,q) = \text{unknown} \).
Section 4.2.3: TAR Simulates GC

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

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

---

Case 2: There is no link $(p, r), r \neq q$, with $\text{Istatus}((p, r)) = \text{unknown}$.

$(3c) A_3(s', \pi) = \text{TestNode}(p)$.

Claims about $s'$:

1. REJECT is in $\text{tarqueue}((q, p))$, by precondition.
2. $\text{testlink}(p) = (p, q)$, by Claim 1 and TAR-D.
3. $p \in \text{testset}(f)$, by Claim 2 and TAR-C(b)
4. $\text{minlink}(f) = \text{nil}$, by Claim 3 and GC-C.
5. $\text{fragment}(q) = f$, by Claim 1 and TAR-G.
6. $(p, q)$ is not external, by Claim 5.
7. There is no external link $(p, r), r \neq q$, 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. $p \notin \text{testset}(f)$, by code.
9. There is no external link of $p$, by Claims 6 and 7 and code.
10. $\text{accmin}(f)$ 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 $s$, $\text{testlink}(p) = \text{nil}$. We must show there is no protocol message for any link of $p$. In $s'$, the REJECT message in $\text{tarqueue}((q, p))$ 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.
Section 4.2.3: TAR Simulates GC

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) \( \pi \) is ComputeMin(\( f \)).

(3c) \( A_3(s', \pi) = \pi \). Since \( accmin(\( f \)) = \text{nil} \) in \( s \) because \( minlink(\( f \)) = \text{nil} \) in \( s \), it is easy to see that \( \pi \) is enabled in \( S_3(s') \) and that its effects are mirrored in \( S_3(s) \).

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

viii) \( \pi \) is ChangeRoot(\( f \)).

(3c) \( A_3(s', \pi) = \pi \). It is easy to see that \( \pi \) is enabled in \( S_3(s') \) and that its effects are mirrored in \( S_3(s) \).

(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 \( \text{awake} = \text{true} \) in \( s' \), and is still true in \( s \).

ix) \( \pi \) is Merge(\( f, g \)).

(3c) \( A_3(s', \pi) = \pi \). After noting that \( accmin(\( h \)) = \text{nil} \) in \( s \) because \( testset(\( h \)) = \text{nodes}(\( h \)) \) in \( s \), it is easy to see that \( \pi \) is enabled in \( S_3(s') \) and that its effects are mirrored in \( S_3(s) \).

(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 \( \text{nodes}(\( f \)) \) or \( \text{nodes}(\( g \)) \) is in \( \text{testset}(\( f \)) \) or \( \text{testset}(\( g \)) \) in \( s' \). By TAR-C(b), \( testlink(\( r \)) = \text{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 \( \text{tarqueue}(\( \langle p, q \rangle \)) \) or in \( \text{tarqueue}(\( \langle q, p \rangle \)) \), where \( \langle p, q \rangle = minlink(\( f \)) \) in \( s' \). Since \( \langle p, q \rangle = \text{core}(\( h \)) \) in \( s \), done.

TAR-E(b): By TAR-O, there is no TEST(\( l, c \)) message in \( \text{tarqueue}(\( \langle p, q \rangle \)) \) with \( lstatus(\( \langle p, q \rangle \)) \neq \text{rejected} \) in \( s' \), for any \( p \) in \( \text{nodes}(\( f \)) \) or \( \text{nodes}(\( g \)) \). Thus, the same is true in \( s \) for any \( p \) in \( \text{nodes}(\( h \)) \), and the predicate is vacuously true in \( s \) for \( h \).
Section 4.2.3: TAR Simulates GC

TAR-E(c): If test(l, c) is in tarqueue((p, q)) and lstatus((p, q)) = rejected in s', then it is a protocol message for (q, p) 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((p, q)) in s', it is a protocol message for (q, p) 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) ≥ level(fragment(q)) in s'. Then fragment(p) = h ≠ fragment(q) in s, and level(h) (in s) > level(f) (in s') ≥ level(fragment(q)) (in s' and s).

TAR-H: By code, rootchanged(h) = false. Since minlink(h) = nil by code, lstatus(minlink(f)) ≠ 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) A_3(s', π) = π. It is easy to see that π is enabled in 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 S_3(s).

Let (q, p) = minlink(g). If p ∈ 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 ∉ testset(f) in s'. If minlink(f) ≠ 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 ∈ nodes(f), by precondition.
3. p ∉ testset(f), by assumption.
4. minlink(f) = nil, by assumption.
5. q ∈ nodes(g), by COM-A.
6. f ≠ g, by Claim 1.
7. accmin(f) = (r, t), for some r and t, by Claims 2 through 6.
8. fragment(t) ≠ g, by Claims 1 and 7 and GC-A.
Section 4.2.3: TAR Simulates GC

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

By Claims 7, 8 and 12, \(\text{accmin}(f) = (r, t)\) 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 \(\text{minlink}(g) \neq \text{nil}\), \(\text{testset}(g) = \emptyset\) in \(s'\). By TAR-C(b), \(\text{testlink}(p) = \text{nil}\) in \(s'\) for all \(p \in \text{nodes}(g)\). There is no change for \(p \in \text{nodes}(f)\) in \(s'\) in going from \(s'\) to \(s\). Thus the predicate is true in \(s\) for \(f\).

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

TAR-E(a): OK because \(\text{core}(f)\) does not change.

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

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

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

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

Suppose \(\text{fragment}(q) = f\). Since \(\text{fragment}(q)\) is the same in \(s\) as in \(s'\), and since \(\text{fragment}(p) \neq g\) in \(s'\), the predicate is still true in \(s\).
Section 4.2.3: TAR Simulates GC

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

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

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

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

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

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

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

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

This algorithm (as well as the original one) correctly handles “leftover” $\text{REPORT}$ messages. Recall that a $\text{REPORT}$ message is sent in both directions over the core $(p, q)$ of a fragment $f$. Suppose the root $p$ receives its $\text{REPORT}$ message first, and the other $\text{REPORT}$ 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 $\text{REPORT}$ and $\text{FIND}$ messages, the state of $q$ remains such that when the leftover $\text{REPORT}$ message is received, the only change is the removal of the message.

Define automaton $DC$ (for “Distributed ComputeMin”) as follows.
Section 4.2.4: DC Simulates GC

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

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

For each node \( p \), there are the following variables:

- \( \text{dcstatus}(p) \), either find or unfind;
- \( \text{findcount}(p) \), a nonnegative integer;
- \( \text{bestlink}(p) \), a link of \( G \) or nil;
- \( \text{bestwt}(p) \), a weight or \( \infty \); and
- \( \text{inbranch}(p) \), a link of \( G \) or nil.

For each link \( (p, q) \), there are associated three variables:

- \( \text{dcqueue}_p((p, q)) \), a FIFO queue of messages from \( p \) to \( q \) waiting at \( p \) to be sent;
- \( \text{dcqueue}_{pq}((p, q)) \), a FIFO queue of messages from \( p \) to \( q \) that are in the communication channel; and
- \( \text{dcqueue}_q((p, q)) \), 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, \( \text{answered}(l) \), one for each \( l \in L(G) \), and Boolean variable \( \text{awake} \).

In the start state of DC, fragments has one element for each node in \( V(G) \); for fragment \( f \) corresponding to node \( p \), \( \text{subtree}(f) = \{p\} \), \( \text{core}(f) = \text{nil} \), \( \text{level}(f) = 0 \), \( \text{rootchanged}(f) \) is false, and \( \text{testset}(f) \) is empty. For each \( p \), \( \text{dcstatus}(p) = \text{unfind} \), \( \text{findcount}(p) = 0 \), \( \text{bestlink}(p) \) is the minimum-weight external link of \( p \), \( \text{bestwt}(p) \) is
Section 4.2.4: DC Simulates GC

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 $dcqueue((p,q))$ is defined to be $dcqueue_q((p,q)) || dcqueue_{pq}((p,q)) || dcqueue_p((p,q))$.

A REPORT($w$) message is headed toward $p$ if either it is in $dcqueue((q,p))$ for some $q$, or it is in some $dcqueue((q,r))$, where $q \in subtree(r)$ and $r \in subtree(p)$. A FIND message is headed toward $p$ if it is in some $dcqueue((q,r))$ and $p$ is in $subtree(r)$. A message is said to be in $subtree(f)$ if it is in some $dcqueue((q,p))$ 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 $dcstatus(p) = unfnd$ 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($q,p$) 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($p,q$), $(p,q) \in L(G)$
  Preconditions:
  awake = true
  $(p,q) \in subtree(fragment(p))$ or $(p,q) = minlink(fragment(p))$
  answered($p,q$) = false
Section 4.2.4: DC Simulates GC

Effects:
\[ \text{answered}((p, q)) := true \]

- NotInTree((p, q)), \((p, q) \in L(G)\)
  Preconditions:
  \[ \text{fragment}(p) = \text{fragment}(q) \text{ and } (p, q) \notin \text{subtree}(\text{fragment}(p)) \]
  \[ \text{answered}((p, q)) = false \]
  Effects:
  \[ \text{answered}((p, q)) := true \]

Internal actions:

- ChannelSend((p, q), m), \((p, q) \in L(G), m \in M\)
  Preconditions:
  \[ m \text{ at head of } \text{dcqueue}_p((p, q)) \]
  Effects:
  \[ \text{dequeue}(\text{dcqueue}_p((p, q))) \]
  \[ \text{enqueue}(m, \text{dcqueue}_{pq}((p, q))) \]

- ChannelRecv((p, q), m), \((p, q) \in L(G), m \in M\)
  Preconditions:
  \[ m \text{ at head of } \text{dcqueue}_{pq}((p, q)) \]
  Effects:
  \[ \text{dequeue}(\text{dcqueue}_{pq}((p, q))) \]
  \[ \text{enqueue}(m, \text{dcqueue}_q((p, q))) \]

- TestNode(p), \(p \in V(G)\)
  Preconditions:
  \[ - \text{let } f = \text{fragment}(p) - \]
  \[ p \in \text{testset}(f) \]
  if \((p, q)\), the minimum-weight external link of \(p\), exists
  then \[ \text{level}(f) \leq \text{level}(\text{fragment}(q)) \]
  \[ \text{dcstatus}(p) = \text{find} \]
  Effects:
  \[ \text{testset}(f) := \text{testset}(f) - \{p\} \]
  if \((p, q)\), the minimum-weight external link of \(p\), exists then
  if \[ \text{wt}(p, q) < \text{bestwt}(p) \]
  then \[ \text{bestlink}(p) := (p, q) \]
  \[ \text{bestwt}(p) := \text{wt}(p, q) \]
  execute procedure \text{Report}(p)
Section 4.2.4: DC Simulates GC

- **ReceiveReport**\((q, p), (q, p) \in L(G)\)

  Preconditions:
  
  \[
  \text{REPORT}(w) \text{ message at head of } dcqueue_p((q, p))
  \]

  Effects:
  
  \[
  \begin{align*}
  &\text{dequeue}(dcqueue_p((q, p))) \\
  &\text{if } (p, q) \neq \text{inbranch}(p) \text{ then } [ \\
  &\quad \text{findcount}(p) := \text{findcount}(p) - 1 \\
  &\quad \text{if } w < \text{bestwt}(p) \text{ then } [ \\
  &\quad\quad \text{bestwt}(p) := w \\
  &\quad\quad \text{bestlink}(p) := (p, q) ] \\
  &\quad \text{execute procedure } \text{Report}(p) ] \\
  &\text{else } [ \\
  &\quad \text{if } dcstatus(p) = \text{find} \text{ then enqueue(REPORT}(w), dcqueue_p((q, p))) ]
  \end{align*}
  \]

- **ReceiveFind**\((q, p), (q, p) \in L(G)\)

  Preconditions:
  
  \[
  \text{FIND message at head of } dcqueue_p((q, p))
  \]

  \[
  \text{AfterMerge}(p, q) \text{ not enabled}
  \]

  Effects:
  
  \[
  \begin{align*}
  &\text{dequeue}(dcqueue_p((q, p))) \\
  &\text{dcstatus}(p) := \text{find} \\
  &\text{inbranch}(p) := (p, q) \\
  &\text{bestlink}(p) := \text{nil} \\
  &\text{bestwt}(p) := \infty \\
  &\quad \text{let } S = \{(p, r) : (p, r) \in \text{subtree}(\text{fragment}(p)), r \neq q\} \quad - \\
  &\quad \text{findcount}(p) := |S| \\
  &\quad \text{enqueue(FIND, dcqueue_p(l)) for all } l \in S
  \end{align*}
  \]

- **Procedure** \(\text{Report}(p), p \in V(G)\)

  \[
  \begin{align*}
  &\text{if findcount}(p) = 0 \text{ and } p \not\in \text{testset}(\text{fragment}(p)) \text{ then } [ \\
  &\quad \text{dcstatus}(p) := \text{unfind} \\
  &\quad \text{enqueue(REPORT(bestwt}(p)), dcqueue_p(\text{inbranch}(p)))) ]
  \end{align*}
  \]

- **ChangeRoot**\((f), f \in \text{fragments}\)

  Preconditions:
  
  \[
  \begin{align*}
  &awake = \text{true} \\
  &\text{rootchanged}(f) = \text{false} \\
  &\text{minlink}(f) \neq \text{nil}
  \end{align*}
  \]

  Effects:
Section 4.2.4: DC Simulates GC

rootchanged(f) := true

- **Merge**(*f*, *g*), *f*, *g* ∈ fragments
  
  Preconditions:
  
  \( f \neq g \)
  
  rootchanged(f) = rootchanged(g) = true
  
  minege(f) = minege(g)
  
  Effects:
  
  add a new element *h* to fragments
  
  subtree(h) := subtree(f) ∪ subtree(g) ∪ minege(f)
  
  core(h) := minege(f)
  
  level(h) := level(f) + 1
  
  rootchanged(h) := false
  
  testset(h) := nodes(h)
  
  - let \( (p, q) = \text{minlink}(f) \) —
  
  enqueue(FIND, dcqueue((p, q)))
  
  delete *f* and *g* from fragments

- **AfterMerge**(*p*, *q*), *p*, *q* ∈ \( V(G) \)

  Preconditions:

  \( (p, q) = \text{core(fragment}(p)) \)

  NO FIND message in dcqueue((q, p))

  dcstatus(q) = unfind

  NO REPORT message in dcqueue((q, p))

  Effects:

  enqueue(FIND, dcqueuep((p, q)))

- **Absorb**(*f*, *g*), *f*, *g* ∈ fragments

  Preconditions:

  rootchanged(g) = true

  level(g) < level(f)

  - let \( (q, p) = \text{minlink}(g) \) —

  fragment(p) = *f*

  Effects:

  subtree(f) := subtree(f) ∪ subtree(g) ∪ minege(g)

  if \( p \in \text{testset}(f) \) then [

  testset(f) := testset(f) ∪ nodes(g)

  if dcstatus(p) = find then [
Section 4.2.4: DC Simulates GC

\[ \text{enqueue(FIND, dcqueue}_{p}(p,q)) \]
\[ \text{findcount}(p) := \text{findcount}(p) + 1 \]
\[ \text{delete } g \text{ from fragments} \]

Define the following predicates on \( \text{states}(DC) \), using these definitions.

A child \( q \) of \( p \) is completed if no node in \( \text{subtree}(q) \) is in \( \text{testset}(\text{fragment}(p)) \), and no REPORT is headed toward \( p \) in \( \text{subtree}(q) \) or in \( \text{dcqueue}((q,p)) \). Node \( p \) is up-to-date if either \( \text{subtree}(\text{fragment}(p)) = \{p\} \), or the following two conditions are met:
(1) following \( \text{inbranches} \) from \( p \) leads along edges of \( \text{subtree}(\text{fragment}(p)) \) toward and over \( \text{core}(f) \), and
(2) if \( p \in \text{testset}(\text{fragment}(p)) \), then \( \text{destatus}(p) = \text{find} \).

Given node \( p \), define \( C_p \) to be the set \( \{r: \text{either } r = p \text{ and } p \in \text{testset}(\text{fragment}(p)) \text{, or } r \text{ is in } \text{subtree}(q) \text{ for some completed child } q \text{ of } p\} \).

All free variables are universally quantified, except that \( f = \text{fragment}(p) \), 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 \( \text{REPORT}(w) \) is in \( \text{dcqueue}((q,p)) \) and \( \text{inbranch}(p) \neq (p,q) \), then
  (a) if \( (p,q) = \text{core}(f) \), then a FIND message is ahead of the REPORT in \( \text{dcqueue}((q,p)) \);
  (b) \( (q,p) = \text{inbranch}(q) \);
  (c) \( \text{bestwt}(q) = w \);
  (d) \( \text{destatus}(q) = \text{unfind} \);
  (e) every child of \( q \) is completed;
  (f) \( q \notin \text{testset}(f) \); and
  (g) if \( (p,q) \neq \text{core}(f) \), then \( \text{destatus}(p) = \text{find} \), and \( q \) is a child of \( p \).

- **DC-B:** If \( \text{REPORT}(w) \) is in \( \text{dcqueue}((q,p)) \) and \( \text{inbranch}(p) = (p,q) \), then
  (a) either \( (p,q) = \text{core}(f) \) or \( p \) is a child of \( q \); and
  (b) if \( (p,q) \neq \text{core}(f) \), then \( \text{destatus}(p) = \text{unfind} \).

- **DC-C:** If \( \text{REPORT}(w) \) is in \( \text{dcqueue}((q,p)) \) and \( (p,q) = \text{core}(f) \), then
  (a) \( q \) is up-to-date;
  (b) \( \text{destatus}(q) = \text{unfind} \); and
  (c) \( \text{bestwt}(q) = w \).

- **DC-D:** If \( \text{FIND} \) is in \( \text{dcqueue}((q,p)) \), then
  (a) if \( (p,q) \neq \text{core}(f) \) then \( p \) is a child of \( q \) and \( \text{destatus}(q) = \text{find} \).
Section 4.2.4: DC Simulates GC

(b) \textit{dcstatus}(p) = \textit{unfind}; and
(c) every node in \textit{subtree}(p) is in \textit{testset}(f).

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

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

- **DC-G:** If \textit{AfterMerge}(p, q) is enabled, then every node in \textit{subtree}(q) is in \textit{testset}(f).

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

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

- **DC-J:** If \textit{dcstatus}(p) = \textit{find} and \( \textit{core}(f) = (p, q) \), then a \textit{FIND} message is in \textit{dcqueue}((p, q)), or \textit{dcstatus}(q) = \textit{find}, or a \textit{REPORT} message is in \textit{dcqueue}((q, p)).

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

- **DC-L:** If \( \textit{inbranch}(p) \neq \textit{nil} \), then \( \textit{inbranch}(p) = (p, q) \) for some \( q \), and \( (p, q) \in \textit{subtree}(f) \).

- **DC-M:** \textit{findcount}(p) \geq 0.

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

- **DC-O:** The only possible values of \textit{dcqueue}((p, q)) are empty, or \textit{FIND}, or \textit{REPORT}, or \textit{FIND} followed by \textit{REPORT} (only if \( (p, q) = \textit{core}(f) \)), or \textit{REPORT} followed by \textit{FIND} (only if \( (p, q) \neq \textit{core}(f) \)).
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 $\mathcal{M}_4 = (S_4, A_4)$ from DC to GC.

Define the function $S_4$ from states(DC) to states(GC) by ignoring the message queues, and the variables destatus, 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 $A_4$ as follows. Let $s$ be a state of DC and $\pi$ 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 $\pi = \text{ReceiveReport}((q,p),w)$. If $(p,q) = \text{core}(f)$ and destatus($p$) = unfind and $w > \text{bestwt}(p)$, then $A_4(s,\pi) = \text{ComputeMin}($fragment$(p))$. Otherwise $A_4(s,\pi)$ is empty.

- If $\pi = \text{ChannelSend}((q,p),m)$, $\text{ChannelRecv}((q,p),m)$, $\text{ReceiveFind}((q,p))$ or $\text{AfterMerge}(p,q)$, then $A_4(s,\pi)$ is empty.

- For all other values of $\pi$, $A_4(s,\pi) = \pi$.

The following predicates are true in any state of DC satisfying $(P'_{GC} \circ S_4) \land P_{DC}$. Recall that $P_{GC} = (P'_{COM} \circ S_2) \land P_{GC}$. If $P_{GC}(S_4(s))$ is true, then the GC predicates are true in $S_4(s)$, the COM predicates are true in $S_2(S_4(s))$, and the HI predicates are true in $S_1(S_2(S_4(s)))$. 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 dequeue($(q,p)$) and $(p,q) = \text{core}(f)$ and destatus($p$) = unfind, then
  (a) if $w < \text{bestwt}(p)$, then the minimum-weight external link $l$ of $f$ is closer to $q$ than to $p$, and $\text{wt}(l) = w$;
  (b) if $w > \text{bestwt}(p)$, then the minimum-weight external link $l$ of $f$ is closer to $p$ than to $q$, and $\text{wt}(l) = \text{bestwt}(p)$; and
  (c) if $w = \text{bestwt}(p)$, then $w = \infty$ and there is no external link of $f$.

Proof:
Section 4.2.4: DC Simulates GC

1. REPORT(w) is at head of dcqueue((q,p)), 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 /∈ testset(f), by Claims 4 and 5.
8. No FIND is in dcqueue((q,p)), 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 /∈ 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) = ∞, 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) ≠ ∞, 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 = ∞, 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 ≠ ∞, 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((q,p)) and is not headed toward mw-root(f), then inbranch(p) = (p,q).

Proof: If (p,q) = core(f), then inbranch(p) = (p,q) by DC-A(a). Suppose (p,q) ≠ core(f), and, in contradiction, that inbranch(p) ≠ (p,q). 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((q,p)) 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) = (p,q).
Section 4.2.4: DC Simulates GC

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

\( \square \)

- 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 \( \text{dcqueue}(q, r) \), by assumption.
2. \( p \in \text{subtree}(r) \), by assumption.
3. \( \text{destatus}(r) = \text{unfind} \), by Claim 1 and DC-D(b).
4. \( \text{destatus}(t) = \text{unfind} \) for all \( t \in \text{subtree}(r) \), by Claim 3 and DC-H(a).
5. No FIND message is in \( \text{dcqueue}(t, u) \), for any \( (t, u) \in \text{subtree}(r) \), by Claim 4 and DC-D(a).

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

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

Claim 9 completes the proof. \( \square \)

- DC-T: If \( (p, q) = \text{core}(f) \), no FIND is in \( \text{dcqueue}(p, q) \), \( p \) is up-to-date, and \( \text{destatus}(q) = \text{unfind} \), then \( q \) is up-to-date.

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

70
Section 4.2.4: DC Simulates GC

6. No FIND is in dequeue((q, p)), by Claim 3 and DC-D(b) and (c).
7. AfterMerge(p, q) is not enabled, by Claim 6.
8. inbranch(q) = (q, p), by Claims 5 and 7 and DC-F.
9. q ∉ 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 M4, PDC, and PGC.

Proof: By inspection, the types of DC, GC, M4, and PDC are correct. By Corollary 16, PPGC is a predicate true in every reachable state of GC.

(1) Let s be in start(DC). Obviously, PDC is true in s, and S4(s) is in start(GC).

(2) Obviously, A4(s, π)|ext(GC) = π|ext(DC).

(3) Let (s', π, s) be a step of DC such that PPGC is true of S4(s') and PDC 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). A4(s', π) = π. Obviously, S4(s')πS4(s) is an execution fragment of GC and PDC is true in s.

ii) π is ChannelSend(l,m) or ChannelRecv(l,m). A4(s', π) is empty. Obviously S4(s) = S4(s') and PDC is true in s.

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

(3c) A4(s', π) = π. Obviously, π is enabled in S4(s'). To show the effects are mirrored in S4(s), we must show that accmin(f) is updated properly (which is obvious) and that minlink(f) is unchanged. Since p ∈ testset(f) in s', minlink(f) = nil in s' by GC-C. If accmin(f) = nil, or if p has an external link in s', then accmin(f) = nil in s, and minlink(f) is still nil in s. If some q ≠ 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':

71
Section 4.2.4: DC Simulates GC

1. \( \text{dstatus}(p) = \text{find} \), by precondition.
2. \( p \in \text{testset}(f) \), by precondition.
3. If \((p, u)\), the minimum-weight external link of \( p \), exists, then \( \text{level}(f) \leq \text{level}(\text{fragment}(u)) \), 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 \((p, r) = \text{core}(f)\), then no REPORT is in \( \text{dcqueue}((p, r)) \), for any \( r \), by Claim 1 and DC-C(b).
7. If a REPORT is in \( \text{dcqueue}((p, r)) \), then \( \text{inbranch}(r) = (r, p) \), for any \( r \), by Claim 1 and DC-A(d).
8. AfterMerge\((r, t)\), where \( p \in \text{subtree}(t) \), is not enabled, by Claim 1 and DC-H(a).
9. If \( \text{bestlink}(p) = \text{nil} \), then \( \text{bestwt}(p) = \infty \) 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 \( \text{bestlink}(p) \neq \text{nil} \), 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 \( p \); \( \text{wt}(l) = \text{bestwt}(p) \) and \( \text{level}(f) \leq \text{level}(\text{fragment}(\text{target}(l))) \), by Claims 2 and 4 and DC-K(c).

Case 1: \( \text{findcount}(p) \neq 0 \) in \( s' \).

More claims about \( s' \):

11. \( \text{findcount}(p) \neq 0 \), by assumption.
12. \( \text{findcount}(p) > 0 \), 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 \( \text{subtree}(r) \) is in \( \text{testset}(f) \), or a REPORT is in \( \text{subtree}(r) \) or \( \text{dcqueue}((r, p)) \) headed toward \( p \), by Claim 13.

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

DC-C: By Claim 6, changing \( \text{bestwt}(p) \) is OK.

DC-D(c): By Claim 5, removing \( p \) from \( \text{testset}(f) \) is OK.

DC-G: By Claim 8 and the fact that \( \text{dstatus}(p) \) is still find in \( s \), removing \( p \) from \( \text{testset}(f) \) is OK.
Section 4.2.4: DC Simulates GC

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-minnode(f), then by Claim 4, removing p from testset(p) is OK.

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

More claims about s':

15. findcount(p) = 0, by assumption.
16. If (p, q) = core(f) and inbranch(q) ≠ (q, p), then a FIND is in dcqueue((p, q)), 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) ≠ core(f), then destatus(q) = find, by Claim 1 and DC-H(a).
19. If REPORT is in dcqueue((q, p)), then (p, q) = core(f), by Claim 4 and DC-B(a).
20. No REPORT is in dcqueue((p, q)), by Claim 1 and DC-R.
21. If FIND is in dcqueue((p, q)), then (p, q) = core(f), by Claim 4 and DC-D(a).
22. Every node r ≠ p in subtree(p) has destatus(r) = unfind, by Claims 1 and 17 and DC-I(b).
23. Every node r ≠ 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) = (p, q), we need only consider the REPORT added to dcqueue((p, q)). (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((p, q)): If inbranch(q) = (q, p), then (p, q) = core(f), by Claim 4.

DC-B for REPORT that might be in dcqueue((q, p)): 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 destatus(p) to unfind and remove p from testset(f).

DC-E: The addition of a REPORT to dcqueue((p, q)) in s cannot cause After-Merge(q, p) to go from enabled in s' to disabled in s, by Claim 1.
Section 4.2.4: DC Simulates GC

DC-F: Cf. DC-E.

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

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

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

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

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((q,p),w). Let f = fragment(p) in s'.

(3b)/(3c) Case 1: (p, q) = core(f) and destatus(p) = unfind and w > bestwt(p) in s'. A_4(s', π) = ComputeMin(f).

Let (r, t) 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((q,p)), by precondition.
2. (p, q) = core(f), by assumption.
3. destatus(p) = unfind, by assumption.
4. w > bestwt(p), by assumption.
5. No FIND is in dcqueue((q,p)), 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. destatus(q) = unfind, by Claims 1 and 2 and DC-C(b).
9. bestwt(q) = w, by Claims 1 and 2 and DC-C(c).
10. p = mw-root(f) (so (r, t) exists), by Claims 1, 2, 3 and 4 and DC-P(b).
11. minlink(f) = nil, by Claims 1 and 10.
4.2.4: DC Simulates GC

12. \( \text{findcount}(p) = 0 \), by Claim 3 and DC-H(b).
13. \( \text{findcount}(q) = 0 \), 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. \( p \notin \text{testset}(f) \), by Claims 3 and 7.
17. \( q \notin \text{testset}(f) \), by Claims 6 and 8.
18. \( \text{testset}(f) = \emptyset \), by Claims 14 through 17.
19. \( \text{accmin}(f) = (r,t) \), by Claims 11 and 18.

By Claims 11, 18 and 19, \( \text{ComputeMin}(f) \) is enabled in \( s' \).

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

More claims about \( s' \):

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

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

Case 2: \( (p,q) \notin \text{core}(f) \) or \( \text{dcsstatus}(p) = \text{find} \) or \( w \leq \text{bestwt}(p) \) in \( s' \). \( A_4(s', \pi) \) is empty. We just need to verify that \( \text{minlink}(f) \) and \( \text{accmin}(f) \) are unchanged in order to show that \( S_4(s') = S_4(s) \).

Subcase 2a: \( (p,q) \notin \text{inbranch}(p) \) in \( s' \).

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

Suppose \( (p,q) \notin \text{inbranch}(p) \) in \( s' \).
Section 4.2.4: DC Simulates GC

Claims about s':

1. REPORT(w) is at the head of dequeue((q,p)), by precondition.
2. (p,q) \not\in inbranch(p), by assumption.
3. (p,q) \not\in core(f), by assumption.
4. destatus(p) = find, by Claims 1, 2 and 3 and DC-A(a).
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-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 dequeue((p,t)), destatus(t) = find, or REPORT is in dequeue((t,p)), 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-root(f), then REPORT is headed toward mw-root(f), by Claims 11 and 12.
14. If p = mw-root(f), then FIND is in dequeue((p,t)), destatus(t) = find, or REPORT is in dequeue((t,p)), 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 destatus(p) = find in s'. Since REPORT(w) is at the head of dequeue((q,p)), DC-A(a) implies that inbranch(p) = (p,q). 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 destatus(p) = unfind and w \leq bestval(p) in s'. As in Subcase 2b, inbranch(p) = (p,q). The only change is that the REPORT message is removed. If w = bestval(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.
Section 4.2.4: DC Simulates GC

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

(3a) Case 1: \( (p,q) = \text{inbranch}(p) \) in \( s' \).

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

Suppose \( \text{dcstatus}(p) = \text{unfind} \). The only change is the removal of the REPORT message from \( \text{dcqueue}((q,p)) \). By DC-B(a), either \( (p,q) = \text{core}(f) \), or \( p \in \text{subtree}(q) \) 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 \( \text{dcqueue}((q,p)) \), \( \text{dcstatus}(p) = \text{unfind} \) in \( s \).

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

Case 2: \( (p,q) \neq \text{inbranch}(p) \). Let \( (p,r) = \text{inbranch}(p) \).

Claims about \( s' \):

1. \( \text{REPORT}(w) \) is at head of \( \text{dcqueue}((q,p)) \), by precondition.
2. \( (p,q) \neq \text{inbranch}(p) \), by assumption.
3. \( (p,q) \neq \text{core}(f) \), by Claims 1 and 2 and DC-A(a).
4. \( (q,p) = \text{inbranch}(q) \), by Claims 1 and 2 and DC-A(b).
5. \( w = \text{bestwt}(q) \), by Claims 1 and 2 and DC-A(c).
6. \( \text{dcstatus}(q) = \text{unfind} \), 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. \( q \notin \text{testset}(f) \), by Claims 1 and 2 and DC-A(f).
9. \( \text{dcstatus}(p) = \text{find} \), by Claim 3 and DC-A(g).
10. If \( \text{REPORT} \) is in \( \text{dcqueue}(p,t) \), then \( \text{inbranch}(t) = (t,p) \), for any \( t \), by Claim 9 and DC-A(d).
Section 4.2.4: DC Simulates GC

11. \( p \) is up-to-date, by Claim 9 and DC-I(a).
12. \( \text{inbranch}(p) \) leads toward and over \( \text{core}(f) \), by Claim 11.
13. \( q \) is an uncompleted child of \( p \), by Claims 1, 2 and 12.
14. \( \text{findcount}(p) \geq 1 \), by Claims 11 and 13 and DC-K(a).
15. Only one \( \text{REPORT} \) is in \( \text{dcqueue}((q,p)) \), by Claim 1 and DC-O.
16. \( q \) is up-to-date, by Claims 4, 8 and 12.
17. If \( \text{REPORT} \) is in \( \text{dcqueue}((p,t)) \), then \( (p,t) \neq \text{core}(f) \), for all \( t \), by Claim 9 and DC-C(b).
18. If \( \text{bestwt}(p) = \infty \), then there is no external link of \( p \) (if \( p \not\in \text{testset}(f) \)) or of any node in the subtree of any completed child of \( p \), by Claim 11 and DC-F(b) and (c).
19. If \( \text{bestwt}(p) \neq \infty \), then following \( \text{bestlinks} \) from \( p \) leads to the minimum-weight external link \( l \) of all nodes in \( C_p \); \( wt(l) = \text{bestwt}(p) \); and \( \text{level}(f) \leq \text{level(\text{fragment(target(l)})}, \) by Claim 11 and DC-F(b) and (c).
20. If \( w = \infty \), then there is no external link of \( \text{subtree}(q) \), by Claims 5, 7, 8 and 16 and DC-K(b) and (c).
21. If \( w \neq \infty \), then following \( \text{bestlinks} \) from \( q \) leads to the minimum-weight external link \( l \) of \( \text{subtree}(q) \); \( wt(l) = w \), and \( \text{level}(f) \leq \text{level(\text{fragment(target(l)})}, \) by Claims 5, 7, 8 and 16 and DC-F(b) and (c).

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

More claims about \( s' \):

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

DC-A(c): by Claim 10, any change to \( \text{bestwt}(p) \) is OK.
DC-C: By Claim 17, changing \( \text{bestwt}(p) \) is OK.
DC-F: Cf. DC-G.
DC-G: Removing \( \text{REPORT} \) from \( \text{dcqueue}((q,p)) \) does not cause \( \text{AfterMerge}(p,q) \) to become enabled, by Claim 3.
DC-I(b): Let \( t \) be some node such that \( p \in \text{subtree}(t) \) and \( \text{dstatus}(t) = \text{find in } s' \). By Claims 24 and 25, either a \( \text{REPORT} \) message is in \( \text{subtree}(p) \) headed toward
Section 4.2.4: DC Simulates GC

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

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

DC-K(a): Since $\text{findcount}(p)$ 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: $p \notin \text{testset}(f)$ and $\text{findcount}(p) = 1$.

26. $p \notin \text{testset}(f)$, by assumption.
27. $\text{findcount}(p) = 1$, by assumption.
28. No find is headed toward $p$, by Claim 9 and DC-D(b).
29. If $(p, r) = \text{core}(f)$ and $\text{inbranch}(r) \neq (r, p)$, then find is in $\text{dcqueue}((p, r))$, by Claim 28 and DC-F.
30. No report is in $\text{dcqueue}((p, r))$, 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 $\text{dcqueue}((p, t))$, $t \neq r$, by Claims 7, 8 and 31 and DC-D(c).
33. If report is in $\text{dcqueue}((r, p))$, then $(p, r) = \text{core}(f)$, by Claim 9 and DC-B(a) and (b).
34. If $(p, r) \neq \text{core}(f)$, then $\text{destatus}(r) = \text{find}$, by Claims 9 and 12 and DC-H(a).
35. If find is in $\text{dcqueue}((p, r))$, then $(p, r) = \text{core}(f)$, by Claim 12 and DC-D(a).

DC-A: By Claim 10 and the fact that $\text{inbranch}(p) = (p, r)$, we need only consider the report added to $\text{dcqueue}((p, r))$. (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 $\text{dcqueue}((p, r))$: if $\text{inbranch}(r) = (r, p)$, then by Claim 12, $\text{core}(f) = (p, r)$.

DC-B for report in $\text{dcqueue}((r, p))$: By Claim 33, $\text{core}(f) = (p, r)$.

DC-C: By Claim 12, $\text{inbranch}(p)$ 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.
Section 4.2.4: DC Simulates GC

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

DC-E: The addition of the REPORT to dequeue((p,r)) in s cannot cause AfterMerge(r,p) to go from enabled in s' to disabled in s, because destatus(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 ≠ q of p, is in testset(f). By Claim 23, p ∉ testset(f).

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

DC-I(b): Let t ≠ p be such that p ∈ subtree(t) and destatus(t) = find in s'. By Claim 12, removing the REPORT from dequeue((q,p)) is compensated for by adding the REPORT to dequeue((p,r)).

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

DC-K: As in Subcase 2a.

DC-M: Claim 27 and code.

DC-O: by Claim 30 and DC-O and cc lc.

v) π is ReceiveFind((q,p)). Let f = fragment(p).

(3b) A₄(s', π) is empty. To show that S₄(s') = S₄(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 destatus(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 dequeue((q,p)), by precondition.
2. AfterMerge(p,q) is not enabled, by precondition.
3. If (p,q) ≠ core(f), then p is a child of q, by Claim 1 and DC-D(a).
4. If (p,q) ≠ core(f), then destatus(q) = find, by Claim 1 and DC-D(a).
Section 4.2.4: DC Simulates GC

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

DC-B: By Claim 9 and 12, changing $\text{destatus}(p)$ is OK.

DC-C: By Claim 8, changing $\text{destatus}(p)$ and $\text{bestud}(p)$ 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 $\text{destatus}(p)$ to find), removing FIND from $\text{dequeue}(\langle q, p \rangle)$ is OK.

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

DC-H(a): If $(p, q) = \text{core}(f)$, then changing $\text{destatus}(p)$ to find is OK. If $(p, q) \neq \text{core}(f)$, then Claim 15 implies that it is OK to change $\text{destatus}(p)$ to find.
DC-I: (a) If \((p, q) = \text{core}(f)\), then code gives the result, since \(\text{inbranch}(p)\) is set to \((p, q)\) and \(\text{destatus}(p)\) is set to find. If \((p, q) \neq \text{core}(f)\), 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) \(\text{findcount}(p) = |S| = \text{number of children of } p\). 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 \text{FIND} from \text{dequeue}((q,p)) is OK. Adding \text{FIND} to \text{dequeue}((p,r)), \((p,r) \in S\), is OK by Claim 16.

vi) \(\pi\) is \text{Merge}(f,g).

(3c) \(A_4(s', \pi) = \pi\). Obviously \(\pi\) is enabled in \(S_4(s')\). Effects are mirrored in \(S_4(s)\) if we can show \(\text{accmin}(h) = \text{minlink}(h) = \text{nil}\) in \(s\). Inspecting the code reveals that in \(s\), a \text{FIND} message is in \(\text{subtree}(h)\), so \(\text{minlink}(h) = \text{nil}\), and \(\text{nodes}(h) = \text{testset}(h)\), so \(\text{accmin}(h) = \text{nil}\).

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

1. \(f \neq g\), by precondition.
2. \(\text{rootchanged}(f) = \text{true}\), by precondition.
3. \(\text{rootchanged}(g) = \text{true}\), by precondition.
4. \(\text{minedge}(f) = \text{minedge}(g)\), by precondition.
5. \(\text{minlink}(f) \neq \text{nil}\), by Claim 2 and COM-B.

Let \((p, q) = \text{minlink}(f)\).
6. \(\text{minlink}(g) = (q, p)\), by Claims 1, 4 and 5.
7. No \text{REPORT} is headed toward \(\text{root}(f)\), by Claim 5.
8. No \text{REPORT} is headed toward \(\text{root}(g)\), by Claim 6.
9. No \text{FIND} is in \(\text{subtree}(f)\), by Claim 5.
10. No \text{FIND} is in \(\text{subtree}(g)\), by Claim 6.
11. \(\text{destatus}(r) = \text{unfind for all } r \in \text{nodes}(f)\), by Claim 5.
12. \(\text{destatus}(r) = \text{unfind for all } r \in \text{nodes}(g)\), by Claim 6.
13. \((p, q)\) is the minimum-weight external link of \(f\), by Claim 5 and COM-A.
14. \((q, p)\) is the minimum-weight external link of \(g\), by Claim 6 and COM-A.
15. \(\text{testset}(f) = \emptyset\), by Claim 5 and GC-C.
Section 4.2.4: DC Simulates GC

16. $testset(g) = \emptyset$, by Claim 6 and GC-C.
17. If REPORT is in $dequeque((r,t))$, then $inbranch(t) = (t,r)$, for all $(r,t) \in subtree(f)$, by Claims 9 and 11 and DC-A(a) and (f).
18. If REPORT is in $dequeque((r,t))$, then $inbranch(t) = (t,r)$, for all $(r,t) \in subtree(f)$, by Claims 10 and 12 and DC-A(a) and (f).
19. If REPORT is in $dequeque((r,t))$ and $(r,t) = core(f)$, then $r = root(f)$, by Claim 7.
20. If REPORT is in $dequeque((r,t))$ and $(r,t) = core(g)$, then $r = root(g)$, by Claim 8.
21. If REPORT is in $dequeque((r,t))$ 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 $dequeque((r,t))$ 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 $dequeque((r,t))$, 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 $dequeque((r,t))$, 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. $dequeque((p,q))$ is empty, by Claim 13 and DC-A(g), DC-B(a) and DC-D(a).
26. $dequeque((q,p))$ 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. $dequeque((p,q))$ 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 $dequeque((r,t))$, then $inbranch(t) = (t,r)$, for all $(r,t) \in subtree(h)$, by Claims 17, 18, 25, 26 and 29.
36. If REPORT is in $dequeque((r,t))$, 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. $dequeque((q,p))$ 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.
Section 4.2.4: DC Simulates GC

DC-B: By Claims 34 and 36.

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

DC-D: The only FIND is in $dcqueue((p,q))$, 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) $\pi$ is AfterMerge($p,q$). Let $f = fragment(p)$.

(3b) $A_4(s',\pi)$ 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 $dcqueue((q,p))$, by precondition.
3. No FIND is in $dcqueue((p,q))$, by precondition.
4. $dcstatus(q) = unfind$, by precondition.
5. No REPORT is in $dcqueue((q,p))$, 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).
Section 4.2.4: DC Simulates GC

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

Claims about \(s\):
12. \((p, q) = \text{core}(f)\), by Claim 1.
13. Every node in \texttt{subtree}(q) is in \texttt{testset}(f), by Claim 6.
14. \texttt{dequeue}((q, p)) consists solely of FIND, by Claim 9.
15. \texttt{dequeue}((p, q)) consists solely of FIND, by Claim 10 and code.
16. \texttt{dcstatus}(q) = \text{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 \texttt{dequeue}((p, q)) replaces AfterMerge\(p, q\) being enabled).

DC-F: By Claim 15 (FIND in \texttt{dequeue}((p, q)) replaces AfterMerge\(p, q\) being enabled).

DC-O: By Claim 15.

viii) \(\pi\) is \text{Absorb}(f,g).

(3c) \(A_4(s', \pi) = \pi\). Obviously \(\pi\) is enabled in \(S_4(s')\). Effects are mirrored in \(S_4(s)\) if we can show that \(\text{accmin}(f)\) and \(\text{minlink}(f)\) do not change.

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

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

Suppose \(\text{accmin}(f) = (q, r)\) in \(s'\). By GC-A, \(\text{level}(f) \leq \text{level}(\text{fragment}(r))\). So by precondition, \(\text{fragment}(r) \neq g\). Since all of \text{nodes}(g) is added to \text{testset}(f), there is no change to \(\text{nodes}(f) - \text{testset}(f)\). Thus \(\text{accmin}(f)\) is unchanged.
Section 4.2.4: DC Simulates GC

Case 2: \( p \notin \text{testset}(f) \) in \( s' \).

Claims about \( s' \):

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

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

If \( \text{accmin}(f) = (r,t) \) in \( s' \), then it is unchanged in \( s \) by Claims 9 and 13. Suppose \( \text{accmin}(f) = \text{nil} \) in \( s' \). If this is because \( \text{minlink}(f) \neq \text{nil} \) in \( s' \), then, since we just showed that \( \text{minlink}(f) \) does not change, \( \text{accmin}(f) \) is still \text{nil} in \( s \).

Suppose \( \text{accmin}(f) = \text{nil} \) not because \( \text{minlink}(f) = \text{nil} \), but because no node in \( \text{nodes}(f) - \text{testset}(f) \) has an external link. But by the assumption for this case, \( p \notin \text{testset}(f) \), yet it is in \( \text{nodes}(f) \) by Claim 4, and \( (p,q) \) 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. \( \text{rootchanged}(g) = \text{true} \), by precondition.
2. \( \text{level}(g) < \text{level}(f) \), by precondition.
3. \( \text{minlink}(g) = (q,p) \), by precondition.
4. \( p \in \text{nodes}(f) \), by precondition.
5. No \text{REPORT} message is headed toward \( \text{root}(g) \), by Claim 3.
6. No \text{FIND} is in \( \text{subtree}(g) \), by Claim 3.
Section 4.2.4: DC Simulates GC

7. \( \text{dcstatus}(r) = \text{unfind} \), for all \( r \in \text{nodes}(g) \), by Claim 3.
8. \((q, p)\) is the minimum-weight external link of \( g \), by Claim 3 and COM-A.
9. \( \text{testset}(g) = \emptyset \), 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 \( \text{core}(g) \), by Claim 10.
12. If \( \text{REPORT} \) is in \( \text{dcqueue}((r, t)) \), then \( \text{inbranch}(t) = (t, r) \), for all \( (r, t) \in \text{subtree}(g) \), by Claims 6 and 7 and DC-A(a) and (f).
13. If \( \text{REPORT} \) is in \( \text{dcqueue}((r, t)) \) and \( (r, t) = \text{core}(f) \), then \( r = \text{root}(g) \), for all \( (r, t) \in \text{subtree}(g) \), by Claim 5.
14. If \( \text{REPORT} \) is in \( \text{dcqueue}((r, t)) \) and \( (r, t) \neq \text{core}(f) \), then \( t \) is a child of \( r \), for all \( (r, t) \in \text{subtree}(g) \), by Claim 9 and DC-B(a).
15. If \( \text{REPORT} \) is in \( \text{dcqueue}((r, t)) \), then \( (r, t) \) is not on the path between \( \text{root}(g) \) and \( q \), for all \( (r, t) \in \text{subtree}(g) \), by Claims 3, 5, 8, 9 and DC-N.
16. No \( \text{REPORT} \) is headed toward \( q \), by Claims 5, 14 and 15.
17. \( \text{dcqueue}((p, q)) \) and \( \text{dcqueue}((q, p)) \) are empty, by Claim 8 and DC-A(g), DC-B(a) and DC-D(a).

Case 1: \( p \notin \text{testset}(f) \).

More claims about \( s' \):

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

DC-A: By Claim 12, vacuously true for any \( \text{REPORT} \) in old \( g \). For a \( \text{REPORT} \) that could be in some \( \text{dcqueue}((r, t)) \) with \( p \in \text{subtree}(t) \): (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 \( \text{nodes}(g) \) to \( \text{testset}(f) \).

DC-G: By Claim 19, vacuously true.

DC-H(a): By Claim 7.

DC-K: Choose any up-to-date node \( r \) in \( \text{nodes}(f) \) in \( s \). By Claims 7 and 11 and code, no node that is in \( \text{nodes}(g) \) in \( s' \) is up-to-date in \( s \). Thus \( r \) is in \( \text{nodes}(f) \) in \( s' \), and is up-to-date.
Section 4.2.4: DC Simulates GC

(a) If \( r = p \), then \( \text{findcount}(p) \) 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 \( \text{findcount}(r) \) nor the number of children of \( r \) that are not completed is changed.

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

(c) Suppose \( \text{bestlink}(r) \neq \text{nil} \) 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 \( \text{bestlinks} \) from \( r \) leads to \( l, \text{wt}(l) = \text{bestwt}(r) \), and \( \text{level}(h) \geq \text{level}(f) \), where \( h = \text{fragment}(\text{target}(l)) \), in \( s' \). By the precondition on \( \text{level}(g) \), \( h \neq g \) 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, \( \text{wt}(p, q) \) is less than the weight of any other external link of \( g \), and thus \( \text{wt}(l) \) 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 \( \text{core}(g) \) is OK.

DC-N: Let \( l \) be the minimum-weight external link of \( f \) in \( s' \). If \( l \neq (p, q) \), then \( \text{wt}(l) < \text{wt}(p, q) \), and by Claim 8, \( \text{wt}(l) < \text{wt}(l') \) 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 \( l = (p, q) \). By DC-N and Claim 18, \( p \) is up-to-date. But by DC-K(b) and (c), \( \text{bestlink}(p) = (p, q) \) and \( \text{level}(f) \leq \text{level}(g) \), which contradicts Claim 2.

Case 2: \( p \in \text{testset}(f) \).

More claims about \( s' \):

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

DC-A(e): by Claim 22, the addition of uncompleted child \( q \) to \( p \) is OK.
Section 4.2.5: \textit{NOT} Simulates \textit{COM}

DC-B: As in Case 1.

DC-D: As in Case 1.

DC-E: By Claim 23.

DC-G: By code, since all of $\text{nodes}(g)$ is added to $\text{testset}(f)$.

DC-H: By Claim 7.

DC-K: As in Case 1.

DC-M: By code, since $\text{findcount}(p)$ is incremented.

DC-N: By code, since all of $\text{nodes}(g)$ is added to $\text{testset}(f)$.

DC-O: By Claim 17 and code.

Let $P_{DC} = (P_{GC} \circ S_4) \land P_{DC}$.

\textbf{Corollary 20:} $P_{DC}'$ is true in every reachable state of DC.

\textbf{Proof:} By Lemmas 1 and 19.

4.2.5 \textit{NOT} Simulates \textit{COM}

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

Define automaton \textit{NOT} (for \textquotedblleft Notify	extquotedblright) as follows.

The state consists of a set \textit{fragments}. Each element $f$ of the set is called a \textit{fragment}, and has the following components:
Section 4.2.5: \textit{NOT} Simulates \textit{COM}

- \textit{subtree}(f), a subgraph of \(G\);
- \textit{minlink}(f), a link of \(G\) or \textit{nil}; and
- \textit{rootchanged}(f), a Boolean.

For each node \(p\), there are associated two variables:

- \textit{nlevel}(p), a nonnegative integer; and
- \textit{nfrag}(p), an edge of \(G\) or \textit{nil}.

For each link \((p, q)\), there are associated three variables:

- \textit{nqueue}_p((p, q)), a FIFO queue of messages from \(p\) to \(q\) waiting at \(p\) to be sent;
- \textit{nqueue}_pq((p, q)), a FIFO queue of messages from \(p\) to \(q\) that are in the communication channel; and
- \textit{nqueue}_q((p, q)), a FIFO queue of messages from \(p\) to \(q\) waiting at \(q\) to be processed.

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

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

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

- For link \((p, q)\), \textit{nqueue}((p, q)) is defined to be \textit{nqueue}_p((p, q)) \| \textit{nqueue}_pq((p, q)) \| \textit{nqueue}_q((p, q)).
- For fragment \(f\), \textit{level}(f) = \max\{l : \textit{nlevel}(p) = l\text{ for }p \in \textit{nodes}(f)\}, or a \textit{NOTIFY}(l, c) message is in \textit{subtree}(f) for some \(c\).
Section 4.2.5: NOT Simulates COM

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

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

Input actions:

- \( \text{Start}(p), p \in V(G) \)

  Effects:
  \( \text{awake} := \text{true} \)

Output actions:

- \( \text{InTree}(p,q), (p,q) \in L(G) \)

  Preconditions:
  \( \text{awake} = \text{true} \)
  \( (p,q) \in \text{subtree}(\text{fragment}(p)) \) or \( (p,q) = \text{minlink}(\text{fragment}(p)) \)
  \( \text{answered}(p,q) = \text{false} \)

  Effects:
  \( \text{answered}(p,q) := \text{true} \)

- \( \text{NotInTree}(p,q), (p,q) \in L(G) \)

  Preconditions:
  \( \text{fragment}(p) = \text{fragment}(q) \) and \( (p,q) \notin \text{subtree}(\text{fragment}(p)) \)
  \( \text{answered}(p,q) = \text{false} \)

  Effects:
  \( \text{answered}(p,q) := \text{true} \)

Internal actions:

- \( \text{ChannelSend}(p,q,m), (p,q) \in L(G), m \in M \)

  Preconditions:
  \( m \) at head of \( nqueue_p(\{p,q\}) \)

  Effects:
  \( \text{deque}(nqueue_p(\{p,q\})) \)
  \( \text{enqueue}(m, nqueue_{pq}(\{p,q\})) \)

- \( \text{ChannelRec}(p,q,m), (p,q) \in L(G), m \in M \)

  Preconditions:
Section 4.2.5: NOT Simulates COM

\[ m \text{ at head of } n\text{queue}_{pq}(\langle p, q \rangle) \]

Effects:
- dequeue(n\text{queue}_{pq}(\langle p, q \rangle))
- enqueue(m, n\text{queue}_q(\langle p, q \rangle))

- ReceiveNotify((q, p), l, c), \langle q, p \rangle \in L(G), l \geq 0, c \in E(G)
  
  Preconditions:
  \[ \text{NOTIFY}(l, c) \text{ at head of } n\text{queue}_p(\langle q, p \rangle) \]
  \[ \text{AfterMerge}(p, q) \text{ not enabled} \]
  
  Effects:
  - dequeue(n\text{queue}_p(\langle q, p \rangle))
  - nlevel(p) := l
  - nfrag(p) := c
  - let \( S = \{ (p, r) : (p, r) \in \text{subtree}(\text{fragment}(p)), r \neq q \} \)
  - enqueue(\text{NOTIFY}(l, c), n\text{queue}_p(k)) \text{ for all } k \in S

- ComputeMin(f), f \in \text{fragments}
  
  Preconditions:
  \[ \text{minlink}(f) = \text{nil} \]
  \[ (p, q) \text{ is the minimum-weight external link of } f \]
  \[ n\text{level}(p) = \text{level}(f) \]
  \[ \text{level}(f) \leq n\text{level}(q) \]
  
  Effects:
  - minlink(f) := l

- ChangeRoot(f), f \in \text{fragments}
  
  Preconditions:
  \[ \text{awake} = \text{true} \]
  \[ \text{rootchanged}(f) = \text{false} \]
  \[ \text{minlink}(f) \neq \text{nil} \]
  
  Effects:
  - rootchanged(f) := true

- Merge(f, g), f, g \in \text{fragments}
  
  Preconditions:
  \[ f \neq g \]
  \[ \text{rootchanged}(f) = \text{rootchanged}(g) = \text{true} \]
  \[ \text{minedge}(f) = \text{minedge}(g) \]
  
  Effects:
  - add a new element \( h \) to \text{fragments}
Section 4.2.5: *NOT* Simulates *COM*

\[\text{subtree}(h) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f)\]

\[\text{minlink}(h) := \text{nil}\]

\[\text{rootchanged}(h) := \text{false}\]

- let \((p, q) = \text{minedge}(f)\) —

\[\text{enqueue}(\text{NOTIFY}(\text{nlevel}(p) + 1, (p, q)), \text{nqueue}_p((p, q)))\]

delete \(f\) and \(g\) from \text{fragments}

- \textit{AfterMerge}(p, q), \(p, q \in V(G)\)
  
  **Preconditions:**
  
  \((p, q) = \text{core}(\text{fragment}(p))\)
  
  \(\text{NOTIFY}(\text{nlevel}(p) + 1, (p, q))\) message in \text{nqueue}((q, p))
  
  no \(\text{NOTIFY}(\text{nlevel}(p) + 1, (p, q))\) message in \text{nqueue}((p, q))
  
  \(\text{nlevel}(q) \neq \text{nlevel}(p) + 1\)
  
  **Effects:**
  
  \(\text{enqueue}(\text{NOTIFY}(\text{nlevel}(p) + 1, (p, q)), \text{nqueue}_p((p, q)))\)

- \textit{Absorb}(f, g), \(f, g \in \text{fragments}\)
  
  **Preconditions:**
  
  \(\text{rootchanged}(g) = \text{true}\)
  
  - let \((q, p) = \text{minlink}(g)\) —
  
  \(\text{level}(g) < \text{nlevel}(p)\)
  
  \(\text{fragment}(p) = f\)
  
  **Effects:**
  
  \(\text{subtree}(f) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(g)\)
  
  \(\text{enqueue}(\text{NOTIFY}(\text{nlevel}(p), \text{nfrag}(p)), \text{nqueue}_p((p, q)))\)
  
  delete \(g\) from \text{fragments}

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

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

- **NOT-B:** If \(q \in \text{subtree}(p)\), then \(\text{nlevel}(q) \leq \text{nlevel}(p)\).

- **NOT-C:** If \((p, q) = \text{core}(f)\), then \(\text{nlevel}(p) \geq \text{level}(f) - 1\).

- **NOT-D:** If \(\text{minlink}(f) = (p, q)\), then \(\text{nlevel}(p) = \text{level}(f) \leq \text{nlevel}(q)\).

- **NOT-E:** If \(\text{nfrag}(p) = \text{core}(\text{fragment}(p))\), then \(\text{nlevel}(p) = \text{level}(\text{fragment}(p))\).
Section 4.2.5: NOT Simulates COM

- NOT-F: Either $nlevel(p) = 0$ and $nfrag(p) = nil$, or else $nlevel(p) > 0$ and $nfrag(p) \in \text{subtree(fragment(p))}$. 

- NOT-G: If $nlevel(p) < \text{level(fragment(p))}$, then either a \texttt{NOTIFY(level(fragment(p)), core(fragment(p)))} message is headed toward $p$, or else \texttt{AfterMerge(q,r)} is enabled, where $p \in \text{subtree(r)}$. 

- NOT-H: If a \texttt{NOTIFY(l,c)} message is in $\text{nqueue((q,p))}$, then 
  
  (a) $nlevel(p) < l$; 
  
  (b) if $(p,q) \neq \text{core(fragment(p))}$, then $nlevel(q) \geq l$; 
  
  (c) if $c = \text{core(fragment(p))}$ then $l = \text{level(fragment(p))}$; 
  
  (d) if \texttt{NOTIFY(l',c')} is ahead of the \texttt{NOTIFY(l,c)} in $\text{nqueue((q,p))}$, then $l' < l$; 
  
  (e) $p$ is a child of $q$, or $(p,q) = \text{core(fragment(p))}$; 
  
  (f) if $(p,q) = \text{core(fragment(p))}$, then $l = \text{level(fragment(p))}$; 
  
  (g) $c \in \text{subtree(fragment(p))}$; 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 $M_5 = (S_5, A_5)$ from NOT to COM. Define the function $S_5$ from states(NOT) to states(COM) by simply ignoring the message queues, and mapping the derived variables $\text{level(f)}$ and $\text{core(f)}$ in the NOT state to the (non-derived) variables $\text{level(f)}$ and $\text{core(f)}$ in the COM state. Define the function $A_5$ as follows. Let $s$ be a state of NOT and $\pi$ an action of NOT enabled in $s$.

- If $\pi = \text{ChannelSend(k,m), ChannelRecv(k,m), ReceiveNotify(k,l,c), or AfterMerge(p,q)}$, then $A_5(s,\pi)$ is empty.

- For all other values of $\pi$, $A_5(s,\pi) = \pi$.

The following predicates are true in any state of NOT satisfying $(P_{COM} \circ S_5) \land P_{NOT}$. Recall that $P_{COM}' = (P_{S_1} \circ S_1) \land P_{COM}$. If $P_{COM}'(S_5(s))$ is true, then the COM predicates are true in $S_5(s)$, and the $S_1$ predicates are true in $S_1(S_5(s))$. Thus, these predicates follow from $P_{NOT}$, together with the HI and COM predicates.

- NOT-I: If $p = \text{minnode(f)}$, then no \text{NOTIFY} message is headed toward $p$.

- NOT-J: For all $p$, at most one \text{NOTIFY(l,c)} message is headed toward $p$, for a fixed $l$.

**Lemma 21:** NOT simulates COM via $M_5$, $P_{NOT}$, and $P_{COM}'$. 

94
Section 4.2.5: NOT Simulates COM

**Proof:** By inspection, the types of NOT, COM, $M_5$, 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 $S_5(s)$ is in $start(COM)$.

(2) Obviously, $A_5(s, \pi)|ext(COM) = \pi|ext(NOT)$.

(3) Let $(s', \pi, s)$ be a step of NOT such that $P'_{COM}$ is true of $S_5(s')$ and $P_{NOT}$ is true of $s'$. Below, we only show (3a) for those predicates that are not obviously true in $s$.

i) $\pi$ is $Start(p)$, $InTree(l)$, $NotInTree(l)$, or $ChangeRoot(f)$. $A_5(s', \pi) = \pi$. Obviously, $S_5(s')\pi S_5(s)$ is an execution fragment of COM, and $P_{NOT}$ is true in $s$.

ii) $\pi$ is $ChannelSend(l,m)$ or $ChannelRecv(l,m)$. $A_5(s', \pi)$ is empty. Obviously, $S_5(s') = S_5(s)$, and $P_{NOT}$ is true in $s$.

iii) $\pi$ is $ReceiveNotify((q,p),l,c)$. Let $f = fragment(p)$.

(3b) $A_5(s', \pi)$ is empty. To show that $S_5(s) = S_5(s')$, 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 $nqueue((q,p))$ 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 $l = nlevel(p) \neq 0$. 95
Section 4.2.5: NOT Simulates COM

NOT-G: The \texttt{notify}(l, c) message removed from \texttt{nqueue}((q, p)) is replaced by the \texttt{notify}(l, c) messages added to \texttt{nqueue}((p, r)), for all \((p, r) \in S\).

NOT-H: Suppose \texttt{notify}(l, c) is added to \texttt{nqueue}(p, r) in \(s\). (i.e., \((p, r) \in S\).)

\textit{Claims about }\(s'\):

1. \texttt{notify}(l, c) is at head of \texttt{nqueue}((q, p)), by precondition.
2. \(p \in \text{subtree}(q)\) or \((p, q) = \text{core}(f)\), by Claim 1 and NOT-H(e).
3. \(r \in \text{subtree}(p)\), by Claim 2 and definition of \(S\).
4. \(\text{nlevel}(r) \leq \text{nlevel}(p)\), by Claim 3 and NOT-B.
5. \(\text{nlevel}(p) < l\), by Claim 1 and NOT-H(a).
6. If \texttt{notify}(l', c') is in \texttt{nqueue}((p, r)), then \(l' < l\), by Claims 3 and 5 and NOT-H(b).
7. \(\text{nlevel}(r) < l\), 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 \texttt{notify}(l, c) in \texttt{nqueue}((q, p)) in \(s'\).

iv) \(\pi\) is ComputeMin(f).

(3c) \(\mathcal{A}_5(s', \pi) = \pi\). Obviously \(\pi\) is enabled in \(\mathcal{S}_5(s')\), since by definition \(\text{nlevel}(q) \leq \text{level}((\text{fragment}(q))\). 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) \(\pi\) is Merge(f,g).

(3c) \(\mathcal{A}_5(s', \pi) = \pi\). Obviously \(\pi\) is enabled in \(\mathcal{S}_5(s')\). To show that its effects are mirrored in \(\mathcal{S}_5(s)\), we show that \(\text{level}(h)\) and \(\text{core}(h)\) are correct. Let \(\text{minlink}(f) = (p, q)\) and \(l = \text{level}(f)\) in \(s'\).

\textit{Claims about }\(s'\):

1. \(\text{minedge}(f) = \text{minedge}(g)\), by precondition.
2. \(\text{level}(g) = l\), by Claim 1 and COM-A.
3. \(\text{rootchanged}(f) = \text{true}\), by precondition.
4. \(\text{minlink}(f) \neq \text{nil}\), by Claim 3 and COM-B.
5. \(\text{nlevel}(p) = l\), by Claim 4 and NOT-D.
6. \(\text{nlevel}(r) \leq l\) for all \(r \in \text{nodes}(f)\), by definition of \(\text{level}(f)\).
7. If \texttt{notify}(m, c) is in \texttt{subtree}(f), then \(m \leq l\), by definition of \(\text{level}(f)\).
8. \(\text{rootchanged}(g) = \text{true}\), by precondition.
Section 4.2.5: NOT Simulates COM

9. \( \text{minlink}(g) \neq \text{nil} \), by Claim 8 and COM-B.
10. \( \text{nlevel}(q) = l \), by Claims 2 and 9 and NOT-D.
11. \( \text{nlevel}(r) \leq l \) for all \( r \in \text{nodes}(g) \), by definition of \( \text{level}(g) \).
12. If \( \text{NOTIFY}(m, c) \) is in \( \text{subtree}(g) \), then \( m \leq l \), by definition of \( \text{level}(g) \).
13. \( (p, q) \) is an external link of \( f \), by COM-A.
14. \( \text{nqueue}(\langle p, q \rangle) \) and \( \text{nqueue}(\langle q, p \rangle) \) are empty, by Claim 13 and NOT-H(e).

Claims about \( s \):
15. \( \text{nlevel}(r) < l + 1 \), for all \( r \in \text{nodes}(h) \), by Claims 6 and 11 and code.
16. The only \( \text{NOTIFY} \) message in \( \text{subtree}(h) \) with level greater than \( l \) is the \( \text{NOTIFY}(l + 1, (p, q)) \) message added to \( \text{nqueue}(\langle p, q \rangle) \), by Claims 7, 12 and 14 and code.
17. \( \text{level}(h) = l + 1 \), by Claims 15 and 16.
18. \( \text{core}(h) = (p, q) \), 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, \( \text{nlevel}(r) = l \) for all \( r \) on the path from \( \text{core}(f) \) to \( p \), and all \( r \) on the path from \( \text{core}(g) \) 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 \( \text{minlink}(h) = \text{nil} \) by code.

NOT-E: By NOT-F and Claim 13, \( \text{nfrag}(r) \neq (p, q) \) for all \( r \) in \( \text{nodes}(f) \) or \( \text{nodes}(g) \). So the predicate is vacuously true.

NOT-F: No relevant change.

NOT-G: If \( r \) is in \( \text{nodes}(g) \) in \( s' \), the predicate is true in \( s \) because of Claims 17 and 18 and the \( \text{NOTIFY}(l + 1, (p, q)) \) added to \( \text{nqueue}(\langle p, q \rangle) \) in \( s \). If \( r \) is in \( \text{nodes}(f) \) in \( s' \), then \( \text{AfterMerge}(q, p) \) is enabled in \( s \), by code and Claims 5, 10, 14 and 18.

NOT-H for the \( \text{NOTIFY}(l + 1, (p, q)) \) added to \( \text{nqueue}(\langle p, q \rangle) \): (a) \( \text{nlevel}(q) < l + 1 \), 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 \( l + 1 > 0 \).
Section 4.2.5: NOT Simulates COM

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 \(p \in \text{subtree}(t)\). By NOT-I, the message is not in \(nqueue((r, t))\). As argued in the proof of NOT-I. \(\text{nlevel}(t) = l\). If the message is in \(nqueue((t, r))\), 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) \(\pi\) is AfterMerge\((p, q)\). Let \(f = \text{fragment}(p)\).

(3b) \(A_5(s')\) is empty. Obviously \(S_5(s') = S_5(s)\).

(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 \(nqueue((p, q))\) 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 \(nqueue((p, q))\).

1. \((p, q) = \text{core}(f)\), by precondition.
2. notify\((l, c)\) is in \(nqueue((q, p))\), by precondition.
3. No notify\((l, c)\) is in \(nqueue((p, q))\), by precondition.
4. \(nlevel(q) \neq l\), by precondition.
5. \(l = \text{level}(f)\), 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 \(nqueue((p, q))\), then \(l' = l\), by Claims 1 and 5 and NOT-H\((d)\).
8. If notify\((l', c')\) is in \(nqueue((p, q))\), then \(c' = c\), by Claim 7 and NOT-A.
9. No notify is in \(nqueue((p, q))\), by Claims 3, 7 and 8.
10. \(nlevel(p) \geq 0\), by NOT-F.
Section 4.2.5: \textit{NOT} Simulates \textit{COM}

(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.

\textbf{vii)} $\pi$ is Absorb($f$, $g$).

(3c) $A(s', \pi) = \pi$.

\textit{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 $r \in \text{nodes}(g)$, by definition of level.
6. If \texttt{NOTIFY($l$, $c$)} is in \texttt{subtree($g$)}, then $l \leq$ level($g$), by definition of level.
7. $(q, p)$ is an external link of $g$, by COM-A.
8. nqueue($(p, q)$) and nqueue($(q, p)$) are empty, by Claim 7 and NOT-H(e).

By Claim 4, $\pi$ is enabled in $S_5(s')$. The effects of $\pi$ are mirrored in $S_5(s)$ if core($f$) and level($f$) are unchanged; by code and Claims 6, 7 and 8, they are unchanged.

(3a) Let $l = \text{nlevel}(p)$ and $c = \text{nfrag}(p)$ in $s'$.

\textit{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 $r \in \text{nodes}(g)$, 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.

\textbf{NOT-A}: by code and Claims 6, 7 and 8.

\textbf{NOT-B}: Same argument as for \textit{Merge($f$, $g$)}.

\textbf{NOT-D}: No relevant changes.

\textbf{NOT-E}: By Claim 12, vacuously true for nodes formerly in \text{nodes}(g).

\textbf{NOT-F}: No relevant changes.
Section 4.2.5: NOT Simulates COM

NOT-G: Suppose \( n_{\text{level}}(p) = \text{level}(f) \) in \( s' \). By code, in \( s \) there is a \( \text{NOTIFY}(\text{level}(f), \cdot) \) message headed toward every node formerly in \( \text{nodes}(g) \).

Suppose \( n_{\text{level}}(p) \neq \text{level}(f) \) in \( s' \). By NOT-G, either a \( \text{NOTIFY}(\text{level}(f), \cdot) \) message is headed toward \( p \) in \( s' \), and thus is headed toward all nodes formerly in \( \text{nodes}(g) \) in \( s \), or AfterMerge\((r, t)\) is enabled in \( s' \) with \( p \in \text{subtree}(t) \), and thus in \( s \), AfterMerge\((r, t)\) is still enabled and every node formerly in \( \text{nodes}(g) \) is in \( \text{subtree}(t) \).

NOT-H for the \( \text{NOTIFY}(l, \cdot) \) added to \( \text{nqueue}(p, q) \): (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 \( \text{NOTIFY}(l', \cdot) \) in \( \text{subtree}(g) \) in \( s' \): (a), (d), (g) and (h): no relevant change. (b) and (e) same argument as for \( \text{Merge}(f, g) \). (c) vacuously true, by Claim 11. (f) vacuously true, by code.

Let \( P'_{\text{NOT}} = (P_{\text{COM}} \circ S_5) \land P_{\text{NOT}} \).

Corollary 22: \( P'_{\text{NOT}} \) is true in every reachable state of NOT.

Proof: By Lemmas 1 and 21.
Section 4.2.6: CON Simulates COM

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-passing. 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 (p, q), there are associated three variables:

- cqueue_{p}((p, q)), a FIFO queue of messages from p to q waiting at p to be sent;
- cqueue_{pq}((p, q)), a FIFO queue of messages from p to q that are in the communication channel; and
- cqueue_{q}((p, q)), a FIFO queue of messages from p to q waiting at q to be processed.
Section 4.2.6: CON Simulates COM

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

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

The derived variable \( \text{cqueue}(\langle p, q \rangle) \) is \( \text{cqueue}_q(\langle p, q \rangle) \cup \text{cqueue}_p(\langle p, q \rangle) \cup \text{cqueue}_{p,q}(\langle p, q \rangle) \). For each fragment \( f \), we define the derived Boolean variable \( \text{rootchanged}(f) \) to be true if and only if a \text{connect} message is in \( \text{cqueue}(\langle p, q \rangle) \), for some external link \( \langle p, q \rangle \) of \( f \). Derived variable \( \text{tominlink}(p) \) is defined to be the link \( \langle p, q \rangle \) such that \( \langle p, q \rangle \) is on the path in \( \text{subtree}(\text{fragment}(p)) \) from \( p \) to \( \text{minnode}(\text{fragment}(p)) \).

Message \( m \) is defined to be in \( \text{subtree}(f) \) if \( m \) is in \( \text{cqueue}(\langle q, p \rangle) \) and \( p \in \text{nodes}(f) \).

Input actions:

- **Start(\( p \)), \( p \in V(G) \)**
  
  Effects:
  
  \[ \text{awake} := \text{true} \]

Output actions:

- **InTree(\( \langle p, q \rangle \)), \( (p,q) \in L(G) \)**

  Preconditions:
  
  \[ \text{awake} = \text{true} \]
  
  \[ (p,q) \in \text{subtree}(\text{fragment}(p)) \text{ or } (p,q) = \text{minlink}(\text{fragment}(p)) \]
  
  \[ \text{answered}(\langle p, q \rangle) = \text{false} \]

  Effects:
  
  \[ \text{answered}(\langle p, q \rangle) := \text{true} \]

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

  Preconditions:
  
  \[ \text{fragment}(p) = \text{fragment}(q) \text{ and } (p,q) \notin \text{subtree}(\text{fragment}(p)) \]
  
  \[ \text{answered}(\langle p, q \rangle) = \text{false} \]

  Effects:
  
  \[ \text{answered}(\langle p, q \rangle) := \text{true} \]
Section 4.2.6: CON Simulates COM

Internal actions:

- **ChannelSend**\((p, q), m)\), \((p, q) \in L(G), m \in M\)
  Preconditions:
  \(m\) at head of \(cqueue_p((p, q))\)
  
  Effects:
  
  dequeue\((cqueue_p((p, q)))\)
  enqueue\((m, cqueue_{pq}((p, q)))\)

- **ChannelRecv**\((p, q), m)\), \((p, q) \in L(G), m \in M\)
  Preconditions:
  \(m\) at head of \(cqueue_{pq}((p, q))\)
  
  Effects:
  
  dequeue\((cqueue_{pq}((p, q)))\)
  enqueue\((m, cqueue_q((p, q)))\)

- **ComputeMin**\((f)\), \(f \in \text{fragments}\)
  Preconditions:
  \(\text{minlink}(f) = \text{nil}\)
  \(l\) is the minimum-weight external link of \(\text{subtree}(f)\)
  \(\text{level}(f) \leq \text{level}\(\text{fragment}(\text{target}(l))\)\)
  no CONNECT message is in \(cqueue(k)\), for any internal link \(k\) of \(f\)
  
  Effects:
  
  \(\text{minlink}(f) := l\)
  
  — let \(p = \text{root}(f)\) —
  if \(p \neq \text{minnode}(f)\) then enqueue\(\text{ Changeroot, cqueue}_p(\text{to minlink}(p))\)
  else enqueue\(\text{ Connect(level}(f)), cqueue_p(\text{minlink}(f))\)

- **ReceiveChangeRoot**\((q, p))\), \((q, p) \in L(G)\)
  Preconditions:
  \(\text{Changeroot}\) at head of \(cqueue_p((q, p))\)
  
  Effects:
  
  dequeue\((cqueue_p((q, p)))\)
  
  — let \(f = \text{fragment}(p)\) —
  if \(p \neq \text{minnode}(f)\) then enqueue\(\text{ Changeroot, cqueue}_p(\text{to minlink}(p))\)
  else enqueue\(\text{ Connect(level}(f)), cqueue_p(\text{minlink}(f))\)

- **ChangeRoot**\((f)\), \(f \in \text{fragments}\)
  Preconditions:
  \(\text{awake} = \text{true}\)

103
Section 4.2.6: \textit{CON} Simulates \textit{COM}

\(\text{rootchanged}(f) = \text{false}\)
\(\text{subtree}(f) = \{p\}\)

Effects:
\(\text{enqueue}(\text{CONNECT}(0), \text{cqueue}_{p}(\text{minlink}(f)))\)

- \textit{Merge}(f, g), \(f, g \in \text{fragments}\)

Preconditions:
\(\text{CONNECT}(l)\) in \(\text{cqueue}((p, q))\), \((p, q)\) external link of \(f\)
\(\text{CONNECT}(l)\) at head of \(\text{cqueue}_{p}((q, p))\), \((q, p)\) external link of \(g\)

Effects:
\(\text{dequeue}(\text{cqueue}_{p}((q, p)))\)
add a new element \(h\) to \(\text{fragments}\)
\(\text{subtree}(h) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(f)\)
\(\text{core}(h) := \text{minedge}(f)\)
\(\text{level}(h) := \text{level}(f) + 1\)
\(\text{minlink}(h) := \text{nil}\)
delete \(f\) and \(g\) from \(\text{fragments}\)

- \textit{AfterMerge}(p, q), \(p, q \in V(G)\)

Preconditions:
\(\text{fragment}(p) = \text{fragment}(q)\)
\(\text{CONNECT}(l)\) at head of \(\text{cqueue}_{p}((q, p))\)

Effects:
\(\text{dequeue}(\text{cqueue}_{p}((q, p)))\)

- \textit{Absorb}(f, g), \(f, g \in \text{fragments}\)

Preconditions:
\(-\quad \text{let} \ p = \text{target}(\text{minlink}(g))\)
\(\text{CONNECT}(l)\) at head of \(\text{cqueue}_{p}(\text{minlink}(g))\)
\(l < \text{level}(f)\)
\(f = \text{fragment}(p)\)

Effects:
\(\text{dequeue}(\text{cqueue}_{p}(\text{minlink}(g)))\)
\(\text{subtree}(f) := \text{subtree}(f) \cup \text{subtree}(g) \cup \text{minedge}(g)\)
delete \(g\) from \(\text{fragments}\)

Define the following predicates on states of \textit{CON}. (All free variables are universally quantified.)

- \textit{CON-A}: If \(\text{awake} = \text{false}\), then \(\text{cqueue}((q, p))\) is empty.
Section 4.2.6: CON Simulates COM

- **CON-B:** If \( \text{rootchanged}(f) = \text{false} \) and \( \text{minlink}(f) \neq \text{nil} \), then either \( \text{subtree}(f) = \{p\} \), or else \( \text{minnode}(f) \neq \text{root}(f) \) and there is exactly one CHANGEROOT message in \( \text{subtree}(f) \).

- **CON-C:** If a CHANGEROOT message is in \( \text{cqueue}(\langle q, p \rangle) \), then \( \text{minlink}(f) \neq \text{nil} \), \( \text{rootchanged}(f) = \text{false} \), \( p \) is a child of \( q \), and \( \text{minnode}(f) \in \text{subtree}(p) \), where \( f = \text{fragment}(p) \).

- **CON-D:** If a CONNECT(l) message is in \( \text{cqueue}(k) \), where \( k \) is an external link of \( f \), then \( k = \text{minlink}(f), l = \text{level}(f) \), and only one CONNECT message is in \( \text{cqueue}(k) \).

- **CON-E:** If a CONNECT(l) message is in \( \text{cqueue}(\langle p, q \rangle) \), where \( \langle p, q \rangle \) is an internal link of \( f \), then \( \langle p, q \rangle = \text{core}(f), l < \text{level}(f) \), and only one CONNECT message is in \( \text{cqueue}(\langle p, q \rangle) \).

- **CON-F:** If \( \text{minlink}(f) \neq \text{nil} \), then no CONNECT message is in \( \text{cqueue}(k) \), for any internal link \( k \) of \( f \).

Let \( P_{\text{CON}} \) be the conjunction of CON-A through CON-F.

In order to show that CON simulates COM, we define an abstraction mapping \( M_6 = (S_6, A_6) \) from CON to COM.

Define the function \( S_6 \) from \( \text{states}(\text{CON}) \) to \( \text{states}(\text{COM}) \) by simply ignoring the message queues, and mapping the derived variables \( \text{rootchanged}(f) \) in the CON state to the (non-derived) variables \( \text{rootchanged}(f) \) in the COM state.

Define the function \( A_6 \) as follows. Let \( s \) be a state of CON and \( \pi \) an action of CON enabled in \( s \). If the minimum-weight external link of \( f \) is adjacent to \( \text{core}(f) \), 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 \( \text{minlink}(f) \) receives a CHANGEROOT message.

- If \( \pi = \text{ChannelSend}(\langle p, q \rangle, m), \text{ChannelRecv}(\langle p, q \rangle, m), \) or \( \text{AfterMerge}(p, q) \), then \( A_6(s, \pi) \) is empty.

- If \( \pi = \text{ComputeMin}(f) \) and \( \text{mw-root}(f) = \text{mw-minnode}(f) \) in \( s \), then \( A_6(s, \pi) = \text{ComputeMin}(f) \uplus \text{ChangeRoot}(f) \), where \( t \) is identical to \( S_6(s) \) except that \( \text{minlink}(t) \) equals the minimum-weight external link of \( f \) in \( t \).
Section 4.2.6: CON Simulates COM

- If $\pi = \text{ComputeMin}(f)$ and $mw\text{-}root(f) \neq mw\text{-}minnode(f)$ in $s$, then $A_6(s, \pi) = \text{ComputeMin}(f)$.

- If $\pi = \text{ReceiveChangeRoot}((q,p))$ and $p = \text{minnode(fragment(p))}$ in $s$, then $A_6(s, \pi) = \text{ChangeRoot(fragment(p))}$.

- If $\pi = \text{ReceiveChangeRoot}((q,p); p \neq \text{minnode(fragment(p))})$ in $s$, then $A_6(s, \pi)$ is empty.

- For all other values of $\pi$, $A_6(s, \pi) = \pi$.

Recall that $P'_\text{COM} = (P_{HI} \circ S_1) \wedge P_{COM}$. If $P'_\text{COM}(S_6(s))$ is true, then the COM predicates are true in $S_6(s)$, and the HI predicates are true in $S_1(S_6(s))$.

**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, $P'_\text{COM}$ is a predicate true in every reachable state of COM.

(1) Let $s$ be in $\text{start}(CON)$. Obviously $P_{CON}$ is true in $s$ and $S_6(s)$ is in $\text{start}(COM)$.

(2) Obviously, $A_6(s, \pi)|\text{ext}(COM) = \pi|\text{ext}(CON)$.

(3) Let $(s', \pi, s)$ be a step of CON such that $P'_\text{COM}$ is true of $S_6(s')$ and $P_{CON}$ is true of $s'$. Below we show (3a) only for those predicates that are not obviously true in $s$.

i) $\pi$ is Start(p), InTree(l) or NotInTree(l). $A_6(s', \pi) = \pi$. Obviously, $S_6(s')\pi S_6(s)$ is an execution fragment of COM, and $P_{CON}$ is true in $s$.

ii) $\pi$ is ChannelSend((q,p),m) or ChannelRecv((q,p),m). $A_6(s', \pi)$ is empty. Obviously, $S_6(s') = S_6(s)$, and $P_{CON}$ is true in $s$.

iii) $\pi$ is ComputeMin(f).

*Case 1: $mw\text{-}root(f) \neq mw\text{-}minnode(f)$ in $s'$.*

(3b) $A_6(s', \pi) = \pi$. Obviously $S_6(s')\pi S_6(s)$ is an execution fragment of COM.

(3a) Claims about $s'$:

1. $\text{minlink}(f) = \text{nil}$, by precondition.
2. $l$ is the minimum-weight external link of $f$, by precondition.
3. $\text{level}(f) \leq \text{level}(|\text{target}(l)|)$, by precondition.
4. No \textsc{connect} message is in $\text{cqueue}(k)$, for any internal link $k$ of $f$, by precondition.
5. $p = \text{mw-root}(f)$, by assumption.
6. $p \neq \text{mw-minnode}(f)$, by assumption.
7. $\text{awake} = \text{true}$, by Claim 1 and \textsc{com-c}.
8. No \textsc{changeroot} message is in $\text{subtree}(f)$, by Claim 1 and \textsc{con-c}.
9. $\text{mw-minnode}(f) \in \text{subtree}(p)$, by Claim 5.
10. $\text{rootchanged}(f) = \text{false}$, by Claim 1 and \textsc{com-b}.

\textbf{Claims about $s$:}

11. $\text{minlink}(f) = l$, the minimum-weight external link of $f$, by Claim 2 and code.
12. $\text{level}(f) \leq \text{level}(|\text{target}(l)|)$, by Claim 3.
13. $p = \text{root}(f)$, by Claims 5 and 11.
14. $p \neq \text{minnode}(f)$, by Claims 6 and 11.
15. $\text{awake} = \text{true}$, by Claim 7.
16. Exactly one \textsc{changeroot} message is in $\text{subtree}(f)$, by Claim 8 and code.
17. $\text{minnode}(f) \in \text{subtree}(p)$, by Claims 9 and 11.
18. $\text{rootchanged}(f) = \text{false}$, by Claim 10.
19. No \textsc{connect} message is in $\text{cqueue}(k)$, for any internal link $k$ of $f$, by Claim 4.

\textsc{con-a} is true by Claim 15. \textsc{con-b} is true by Claims 13, 14, and 16. \textsc{con-c} is true by definition of $\text{tominlink}$, Claims 17, 18 and 11. \textsc{con-d} and \textsc{con-e} are true since no relevant changes are made. \textsc{con-f} is true by Claim 19.

\textit{Case 2: $\text{mw-root}(f) = \text{mw-minnode}(f)$ in $s'$.}

(3b) $A_6(s', \pi) = \pi \text{ t ChangeRoot}(f)$, where $t$ is identical to $S_6(s')$ except that $\text{minlink}(f)$ equals the minimum-weight external link of $f$ in $t$.

\textbf{Claims about $s'$:}

1. $\text{minlink}(f) = \text{nil}$, by precondition.
2. $l$ is the minimum-weight external link of $f$, by precondition.
3. $\text{level}(f) \leq \text{level}(|\text{target}(l)|)$, by precondition.
4. $\text{awake} = \text{true}$, by Claim 1 and \textsc{com-c}.
5. $\text{rootchanged}(f) = \text{false}$, by Claim 1 and \textsc{com-b}.  
107
Section 4.2.6: CON Simulates COM

Claims about \( t \):

6. \( \text{minlink}(f) \) is the minimum-weight external link of \( f \), by definition of \( t \).
7. \( \text{awake} = \text{true} \), by Claim 4.
8. \( \text{rootchanged}(f) = \text{false} \), by Claim 5.

Claims about \( s \):

9. \( \text{minlink}(f) \) is the minimum-weight external link of \( f \), by code.
10. A \( \text{CONNECT} \) message is in \( \text{cqueue}(\text{minlink}(f)) \), by code.
11. \( \text{rootchanged}(f) = \text{false} \), by Claims 9 and 10.

By Claims 1, 2 and 3, \( \pi \) is enabled in \( S_6(s') \). By Claim 6 (and definition of \( t \)), the effects of \( \pi \) are mirrored in \( t \). By Claims 6, 7, and 8, \( \text{ChangeRoot}(f) \) is enabled in \( t \). By Claim 11 (and definition of \( t \)), the effects of \( \text{ChangeRoot}(f) \) are mirrored in \( S_6(s) \). Therefore, \( S_6(s') \pi \text{ ChangeRoot}(f) S_6(s) \) is an execution fragment of \( \text{COM} \).

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

12. No \( \text{CHANGEROOT} \) message is in \( \text{subtree}(f) \), by Claim 1 and CON-C.
13. No \( \text{CONNECT} \) message is in any \( \text{cqueue}(k) \), where \( k \) is an external link of \( f \), by Claim 1 and CON-D.
14. No \( \text{CONNECT} \) message is in any \( \text{cqueue}(k) \), where \( k \) is an internal link of \( f \), by precondition.

More claims about \( s \):

15. \( \text{awake} = \text{true} \), by Claim 4.
16. No \( \text{CHANGEROOT} \) message is in \( \text{subtree}(f) \), 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) \( \pi \) is \( \text{RecvChangeRoot}((q,p)) \). Let \( f = \text{fragment}(p) \).

Case 1: \( p \neq \text{minnode}(f) \) in \( s' \).

(3c) \( A_6(s', \pi) \) is empty. Below we show that \( \text{rootchanged}(f) \) is the same in \( s' \) and \( s \), which implies that \( S_6(s) = S_6(s') \).

Claims about \( s' \):

1. A \( \text{CHANGEROOT} \) message is in \( \text{cqueue}((q,p)) \), by precondition.
Section 4.2.6: CON Simulates COM

2. \((p, q) \in \text{subtree}(f)\), by Claim 1 and CON-C.
3. \(\text{rootchanged}(f) = \text{false}\), by Claims 1 and 2 and CON-A.

Claims about \(s\):
4. \(\text{rootchanged}(f) = \text{false}\), by Claim 1 and code.

Claims 2 and 4 give the result.

(3a) Let \((p, r) = \text{tominlink}(p)\).

More claims about \(s'\):
5. \(\text{awake} = \text{true}\), by Claim 1 and CON-A.
6. \(\text{minlink}(f) \neq \text{nil}\), by Claims 1 and 2 and CON-C.
7. \(\text{minnode}(f) \in \text{subtree}(p)\), by Claims 1 and 2 and CON-C.
8. There is exactly one \text{CHANGEROOT} message in \(\text{subtree}(f)\), by Claims 2, 3 and 6 and CON-B.
9. \(r\) is a child of \(p\) and \(\text{minnode}(f) \in \text{subtree}(r)\), by definition of \text{tominlink}(p).

More claims about \(s\):
10. \(\text{awake} = \text{true}\), by Claim 5.
11. There is exactly one \text{CHANGEROOT} message in \(\text{subtree}(f)\), by Claim 8 and code.
12. \(r\) is a child of \(p\), by Claim 9.
13. \(\text{minlink}(f) \neq \text{nil}\), by Claim 6.
14. \((p, r) \neq \text{core}(f)\), by Claim 9.
15. \(\text{minnode}(f) \in \text{subtree}(r)\), 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: \(p = \text{minnode}(f)\) in \(s'\).

(3b) \(A_6(s', \pi) = \text{ChangeRoot}(f)\).

Claims about \(s'\):
1. A \text{CHANGEROOT} message is in \(\text{cqueue}((q, p))\), by precondition.
Section 4.2.6: CON Simulates COM

2. \( p = \text{minnode}(f) \), by assumption.
3. \( \text{awake} = \text{true} \), by Claim 1 and CON-A.
4. \( \text{minlink}(f) \neq \text{nil} \), by Claim 1 and CON-C.
5. \( \text{rootchanged}(f) = \text{false} \), by Claim 1 and CON-C.
6. \( \text{minlink}(f) \) is an external link of \( f \), by Claim 4 and COM-A.

By Claims 3, 4 and 5, \( \text{ChangeRoot}(f) \) is enabled in \( S_6(s') \).

Claims about \( s' \):

7. A \text{CONNECT} message is in \( cqueue(\text{minlink}(f)) \), by code.
8. \( \text{minlink}(f) \) is an external link of \( f \), by Claim 6.
9. \( \text{rootchanged}(f) = \text{true} \), by Claims 7 and 8.

By Claim 9, the effects of \( \text{ChangeRoot}(f) \) are mirrored in \( S_6(s) \).

So \( S_6(s') \text{ ChangeRoot}(f) S_6(s) \) is an execution fragment of \( \text{COM} \).

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

10. \( p \) is a child of \( q \), by Claim 1 and CON-C.
11. Exactly one \text{CHANGEROOT} message is in \( \text{subtree}(f) \), by Claims 5, 4, 10 and CON-B.
12. No \text{CONNECT} message is in any \( cqueue(k) \), where \( k \) is an external link of \( f \), by Claim 5.
13. No \text{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. \( \text{awake} = \text{true} \), by Claim 3.
15. No \text{CHANGEROOT} message is in \( \text{subtree}(f) \); by Claims 1, 10 and 11 and code.
16. No \text{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) \( \pi \) is \text{ChangeRoot}(f).
(3b) $A_o(s', \pi) = \pi$.

Claims about $s'$:
1. \textit{awake} = true, by precondition.
2. \textit{rootchanged}(f) = false, by precondition.
3. \textit{subtree}(f) = \{p\}, by precondition.
4. \textit{minlink}(f) \neq \text{nil}, by Claim 3 and COM-E.
5. \textit{minlink}(f) is an external link of f, by Claim 4 and COM-A.

Claims 1, 2 and 4 imply that $\pi$ is enabled in $S_6(s')$.

Claims about $s$:
6. \textit{minlink}(f) is an external link of f, by Claim 5.
7. A \textit{CONNECT} message is in \textit{cqueue}(\textit{minlink}(f)), by code.
8. \textit{rootchanged}(f) = true, by Claims 6 and 7.

Claim 8 implies that the effects of $\pi$ are mirrored in $S_6(s)$.

So $S_6(s')\pi S_6(s)$ is an execution fragment of COM.

---

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

9. No \textit{CHANGEROOT} message is in \textit{cqueue}((q,p)), for any q, by Claim 3 and CON-C.
10. No \textit{CONNECT} message is in any \textit{cqueue}(k), where k is an external link of f, by Claim 2.
11. No \textit{CONNECT} message is in any \textit{cqueue}(k), where k is an internal link of f, by Claim 3.

More claims about $s$:
12. \textit{awake} = true, by Claim 1 and code.
13. No \textit{CHANGEROOT} message is in \textit{cqueue}((q,p)), for any q, by Claim 9.
14. No \textit{CONNECT} message is in any \textit{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) $\pi$ is \textit{Merge}(f,g).
Section 4.2.6: \( CON \) Simulates \( COM \)

\[(3b) \ A_\delta(s', \pi) = \pi.\]

Claims about \( s' \):

1. A \( \text{CONNECT}(l) \) message is in \( \text{queue}((p,q)) \), by precondition.
2. \( (p,q) \) is an external link of \( f \), by precondition.
3. A \( \text{CONNECT}(l) \) message is in \( \text{queue}((q,p)) \), by precondition.
4. \( (q,p) \) is an external link of \( g \), by precondition.
5. \( f \neq g \), by Claims 2 and 4.
6. \( \text{rootchanged}(f) = \text{true} \), by Claims 1 and 2.
7. \( \text{rootchanged}(g) = \text{true} \), by Claims 3 and 4.
8. \( (p,q) = \text{minlink}(f) \), by Claims 1 and 2 and \( \text{CON-D} \).
9. \( (q,p) = \text{minlink}(g) \), by Claims 3 and 4 and \( \text{CON-D} \).
10. \( \text{minedge}(f) = \text{minedge}(g) \), by Claims 8 and 9.
11. If \( k \neq \text{minlink}(f) \) is an external link of \( f \), then no \( \text{CONNECT} \) message is in \( \text{queue}(k) \), by \( \text{CON-D} \).
12. If \( k \neq \text{minlink}(g) \) is an external link of \( g \), then no \( \text{CONNECT} \) message is in \( \text{queue}(k) \), by \( \text{CON-D} \).

By Claims 5, 6, 7 and 10, \( \pi \) is enabled in \( S_\delta(s') \). By Claims 11 and 12 and definition of \( h \), \( \text{rootchanged}(h) = \text{false} \) in \( s \), so the effects of \( \pi \) are mirrored in \( S_\delta(s) \). Thus, \( S_\delta(s')\pi S_\delta(s) \) is an execution fragment of \( \text{COM} \).

---

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

13. \( \text{awake} = \text{true} \), by Claim 1 and \( \text{COM-A} \).
14. No \( \text{CHANGEROOT} \) message is in \( \text{subtree}(f) \), by Claim 6 and \( \text{CON-C} \).
15. No \( \text{CHANGEROOT} \) message is in \( \text{subtree}(g) \), by Claim 7 and \( \text{CON-C} \).
16. No \( \text{CONNECT} \) message is in \( \text{queue}(k) \), for any internal link \( k \) of \( f \), by Claim 8 and \( \text{CON-F} \).
17. No \( \text{CONNECT} \) message is in \( \text{queue}(k) \), for any internal link \( k \) of \( g \), by Claim 9 and \( \text{CON-F} \).
18. Exactly one \( \text{CONNECT} \) message is in \( \text{queue}((p,q)) \), by Claims 1 and 2 and \( \text{CON-D} \).
19. Exactly one \( \text{CONNECT} \) message is in \( \text{queue}((q,p)) \), by Claims 3 and 4 and \( \text{CON-D} \).
20. \( l = \text{level}(f) \), by Claims 1 and 2 and \( \text{CON-D} \).

Claims about \( s \):
Section 4.2.6: $CON$ Simulates $COM$

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 $cqueue((p,q))$ 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 $cqueue((q,p))$, 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) $\pi$ is $AfterMerge(p,q)$. $A_6(s',\pi)$ is empty. Obviously, $S_6(s) = S_6(s')$, and $P_{CON}$ is true in $s$.

viii) $\pi$ is $Absorb(f,g)$.

(3b) $A_6(s',\pi) = \pi$.

Claims about $s'$:

1. $(q,p) = minlink(g)$, 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 $cqueue((p,q))$, then $(p,q) = minlink(f)$, by Claims 4 and 5 and $CON$-D.
10. If a $CONNECT$ message is in $cqueue((p,q))$, then $level(f) \leq level(g)$, by Claim 9 and $COM$-A.
11. No $CONNECT$ message is in $cqueue((p,q))$, 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.

113
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

By Claims 6, 8, 4 and 1, \( \pi \) is enabled in \( S_6(s') \). By Claims 11 and 12, \( \text{rootchanged}(f) \) remains unchanged, and the effects of \( \pi \) are mirrored in \( S_6(s) \). Thus, \( S_6(s') \pi S_6(s) \) is an execution fragment of \( \text{COM} \).

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

13. \( \text{awake} = \text{true} \), by Claim 2 and \( \text{CON-A} \).
14. \( l \geq 0 \), by \( \text{COM-F} \).
15. \( \text{level}(f) > 0 \), by Claims 7, 8 and 14.
16. \( |\text{nodes}(f)| > 1 \), by Claim 15 and \( \text{COM-F} \).
17. No \( \text{CHANGEROOT} \) message is in \( \text{subtree}(g) \), by Claim 6 and \( \text{CON-C} \).
18. No \( \text{CONNECT} \) message is in \( \text{cqueue}(k) \), where \( k \) is an internal link of \( g \), by Claim 1 and \( \text{CON-F} \).

Claim about \( s \):

19. \( \text{awake} = \text{true} \), by Claim 12 and code.

\( \text{CON-A} \) is true by Claim 19. \( \text{CON-B} \) is true since by Claims 16 and 17 no relevant changes are made. \( \text{CON-C} \) is true since by Claim 11, 12 and 17 no relevant changes are made. \( \text{CON-D} \) is true since by Claim 12 no relevant changes are made. \( \text{CON-E} \) is true since by Claims 11 and 18 no relevant changes are made. \( \text{CON-F} \) is true by Claim 18 and code.

\[ \text{Let } P'_{\text{CON}} = (P'_{\text{COM}} \circ S_6) \land P_{\text{CON}}. \]

Corollary 24: \( P'_{\text{CON}} \) is true in every reachable state of \( \text{CON} \).

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 \( \text{TAR, DC, NOT} \) and \( \text{CON} \) are united into one. All variables that are derived in one of these automata are also derived (in the same way) in \( \text{GHS} \). In addition, there are the following derived variables. The variable \( \text{dstatus}(p) \) of \( \text{DC} \) is refined by the variable \( \text{nstatus}(p) \), and has values sleeping, find, and found; initially, it is sleeping. The \( \text{awake} \) variable is now derived, and is true if and only if at least one
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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 requeuing), 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 \( \text{find} \) is only sent if \( \text{testlink}(p) \neq \text{nil} \) in our version, instead of whenever \( \text{nstatus}(p) = \text{find} \) as in the original. As a result of this change, if \( \text{nstatus}(p) = \text{find} \) and \( \text{testlink}(p) = \text{nil} \), \( p \) 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 \( \text{Node}(p) \), for all \( p \in V(G) \), and \( \text{Link}(l) \), for all \( l \in L(G) \), and then hiding actions appropriately to fit the \( MST(G) \) problem specification.

First we describe the automaton \( \text{Node}(p) \), for \( p \in V(G) \). The state has the following components:

- \( \text{nstatus}(p) \), either sleeping, find, or found;
- \( \text{nfrag}(p) \), an edge of \( G \) or \( \text{nil} \);
- \( \text{nlevel}(p) \), a nonnegative integer;
- \( \text{bestlink}(p) \), a link of \( G \) or \( \text{nil} \);
- \( \text{bestwt}(p) \), a weight or \( \infty \);
- \( \text{testlink}(p) \), a link of \( G \) or \( \text{nil} \);
- \( \text{inbranch}(p) \), a link of \( G \) or \( \text{nil} \); and
- \( \text{findcount}(p) \), a nonnegative integer.

For each link \( (p, q) \in L_p(G) \), there are the following variables:

- \( \text{lstatus}(p, q) \), either unknown, branch or rejected;
- \( \text{queue}_p((p, q)) \), a FIFO queue of messages from \( p \) to \( q \) waiting at \( p \) to be sent;
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

- \( queue_p((q,p)) \), a FIFO queue of messages from \( q \) to \( p \) waiting at \( p \) to be processed; and

- \( answered((p,q)) \), a Boolean.

The set of possible messages \( M \) is \( \{ \text{CONNECT}(l) : l \geq 0 \} \cup \{ \text{INITIATE}(l,c,st) : l \geq 0, c \in E(G), st \text{ is find or found} \} \cup \{ \text{TEST}(l,c) : l \geq 0, c \in E(G) \} \cup \{ \text{REPORT}(w) : w \text{ is a weight or } \infty \} \cup \{ \text{ACCEPT, REJECT, CHANGEROOT} \} \).

In the start state of \( Node(p) \), \( nstatus(p) = \text{sleeping}, nfrag(p) = \text{nil}, nlevel(p) = 0 \), \( \text{bestlink}(p) \) is arbitrary, \( \text{bestwt}(p) \) is arbitrary, \( \text{testlink}(p) = \text{nil}, \text{inbranch}(p) \) is arbitrary, \( \text{findcount}(p) = 0 \), \( lstatus(l) = \text{unknown} \) for all \( l \in L_p(G) \), \( answered(l) = \text{false} \) for all \( l \in L_p(G) \), and both queues are empty.

Now we describe the actions of \( Node(p) \).

Input actions:

- \( \text{Start}(p) \)
  Effects:
  
  if \( nstatus(p) = \text{sleeping} \) then execute procedure \( \text{WakeUp}(p) \)

- \( \text{ChannelRecv}(l), l \in L_p(G), m \in M \)
  Effects:
  
  enqueue(m, \( queue_p(l) \))

Output actions:

- \( \text{InTree}(l), l \in L_p(G) \)
  Preconditions:
  
  \( answered(l) = \text{false} \)
  \( lstatus(l) = \text{branch} \)
  Effects:
  
  \( answered(l) := \text{true} \)

- \( \text{NotInTree}(l), l \in L_p(G) \)
  Preconditions:
  
  \( answered(l) = \text{false} \)
  \( lstatus(l) = \text{rejected} \)
  Effects:
  
  \( answered(l) := \text{true} \)
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

- **ChannelSend** \((l, m), l \in L_p(G), m \in M\)
  
  **Preconditions:**
  
  \(m\) at head of \(queue_p(l)\)
  
  **Effects:**
  
  dequeue\(\left(queue_p(l)\right)\)
  
  **Internal actions:**

- **ReceiveConnect** \((q, p), l, (p, q) \in L_p(G)\)
  
  **Preconditions:**
  
  \(CONNECT(l)\) at head of \(queue_p((q, p))\)
  
  **Effects:**
  
  dequeue\(\left(queue_p((q, p))\right)\)
  
  if \(nstatus(p) = \text{sleeping}\) then execute procedure \(\text{WakeUp}(p)\)
  
  if \(l < nlevel(p)\) then [
  
  \(lstatus((p, q)) := \text{branch}\)
  
  if \(testlink(p) \neq \text{nil}\), then [
  
  enqueue\(\left(INITIATE(nlevel(p), nfrag(p), \text{find}), queue_p((p, q))\right)\)
  
  \(\text{findcount}(p) := \text{findcount}(p) + 1\) ]
  
  else enqueue\(\left(INITIATE(nlevel(p), nfrag(p), \text{found}), queue_p((p, q))\right)\) ]
  
  else
  
  if \(lstatus((p, q)) = \text{unknown}\) then enqueue\(\left(CONNECT(l), queue_p((q, p))\right)\)
  
  else enqueue\(\left(INITIATE(nlevel(p) + 1, (p, q), \text{find}), queue_p((p, q))\right)\) ]

- **ReceiveInitiate** \((q, p), l, c, st), (p, q) \in L_p(G)\)
  
  **Preconditions:**
  
  \(INITIATE(l, c, st)\) at head of \(queue_p((q, p))\)
  
  **Effects:**
  
  dequeue\(\left(queue_p((q, p))\right)\)
  
  \(nlevel(p) := l\)
  
  \(nfrag(p) := c\)
  
  \(nstatus(p) := st\)
  
  — let \(S = \{(p, r) : lstatus((p, r)) = \text{branch}, r \neq q\}\) —
  
  enqueue\(\left(INITIATE(l, c, st), queue_p(k)\right)\) for all \(k \in S\)
  
  if \(st = \text{find}\) then [
  
  \(\text{inbranch}(p) := (p, q)\)
  
  \(\text{bestlink}(p) := \text{nil}\)
  
  \(\text{bestwt}(p) := \infty\)
  
  execute procedure \(\text{Test}(p)\)
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

\[ \text{findcount}(p) := |S| \]

- **ReceiveTest**\((q,p,l,c)\), \((p,q) \in L_p(G)\)
  
  Preconditions:
  
  TEST\((l,c)\) at head of queue\(_p((q,p))\)
  
  Effects:
  
  dequeue(queue\(_p((q,p))\))
  
  if nstatus\((p) = \text{sleeping} \) then execute procedure WakeUp\((p)\)
  
  if \(l > nlevel(p)\) then enqueue\(\text{TEST}(l,c), \text{queue}_p((q,p))\)
  
  else
  
  if \(c \neq nfrag(p)\) then enqueue\(\text{ACCEPT}, \text{queue}_p((p,q))\)
  
  else [  
  
  if lstatus\((p,q)\) = unknown then lstatus\((p,q)\) := rejected
  
  if testlink\((p) \neq (p,q)\) then enqueue\(\text{REJECT}, \text{queue}_p((p,q))\)
  
  else execute procedure Test\((p)\) ]

- **ReceiveAccept**\((q,p)\), \((p,q) \in L_p(G)\)
  
  Preconditions:
  
  ACCEPT at head of queue\(_p((q,p))\)
  
  Effects:
  
  dequeue(queue\(_p((q,p))\))
  
  testlink\((p) := \text{nil} \)
  
  if wt\((p,q) < \text{bestwt}(p)\) then [  
  
  bestlink\((p) := (p,q) \)
  
  bestwt\((p) := \text{wt}(p,q) \) ]
  
  execute procedure Report\((p)\)

- **ReceiveReject**\((q,p)\), \((p,q) \in L_p(G)\)
  
  Preconditions:
  
  REJECT at head of queue\(_p((q,p))\)
  
  Effects:
  
  dequeue(queue\(_p((q,p))\))
  
  if lstatus\((p,q)\) = unknown then lstatus\((p,q)\) := rejected
  
  execute procedure Test\((p)\)

- **ReceiveReport**\((q,p), w\), \((p,q) \in L_p(G)\)
  
  Preconditions:
  
  REPORT\((w)\) at head of queue\(_p((q,p))\)
  
  Effects:
  
  dequeue(queue\(_p((q,p))\))
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

if \((p, q) \neq \text{inbranch}(p)\) then [  
findcount(p) := findcount(p) - 1  
if \(w < \text{bestwt}(p)\) then [  
\text{bestwt}(p) := w  
\text{bestlink}(p) := (p, q)  
execute procedure Report(p) ]  
else  
if \(\text{nstatus}(p) = \text{find}\) then enqueue(REPORT(u \text{queue}_p, ((q, p))))  
else if \(w > \text{bestwt}(p)\) then execute procedure ChangeRoot(p)
]

- ReceiveChangeRoot((q, p)), \((p, q) \in L_p(G)\)  
  Preconditions:  
  CHANGEROOT at head of \text{queue}_p((q, p))  
  Effects:  
  dequeue(queue_p((q, p)))  
  execute procedure ChangeRoot(p)

Procedures

- WakeUp(p)  
  - let \((p, q)\) be the minimum-weight link of \(p\)  
  \(\text{listatus}((p, q)) := \text{branch}\)  
  \(\text{nstatus}(p) := \text{found}\)  
  enqueue(CONNECT(0), queue_p((p, q)))

- Test(p)  
  if \(l\), the minimum-weight link of \(p\) with \(\text{listatus}(l) = \text{unknown}\), exists then [  
  \text{testlink}(p) := l  
  enqueue(TEST(nlevel(p), nfrag(p)), queue_p(l)) ]  
  else [  
  \text{testlink}(p) := nil  
  execute procedure Report(p) ]

- Report(p)  
  if findcount(p) = 0 and testlink(p) = nil then [  
  \text{nstatus}(p) := \text{found}\  
  enqueue(REPORT(\text{bestwt}(p)), queue_p(\text{inbranch}(p))) ]

- ChangeRoot(p)  
  if \(\text{listatus}(%\text{bestlink}(p)) = \text{branch}\) then
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

enqueue(CHANGERoot, queue_p(bestlink(p)))
else |
enqueue(CONNECT(nlevel(p)), queue_p(bestlink(p)))
Istatus(bestlink(p)) := branch |

Now we describe the automaton \textit{Link}(\langle p, q \rangle), for each \langle p, q \rangle \in L(G).

The state consists of the single variable \textit{queue}_{pq}(\langle p, q \rangle), a FIFO queue of messages. The set of messages, \textit{M}, is the same as for \textit{Node}(p). The queue is empty in the start state.

**Input Actions:**

- \textit{ChannelSend}(\langle p, q \rangle, m), m \in \textit{M}
  - Effects:
    enqueue(m, queue_{pq}(\langle p, q \rangle))

**Output Actions:**

- \textit{ChannelRecv}(\langle p, q \rangle, m), m \in \textit{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 \textit{GHS} to be the result of composing the automata \textit{Node}(p), for all \textit{p} \in V(G), and \textit{Link}(l), for all \textit{l} \in L(G), and then hiding all actions except for \textit{Start}(p), \textit{p} \in V(G), \textit{InTree}(l) and \textit{NotInTree}(l), \textit{l} \in L(G).

Given a FIFO queue \textit{q} and a set \textit{M}, define \textit{q} \vert \textit{M} to be the FIFO queue obtained from \textit{q} by deleting all elements of \textit{q} that are not in \textit{M}.

**Derived Variables:**

- \textit{queue}(\langle p, q \rangle) is \textit{queue}_p(\langle p, q \rangle) \parallel \textit{queue}_{pq}(\langle p, q \rangle) \parallel \textit{queue}_q(\langle p, q \rangle).
- \textit{tarqueue}_p(\langle p, q \rangle) is \textit{queue}_p(\langle p, q \rangle) \parallel \textit{MTAR}, where \textit{MTAR} is the set of all possible messages in \textit{TAR}; similarly for \textit{tarqueue}_{pq}(\langle p, q \rangle) and \textit{tarqueue}_q(\langle p, q \rangle).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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((p, q)) = branch and no CONNECT message is in queue((p, q)), or (2) lstatus((q, p)) = branch and no CONNECT message is in queue((p, q)).

- 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((q, r)), 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((p, q)) is empty for all q, and
  (c) lstatus((p, q)) = unknown for all q.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

- **GHS-B**: If \( \text{CONNECT}(l) \) is in \( \text{queue}((q,p)) \), \( \text{lstatus}((p,q)) \neq \text{unknown} \), and no \( \text{CONNECT} \) is in \( \text{queue}((p,q)) \), then
  (a) the state of \( \text{queue}((q,p)) \) is \( \text{CONNECT}(l) \) followed by \( \text{INITIATE}(l+1,(p,q)) \).
  (b) \( \text{queue}((p,q)) \) is empty;
  (c) \( \text{nstatus}(q) \neq \text{find} \); and
  (d) \( \text{nlevel}(p) = \text{nlevel}(q) = l \).

- **GHS-C**: If a \( \text{CONNECT} \) message is in \( \text{queue}(l) \), then no \( \text{FIND} \) message precedes the \( \text{CONNECT} \) in \( \text{queue}(l) \), and no \( \text{TEST} \) or \( \text{REJECT} \) message is in \( \text{queue}(l) \).

- **GHS-D**: If \( \text{INITIATE}(l,c,\text{find}) \) is in \( \text{subtree}(f) \), then \( l = \text{level}(f) \).

- **GHS-E**: If \( \text{INITIATE}(l,c,st) \) is in \( \text{queue}((p,q)) \) and \( (p,q) = \text{core}(\text{fragment}(p)) \), then \( st = \text{find} \).

- **GHS-F**: If \( \text{TEST}(l,c) \) is in \( \text{queue}((q,p)) \), then \( \text{nlevel}(q) \geq l \).

- **GHS-G**: If \( \text{ACCEPT} \) is in \( \text{queue}((q,p)) \), then \( \text{nlevel}(p) \leq \text{nlevel}(q) \).

- **GHS-H**: If \( \text{testlink}(p) \neq \text{nil} \), then \( \text{nstatus}(p) = \text{find} \).

- **GHS-I**: If \( p \) is up-to-date, then \( \text{nlevel}(p) = \text{level}(\text{fragment}(p)) \).

- **S-J**: If \( p \) is up-to-date, \( p \notin \text{testset}(\text{fragment}(p)) \), and \( (p,q) \) is the minimum-weight external link of \( p \), then \( \text{nlevel}(p) \leq \text{nlevel}(q) \).

- **GHS-K**: If \( \text{subtree}(f) = \{p\} \) and \( \text{nstatus}(p) \neq \text{sleeping} \), then \( \text{rootchanged}(f) = \text{true} \).

Let \( P_{\text{GHS}} \) be the conjunction of GHS-A through GHS-K.

We now define \( M_x = (S_x,A_x) \), an abstraction mapping from \( \text{GHS} \) to \( x \), for \( x = \text{TAR, DC, NOT} \) and \( \text{CON} \). \( S_x \) should be obvious for all \( x \), given the above derived functions. We now define \( A_x(s,\pi) \) for all \( x \), states \( s \) of \( \text{GHS} \), and actions \( \pi \) of \( \text{GHS} \) enabled in \( s \).

- \( \pi = \text{InTree}(l) \) or \( \text{NotInTree}(l) \). \( A_x(s,\pi) = \pi \) for all \( x \).

- \( \pi = \text{Start}(p) \). Let \( f = \text{fragment}(p) \).

**Case 1**: \( \text{nstatus}(p) = \text{sleeping} \) in \( s \). For all \( x \), \( A_x(s,\pi) = \text{Start}(p) \) \( t_x \text{ ChangeRoot}(f) \), where \( t_x \) is the same as \( S_x(s) \) except that \( \text{awake} = \text{true} \) in \( t_x \).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

Case 2: nstatus(p) ≠ sleeping in s. A_x(s, π) = π for all x.

- \( \pi = \text{ChannelRecv}(k, m) \). For all x, \( A_x(s, \pi) \) is empty, with the following exceptions: If \( m = \text{CONNECT}(l) \) or \( \text{CHANGEROOT} \), then \( A_{\text{CON}}(s, \pi) = \pi \). If \( m = \text{INITIATE}(l, c, st) \), then \( A_{\text{NOT}}(s, \pi) = \text{ChannelRecv}(k, \text{NOTIFY}(l, c)) \), and if \( st = \text{find} \), then \( A_{\text{DC}}(s, \pi) = \text{ChannelRecv}(k, \text{FIND}) \). If \( m = \text{TEST} \), \( \text{ACCEPT} \) or \( \text{REJECT} \), then \( A_{\text{TAR}}(s, \pi) = \pi \). If \( m = \text{REPORT}(w) \), then \( A_{\text{DC}}(s, \pi) = \pi \).

- \( \pi = \text{ChannelSend}(k, m) \). Analogous to \( \text{ChannelRecv}(k, m) \).

- \( \pi = \text{ReceiveConnect}((q, p), l) \). Let \( f = \text{fragment}(p) \) and \( g = \text{fragment}(q) \). (Later we will show that the following four cases are exhaustive.)

Case 1: nstatus(p) = sleeping in s. If \( (p, q) \) is not the minimum-weight external link of p in s, then \( A_x(s, \pi) = \text{ChangeRoot}(f) \) for all x. If \( (p, q) \) is the minimum-weight external link of p in s, then, for all x, \( A_x(s, \pi) = \text{ChangeRoot}(f) t_x \text{ Merge}(f, g) \), where \( t_x \) is the state of x resulting from applying \( \text{ChangeRoot}(f) \) to \( S_x(s) \).

Case 2: nstatus(p) ≠ sleeping, \( l = \text{nlevel}(p) \), and no \( \text{CONNECT} \) message is in \( \text{queue}((p, q)) \) in s. If \( \text{Istatus}((p, q)) = \text{unknown} \) in s, then \( A_x(s, \pi) \) is empty for all x. If \( \text{Istatus}((p, q)) \neq \text{unknown} \) in s, then \( A_{\text{TAR}}(s, \pi) \) is empty, and \( A_x(s, \pi) = \text{AfterMerge}(p, q) \) for all other x.

Case 3: nstatus(p) ≠ sleeping, \( l = \text{nlevel}(p) \), and a \( \text{CONNECT} \) message is in \( \text{queue}((p, q)) \) in s. \( A_x(s, \pi) = \text{Merge}(f, g) \) for all x.

Case 4: nstatus(p) ≠ sleeping, and \( l < \text{nlevel}(p) \) in s. \( A_x(s, \pi) = \text{Absorb}(f, g) \) for all x.

- \( \pi = \text{ReceiveInitiate}((q, p), l, c, st) \).

\( A_{\text{TAR}}(s, \pi) = \text{SendTest}(p) \) if \( st = \text{find} \), and is empty otherwise.

If \( st \neq \text{find} \), then \( A_{\text{DC}}(s, \pi) \) is empty; if \( st = \text{find} \) and there is a link \( (p, r) \) such that \( \text{Istatus}((p, r)) = \text{unknown} \) in s, then \( A_{\text{DC}}(s, \pi) = \text{ReceiveFind}((q, p)) \); if \( st = \text{find} \) and there is no link \( (p, r) \) such that \( \text{Istatus}((p, r)) = \text{unknown} \) in s, then \( A_{\text{DC}}(s, \pi) = \text{ReceiveFind}((q, p)) t \text{ TestNode}(p) \), where \( t \) is the state of DC resulting from applying \( \text{ReceiveFind}((q, p)) \) to \( S_{\text{DC}}(s) \).

\( A_{\text{NOT}}(s, \pi) = \text{ReceiveNotify}((q, p), l, c) \).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

$\mathcal{A}_{CON}(s, \pi)$ is empty.

- $\pi = ReceiveTest((q, p), l, c)$. Let $f = fragment(p)$.

Case 1: $nstatus(p) = \text{sleeping in } s$.

$\mathcal{A}_{TAR}(s, \pi) = ChangeRoot(f) \triangleright \pi$, where $t$ is the same as $S_{TAR}(s)$ except that $\text{rootchanged}(f) = \text{true}$ and $\text{lstatus}(\text{minlink}(f)) = \text{branch in } t$.

$\mathcal{A}_{x}(s, \pi) = ChangeRoot(f)$ for all other $x$.

Case 2: $nstatus(p) \neq \text{sleeping in } s$.

$\mathcal{A}_{TAR}(s, \pi) = \pi$ if $l \leq nlevel(p)$ or $nlevel(p) = \text{level}(f)$ in $s$, and is empty otherwise.

$\mathcal{A}_{DC}(s, \pi) = TestNode(p)$ if $l \leq nlevel(p)$, $c = nfrag(p)$, $\text{testlink}(p) = (p, q)$, and $\text{lstatus}(p, r) \neq \text{unknown}$ for all $r \neq q$, in $s$, and is empty otherwise.

$\mathcal{A}_{x}(s, \pi)$ is empty for all other $x$.

- $\pi = ReceiveAccept((q, p))$.

$\mathcal{A}_{TAR}(s, \pi) = \pi$.

$\mathcal{A}_{DC}(s, \pi) = TestNode(p)$.

$\mathcal{A}_{x}(s, \pi)$ is empty for all other $x$.

- $\pi = ReceiveReject((q, p))$.

$\mathcal{A}_{TAR}(s, \pi) = \pi$.

$\mathcal{A}_{DC}(s, \pi) = TestNode(p)$ if there is no $r \neq q$ such that $\text{lstatus}(p, r) = \text{unknown}$ in $s$, and is empty otherwise.

$\mathcal{A}_{x}(s, \pi)$ is empty for all other $x$.

- $\pi = ReceiveReport((q, p), w)$. Let $f = fragment(p)$.

Case 1: $(p, q) = \text{core}(f)$, $nstatus(p) \neq \text{find}$, $w > bestwt(p)$, and $\text{lstatus}(\text{bestlink}(p)) = \text{branch in } s$.

$\mathcal{A}_{DC}(s, \pi) = \pi$. 

125
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

\( A_{x}(s, \pi) = ComputeMin(f) \) for all other \( x \).

**Case 2:** \((p,q) = \text{core}(f), nstatus(p) \neq \text{find}, w > \text{bestwt}(p), \) and \( lstatus(\text{bestlink}(p)) \neq \text{branch} \) in \( s \).

\[
A_{DC}(s, \pi) = \pi \ t_{DC} \ ChangeRoot(f)
\]
where \( t_{DC} \) is the state of DC resulting from applying \( \pi \) to \( S_{DC}(s) \).

\[
A_{CON}(s, \pi) = ComputeMin(f).
\]

\( A_{x}(s, \pi) = ComputeMin(f) \ t_{x} \ ChangeRoot(f) \) for all other \( x \), where \( t_{x} \) is the state of \( x \) resulting from applying \( ComputeMin(f) \) to \( S_{x}(s) \).

**Case 3:** \((p,q) \neq \text{core}(f) \) or \( nstatus(p) = \text{find} \) or \( w \leq \text{bestwt}(p) \) in \( s \).

\[
A_{DC}(s, \pi) = \pi.
\]

\( A_{x}(s, \pi) \) is empty for all other \( x \).

- \( \pi = \text{ReceiveChangeRoot}((q,p)) \). Let \( f = \text{fragment}(p) \).

\[
A_{CON}(s, \pi) = \pi.
\]

For all other \( x \), \( A_{x}(s, \pi) = ChangeRoot(f) \) if \( lstatus(\text{bestlink}(p)) \neq \text{branch} \) 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 \( \bigwedge_{x \in I} (P_{x} \circ S_{x}) \land P_{GHS} \). I.e., they are derivable from \( P_{GHS} \), together with the TAR, DC, NOT, CON, GC, COM and HI predicates.

- GHS-L: If \( \text{AfterMerge}(p,q) \) is enabled for DC or NOT, then a CONNECT message is at the head of \( \text{queue}((q,p)) \).

**Proof:** First we show the predicate for DC. Let \( f = \text{fragment}(p) \).

1. \((p,q) = \text{core}(f) \), by precondition.
2. FIND is in \( \text{dcqueue}((q,p)) \), by precondition.
3. No FIND is in \( \text{dcqueue}((p,q)) \), by precondition.
4. dcstatus(q) = unfind, by precondition.
5. No REPORT is in \( \text{dcqueue}((q,p)) \), by precondition.
6. \( q \in \text{testset}(f) \), by Claims 1 through 5 and DC-G.
7. testlink(p) = nil, by Claim 4 and GHS-H.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

8. A CONNECT is in queue((q, p)), by Claims 1, 3, 6 and 7.
9. (p, q) ∈ subtree(f), by Claim 1 and COM-F.
10. No INITIATE(*, *, found) is in queue((q, p)), by Claim 1 and GHS-E.
11. No CHANGEROOT is in queue((q, p)), by Claim 1.
12. No ACCEPT is in queue((q, p)), by Claim 9 and TAR-F.
13. CONNECT precedes any FIND, TEST, or REJECT in queue((q, p)), 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((q, p)), by precondition.
3. No NOTIFY(nlevel(p) + 1, (p, q)) is in nqueue((p, q)), by precondition.
4. nlevel(q) ≠ nlevel(p) + 1, by precondition.
5. INITIATE(nlevel(p) + 1, (p, q), find) is in queue((q, p)), 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((p, q)), by Claims 3 and 6 and GHS-D.
8. q is not up-to-date, by Claims 4 and 6 and GHS-I.
9. destatus(q) ≠ find, by Claim 8 and DC-I(a).
10. No REPORT is in queue((q, p)), 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) ≠ nil or findcount(p) > 0, then no FIND message is headed toward p, and no CONNECT message is in queue((q, r)), where (q, r) = core(fragment(p)) and p ∈ subtree(q).

Proof:
1. testlink(p) ≠ nil or findcount(p) > 0, by assumption.
2. nstatus(p) = find, by Claim 1 and either GHS-H or DC-H(b).
3. destatus(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((q, r)), or lstatus((r, q)) = unknown, or CONNECT is in queue((r, q)), by Claim 3 and GHS-B(c).
6. (q, r) ∈ subtree(fragment(p)), by COM-F.
7. lstatus((r, q)) ≠ unknown, by Claim 6 and TAR-A(b).
8. If CONNECT is in queue((r, q)) then no CONNECT is in queue((q, r)), by Claim 6.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

9. If no CONNECT is in queue((r,q)) then no CONNECT is in queue((q,r)), 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 \{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 \(S_x(s)\) is in start(\(x\)) for all \(x\).

(2) Obviously, \(A_x(s,\pi)|ext(x) = \pi|ext(GHS)\) for all \(x\).

(3) Let \((s',\pi,s)\) be a step of GHS such that \(\bigwedge_{x \in I} P'_x(S_x(s'))\) 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) \(\pi\) is InTree((\(p,q\))). Let \(f = fragment(p)\) in \(s'\).

(3a) Obviously, \(P_{GHS}\) is true in \(s\).

(3b)/(3c) \(A_x(s',\pi) = \pi\) for all \(x\).

Claims about \(s'\):

1. answered((\(p,q\))) = false, by precondition.
2. lstatus((\(p,q\))) = brauch, by precondition.
3. nstatus(p) \(\neq\) sleeping, by Claim 2 and GHS-A(c).
4. awake = true, by Claim 3.
5. \((p,q) \in subtree(f)\) or \((p,q) = minlink(f)\), by Claim 2 and TAR-A(a).

\(\pi\) is enabled in \(S_x(s')\) 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 \(S_x(s)\) for all \(x\).

ii) \(\pi\) is NotInTree((\(p,q\))). Let \(f = fragment(p)\) in \(s'\).

(3a) Obviously, \(P_{GHS}\) is true in \(s\).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

(3b)/(3c) $A_x(s, \pi) = \pi$ for all $x$.

Claims about $s'$:

1. $\text{answered}((p, q)) = \text{false}$, by precondition.
2. $\text{lstatus}((p, q)) = \text{rejected}$, by precondition.
3. $\text{nstatus}(p) \neq \text{sleeping}$, by Claim 2 and GHS-A(c).
4. $\text{awake} = \text{true}$, by Claim 3.
5. $\text{fragment}(p) = \text{fragment}(q)$ and $(p, q) \neq \text{subtree}(f)$, by Claim 2 and TAR-B.

$\pi$ is enabled in $S_z(s')$ 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 $S_z(s)$ for all $x$.

iii) $\pi$ is Start($p$). Let $f = \text{fragment}(p)$.

Case 1: $\text{nstatus}(p) \neq \text{sleeping}$ in $s'$. $A_x(s', \pi) = \pi$ for all $x$. Obviously $S_x(s')\pi S_x(s)$ is an execution fragment of $x$ for all $x$, and $PGHS$ is true in $s$.

Case 2: $\text{nstatus}(p) = \text{sleeping}$ in $s'$.

(3b)/(3c) For all $x$, $A_x(s', \pi) = \pi t_x \text{ ChangeRoot}(f)$, where $t_x$ is the same as $S_x(s')$ except that $\text{awake} = \text{true}$ in $t_x$. For all $x$, we must show that $\pi$ is enabled in $S_x(s')$ (which is true because $\pi$ is an input action), that its effects are mirrored in $t_x$ (which is true by definition of $t_x$), that $\text{ChangeRoot}(f)$ is enabled in $t_x$, and that its effects are mirrored in $S_x(s)$.

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. $\text{nstatus}(p) = \text{sleeping}$, by assumption.
2. $\text{subtree}(f) = \{p\}$, by Claim 1 and GHS-A.
3. $\text{minlink}(f) = l$, by Claim 2 and definition.
4. $\text{lstatus}((p, q)) = \text{unknown}$, for all $q$, by Claim 1 and GHS-A(c).
5. $\text{rootchanged}(f) = \text{false}$, by Claim 4 and TAR-H.

Claims about $t_x$, for all $x$:

6. $\text{awake} = \text{true}$, by definition.
7. $\text{subtree}(f) = \{p\}$, by Claim 2.
8. $\text{rootchanged}(f) = \text{false}$, by Claim 5.
Section 4.2.7: \textit{GHS} Simultaneously Simulates \textit{TAR}, \textit{DC}, \textit{NOT}, \textit{CON}


\textit{ChangeRoot}(f) is enabled in \(t_{\text{CON}}\) by Claims 6, 7 and 8. For all other \(x\), \textit{ChangeRoot}(f) is enabled in \(t_x\) by Claims 6, 8 and 9.

\textit{Claims about s:}

10. \textit{CONNECT}(0) is in \textit{queue}(l), by code.
12. \textit{rootchanged}(f) = \textit{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 \textit{branch}, \(MSF\) does not change, since a \textit{CONNECT} message is in \textit{queue}(l).

For \(x = \textit{TAR}\), the effects of \textit{ChangeRoot}(f) are mirrored in \(S_x(s)\) by Claims 11 and 12. For \(x = \textit{CON}\), the effects of \textit{ChangeRoot}(f) are mirrored in \(S_x(s)\) by Claim 10. For all other \(x\), the effects of \textit{ChangeRoot}(f) are mirrored in \(S_x(s)\) by Claim 12.

(3a) \textit{More Claims about s':}

13. \textit{lstatus}((q,p)) \neq \textit{rejected}, for all q, by Claim 2 and TAR-B.
14. If \textit{lstatus}((q,p)) = \textit{branch}, then a \textit{CONNECT} is in \textit{queue}((q,p)), for all q, by Claim 2.
15. \textit{testset}(f) = \emptyset, by Claim 3 and GC-C.
17. \textit{queue}(l) is empty, by Claim 1 and GHS-A(b).

GHS-A is vacuously true since \(nstatus(p) = \text{found}\) in \(s\).

GHS-B: vacuously true for CONNECT added to \textit{queue}(l) by Claims 13 and 14; vacuously true for any CONNECT already in \textit{queue}(\textit{reverse}(l)) by Claim 10; vacuously true for any CONNECT already in \textit{queue}((q,p)), for any q such that \((p,q) \neq l\), by Claim 4.

GHS-C is true by Claim 17 and code.

GHS-H is vacuously true by Claim 16.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

No change affects the others.

iv) \( \pi \) is \( \text{ChannelRecv}(k,m) \) or \( \text{ChannelSend}(k,m) \). Obviously \( P_{GHS}(s) \) is true, and the step simulations are correct.

v) \( \pi \) is \( \text{ReceiveConnect}((q,p),l) \). Let \( f = \text{fragment}(p) \), and \( g = \text{fragment}(q) \) in \( s' \). We consider four cases. We now show that they are exhaustive, i.e., that \( l > n\text{level}(p) \) is impossible. First, suppose \((q,p)\) is an external link of \( g \). By CON-D, \( l = \text{level}(g) \) and \((q,p) = \text{minlink}(g) \). By NOT-D, \( \text{level}(g) \leq n\text{level}(p) \). Second, suppose \((q,p)\) is an internal link of \( g = f \). By CON-E, \((p,q) = \text{core}(f) \), and \( l < \text{level}(f) \). But by NOT-C, \( n\text{level}(p) \geq \text{level}(f) - 1 \).

Case 1: \( n\text{status}(p) = \text{sleeping} \). 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. \( \text{CONNECT}(l) \) is at head of \( \text{queue}_p((q,p)) \), by precondition.
2. \( n\text{status}(p) = \text{sleeping} \), by assumption.
3. \( \text{subtree}(f) = \{p\} \), by Claim 2 and GHS-A.
4. \( \text{rootchanged}(f) = \text{false} \), by Claim 2, GHS-A(c) and TAR-H.
5. \( \text{minlink}(f) = k \), by Claim 3 and definition.
6. \( \text{awake} = \text{true} \), by Claim 1 and CON-A.
7. No \( \text{FIND} \) is in \( \text{queue}((q,p)) \), by Claim 3 and DC-D(a).
8. \( f \neq g \), by Claim 3.
9. \( (q,p) \) is an external link of \( g \), by Claim 8.
10. \( \text{minlink}(g) = (q,p) \), by Claims 1 and 9 and CON-D
11. \( \text{level}(g) \leq \text{level}(f) \), by Claim 10 and COM-A.
12. \( l = \text{level}(g) \), by Claims 1 and 9 and CON-D.
13. \( \text{level}(f) = 0 \), 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. \( n\text{level}(p) = 0 \), by Claims 3 and 13.

Subcase 1a: \((p,q) \neq k \). By Claim 2 and GHS-A(c), \( l\text{status}((p,q)) = \text{unknown} \) 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 \( \text{CONNECT}(l) \) message is requeued.

131
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

(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 $x$, $A_x(s', \pi) = ChangeRoot(f)$. For $x = CON$, $ChangeRoot(f)$ is enabled in $S_x(s')$ by Claims 6, 4 and 3; for all other $x$, it is enabled by Claims 6, 4 and 5.

Claims about $s$:

17. $Istatus(k) = \text{branch}$, by code.
18. $\text{CONNECT}(0)$ is added to the end of $queue(k)$, by code.
19. $\text{rootchanged}(f) = \text{true}$, 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, $destatus(p)$ remains unchanged. Even though $Istatus(k)$ changes to branch, $MSF$ does not change, since a $CONNECT$ message is in $queue(k)$.

The effects of $ChangeRoot(f)$ are mirrored in $S_x(s)$ by Claims 17 and 19 for $x = TAR$, by Claim 18 for $x = CON$, and by Claim 19 for all other $x$.

Subcase 1b: $(p, q) = k$.

(3b)/(3c) For all $x$, $A_x(s', \pi) = ChangeRoot(f) \cdot tx \cdot Merge(f, g)$, where $tx$ is the result of applying $ChangeRoot(f)$ to $S_x(s')$. $ChangeRoot(f)$ is enabled in $S_x(s')$ 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 $tx$.

More claims about $s'$:

20. $k = (p, q)$, by assumption.
21. $(p, q)$ is an external link of $f$, by Claim 8.
22. $\text{rootchanged}(g) = \text{true}$, by Claim 1 and Claim 9.
23. Only one $CONNECT$ message is in $queue((q, p))$, by Claims 1 and 9 and CON-D.
24. $Istatus((q, p)) = \text{branch}$, 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.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

28. No INITIATE message is in queue((p, q)) or queue((q, p)), by Claims 9 and 21 and NOT-H(e).
29. No CONNECT message is in queue((p, r)) for any r ≠ q, by Claims 3 and 20 and CON-D.
30. No CONNECT message is in queue((q, r)) for any r ≠ p, by Claims 10 and 26 and CON-D.

Claims about t_x:

31. f ≠ 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((p, q)), by definition of t_x.
36. If x = CON, then connect(0) is at the head of cqueue((q, p)), 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((q, p)), by Claim 23 and code.
38. lstatus((q, p)) = branch, by Claim 24 and code.
39. (p, q) ∈ 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((p, q)), 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((p, q)), 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 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
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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), \( nstatus(q) \neq \text{sleeping in } s' \); 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 \text{CONNECT} added to \( queue((p, q)) \).

(a) \( queue((p, q)) \) has the correct contents in \( s \) because of the code and the fact that \( queue((p, q)) \) is empty in \( s' \) by Claim 2 and GHS-A(b).

(b) To show that \( queue((q, p)) \) is empty in \( s \), we must show that it contains only the \text{CONNECT} in \( s' \). By Claim 1 and GHS-C, there is no \text{TEST} or \text{REJECT} in \( queue((q, p)) \). By Claim 2 and GHS-H, \( testlink(p) = \text{nil} \); thus, by TAR-D, no \text{ACCEPT} is in \( queue((q, p)) \). By Claim 3, DC-A(g) and DC-B(a), there is no \text{REPORT} in \( queue((q, p)) \). By Claim 3 and NOT-H(e), there is no \text{NOTIFY} in \( queue((q, p)) \). By Claim 3 and CON-C, there is no \text{CHANGEROOT} in \( queue((q, p)) \). By Claim 1, CON-D and CON-E, there is only one \text{CONNECT} in \( queue((q, p)) \).

(c) \( nstatus(p) \neq \text{find in } s \) by code.

(d) By Claims 16 and 27, \( nlevel(p) = nlevel(q) = 0 \).

GHS-C: No \text{FIND} is in \( queue((p, q)) \) in \( s' \) by Claim 3 and DC-D(a). No \text{REJECT} is in \( queue((p, q)) \) in \( s' \) by Claim 3 and TAR-G. No \text{TEST}(l, c), for any \( l \) and \( c \), is in \( queue((p, q)) \) 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 \text{INITIATE} added to \( queue((p, q)) \). By Claim 28, this is the only relevant message affected.

GHS-H is true in \( s \) since \( nstatus(p) \) goes from \text{sleeping} to \text{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 \( nstatus(p) \neq \text{find} \) and \( nstatus(q) \neq \text{find} \). Then by Claim 40, no node in \( subtree(h) \) is
Section 4.2.7: *GHS* Simultaneously Simulates *TAR, DC, NOT, CON*

up-to-date, so the predicate is vacuously true (for h). By code, $d\text{c}\text{st}\text{a}\text{t}\text{u}\text{s}(p) =$ 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), $d\text{c}\text{st}\text{a}\text{t}\text{u}\text{s}(q) \neq \text{find}$ 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: $n\text{st}\text{a}\text{t}\text{u}\text{s}(p) \neq \text{sleeping}, l = n\text{le}\text{v}\text{e}\text{l}(p)$, and no CONNECT message is in queue((p, q)) in $s'$.

Subcase 2a: $l\text{st}\text{a}\text{t}\text{u}\text{s}(p, q) = \text{unknown}$ 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) $\neq \text{subtree}(f)$. Thus, by DC-D(a), no FIND is in queue((q, p)) in $s'$, and the predicates are still true in $s$.

(3b)/(3c) $A_x(s', \pi)$ is empty for all $x$. We now show that $S_x(s') = S_x(s)$ for all $x$, by showing that cqueue((q, p)) 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((q, p)). By CON-D and CON-E, only one CONNECT message is in cqueue((q, p)).

Subcase 2b: $l\text{st}\text{a}\text{t}\text{u}\text{s}(p, q) \neq \text{unknown}$ in $s'$.

(3b)/(3c) $A_{\text{TAR}}(s', \pi)$ is empty, and $A_x(s', \pi) = \text{AfterMerge}(p, q)$ for all other $x$.

Claims about $s'$:

1. CONNECT is at head of queue$_p((q, p))$, by precondition.
2. $n\text{st}\text{a}\text{t}\text{u}\text{s}(p) \neq \text{sleeping}$, by assumption.
3. $n\text{le}\text{v}\text{e}\text{l}(p) = l$, by assumption.
4. No CONNECT is in queue((p, q)), by assumption.
5. $l\text{st}\text{a}\text{t}\text{u}\text{s}(p, q) \neq \text{unknown}$, by assumption.
6. If $l\text{st}\text{a}\text{t}\text{u}\text{s}(p, q) = \text{rejected}$, then fragment(p) = fragment(q), by TAR-B.
7. If \(\text{istatus}(p, q) = \text{branch}\), then \((p, q) \in \text{subtree}(f)\), by Claim 4 and definition of MSF.

8. \((p, q)\) is an internal link of \(f\), by Claims 5, 6 and 7.

9. \((p, q) = \text{core}(f)\), by Claims 1 and 8 and CON-E.

10. \(\text{INITIATE}(nlevel(p) + 1, (p, q), \text{find})\) is in \(\text{queue}(\langle q, p \rangle)\), by Claims 1, 3, 4 and 5 and GHS-B(a).

11. No \(\text{INITIATE}(nlevel(p) + 1, (p, q), \ast)\) is in \(\text{queue}(\langle p, q \rangle)\), by Claims 1, 3, 4 and 5 and GHS-B(b).

12. \(\text{destatus}(q) \neq \text{find}\), by Claims 1, 4 and 5 and GHS-B(c).

13. No \(\text{REPORT}\) is in \(\text{queue}(\langle q, p \rangle)\), by Claims 1, 4 and 5 and GHS-B(a).

14. \(nlevel(q) = 1\), by Claims 1, 4 and 5 and GHS-B(d).

\(\text{AfterMerge}(p, q)\) is enabled in \(S_i(s')\) by Claims 9, 10, 11, 12 and 13 for \(x = \text{DC}\); by Claims 3, 9, 10, 11 and 14 for \(x = \text{NOT}\); and by Claims 1 and 9 for \(x = \text{CON}\).

Claims about \(s\):

15. \(\text{CONNECT}(l)\) is dequeued from \(\text{queue}_p(\langle q, p \rangle)\), by code.

16. \(\text{FIND}\) is in \(\text{queue}(\langle p, q \rangle)\), by code.

17. \(\text{INITIATE}(nlevel(p) + 1, (p, q), \text{find})\) is in \(\text{queue}(\langle p, q \rangle)\), by code.

The only derived variables that are not obviously unchanged are \(\text{testset}(f)\), \(\text{level}(f)\) and \(\text{core}(f)\). Claims 15 and 16 show that \(\text{testset}(f)\) is unchanged. Claims 10 and 17 show that \(\text{level}(f)\) and \(\text{core}(f)\) are unchanged.

The effects of \(\text{AfterMerge}(p, q)\) are mirrored in \(S_i(s)\) by Claim 16 for \(x = \text{DC}\); by Claim 17 for \(x = \text{NOT}\); and by Claim 15 for \(x = \text{CON}\). It is easy to see that \(S_{\text{TAR}}(s') = S_{\text{TAR}}(s)\).

(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 \(\text{CONNECT}\) message in \(\text{queue}(\langle q, p \rangle)\) in \(s'\). Since it is removed in \(s\), the predicate is vacuously true for a \(\text{CONNECT}\) in \(\text{queue}(\langle q, p \rangle)\). By Claim 4, the predicate is vacuously true for a \(\text{CONNECT}\) in \(\text{queue}(\langle p, q \rangle)\).

GHS-C: By Claim 4, vacuously true for \(\text{queue}(\langle p, q \rangle)\).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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 8: \( nstatus(p) \neq \) sleeping, \( l = nlevel(p) \), and a connect message is in queue\((p, q)\) in \( s' \).

\( (3b)/(3c) \ A_x(s', \pi) = \text{Merge}(f, g) \) for all \( x \).

Claims about \( s' \):

1. \( \text{connect}(l) \) is at head of queue\((q, p)\), by precondition.
2. \( l = nlevel(p) \), by assumption.
3. \( \text{connect}(m) \) is in queue\((p, q)\), by assumption.
4. \( (p, q) \) is an external link of \( p \), by Claims 1 and 3.
5. \( (q, p) \) is an external link of \( q \), by Claims 1 and 3.
6. \( f \neq g \), by Claim 4.
7. \( \text{rootchanged}(f) = \text{true} \), by Claims 1 and 4.
8. \( \text{rootchanged}(g) = \text{true} \), by Claims 3 and 5.
9. \( (q, p) = \text{minlink}(g) \), by Claims 1 and 5 and CON-D.
10. \( (p, q) = \text{minlink}(f) \), by Claims 3 and 4 and CON-D.
11. \( \text{minedge}(f) = \text{minedge}(g) \), 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.

\( \text{Merge}(f, g) \) is enabled in \( SCON(s') \) 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\((q, p)\), by Claim 1 and CON-D.
16. \( lstatus((q, p)) = \text{branch} \), by Claims 8 and 9 and TAR-H.
17. \( lstatus((p, q)) = \text{branch} \), by Claims 7 and 10 and TAR-H.
18. \( level(g) = l \), by Claims 1 and 5 and CON-D.
19. If \( \text{initiate}(l', c, *) \) is in subtree\((f) \), then \( l' \leq l \), by Claims 12 and 14.
20. If \( \text{initiate}(l', c, *) \) is in subtree\((g) \), then \( l' \leq l \), by Claim 18.
21. \( nlevel(r) \leq l \) for all \( r \in \text{nodes}(f) \), by Claims 12 and 14.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

22. \( nlevel(r) \leq l \) for all \( r \in \text{nodes}(g) \), by Claim 18.
23. No INITIATE message is in \( \text{queue}((q,p)) \) or \( \text{queue}((p,q)) \), by Claims 4 and 5 and NOT-H(e).
24. No CONNECT is in \( \text{queue}((r,t)) \), where \( r \in \text{nodes}(f) \), and \( (r,t) \neq (p,q) \), by Claim 10 and CON-D and CON-F.
25. No CONNECT is in \( \text{queue}((r,t)) \), where \( r \in \text{nodes}(g) \) and \( (r,t) \neq (q,p) \), by Claim 9 and CON-D and CON-F.
26. \( (p,q) \neq \text{core}(f) \), by Claim 4 and COM-F.
27. \( (p,q) \neq \text{core}(g) \), 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 \( \text{queue}((q,p)) \), by Claim 15 and code.
29. \( \text{lstatus}((q,p)) = \text{branch} \), by Claim 16.
30. \( (p,q) \in \text{MSF} \), by Claims 28 and 29.
31. \( \text{subtree}(h) \) is the union of the old \( \text{subtree}(f) \) and \( \text{subtree}(g) \) and \( (p,q) \), by Claim 30.
32. INITIATE\( (l + 1, (p,q), \text{find}) \) is in \( \text{queue}((p,q)) \), by Claim 2 and 17 and code.
33. if INITIATE\( (l', c, *) \) is in \( \text{subtree}(h) \), then \( l' \leq l + 1 \), by Claims 19, 20, 23, 31 and 32.
34. \( nlevel(r) \leq l \) for all \( r \in \text{nodes}(h) \), by Claims 21, 22 and 31.
35. \( \text{level}(h) = l + 1 \), by Claims 33 and 34.
36. \( \text{core}(h) = (p,q) \), by Claims 19, 20, 23, 31, 32, and 34.
37. CONNECT\( (l) \) is in \( \text{queue}((p,q)) \), by Claims 3 and 14.
38. \( \text{testset}(h) = \text{nodes}(h) \), by Claims 31, 32 and 37.
39. \( \text{minlink}(h) = \text{nil} \), by Claim 38.
40. \( \text{rootchanged}(h) = \text{false} \), 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 \( S_x(s) \) 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).

138
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

GHS-B: Obviously, the only situation affected is the connect in \( \text{queue}(\langle p, q \rangle) \).

(a) We must show that in \( s' \), \( \text{queue}(\langle p, q \rangle) \) consists only of a \text{CONNECT}(l) message. (The code adds the appropriate \text{INITIATE} message.) By Claim 3 and GHS-C, no \text{TEST} or \text{REJECT} is in \( \text{queue}(\langle p, q \rangle) \). By Claim 4, DC-A(g) and DC-B(a), no \text{REPORT} is in \( \text{queue}(\langle p, q \rangle) \). By Claim 23, no \text{NOTIFY} is in \( \text{queue}(\langle p, q \rangle) \). By Claim 4 and CON-C, no \text{CHANGEROOT} is in \( \text{queue}(\langle p, q \rangle) \). By Claims 3 and 14, a \text{CONNECT}(l) message is in \( \text{queue}(\langle p, q \rangle) \), and by CON-E and CON-F, it is the only \text{CONNECT} message in that queue.

(b) A very similar argument to that in (a) shows that in \( s' \), \( \text{queue}(\langle q, p \rangle) \) consists only of a \text{CONNECT}(l) message. (Since it is removed in \( s \), the queue is then empty.)

(c) If \( |\text{nodes}(f)| > 1 \), then \text{dcstatus}(p) \neq \text{find} by Claim 10. Suppose \( \text{subtree}(f) = \{p\} \). Obviously, no \text{REPORT} message is headed toward \( p \) in \( s' \). By Claim 10 and GC-C, \( \text{testset}(f) = 0 \) in \( s' \). Thus, by DC-I(b), \text{dcstatus}(p) \neq \text{find} in \( s' \). In both cases, \text{nstatus}(p) does not change in \( s \).

(d) \( \text{nlevel}(q) = l \) in \( s' \) by assumption. \( \text{nlevel}(q) = l \) 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 \text{INITIATE} message is OK.

GHS-D: by Claim 35.

GHS-E: By code, for the \text{INITIATE} added. By Claim 23, there are no leftover \text{INITIATE} messages affected by the change of core.

GHS-I: We show no \( r \in \text{nodes}(h) \) in \( s \) is up-to-date. By Claim 38, \( r \) is in \( \text{testset}(h) \). By the same argument as in GHS-B(c), \text{dcstatus}(r) \neq \text{find}.

GHS-J: Vacuously true by Claim 38.

No change affects the rest.

\hline

Case 4: \text{nstatus}(p) \neq \text{sleeping}, and \( l < \text{nlevel}(p) \) in \( s' \).

(3b)/(3c) \( A_x(s', \pi) = \text{Absorb}(f, g) \) for all \( x \).

Claims about \( s' \):
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

1. CONNECT(l) is at head of queue((q,p)), by precondition.
2. \( l < nlevel(p) \), by assumption.
3. \( lstatus((p,q)) = \text{unknown} \), or a CONNECT is in queue((p,q)), by Claims 1 and 2 and GHS-B(d).
4. \((q,p)\) is an external link of \( g \), by Claims 1 and 3.
5. \( \text{minlink}(g) = (q,p) \), by Claims 1 and 4 and CON-D.
6. \( l = \text{level}(g) \), by Claims 1 and 4 and CON-D.
7. \( \text{rootchanged}(g) = \text{true} \), by Claims 1 and 4.
8. \( nlevel(p) \leq \text{level}(f) \), by definition of \( \text{level}(f) \).
9. \( \text{level}(g) < \text{level}(f) \), by Claims 2, 6 and 8.
10. \( lstatus((q,p)) = \text{branch} \), by Claims 5 and 7 and TAR-H.
11. If \( \text{INITIATE}(l',c,\ast) \) is in subtree\((g)\), then \( l' < \text{level}(f) \), by Claims 6 and 9.
12. If \( \text{INITIATE}(l',c,\ast) \) is in subtree\((f)\), then \( l' \leq \text{level}(f) \), by definition of \( \text{level}(f) \).
13. \( nlevel(r) < \text{level}(f) \), for all \( r \in \text{nodes}(g) \), by Claims 6 and 9.
14. \( nlevel(r) \leq \text{level}(f) \), for all \( r \in \text{nodes}(f) \), by definition of \( \text{level}(f) \).
15. No \( \text{INITIATE} \) message is in queue((q,p)) or queue((p,q)), by Claim 4 and NOT-H(e).
16. No CONNECT message is in queue((r,t)), where \( r \in \text{nodes}(g), (r,t) \neq (q,p) \), 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. \( \text{level}(f) > 0 \), by Claims 18 and 9.
20. \( \text{core}(f) \neq \text{nil} \), by Claim 19 and COM-F.
21. \( \text{core}(f) \in \text{subtree}(f) \), by Claim 20 and COM-F.
22. If \( \text{subtree}(g) = \{q\} \), then \( \text{core}(g) = \text{nil} \), by COM-F.
23. If \( \text{subtree}(g) \neq \{q\} \), then \( \text{core}(g) \in \text{subtree}(g) \), by COM-F.
24. Only one CONNECT message is in queue((q,p)), by Claims 1 and 4 and CON-D.
25. \( \text{testset}(g) = \emptyset \), by Claim 5 and GC-C.
26. \( \text{testlink}(r) = \text{nil} \), for all \( r \in \text{nodes}(g) \), by Claim 25.
27. If \( \text{testlink}(p) \neq \text{nil} \), then \( p \in \text{testset}(f) \), by definition.
28. If \( \text{testlink}(p) \neq \text{nil} \), then \( nstatus(p) = \text{find} \), by GHS-H.
29. If \( nstatus(p) = \text{find} \), then no \( \text{FIND} \) message is headed toward \( p \), by DC-D(b) and DC-H(a).
30. \( lstatus((r,t)) \neq \text{unknown} \), where \( (r,t) = \text{core}(f) \), by Claim 21 and TAR-A(b).
31. If CONNECT is in queue((r,t)), then no CONNECT is in queue((t,r)), where \( (r,t) = \text{core}(f) \), by Claim 21.
32. If \( nstatus(p) = \text{find} \) and \( p \in \text{subtree}(r) \), then \( nstatus(r) = \text{find} \), for all \( r \), by DC-H(a).

140
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

33. If \( \text{nstatus}(p) = \text{find} \), then no \text{CONNECT} is in \( \text{queue}((r, t)) \), where \( (r, t) = \text{core}(f) \) and \( p \in \text{subtree}(r) \), by Claims 30, 31 and 32 and GHS-B(c).

34. If \( \text{nstatus}(p) = \text{find} \) and \( p \in \text{testset}(f) \), then \( \text{testlink}(p) \neq \text{nil} \), by Claims 29 and 33.

\( \text{Absorb}(f, g) \) 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 \( s' \).

\textit{Claims about }s:\

35. No \text{CONNECT} message is in \( \text{queue}((q, p)) \), by Claim 24 and code.

36. \( \text{status}((q, p)) = \text{branch} \), by Claim 10.

37. \( (p, q) \in \text{MSF} \), by Claims 35 and 36.

38. \( \text{subtree}(h) \) is the union of the old \( \text{subtree}(f) \) and \( \text{subtree}(g) \) and \( (p, q) \), by Claim 37.

39. \( \text{INITIATE}(nlevel(p), nfrag(p), \text{nstatus}(p)) \) is in \( \text{queue}((p, q)) \), by code.

40. \( \text{level}(h) = \text{old level}(f) \), by Claims 11, 12, 13, 14, 15 and 38.

41. \( \text{core}(h) = \text{old core}(f) \), by Claims 11, 12, 13, 14, 15 and 38.

42. \( h = f \), by Claim 41.

43. \( g \not\in \text{fragments} \), by Claims 38 and 41.

44. \( \text{NOTIFY}(nlevel(p), nfrag(p)) \) is added to \( \text{queue}_p((p, q)) \), by code.

First, we discuss how \( \text{testset}(f) \) changes. If \( p \in \text{testset}(f) \) in \( s' \) because of a \text{FIND} or \text{CONNECT} message, then every node in \( \text{nodes}(g) \) in \( s' \) is in \( \text{testset}(f) \) in \( s \) because of the same FIND or CONNECT message. If \( p \in \text{testset}(f) \) in \( s' \) because \( \text{testlink}(p) \neq \text{nil} \), then a \text{FIND} message is added to \( \text{queue}((p, q)) \) in \( s \), causing every node formerly in \( \text{nodes}(g) \) to be in \( \text{testset}(f) \). If \( p \) is not in \( \text{testset}(f) \) in \( s' \), then no \text{FIND} message is headed toward \( p \), and no \text{CONNECT} message is in \( \text{queue}((r, t)) \), with \( p \in \text{subtree}(r) \); thus, Claim 25 implies that in \( s \), no node formerly in \( \text{nodes}(g) \) is in \( \text{testset}(f) \).

By the previous paragraph, and inspection, the effects of \( \text{Absorb}(f, g) \) are mirrored in \( S_x(s) \) by Claims 36, 38, 42 and 43 for \( x = \text{TAR} \); by Claims 27, 28, 34, 38, 42 and 43 for \( x = \text{DC} \); by Claims 38, 42, 43 and 44 for \( x = \text{NOT} \); and by Claims 35, 38, 42 and 43 for \( x = \text{CON} \).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, 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 $queue((q,p))$ by Claim 35. By Claim 4 and CON-D, if CONNECT is in $queue((p,q))$, then $minlink(f) = (p,q)$. But by Claim 9 and COM-A, this cannot be. Thus the predicate is vacuously true for a CONNECT in $queue((p,q))$.

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 $(r,t)$ 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 $s$, $(r,t)$ 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) $\pi$ is $ReceiveInitiate((q,p),l,c,st)$. Let $f = fragment(p)$.

(3b)/(3c) Case 1: $st = find$. $A_{TAR}(s', \pi) = SendTest(p)$.

If there is a link $(p,r)$ such that $lstatus((p,r))$ = unknown in $s'$, then $A_{DC}(s', \pi) = ReceiveFind((q,p))$; otherwise $A_{DC}(s', \pi) = ReceiveFind((q,p)) t TestNode(p)$, where $t$ is the state resulting from applying $ReceiveFind((q,p))$ to $S_{DC}(s')$.

$A_{NOT}(s', \pi) = ReceiveNotify((q,p),l,c)$.

$A_{CON}(s', \pi)$ is empty.

Claims about $s'$:
1. $initiate(l,c,find)$ is at the head of $queue_p((q,p))$, by precondition.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

2. \((p, q) \in \text{subtree}(f)\), by Claim 1 and DC-D(a).
3. \(\text{minlink}(f) = \text{nil}\), by Claims 1 and 2.
4. If \(lstatus((p, r)) = \text{rejected}\) then \(\text{fragment}(p) = \text{fragment}(r)\), for all \(r\), by TAR-B.
5. If \(lstatus((p, r)) = \text{branch}\), then \((p, r) \in \text{subtree}(f)\), for all \(r\), by Claim 3 and TAR-A(a).
6. If \((p, r) \in \text{subtree}(f)\), then \(lstatus((p, r)) = \text{branch}\) for all \(r\), by TAR-A(b).
7. \(\minlink(f) = \text{nil}\), by Claims 1 and 2.

4. If \(lstatus((p, r)) = \text{rejected}\) then \(\text{fragment}(p) = \text{fragment}(r)\), for all \(r\), by TAR-B.
8. \(p \in \text{testset}(f)\), by Claims 1 and 2.
9. \(d\text{cstatus}(p) = \text{unfind}\), by Claim 1 and DC-D(b).
10. \(\text{testlink}(p) = \text{nil}\), by Claim 9 and GHS-H.
11. \(l = \text{level}(f)\), by Claims 1 and 2 and GHS-D.
12. \(c = \text{core}(f)\), 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. \(\text{core}(f) \neq \text{nil}\), by Claim 2 and COM-F.

Let \((r, t) = \text{core}(f)\).

15. \((r, t) \in \text{subtree}(f)\), by Claim 14 and COM-F.

Let \(p\) be in \(\text{subtree}(r)\).

16. If \((p, q) \neq (r, t)\) then \(d\text{cstatus}(q) = \text{find}\), by Claim 1 and DC-D(a).
17. If \((p, q) \neq (r, t)\) then \(d\text{cstatus}(r) = \text{find}\), by Claim 16 and DC-H(a).
18. If \((p, q) \neq (r, t)\) then either no CONNECT is in \(\text{queue}((r, t))\), or \(lstatus((t, r)) = \text{unknown}\), or a CONNECT is in \(\text{queue}((t, r))\), by Claim 17 and GHS-B(c).
19. If \((p, q) = (r, t)\) then either no CONNECT is in \(\text{queue}((r, t))\), or \(lstatus((t, r)) = \text{unknown}\), or a CONNECT is in \(\text{queue}((t, r))\), by Claim 1 and GHS-B(b).
20. Either no CONNECT is in \(\text{queue}((r, t))\), or \(lstatus((t, r)) = \text{unknown}\), or a CONNECT is in \(\text{queue}((t, r))\), by Claims 18 and 19.
21. \(lstatus((t, r)) \neq \text{unknown}\), by Claim 15 and TAR-A(b).
22. If CONNECT is in \(\text{queue}((t, r))\) then no CONNECT is in \(\text{queue}((r, t))\), by Claim 15.
23. If no CONNECT is in \(\text{queue}((t, r))\) then no CONNECT is in \(\text{queue}((r, t))\), by Claims 20, 21 and 22.
24. No CONNECT is in \(\text{queue}((r, t))\), by Claims 22 and 23.
25. If \((p, q) \neq (r, t)\) then \(\text{AfterMerge}(p, q)\) is not enabled (for DC or NOT), since \((r, t) = \text{core}(f)\).
26. If \((p, q) = (r, t)\) then \(\text{AfterMerge}(p, q)\) is not enabled (for DC or NOT), by Claim 24 and GHS-L.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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).

\(\text{SendTest}(p)\) is enabled in \(S_{TAR}(s')\) by Claims 8 and 10. \(\text{ReceiveFind}(\langle q, p \rangle)\) is enabled in \(S_{DC}(s')\) by Claims 1, 25 and 26. \(\text{ReceiveNotify}(\langle q, p, l, c \rangle)\) is enabled in \(S_{NOT}(s')\) by Claims 1, 25 and 26.

Claims about \(t\): (only defined when there are no unknown links of \(p\) in \(s'\))

29. \(p \in \text{testset}(f)\), by Claim 8.
30. There is no external link of \(p\), by Claim 27.
31. \(\text{dcstatus}(p) = \text{find}\), by definition of \(t\).

\(\text{TestNode}(p)\) is enabled in \(t\) by Claims 29, 30 and 31.

Claims about \(s\):

32. \(\text{level}(f) = 1\), by Claim 11 and code.
33. \(\text{core}(f) = c\), by Claim 12 and code.
34. No \text{FIND} message is headed toward \(p\), by Claim 13 and code.
35. No \text{CONNECT} is in \(\text{queue}((t, r))\), by Claim 24 and code.
36. There is no unknown link of \(p\) (in \(s'\)) if and only if \(\text{testlink}(p) = \text{nil}\) (in \(s\)), by Claim 10 and code.
37. There is no unknown link of \(p\) (in \(s'\)) if and only if \(p \not\in \text{testset}(f)\) (in \(s\)), by Claims 34, 35 and 36.
38. If \(|S| > 0\) (in \(s'\)) then a \text{FIND} message is in \(\text{subtree}(f)\), by Claim 5 and code.
39. If \(|S| = 0\) and there is no unknown link of \(p\) (in \(s'\)), then \(p \neq \text{mw-root}(f)\) (in \(s\)), by Claim 7 and code.
40. If \(|S| = 0\) and there is no unknown link of \(p\) (in \(s'\)), then either a \text{REPORT} message is headed toward \(\text{mw-root}(f)\), 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 \(\text{nstatus}(p) = \text{find}\) (in \(s\)), by code.
42. \(\text{minlink}(f) = \text{nil}\), by Claims 38, 40 and 41.

The changes (or lack of changes) to the remaining derived variables are obvious.

The effects of \(\text{SendTest}(p)\) are mirrored in \(S_{TAR}(s)\) 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 \(\text{ReceiveFind}(\langle q, p \rangle)\) are mirrored in \(S_{DC}(s)\) by Claims 5, 6, 36 and 37 for changes, and Claims 3, 11, 12, 32, 33, 37 and
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

42 for lack of changes. If there is no unknown link of p in s', then the effects of
ReceiveFind((q, p)) followed by TestNode(p) are mirrored in 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((q, p), l, c) are mirrored in S_{NOT}(s) by Claims 3, 4 and 42.
S_{CON}(s') = S_{CON}(s) by Claims 3, 11, 12, 32, 33, and 42.

Case 2: st ≠ find.

A_{NOT}(s', π) = ReceiveNotify((q, p), l, c). A_{x}(s', π) is empty for all other x.

Claims about s':

1. INITIATE(l, c, found) is at the head of queue_p((q, p)), by precondition.
2. (p, q) ∈ 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 ≠ minnode(f), by Claims 1 and 2 and NOT-I.
6. If lstatus((p, r)) = branch, then (p, r) ∈ subtree(f), for all r ≠ q, by Claim 5 and
   TAR-A(a).
7. If (p, r) ∈ subtree(f), then lstatus((p, r)) = branch, for all r ≠ q, by TAR-A(b).
8. p is not up-to-date, by Claim 4 and GHS-I.
9. nstatus(p) ≠ find, by Claim 8 and DC-I(a).
10. (p, q) ≠ 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((q, p), l, c) is enabled in S_{NOT}(s'). Its effects
are mirrored in S_{NOT}(s) by Claims 6 and 7.

It is easy to see that S_{x}(s') = S_{x}(s) for all other x.

(3a) GHS-A: By DC-D(a), (p, q) ∈ subtree(f). So by GHS-A(a), nstatus(p) ≠
sleeping in s'. Since the same is true in s, the predicate is vacuously true.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

GHS-B: Vacuously true for a connect in queue((q,p)) by GHS-B(a) and the fact that initiate is first in the queue. Vacuously true for a connect in queue((p,q)) by GHS-B(b) and the presence of initiate in queue((q,p)). The only other situation to consider is the addition of an initiate message to queue((p,r)), r ≠ q, with lstatus((p,r)) = branch. As shown in (b)/(c), (p,r) ∈ subtree(f). By NOT-H(e), either (p,q) = core(f) or p is a child of q, so (p,r) ≠ core(f). Thus by CON-E, no connect is in queue((p,r)), or in queue((r,p)).

GHS-C: Adding a find message does not falsify the predicate. Suppose a test message is added to queue((p,r)). Then in s', st = find.

Case 1: (p,r) is an internal link of f. By TAR-A(b), (p,r) ≠ subtree(f). By COM-F, (p,r) ≠ core(f). By CON-E, no connect is in queue((p,r)).

Case 2: (p,r) 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((p,r)).

GHS-D: Since it is true for the initiate in queue((q,p)) in s', it is true for any initiate added in s.

GHS-E: As shown in GHS-B, (p,r) ≠ 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((p,r)). By NOT-H(a), nlevel(p) increases, so the predicate is still true.

Case 2: An accept is in queue((r,p)). By TAR-D, testlink(p) = (p,r). By GHS-H, nstatus(p) = find. But by Claim 9 (for both Case 1 and Case 2 of (3b)/(3c)), nstatus(p) ≠ find. So there is no accept in queue((r,p)), 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 ≠ 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) ≠ find in s, p is still not up-to-date.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

GHS-J: Case 1: \( st = \text{find} \). By Claim 37 in Case 1 of (3b)/(3c), \( p \notin \text{testset}(f) \) in \( s \) if and only if there is no external link of \( p \), so the predicate is vacuously true.

Case 2: \( st \neq \text{find} \). As in GHS-I, Case 2, \( p \) is not up-to-date, so the predicate is vacuously true.

vii) \( \pi \) is ReceiveTest((q,p),l,c). Let \( f = \text{fragment}(p) \).

Case 1: \( \text{nstatus}(p) = \text{sleeping in } s' \).

\( (3b)/(3c) \ A_{\text{TAR}}(s', \pi) = \text{ChangeRoot}(f) t \pi \), where \( t \) is the same as \( S_{\text{TAR}}(s') \) except that \( \text{rootchanged}(f) = \text{true} \) and \( \text{lstatus(minlink}(f)) = \text{branch in } t \).

\( A_x(s', \pi) = \text{ChangeRoot}(f) \) for all other \( x \).

Claims about \( s' \):
1. \( \text{TEST}(l, c) \) is at the head of \( \text{queue}_p((q, p)) \), by precondition.
2. \( \text{nstatus}(p) = \text{sleeping} \), by assumption.
3. \( \text{subtree}(f) = \{p\} \), by Claim 2 and GHS-A.
4. \( \text{minlink}(f) \neq \text{nil} \), by Claim 3 and definition.
5. \( \text{rootchanged}(f) = \text{false} \), by Claim 2, GHS-A(c) and TAR-H.
6. \( \text{level}(f) = 0 \), by Claim 3 and COM-F.
7. \( \text{nlevel}(p) = 0 \), by Claims 3 and 6.
8. \( l \geq 1 \), by TAR-M.
9. \( l > \text{nlevel}(p) \), by Claims 7 and 8.
10. \( l > \text{level}(f) \), by Claims 6 and 8.
11. \( \text{awake} = \text{true} \), by Claim 1 and GHS-A(b).

Claims about \( s \):
12. The \( \text{TEST} \) message is requeued, by Claim 9.
13. \( \text{lstatus}(\text{minlink}(f)) = \text{branch} \), by code.
14. \( \text{CONNECT}(0) \) is in \( \text{queue}(\text{minlink}(f)) \), by code.
15. \( \text{minlink}(f) \) does not change (i.e., is still external), by Claims 13 and 14.
16. \( \text{rootchanged}(f) = \text{true} \), by Claims 14 and 15.

\( \text{ChangeRoot}(f) \) is enabled in \( S_x(s') \) by Claims 11, 3 and 5 for \( x = \text{CON} \), and by Claims 11, 4 and 5 for all other \( x \).

TAR: Effects of \( \text{ChangeRoot}(f) \) are mirrored in \( t \) by its definition. \( \pi \) is enabled in \( t \) by definition. Its effects are mirrored in \( S_{\text{TAR}}(s) \) by Claim 12.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

For all other \( x \), the effects of \( \text{ChangeRoot}(f) \) are mirrored in \( S_x(s) \) by Claim 16 for DC and NOT, and by Claim 14 for CON.

\[(\text{3a}) \ P_{GHS} \text{ is true in } s \text{ by essentially the same argument as in } \pi = \text{Start}(p). \]

Case 2.

\[(\text{3b})/(\text{3c}) \ A_{\text{TAR}}(s', \pi) = \pi \text{ if } l \leq nlevel(p) \text{ or } nlevel(p) = \text{level}(f) \text{ in } s'. \text{ and is empty otherwise.} \]

\[A_{\text{DC}}(s', \pi) = \text{TestNode}(p) \text{ if } l \leq nlevel(p), \text{ } c = nfrag(p), \text{ testlink}(p) = (p,q) \text{ and lstatus((p,r)) \neq unknown for all } r \neq q, \text{ in } s', \text{ and is empty otherwise.} \]

\[A_{\pi}(s', \pi) \text{ is empty for all other } x. \]

First we discuss what happens to \( \text{testset}(f) \) and \( \text{minlink}(f) \).

We show \( \text{testset}(f) \) is unchanged, except that \( p \) is removed from \( \text{testset}(f) \) if and only if \( l \leq nlevel(p), c = nfrag(p), \text{ testlink}(p) = (p,q), \text{ and there is no link } (p,r), r \neq q, \text{ with lstatus((p,r)) = unknown.} \) If \( \text{testlink}(p) \) does not change from non-nil to nil (or vice versa), then obviously \( \text{testset}(f) \) is unchanged. The only place \( \text{testlink}(p) \) 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 \( l \leq nlevel(p), c = nfrag(p), \text{ and testlink}(p) = (p,q) \text{ in } s'. \) Suppose \( \text{testlink}(p) \) is changed from non-nil to nil. Since \( \text{testlink}(p) \neq nil \text{ in } s' \), GHS-M implies that no FIND message is headed toward \( p \), and no CONNECT message is in queue((r,t)), where \((r,t) = core(f) \) and \( p \in subtree(r) \). Thus in \( s \), since \( \text{testlink}(p) = nil, \text{ } p \text{ is not in } \text{testset}(f). \)

Now we show that \( \text{minlink}(f) \) does not change. If destatus\((p)\) does not change, and no REPORT message is added to any queue, then obviously \( \text{minlink}(f) \) does not change. Suppose destatus\((p)\) changes, and a REPORT message is added to a queue (in procedure Report\((p)\)). Then \( l \leq nlevel(p), c = nfrag(p), \text{ testlink}(p) = (p,q), \text{ there are no more unknown links of } p \) (so testlink\((p)\) is set to nil), and findcount\((p)\) = 0.

Claims about \( s' \):

1. \( \text{testlink}(p) = (p,q) \), by assumption.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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((p, r)), or destatus(r) = find, or a REPORT message is in queue((r, p)), 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 ≠ mw-root(f), then either there is no external link of f, or a REPORT is headed toward mw-root(f), by Claim 5 and code.
7. If p = mw-root(f), then either a FIND is in queue((p, r)), or destatus(r) = find, or a REPORT is in queue((r, p)), 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) ≠ level(f).

Claims about s’:

1. l > nlevel(p), by assumption.
2. nlevel(p) ≠ level(f), by assumption.
3. p is not up-to-date, by Claim 2 and GHS-I.
4. nstatus(p) ≠ 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 (p, q), by Claim 5 and TAR-D.
7. The TEST message in queue((q, p)) is a protocol message for (q, p), by Claim 6.
8. testlink(q) = (q, p), by Claim 7 and TAR-D.
9. There is exactly one protocol message for (q, p), by Claim 8 and TAR-C(c).
10. There is only one TEST message in tarqueue((q, p)), by Claim 9.

Claims 6 and 10, the TEST is the only TAR message in tarqueue((q, p)). Since the TEST message is requeued in GHS, tarqueue((q, p)) is unchanged. By earlier remarks about testset(f) and minlink(f), and by inspection, the other derived variables (for TAR) are unchanged. Thus, $S_{TAR}(s') = S_{TAR}(s)$.

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, $S_{TAR}(s') \pi S_{TAR}(s)$ is an execution fragment of TAR.

Third, suppose l ≤ nlevel(p). Let g = fragment(q).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

**Claims about s':**

1. \( \text{test}(l, c) \) is at the head of \( \text{queue}_p((q, p)) \), by precondition.
2. \( l \leq n\text{level}(p) \), by assumption.
3. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( c = \text{core}(g) \) and \( l = \text{level}(g) \), by Claim 1 and TAR-E(b).
4. If \( \text{lstatus}((q, p)) = \text{rejected} \), then \( c = \text{core}(f) \) and \( l = \text{level}(f) \), by Claim 1 and TAR-E(c).
5. \( c \neq \text{nil} \), by Claim 1 and TAR-M.

Next we show that \( c = \text{core}(f) \) if and only if \( c = n\text{frag}(p) \). First, suppose \( c = \text{core}(f) \).

6. \( c = \text{core}(f) \), by assumption.
7. If \( \text{lstatus}((q, p)) = \text{rejected} \), then \( n\text{level}(p) = \text{level}(f) \), by Claims 2 and 4 and definition of \( \text{level}(f) \).
8. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( \text{core}(g) = \text{core}(f) \), by Claims 3 and 6.
9. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( c \in \text{subtree}(g) \) and \( c \in \text{subtree}(f) \), by Claims 5, 6 and 8 and COM-F.
10. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( f = g \), by Claim 9 and COM-G.
11. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( l = \text{level}(f) \), by Claims 3 and 10.
12. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( n\text{level}(p) = \text{level}(f) \), by Claims 2 and 11 and definition of \( \text{level}(f) \).
13. \( n\text{level}(p) = \text{level}(f) \), by Claims 8 and 12.
14. \( n\text{frag}(p) = \text{core}(f) \), by Claim 13 and NOT-A.
15. \( n\text{frag}(p) = c \), by Claims 6 and 14.

Now suppose \( c = n\text{frag}(p) \).

16. \( c = n\text{frag}(p) \), by assumption.
17. \( c \in \text{subtree}(f) \), by Claims 5 and 16 and NOT-F.
18. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( c \in \text{subtree}(g) \), by Claims 5 and 3 and COM-F.
19. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( f = g \), by Claims 17 and 18 and COM-G.
20. If \( \text{lstatus}((q, p)) \neq \text{rejected} \), then \( c = \text{core}(f) \), by Claims 3 and 19.
21. \( c = \text{core}(f) \), by Claims 4 and 20.

\( \pi \) is enabled in \( S_{\text{TAR}}(s') \) by Claim 1. We now verify that the effects are mirrored in \( S_{\text{TAR}}(s) \). By the above argument, \( c \neq \text{frag}(p) \) if and only if \( c \neq \text{core}(f) \). Thus, the body of \( \text{ReceiveTest} \) for \( \text{TAR} \) is simulated correctly. Consider procedure \( \text{Test}(p) \). If it is executed, then \( c = n\text{frag}(p) \) in \( s' \). By Claim 21, \( n\text{frag}(p) = \text{core}(f) \), and by
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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\((p) = (p, q) \), and lstatus\((p, r)\) \( \neq \) unknown for all \( r \neq q \), in \( s' \).

Claims about \( s' \):

1. TEST\((l, c)\) is at the head of queue\(_p\)((q, p)) \), by precondition.
2. \( l \leq nlevel(p) \), by assumption.
3. \( c = nfrag(p) \), by assumption.
4. testlink\((p) = (p, q) \), by assumption.
5. lstatus\((p, r)\) \( \neq \) unknown, for all \( r \neq q \), by assumption.
6. \( p \in \text{testset}(f) \), by Claim 4 and TAR-C(b).
7. minlink\((f) = \text{nil} \), by Claim 6 and GC-C.
8. If lstatus\((p, r)\) = branch, then \( (p, r) \in \text{subtree}(f) \), for all \( r \neq q \), by Claim 7 and TAR-A(a).
9. If lstatus\((p, q)\) = rejected, then fragment\((r) = f \), for all \( r \neq q \), by TAR-B.
10. \( c = \text{core}(f) \), 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\((p) = \text{find} \), by Claim 4 and GHS-H.

TestNode\((p)\) is enabled in \( S_{DC}(s') \) by Claims 6, 12 and 13. Its effects are mirrored in \( S_{DC}(s) \) 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\((p) \neq (p, q) \) or there is a link \( (p, r) \) with lstatus\((p, r)\) = unknown and \( r \neq q \). Then \( S_{DC}(s') = S_{DC}(s) \) by inspection and earlier discussion of testset\((f)\) and minlink\((f)\).

NOT and CON: We want to show \( S_x(s') = S_x(s) \) for \( x = \text{NOT} \) and \( \text{CON} \). 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\((q, p)\), 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.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

(3a) GHS-A is vacuously true by the assumption that nstatus(p) ≠ 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((p, q)) going from unknown to rejected. But since test is in queue((q, p)) in s', by GHS-C no connect is in queue((q, p)) 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((p, q)). We already showed that no connect is in queue((q, p)). Because of the TEST in queue((q, p)), the preconditions of the predicate are not true for a connect in queue((q, p)) in s'.

Case C: TEST is added to some queue((p, r)). Since lstatus((p, r)) = unknown, the preconditions are not true in s' for a connect in queue((r, p)). Since the TEST is added, testlink(p) = (p, q) 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((p, q)).

Case D: REPORT is added to queue(inbranch(p)). Let (p, r) = inbranch(p) in s'. As in Case 3, the predicate is vacuously true for a connect in queue((p, r)). 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) ∈ subtree(f), there cannot be an initiate(nlevel(p) + 1, *, *, *) message in queue((r, p)). By GHS-B(a), the preconditions are not true for a connect in queue((r, p)).

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((p, q)). Then l ≤ nlevel(p), c = nfrag(p), and testlink(p) ≠ (p, q) in s'. As argued in Lemma 17, verifying (3c) of Case 1 for π = ReceiveTest, (p, q) is an internal link of f. By TAR-E(a), (p, q) ≠ core(f), so by CON-E, no connect is in queue((p, q)).

Case 2: TEST is added to queue((p, r)). Then in s', l ≤ nlevel(p), c = nfrag(p), testlink(p) = (p, q), and lstatus((p, r)) = unknown.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

Case 2a: \( (p, r) \) is an internal link of \( f \). By TAR-A(b), \( (p, r) \notin \text{subtree}(f) \). By COM-F, \( (p, r) \neq \text{core}(f) \). By CON-E, no \text{CONNECT} is in \( \text{queue}((p, r)) \).

Case 2b: \( (p, r) \) is an external link of \( f \). By GHS-H, \( nstatus(p) = \text{find} \). Thus \( \text{minlink}(f) = \text{nil} \). By CON-D, no \text{CONNECT} is in \( \text{queue}((p, r)) \).

GHS-G: Suppose \text{ACCEPT} is added to \( \text{queue}((p, q)) \). Then \( l \leq \text{nlevel}(p) \) in \( s' \). As argued in Lemma 17, verifying TAR-F for \( \pi = \text{Receive Test} \), \( l = \text{level}((\text{fragment}(q))) \). By GHS-F, \( l \leq \text{nlevel}(q) \). So \( l = \text{nlevel}(q) \).

No changes affect the rest.

viii) \( \pi \) is \text{Receive Accept}((q, p)). Let \( f = \text{fragment}(p) \).

(3b)/(3c) \( \mathcal{A}_{\text{TAR}}(s', \pi) = \pi \). \( \mathcal{A}_{\text{DC}}(s', \pi) = \text{Test Node}(p) \). \( \mathcal{A}_x(s', \pi) \) is empty for all other \( x \).

An argument similar to that used in \( \pi = \text{Receive Test}((q, p), l, c) \), Case 2, shows that \( \text{minlink}(f) \) is unchanged.

TAR: Claims about \( s' \):

1. ACCEPT is at the head of \( \text{queue}_p((q, p)) \), by precondition.
2. There is a protocol message for \( (p, q) \), by Claim 1.
3. \( \text{testlink}(p) = (p, q) \), by Claim 2 and TAR-D.
4. No \text{FIND} message is headed toward \( p \), by Claim 3 and GHS-M.
5. No \text{CONNECT} message is in \( \text{queue}((r, t)) \), where \( (r, t) = \text{core}(f) \) and \( p \in \text{subtree}(r) \), by Claim 3 and GHS-M.

Claims about \( s' \):

6. \( \text{testlink}(p) = \text{nil} \), by code.
7. No \text{FIND} message is headed toward \( p \), by Claim 4.
8. No \text{CONNECT} message is in \( \text{queue}((r, t)) \), where \( (r, t) = \text{core}(f) \) and \( p \in \text{subtree}(r) \), by Claim 5 and code.
9. \( p \notin \text{testset}(f) \), by Claims 6, 7 and 8.

\( \pi \) is enabled in \( \mathcal{S}_{\text{TAR}}(s') \) by Claim 1; its effects are mirrored in \( \mathcal{S}_{\text{TAR}}(s) \) by Claims 6 and 9, and discussion of \( \text{minlink}(f) \). (The disposition of the remaining derived variables should be obvious.)

DC: More Claims about \( s' \):

153
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

10. \( p \in \text{testset}(f) \), by Claim 3.
11. \( \text{minlink}(f) = \text{nil} \), by Claim 10.
12. \( \text{fragment}(q) \neq f \), by Claim 1 and TAR-F.
13. \( \text{level}(f) \leq \text{level}(\text{fragment}(q)) \), by Claim 1 and TAR-F.
14. \( \text{lstatus}(\langle p, q \rangle) \neq \text{branch} \), 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 \( \text{lstatus}(\langle p, r \rangle) = \text{rejected} \), then \( \langle p, r \rangle \) is not external, for all \( r \), by TAR-B.
17. If \( \text{lstatus}(\langle p, r \rangle) = \text{branch} \), 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 \( \text{lstatus}(\langle p, r \rangle) = \text{unknown} \), 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. \( \text{nstatus}(p) = \text{find} \), by Claim 3 and GHS-H.

TestNode\((p)\) is enabled in \( S_{DC}(s') \) by Claims 10, 19 and 13, and 20. Its effects are mirrored in \( S_{DC}(s) \) by Claims 9, 19 and 6.

NOT and CON: It is easy to verify that \( S_x(s') = S_x(s) \) for \( x = \text{NOT} \) and CON.

(3a) GHS-A. By Claim 20, vacuously true in \( s \).

GHS-B: Suppose a \text{REPORT} message is added to \( \text{queue}(\langle p, r \rangle) \) in \( s \). Let \( \langle p, r \rangle = \text{inbranch}(p) \). By Claim 20 and DC-I(a), \( p \) is up-to-date in \( s' \). By GHS-1, \( \text{nlevel}(p) = \text{level}(f) \). By DC-L, \( (p, r) \in \text{subtree}(f) \), so no \text{INITIATE}(\text{nlevel}(p) + 1, *, *) \) can be in \( \text{queue}(\langle p, r \rangle) \) or \( \text{queue}(\langle r, p \rangle) \). By GHS-B(a), the preconditions for a \text{CONNECT} in \( \text{queue}(\langle p, r \rangle) \) or \( \text{queue}(\langle r, p \rangle) \) are not true in \( s' \), or in \( s \).

GHS-H: By code, \( \text{testlink}(p) = \text{nil} \).

GHS-J: By Claim 19 and GHS-G.

No changes affect the rest.

ix) \( \pi \) is \text{ReceiveReject}(\langle q, p \rangle). Let \( f = \text{fragment}(p) \).

(3b)/(3c) \( \mathcal{A}_{TAR}(s', \pi) = \pi \).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

\[ A_{DC}(s', \pi) = \text{TestNode}(p) \text{ if there is no } r \neq q \text{ such that } \text{status}(p, r) = \text{unknown in } s, \text{ and is empty otherwise.} \]

\[ A_{x}(s', \pi) \text{ is empty for all other } x. \]

An argument similar to that in \( \pi = \text{ReceiveTest}(q, p, l, c) \). Case 2, shows that minlink\( (f) \) is unchanged.

\text{TAR: Claims about } s':

1. \text{REJECT is at the head of } queue_{p}(q, p), \text{ by precondition.}
2. There is a protocol message for \( (p, q) \), by Claim 1.
3. testlink\( (p) = (p, q) \), 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 \( queue(r, t) \), where \( (r, t) = \text{core}(f) \) and \( p \in \text{subtree}(r) \), by Claim 3 and GHS-M.
6. nstatus\( (p) = \text{find}, \text{ by Claim 3 and GHS-H.} \)
7. nlevel\( (p) = \text{level}(f), \text{ by Claim 6, DC-I(a) and GHS-I.}\)
8. nfrag\( (p) = \text{core}(f), \text{ by Claim 7 and NOT-A.}\)

\text{Claims about } s:

9. If there is no link \( (p, r) \) with \( \text{status}(p, r) = \text{unknown} \) (in \( s' \)), then testlink\( (p) = \text{nil} \) (in \( s \)), by code.
10. No FIND message is headed toward \( p \), by Claim 4.
11. No CONNECT message is in \( queue(r, t) \), by Claim 5.
12. If there is no link \( (p, r) \) with \( \text{status}(p, r) = \text{unknown} \) (in \( s' \)), then \( p \notin \text{testset}(f) \) (in \( s \)), by Claims 9, 10 and 11.

\( \pi \) is enabled in \( S_{TAR}(s') \) by Claim 1. Its effects are mirrored in \( S_{TAR}(s) \) by Claims 9, 12, 7 and 8, and earlier discussion of minlink\( (f) \).

\text{DC: If there is a link } \( (p, r) \) \text{ such that } \text{status}(p, r) = \text{unknown and } r \neq q, \text{ then it is easy to check that } S_{DC}(s') = S_{DC}(s) \text{. Suppose there is no unknown link (other than } \( (p, q) \)) \text{.}

\text{More claims about } s':

13. \( \text{status}(p, r) \neq \text{unknown}, \text{ for all } r \neq q, \text{ by assumption.} \)
14. minlink\( (f) = \text{nil}, \text{ by Claim 6.} \)
15. If \( \text{status}(p, r) = \text{branch}, \text{ then } (p, r) \in \text{subtree}(f), \text{ for all } r \neq q, \text{ by Claim 14 and TAR-A(a).} \)
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

16. If \( lstatus((p,r)) = \) 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 \( S_{DC}(s') \) by Claims 19, 18 and 6. Its effects are mirrored in \( S_{DC}(s) \) by Claims 9 and 12.

\( NOT \) and \( CON \): It is easy to show that \( S_x(s') = S_x(s) \) 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 \( \pi = ReceiveTest((q,p),l,c) \), Case 2 of (a).

GHS-C: Only affected if a \( TEST \) is added. The argument is very similar to that in \( \pi = ReceiveTest((q,p),l,c) \), Case 2 of (a).

GHS-H: The argument is very similar to that in \( \pi = ReceiveTest((q,p),l,c) \), 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.

(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 \( (r,t) \) 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 \( queue((q,p)) \), by assumption.
2. \( (p,q) = \) core(f), by assumption.
3. \( nstatus(p) \neq \) find, by assumption.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

4. $w > \text{bestwt}(p)$, by assumption.
5. $\text{ReceiveReport}((q,p), w)$ is enabled in $S_{DC}(s')$, by Claim 1.
6. $\text{ComputeMin}(f)$ (for GC) is enabled in $S_4(S_{DC}(s'))$, by Claims 2, 3, 4 and 5 and argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = \text{ReceiveReport}$.
7. $\text{minlink}(f) = \text{nil}$, by Claim 6.
8. $\text{accmin}(f)$ $\neq$ nil, by Claim 6.
9. $\text{testset}(f) = \emptyset$, by Claim 6.
10. $\text{ComputeMin}(f)$ (for COM) is enabled in $S_2(S_{DC}(s'))$, by Claim 6 and argument in proof of Lemma 15, verifying (3c) for $\pi = \text{ComputeMin}$.
11. $\text{level}(f) \leq \text{level}(\text{fragment}(t))$, by Claim 10.
12. $\text{accmin}(f) = (r, t)$, by Claims 8 and 9 and GC-A.
13. $r$ is up-to-date, by Claim 9, DC-N, and choice of $(r, t)$.
14. $\text{nlevel}(r) = \text{level}(f)$, by Claim 13 and GHS-I.
15. $\text{nlevel}(f) \leq \text{nlevel}(t)$, by Claims 9 and 13 and GHS-J.
16. No CONNECT message is in either queue of $\text{core}(f)$, by Claim 9.
17. No CONNECT message is in any internal queue of $f$, by Claim 16 and CON-E.
18. $\text{inbranch}(p) = (p, q)$, by Claims 1 and 2 and DC-A(a).
19. $p$ is up-to-date, by Claims 2, 9 and 18.
20. $\text{findcount}(p) = 0$, by Claim 3 and DC-H(b).
21. All children of $p$ are completed, by Claims 19 and 20 and DC-K(a).
22. $r \in \text{subtree}(p)$, by Claims 1, 2, 3 and 4 and DC-P(b).
23. Following $\text{bestlinks}$ from $p$ leads along edges of $\text{subtree}(f)$ to $(r, t)$, by Claims 9, 19, 21 and 22, choice of $(r, t)$, and DC-K(b) and (c).

The following remarks apply to both Subcase 1a and Subcase 1b: $\text{ComputeMin}(f)$ is enabled in $S_x(s')$ by Claims 7, 8 and 9 for $x = \text{TAR}$; by Claims 7, 14 and 15 (and definition of $(r, t)$) for $x = \text{NOT}$; and by Claims 7, 11 and 17 for $x = \text{CON}$. $\pi$ is obviously enabled in $S_{DC}(s')$.

Subcase 1a: $\text{Istatus}(\text{bestlink}(p)) = \text{branch}$. $\mathcal{A}_{DC}(s', \pi) = \pi$. $\mathcal{A}_x(s', \pi) = \text{ComputeMin}(f)$ for all other $x$.

More Claims about $s'$:

24. $\text{Istatus}(\text{bestlink}(p)) = \text{branch}$, by assumption.
25. $\text{bestlink}(p) \in \text{subtree}(f)$, by Claims 7 and 24 and TAR-A(a).
26. $p \neq r = \text{mw-minnode}(f)$, by Claims 23 and 25.

Claims about $s$:
27. The effects of $\pi$ are reflected in $S_{DC}(s)$, by code.
28. The effects of $\text{ComputeMin}(f)$ are reflected in $S_{d}(S_{DC}(s))$, by Claim 27 and argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = \text{ReceiveReport}$.
29. $\text{minlink}(f) = (r,t)$, by Claims 28 and 12.
30. Following bestlinks from $p$ leads to $(r,t)$, by Claim 23.
31. $\text{tominlink}(p) = \text{bestlink}(p)$, by Claims 30 and 24.
32. $p \neq \text{minnode}(f)$, by Claims 29 and 24.
33. $p = \text{root}(f)$, by Claims 2, 22 and 29.

By Claims 3, 4 and 17, procedure $\text{ChangeRoot}(p)$ is executed in $\text{GHS}$. The effects of $\text{ComputeMin}(f)$ are reflected in $S_{x}(s)$ by Claims 29 and 12 for $x = \text{TAR}$; by Claim 29 and choice of $(r,t)$ for $x = \text{NOT}$; and by Claims 29, 31, 32, 33 and choice of $(r,t)$ for $x = \text{CON}$. The effects of $\pi$ are reflected in $S_{DC}(s)$ by Claim 27.

---

**Subcase 1b: lstatus(bestlink(p)) \neq \text{branch.}**

$A_{DC}(s',\pi) = \pi \ t_{DC} \ \text{ChangeRoot}(f)$, where $t_{DC}$ is the result of applying $\pi$ to $S_{DC}(s')$.

$A_{CON}(s',\pi) = \text{ComputeMin}(f)$.

For all other $x$, $A_{x}(s',\pi) = \text{ComputeMin}(f) \ t_{x} \ \text{ChangeRoot}(f)$, where $t_{x}$ is the result of applying $\text{ComputeMin}(f)$ to $S_{x}(s')$.

**More claims about $s'$:**

34. $\text{lstatus(bestlink(p))} \neq \text{branch.}$
35. $\text{bestlink}(p) = (r,t)$, by Claims 23, 34 and 7 and TAR-A(b).
36. $p = r = \text{mw-minnode}_{x}(f)$, by Claim 35.
37. $\text{nstatus}(q) \neq \text{sleeping}$, by Claim 1 and GHS-A.
38. $\text{awake} = \text{true}$, by Claim 37.
39. $\text{rootchanged}(f) = \text{false}$, by Claim 7 and COM-B.

**Claims about $t_{x}$, $x \neq \text{CON}$:**

40. If $x = \text{TAR}$, then $\text{minlink}(f) = (r,t)$, by Claim 12.
41. If $x = \text{NOT}$, then $\text{minlink}(f) = (r,t)$, by choice of $(r,t)$.
42. If $x = \text{DC}$, then $\text{minlink}(f) = (r,t)$, by Claims 6 and 12 and argument in proof of Lemma 15, verifying (3c) for $\pi = \text{ComputeMin}$.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

43. awake = true, by Claim 38.
44. rootchanged(f) = false, by Claim 39.

The effects of π are mirrored in $t_{DC}$ and of $\text{ComputeMin}(f)$ 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. $\text{minlink}(f) = (r, t)$, by argument in proof of Lemma 19, Case 1 of verifying (3c) for $\pi = \text{ReceiveReport}$.
46. $\text{Istatus(bestlink}(p)) = \text{branch}$, by code.
47. $\text{Istatus(minlink}(p)) = \text{branch}$, by Claims 35 and 45.
48. CONNECT is added to $\text{queue}(\text{bestlink}(p))$, by code.
49. rootchanged(f) = true, by Claims 45 and 48.

The effects of ChangeRoot(f) are mirrored in $S_x(s)$ by Claims 47 and 49 for $x = TAR$; by Claim 49 for $x = DC$ and $NOT$. The effects of $\text{ComputeMin}(f)$ are mirrored in $S_{CON}(s)$ by Claims 36, 14 and 45.

---

**Case 2:** $(p, q) \not\in \text{core}(f)$ or $\text{nstatus}(p) = \text{find}$ or $w \leq \text{bestwt}(p)$ in $s'$.

$A_{DC}(s', \pi) = \pi$. $A_{x}(s', \pi)$ is empty for all other $x$.

Subcase 2a: $(p, q) \not\in \text{core}(f)$ in $s'$. Suppose $(p, q) = \text{inbranch}(p)$ in $s'$. By DC-B(b), dcstatus(p) = unfind. Thus, the only effect is to remove the REPORT message. Thus $S_{DC}(s')\pi S_{DC}(s)$ is an execution fragment of DC. As proved in Lemma 19, Case 2a of verifying (3b) for $\pi = \text{ReceiveReport}$, $\text{minlink}(f)$ is unchanged. Thus $S_x(s') = S_x(s)$ for all $x \neq DC$.

Now suppose $(p, q) \not\in \text{inbranch}(p)$.

**Claims about s':**

1. REPORT is at head of $\text{queue}((q, p))$, by precondition.
2. $(p, q) \not\in \text{core}(f)$, by assumption.
3. $(p, q) \not\in \text{inbranch}(p)$, 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.
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

7. \( \text{findcount}(p) > 0 \), by Claims 1, 5 and 6 and DC-K(a).
8. No \text{FIND} message is headed toward \( p \), by Claim 7 and GHS-M.
9. No \text{CONNECT} is in \( \text{queue}(r, t) \), where \( (r, t) = \text{core}(f) \) and \( p \in \text{subtree}(r) \), by Claim 7 and GHS-M.
10. \( p \in \text{testset}(f) \) if and only if \( \text{testlink}(p) \neq \text{nil} \), by Claims 8 and 9.

Obviously, \( \pi \) is enabled in \( S_{DC}(s') \). By Claim 10 and inspection, the effects of \( \pi \) are mirrored in \( S_{DC}(s) \). Since the proof of Lemma 19, Case 2a of verifying (3b) for \( \pi = \text{ReceiveReport} \), shows \( \text{minlink}(f) \) is unchanged, \( S_x(s') = S_x(s) \) for all \( x \neq DC \).

\text{Subcase 2b}: \( (p, q) = \text{core}(f) \) and \( \text{nstatus}(p) = \text{find} \) in \( s' \). Since \( \text{report}(w) \) is at the head of \( \text{queue}(q, p) \), DC-A(a) implies that \( \text{inbranch}(p) = (p, q) \). Thus, the only change is that the \text{REPORT} message is requeued. Obviously \( S_{DC}(s') \pi S_{DC}(s) \) is an execution fragment of \( DC \), and \( S_x(s') = S_x(s) \) for all \( x \neq DC \).

\text{Subcase 2c}: \( (p, q) = \text{core}(f) \), \( \text{nstatus}(p) = \text{find} \) and \( w \leq \text{bestwt}(p) \) in \( s' \). As in Subcase 2b, \( \text{inbranch}(p) = (p, q) \). The only change is that the \text{REPORT} message is removed. Thus \( S_{DC}(s') \pi S_{DC}(s) \) is an execution fragment of \( DC \). As proved in Lemma 19, Case 2c of verifying (3b) for \( \pi = \text{ReceiveReport} \), \( \text{minlink}(f) \) is unchanged in \( s \). Thus \( S_x(s') = S_x(s) \) for all \( x \neq DC \).

(3a) \text{Case 1}: \text{inbranch}(p) \neq (p, q).

GHS-A: By DC-A(a), \( (p, q) \neq \text{core}(f) \). By DC-A(g), \( \text{dcstatus}(p) = \text{find} \). The predicate is vacuously true.

GHS-B: Only the addition of a \text{REPORT} message affects this predicate. The argument is very similar to that in \( \pi = \text{ReceiveTest}(q, p, l, c) \), Case 2, of (3a).

GHS-H: By code (in procedure Report(\( p \))).

No change affects the rest.

\text{Case 2}: \text{inbranch}(p) = (p, q). \text{If nstatus}(p) = \text{find} \text{or } w \leq \text{bestwt}(p), \text{then no change affects any predicate. Suppose nstatus}(p) \neq \text{find} \text{and } w > \text{bestwt}(p).\)
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

GHS-A: By DC-B(a), subtree(p) ≠ {p}. By GHS-A(a), nstatus(p) ≠ sleeping, so the predicate is vacuously true.

GHS-B: Let \((p, r) = \text{bestlink}(p)\) in \(s'\). If lstatus((p, r)) = branch, then no change affects this predicate. Suppose lstatus((p, r)) ≠ branch. As shown in (3b)/(3c), Claim 35 of Case 1b, bestlink(p) is the minimum-weight external link of \(f\). Thus lstatus((r, p)) ≠ rejected by TAR-B, and if lstatus((r, p)) = branch, then there is a \text{CONNECT} in queue((r, p)). So the predicate is vacuously true for the \text{CONNECT} added to queue((p, r)). If there is a leftover \text{CONNECT} in queue((r, p)), then the predicate is vacuously true because of the new \text{CONNECT} in queue((p, r)).

GHS-C: Let \((p, r) = \text{bestlink}(p)\) in \(s'\). Since bestlink(p) is external (as shown in (3b)/(3c)), no reject is in queue((p, r)) by TAR-G. Also since it is external, lstatus((p, r)) ≠ rejected by TAR-B. Suppose a test is in queue((p, r)). By TAR-D, testlink(p) = (p, r), and by GHS-H, nstatus(p) = find, which contradicts the assumption for this case. Also since the link is external, no \text{FIND} is in queue((p, r)) by DC-D(a).

No change affects the rest.

\(\pi\) is ReceiveChangeRoot((q, p)).

(3b)/(3c) There are two cases. First we prove some facts true in both cases.

Claims about \(s'\):

1. \text{CHANGEROOT} is at the head of queue((q, p)), by precondition.
2. minlink(f) ≠ nil, by Claim 1 and CON-C.
3. rootchanged(f) = false, by Claim 1 and CON-C.
4. \(p \in \text{subtree}(q)\), by Claim 1 and CON-C.
5. minnode(f) ∈ subtree(p), by Claim 1 and CON-C.
6. nlevel(minnode(f)) = level(f), by NOT-D.
7. \text{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 \text{REPORT} message is headed toward mw-root(f), by Claim 2.
12. No \text{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).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

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 $\text{subtree}(f)$ to the minimum-weight external link of $\text{subtree}(p)$, by Claims 7, 10 and 15 and DC-K(b) and (c).

---

Case 1: $lstatus(bestlink(p)) \neq \text{branch in } s'$.

$A_{\text{CON}}(s', \pi) = \pi$. $A_\pi(s', \pi) = \text{ChangeRoot}(f)$ for all other $x$.

More claims about $s'$:

17. $lstatus(bestlink(p)) \neq \text{branch},$ by assumption.
18. $bestlink(p)$ is not in $\text{subtree}(f)$, by Claim 17 and TAR-A(b).
19. $bestlink(p) = \text{minlink}(f)$, by Claims 5, 8, 16 and 18.
20. $nstatus(q) \neq \text{sleeping},$ by Claim 1 and GHS-A(b).
21. $awake = \text{true},$ by Claim 20.

Claims about $s$:

22. $lstatus(bestlink(p)) = \text{branch},$ by code.
23. $\text{CONNECT}$ is in $\text{queue(bestlink(p))},$ by code.
24. $MSF$ does not change, Claims 22 and 23.
25. $bestlink(p) = \text{minlink}(f)$, by Claims 19 and 24.
26. $\text{rootchanged}(f) = \text{true},$ by Claims 23 and 25.

$\text{ChangeRoot}(f)$ is enabled in $S_\pi(s')$ by Claims 2, 3 and 21, for all $x \neq \text{CON}$. The effects of $\text{ChangeRoot}(f)$ are mirrored in $S_\pi(s)$ by Claims 22, 25 and 26 for $x = \text{TAR}$; and by Claim 26 for $x = \text{DC}$ and NOT. $\pi$ is enabled in $S_{\text{CON}}(s')$ by Claim 1; its effects are mirrored in $S_{\text{CON}}(s)$ by Claims 6 and 19.

---

Case 2: $lstatus(bestlink(p)) = \text{branch in } s'$.

$A_{\text{CON}}(s', \pi) = \pi$. $A_\pi(s', \pi)$ is empty for all other $x$.

More Claims about $s'$:

27. $lstatus(bestlink(p)) = \text{branch},$ by assumption.
28. $lstatus(\text{minlink}(f)) \neq \text{branch},$ by Claim 3 and TAR H.
29. $bestlink(p)$ is in $\text{subtree}(f)$, by Claims 27 and 28 and TAR-A(a).
Section 4.2.7: GHS Simultaneously Simulates TAR, DC, NOT, CON

30. $p \neq \text{minnode}(f)$, by Claims 16 and 29.
31. $\text{bestlink}(p) = \text{tominlink}(f)$, by Claims 8, 16 and 29.
32. $\text{nlevel}(p) = \text{level}(f)$, by Claim 10 and GHS-I.

Obviously, all derived (and non-derived) variables are unchanged, except $cqueue$. Thus, $S_x(s') = S_x(s)$ for all $x \neq CON$. $\pi$ is enabled in $S_{CON}(s')$ by Claim 1; its effects are mirrored in $S_x(s)$ by Claims 30, 31 and 32.

---

(3a) GHS-A: By CON-C, $(p, q) \in \text{subtree}(f)$. By GHS-A(a), $\text{nstatus}(p) \neq \text{sleeping in } s'$, so the predicate is vacuously true in $s$.

GHS-B: Essentially the same argument as in $\pi = \text{ReceiveReport}((q, p), w)$, Case 2 of (3a).

GHS-C: Essentially the same argument as in $\pi = \text{ReceiveReport}((q, p), w)$, Case 2 of (3a).

No change affects the rest. \[ \Box \]

Let $P'_{GHS} = \bigwedge_{x \in I}(P'_x \circ S_x) \land P_{GHS}$.

Corollary 26: $P'_{GHS}$ is true in every reachable state of GHS.

Proof: By Lemmas 1 and 25. \[ \Box \]
Section 4.3.1: COM is Equitable for HI

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 $M_1$.

Proof: By Corollary 14, $(P_{HI} \circ S_1) \land P_{COM}$ 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 $\varphi$ of HI, we must show that COM is equitable for $\varphi$ via $M_1$.

i) $\varphi$ is Start(p) or NotInTree(l). Since $\varphi$ is enabled in $S_1(s)$ if and only if it is also enabled in $s$, and since $A_1(s, \varphi)$ includes $\varphi$, for any state $s$, Lemma 5 shows that COM is equitable for $\varphi$ via $M_1$.

ii) $\varphi$ is Combine(F,F',e). We show COM is progressive for $\varphi$ via $M_1$: Lemma 6 implies COM is equitable for $\varphi$ via $M_1$. 164
Section 4.3.1: COM is Equitable for HI

Let $\Psi_\varphi$ be the set of all pairs $(s, \psi)$ of reachable states $s$ of COM and internal actions $\psi$ 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 $\text{rootchanged}(f) = \text{false}$ in $s$, and $z$ is the number of fragments $f$ with $\text{minlink}(f) = \text{nil}$ in $s$. (Two triples are compared lexicographically.)

1. Let $s$ be a reachable state of COM in $E_\varphi$. We now demonstrate that some action $\psi$ is enabled in $s$ with $(s, \psi) \in \Psi_\varphi$.

Claims:

1. $\text{awake} = \text{true}$ in $S_1(s)$, by precondition.
2. $F \neq F'$ in $S_1(s)$, by precondition.
3. $\text{awake} = \text{true}$ in $s$, by Claim 1 and definition of $S_1$.
4. There exist $f$ and $g$ in $\text{fragments}$ such that $\text{subtree}(f) = F$ and $\text{subtree}(g) = F'$ in $s$, by Claim 2 and definition of $S_1$.
5. $f \neq g$ in $s$, by Claims 2 and 4.

Let $l = \min\{\text{level}(f') : f' \in \text{fragments}\}$ in $s$. (By Claim 4, $\text{fragments}$ is not empty in $s$, so $l$ is defined.) Let $L = \{f' \in \text{fragments} : \text{level}(f') = l\}$.

Case 1: There exists $f' \in L$ with $\text{minlink}(f') = \text{nil}$. Let $\psi = \text{ComputeMin}(f')$. We now show $\psi$ is enabled in $s$. By Claim 5, the minimum-weight external link $(p, q)$ of $f'$ exists. By choice of $l$, $\text{level}(f') \leq \text{level}(\text{fragment}(q))$. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2: For all $f' \in L$, $\text{minlink}(f') \neq \text{nil}$.

Case 2.1: There exists $f' \in L$ with $\text{rootchanged}(f') = \text{false}$. Let $\psi = \text{ChangeRoot}(f')$. $\psi$ is enabled in $s$ by Claim 3 and the assumption for Case 2. Obviously $(s, \psi) \in \Psi_\varphi$.

Case 2.2: For all $f' \in L$, $\text{rootchanged}(f') = \text{true}$.

Case 2.2.1: There exists fragment $g' \in L$ with $\text{level}(f') > l$, where $f' = \text{fragment}(\text{target}(\text{minlink}(g')))$. (By COM-G, $f'$ is uniquely defined.) Let $\psi = \text{Absorb}(f', g')$. Obviously $\psi$ is enabled in $s$, and $(s, \psi) \in \Psi_\varphi$.

Case 2.2.2: There is no fragment $g' \in L$ such that $\text{level}(f') > l$, where $f' = \text{fragment}(\text{target}(\text{minlink}(g')))$. Pick any fragment $f_1$ such that $\text{level}(f_1) = l$. For $i > 1$, define $f_i$ to be $\text{fragment}(\text{target}(\text{minlink}(f_{i-1})))$.

More claims about $s'$:
Section 4.3.1: COM is Equitable for HI

6. \( f_i \) is uniquely defined, for all \( i \geq 1 \). \textbf{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 \( \text{minlink}(f_i) \) is well-defined and non-nil.

7. \( \text{minlink}(f_i) \) 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 \( \text{miedge}(f_i) \neq \text{miedge}(f_{i-1}) \) 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 \( \text{miedge}(f_i) = \text{miedge}(f_{i-1}) \).

Let \( \psi = \text{Merge}(f_i, f_{i-1}) \). Obviously \( \psi \) is enabled in \( s \), and \( (s, \psi) \in \Psi_\varphi \).

(2) Consider a step \((s', \pi, s)\) of \( \text{COM} \), where \( s' \) is reachable and in \( E_\varphi \), \((s', \pi) \notin X_\varphi \), and \( s \in E_\varphi \).

(a) \( v_\varphi(s) \leq v_\varphi(s') \), because there is no action of \( \text{COM} \) that increases the number of fragments; only a Merge action increases the number of fragments with \( \text{minlink} \) equal to \text{nil} or \text{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 \text{rootchanged} \) = false and decreases the number with \( \text{minlink} = \text{nil} \), and ChangeRoot maintains the number of fragments and decreases the number with \text{rootchanged} \) = false.

(c) Suppose \((s', \pi) \notin \Psi_\varphi \), \( \psi \) is enabled in \( s' \), and \((s', \psi) \in \Psi_\varphi \). Then \( \psi \) is still enabled in \( s \), since the only possible values of \( \pi \) are \text{Start}(p), \text{InTree}(l) \) and \text{NotInTree}(l), none of which disables \( \psi \). By definition, \((s, \psi) \in \Psi_\varphi \).

\text{iii) } \varphi \text{ is InTree}((p,q)). \text{ We show COM is progressive for } \varphi \text{ via } M_1; \text{ Lemma 6 implies that COM is equitable for } \varphi \text{ via } M_1.

Let \( \Psi_\varphi \) be the set of all pairs \((s, \psi)\) of reachable states \( s \) of \( \text{COM} \) and actions \( \psi \) of \( \text{COM} \) enabled in \( s \) such that \( \psi \) is either an internal action or is \( \varphi \).

For reachable state \( s \), let \( v_\varphi(s) = v_{\text{Combine}(F,F',\varphi)}(s) \).
Section 4.3.2: GC is Equitable for COM

(1) Let \( s \) be a reachable state of \( COM \) in \( E_\varphi \). We now demonstrate that some action \( \psi \) is enabled in \( s \) with \((s, \psi) \in \mathcal{F}_\varphi\).

If \((p, q) \in F\) for some \( F \) in \( S_1(s) \), then \((p, q) \in\) subtree(fragment\((p)\)) in \( s \). Let \( \psi = \text{InTree}(\langle p, q \rangle) \).

Suppose \((p, q)\) is the minimum-weight external link of some \( F \) in \( S_1(s) \). Then there is more than one fragment. Essentially the same argument as in \( \varphi = \text{Combine}(F, F', e) \) shows that some \( \text{Absorb}(f', g') \), or \( \text{Merge}(f_i, f_{i+1}) \), or \( \text{ChangeRoot}(f') \), or \( \text{ComputeMin}(f') \) is enabled in \( s \).

(2) As in \( \varphi = \text{Combine}(F, F', e) \), after noting that \( \pi \neq \text{InTree}(\langle p, q \rangle) \).

\[ \square \]

4.3.2 GC is Equitable for COM

The main part of the proof is showing that eventually every node is removed from \( \text{testset}(f) \), so that eventually \( \text{ComputeMin}(f) \) 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 \( M_2 \).

Proof: By Corollary 16, \((P'_{COM} \circ S_2) \land P_{GC}\) 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 \( \varphi \) of \( COM \), we must show that \( GC \) is equitable for \( \varphi \) via \( M_2 \).

i) \( \varphi \) is not \( \text{ComputeMin}(f) \) for any \( f \). Since \( \varphi \) is enabled in \( s \) if and only if \( \varphi \) is enabled in \( S_2(s) \), and since \( A_2(s, \varphi) \) includes \( \varphi \), for all \( s \), Lemma 5 shows that \( GC \) is equitable for \( \varphi \) via \( M_2 \).

ii) \( \varphi \) is \( \text{ComputeMin}(f) \). We show \( GC \) is progressive for \( \varphi \) via \( M_2 \); Lemma 6 implies that \( GC \) is equitable for \( \varphi \) via \( M_2 \).

Let \( \mathcal{F}_\varphi \) be the set of all pairs \((s, \pi)\) of reachable states \( s \) of \( GC \) and internal actions \( \pi \) 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 \text{rootchanged} = false;
Section 4.3.2: GC is Equitable for COM

3. the number of fragments with \( \text{minlink} = \text{nil} \); and
4. the sum of the number of nodes in each fragment’s \text{testset}.

(1) Let \( s \) be a reachable state of GC in \( E_\phi \). So ComputeMin\((f)\) is enabled in \( S_2(s) \). We now show that some \( \psi \) is enabled in \( s \) with \((s, \psi) \in \Psi_\phi\).

Let \( G \) be the directed graph defined as follows. There is one vertex of \( G \) for each element of \text{fragments} in \( s \). We now specify the directed edges of \( G \). Let \( v \) and \( w \) be two vertices of \( G \), corresponding to fragments \( f' \) and \( g' \). There is a directed edge from \( v \) to \( w \) in \( G \) if and only if there is a node \( p \) in \text{testset}(f') whose minimum-weight external link is \((p, q), \text{fragment}(q) = g', \text{level}(f') > \text{level}(g')\). We will call fragment \( f' \) a sink if its corresponding vertex in \( G \) is a sink. (It should be obvious that there is at least one sink.)

Case 1: There is a sink \( f' \) such that \text{testset}(f') \( \neq \emptyset \). Let \( \psi = \text{TestNode}(p) \) for some \( p \in \text{testset}(f') \). Since \( f' \) is a sink, \( \psi \) is enabled in \( s \). Obviously \((s, \psi) \in \Psi_\phi\).

Case 2: For all sinks \( f' \), \text{testset}(f') \( = \emptyset \).

Case 2.1: There is a sink \( f' \) such that \text{minlink}(f') = \text{nil}. Let \( \psi = \text{ComputeMin}(f') \). Since ComputeMin\((f)\) is enabled in \( S_2(s) \), there are at least two fragments, so there is an external link of \( f' \). By GC-B, \( \text{accmin}(f') \neq \text{nil} \). Thus \( \psi \) is enabled in \( s \). Obviously \((s, \psi) \in \Psi_\phi\).

Case 2.2: For all sinks \( f' \), \text{minlink}(f') \( \neq \text{nil} \).

Case 2.2.1: There is a sink \( f' \) such that rootchanged\((f') = \text{false}. Let \( \psi = \text{ChangeRoot}(f') \). Since ComputeMin\((f)\) is enabled in \( S_2(s) \), \text{minlink}(f') \( = \text{nil} \). By COM-C then, \text{awake} = \text{true}. Thus \( \psi \) is enabled in \( s \). Obviously \((s, \psi) \in \Psi_\phi\).

Case 2.2.2: For all sinks \( f' \), rootchanged\((f') = \text{true}. By COM-A, the following two cases are exhaustive.

Case 2.2.2.1: There is a sink \( f' \) such that \text{level}(g') > \text{level}(f'), where \( g' = \text{fragment}(\text{target} \text{minlink}(f'))) \). Let \( \psi = \text{Absorb}(g', f') \). Since \( f' \) is a sink, \( \psi \) is enabled in \( s \). Obviously \((s, \psi) \in \Psi_\phi\).

Case 2.2.2.2: For all sinks \( f' \), \text{level}(g') \( = \text{level}(f') \), where \( g' = \text{fragment}(\text{target} \text{minlink}(f'))) \). Let \( m = \min \{\text{level}(f') : f' \text{ is a sink}\} \). Let \( f' \) be a sink with \text{level}(f') \( = m \). and let \( g' = \text{fragment}(\text{target} \textminlink(f'))) \). If \( g' \) is not a sink, then from the vertex in \( G \) corresponding to \( g' \) a sink is reachable (along the directed edges)
Section 4.3.3: TAR is Equitable for GC

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 \( \text{minedge}(f') = \text{minedge}(g') \). Let \( \psi = \text{Merge}(f', g') \). Obviously \( \psi \) is enabled in \( s \), and \( (s, \psi) \in \Psi_\varphi \).

\[ \]

(2) Consider step \( (s', \pi, s) \) of GC, where \( s' \) is reachable and in \( E_\varphi \), \( (s', \pi) \notin X_\varphi \), and \( s \in E_\varphi \).

(a) Obviously the external actions of GC do not change \( \nu_\varphi \). This fact, together with (b) below, shows that \( \nu_\varphi(s) \leq \nu_\varphi(s') \).

(b) Suppose \( (s', \pi) \in \Psi_\varphi \). If \( \pi = \text{TestNode}(p) \), then component 4 of \( \nu_\varphi \) decreases and the rest stay the same. If \( \pi = \text{ComputeMin}(f') \), then component 3 of \( \nu_\varphi \) decreases and the rest stay the same. If \( \pi = \text{ChangeRoot}(f') \), then component 2 of \( \nu_\varphi \) decreases and the rest stay the same. If \( \pi = \text{Merge}(f', g') \) or \( \text{Absorb}(f', g') \), then component 1 of \( \nu_\varphi \) decreases.

(c) Suppose \( (s', \pi) \notin \Psi_\varphi \), \( \psi \) is enabled in \( s' \), and \( (s', \psi) \in \Psi_\varphi \). Since the only choice for \( \pi \) is an external action of GC, obviously \( \psi \) 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 \( p \)'s 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 \( \Psi \) 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 \( \Psi \).

**Lemma 29:** TAR is equitable for GC via \( M_3 \).

**Proof:** By Corollary 18, \( (P_{GC} \circ S_3) \land P_{TAR} \) 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 \( \varphi \) of GC, we must show that TAR is equitable for \( \varphi \) via \( M_3 \).
Section 4.3.3: TAR is Equitable for GC

i) \( \varphi \) is not TestNode(p) for any p, or InTree(l) or NotInTree(l) for any l. Since \( \varphi \) is enabled in \( s \) if and only if \( \varphi \) is enabled in \( S_3(s) \), and since \( A_3(s, \varphi) \) includes \( \varphi \), for all \( s \), Lemma 5 implies that TAR is equitable for \( \varphi \) via \( M_3 \).

ii) \( \varphi \) is TestNode(p). We show TAR is progressive for \( \varphi \) via \( M_3 \); Lemma 6 implies that TAR is equitable for \( \varphi \) via \( M_3 \). In the worst case, we have to wait for the levels to have the correct relationship. This requires a "global" argument.

Let \( \Psi_\varphi \) be the set of all pairs \((s, \pi)\) of reachable states \( s \) of TAR and internal actions \( \pi \) of TAR enabled in \( s \), such that if \( \pi = \text{ReceiveTest}((q, r), l, c) \), then in \( s \) either \( \text{level(fragment(r))} \geq l \), or there is more than one message in \( \text{tarqueue}_r((q, r)) \).

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 \( \text{rootchanged}(f) = \text{false} \) in \( s \),
3. the number of fragments \( f \) with \( \text{minlink}(f) = \text{nil} \) in \( s \),
4. the number of nodes \( q \) such that \( q \in \text{testset(fragment(q))} \) in \( s \),
5. the number of links \( l \) such that either \( \text{lstatus}(l) = \text{unknown} \), or else \( \text{lstatus}(l) = \text{branch} \) 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 \( \text{tarqueue}(l) \) in \( s \),
7. the number of links \( l \) such that no TEST message is in \( \text{tarqueue}(l) \) in \( s \),
8. the number of messages in \( \text{tarqueue}_q((q, r)) \), for all \( (q, r) \in L(G) \), in \( s \),
9. the number of messages in \( \text{tarqueue}_{q_r}((q, r)) \), for all \( (q, r) \in L(G) \), in \( s \),
10. the number of messages in \( \text{tarqueue}_{r}((q, r)) \), for all \( (q, r) \in L(G) \), that are behind a TEST message in \( s \).

(1) Let \( s \) be a reachable state of TAR in \( E_\varphi \). We show that there exists an action \( \psi \) enabled in \( s \) such that \( (s, \psi) \in \Psi_\varphi \).

Let \( l = \min\{\text{level}(f) : f \in \text{fragments}\} \).

Case 1: All fragments \( f \) at level \( l \) have \( \text{rootchanged}(f) = \text{true} \). Then some \( \text{Absorb}(f, g) \) or \( \text{Merge}(f, g) \) is enabled in \( s \), as argued in Lemma 27, Case 2.2.1 for \( \varphi = \text{Combine} \). Let \( \psi \) be one of these enabled actions.

Case 2: level(f) = l and rootchanged(f) \( \neq \text{true} \), for some \( f \in \text{fragments} \).

Claims about \( s \):

1. \( p \in \text{testset(fragment(q))} \), by precondition of \( \varphi \).
Section 4.3.3: TAR is Equitable for GC

2. awake = true, by Claim 1 and GC-C and COM-C.

Case 2.1: minlink(f) ≠ nil. Let ψ = 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) = ∅.

3. Either there is no external link of f, or accmin(f) ≠ nil, by GC-B and assumption for Case 2.2.1.

4. fragment(p) ≠ f, by Claim 1 and assumption for Case 2.2.1.

5. accmin(f) ≠ nil, by Claims 3 and 4.

Let ψ = ComputeMin(f). It is enabled in s by Claim 5 and assumption for Case 2.2.1.

Case 2.2.2: testset(f) ≠ ∅. Let q be some element of testset(f).

Case 2.2.2.1: testlink(q) = nil. Let ψ = SendTest(q). It is enabled in s by assumptions for Case 2.2.2.1.

Case 2.2.2.2: testlink(q) ≠ nil. By TAR-C(a). testlink(q) = (q, r), for some r. There is a protocol message for (q, r), by TAR-C(c). So there is some message at the head of at least one of the six queues comprising tarqueue((q, r)) and tarqueue((r, q)). 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 ⟨q, ¬⟩ or ⟨r, q⟩, and m ∈ M.

Suppose in contradiction that there is no ψ enabled in s such that (s, ψ) ∈ Ψ. That is, by definition of Ψ, the only message in tarqueue((q, r)) (if any) is a TEST(l', c') in tarqueue((q, r)) with l' > level(fragment(r)); and the only message in tarqueue((r, q)) (if any) is a TEST(l'', c'') in tarqueue((r, q)) with l'' > fragment(q)).

Suppose the protocol message for ⟨q, r⟩ is a TEST(l', c') in tarqueue((q, r)), with lstatus((q, r)) ≠ rejected. By TAR-E(b), l' = level(fragment(r)). Since fragment(q) = f, l' = l by choice of f. But l' > level(fragment(r)), by definition of Ψ, which contradicts the definition of l.

Suppose the protocol message for ⟨q, r⟩ is a TEST(l'', c'') in tarqueue((r, q)), with lstatus((r, q)) = rejected. By TAR-E(c), l'' = level(fragment(q)). But by definition of Ψ, l'' > level(fragment(q)).
Section 4.3.3: TAR is Equitable for GC

(2) Let \((s', \pi, s)\) be a step of TAR, where \(s'\) is reachable and is in \(E_\varphi\), \((s', \pi) \notin X_\varphi\), and \(s \in E_\varphi\).

(a) If \((s', \pi) \notin \Psi_\varphi\), then \(\pi\) is either \(\text{InTree}(l)\), \(\text{NotInTree}(l)\), or \(\text{Start}(p)\), or else \(\pi\) is \(\text{ReceiveTest}((q, r), l, c)\) and in \(s, l > \text{level(fragment}(r))\) and there is only one message in \(\text{tarqueue}_r((q, r))\). In all cases, no component of \(v_\varphi\) 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 \(\pi = \text{ChannelSend}(l, m)\). Component 8 of \(v_\varphi\) decreases and components 1 through 7 do not change.

- Suppose \(\pi = \text{ChannelRecv}(l, m)\). Component 9 of \(v_\varphi\) decreases and components 1 through 8 do not change.

- Suppose \(\pi = \text{SendTest}(q)\). Let \((q, r)\) be the minimum-weight link of \(q\) with \(\text{lsstatus}\) unknown in \(s'\). By precondition, \(\text{testlink}(q) = \text{nil}\) in \(s'\). By TAR-D, there is no protocol message for \((q, r)\) in \(s'\), so there is no \text{test} message in \(\text{tarqueue}((q, r))\) in \(s'\). One is added in \(s\). Thus component 7 of \(v_\varphi\) decreases and components 1 through 6 do not change. If there is no link of \(q\) with \(\text{lsstatus}\) unknown, then \(q\) is removed from \(\text{testset(fragment}(q))\). Thus component 4 of \(v_\varphi\) decreases and components 1 through 3 do not change.

- Suppose \(\pi = \text{ReceiveTest}((q, r), l, c)\) and in \(s'\) either \(l \leq \text{level(fragment}(r))\) or there is more than one message in \(\text{tarqueue}_r((q, r))\).

\textbf{Case 1:} \(l \leq \text{level(fragment}(r))\) and either \(c \neq \text{core(fragment}(r))\) or \(\text{testlink}(r) \neq (r, q)\) in \(s'\).

\textbf{Claims about} \(s'\):

1. \(\text{TEST}(l, c)\) message is in \(\text{tarqueue}((q, r))\), by precondition.
2. \(c \neq \text{core(fragment}(r))\) or \(\text{testlink}(r) \neq (r, q)\), by assumption.
3. If \(c \neq \text{core(fragment}(r))\), then \(\text{lsstatus}((q, r))\) is not rejected, by TAR-E(c).
4. If \(\text{testlink}(r) \neq (r, q)\), then there is no protocol message for \((r, q)\), by TAR-D.
5. If \(\text{testlink}(r) \neq (r, q)\), then \(\text{lsstatus}((q, r))\) is not rejected, by Claim 4 and definition.
Section 4.3.3: TAR is Equitable for GC

6. The \texttt{test}(l,c) message in \texttt{tarqueue}((q,r)) is a protocol message for \texttt{(q,r)}, by Claims 2, 3 and 5.
7. \texttt{testlink}(q) = (q,r), by Claim 6 and TAR-D.
8. There is no \texttt{ACCEPT} or \texttt{REJECT} message in \texttt{tarqueue}((r,q)), by Claims 6 and 7 and TAR-C(c).

If \texttt{lstatus}((q,r)) is changed from unknown to rejected, then component 5 of \( v_\varphi \) decreases and components 1 through 4 are unchanged. Otherwise, an \texttt{ACCEPT} or \texttt{REJECT} message is added to \texttt{tarqueue}((r,q)) in \( s \), causing component 6 of \( v_\varphi \) to decrease by Claim 8, while components 1 through 5 stay the same.

---

\textbf{Case 2:} \( l \leq \texttt{level(fragment(r))} \) and \( c = \texttt{core(fragment(r))} \) and \( \texttt{testlink(r)} = (r,q) \) in \( s' \).

\textit{Claims about} \( s' \):

1. \texttt{test}(l,c) is in \texttt{tarqueue}((q,r)), by precondition.
2. \( c = \texttt{core(fragment(r))} \), by assumption.
3. \texttt{testlink(r)} = (r,q), by assumption.

\textit{Case 2.1:} There is no link \texttt{(r,t)}, \( t \neq q \), with \texttt{lstatus} unknown in \( s' \). Then \( q \) is removed from \texttt{testset(fragment(q))} in \( s \), causing component 4 of \( v_\varphi \) to decrease while components 1 through 3 do not change.

\textit{Case 2.2:} There is a link \texttt{(r,t)}, \( t \neq q \), with \texttt{lstatus}(t) = unknown in \( s' \).

4. \texttt{lstatus}(r,q) \neq \texttt{rejected}, by Claim 3 and TAR-K.

By Claim 4, Cases 2.2.1 and 2.2.2 are exhaustive.

\textit{Case 2.2.1:} \texttt{lstatus}(r,q) = unknown in \( s' \). It is changed to rejected in \( s \), causing component 5 of \( v_\varphi \) to decrease and components 1 through 4 to stay the same.

\textit{Case 2.2.2:} \texttt{lstatus}(r,q) = branch.

\textit{Case 2.2.2.1:} The \texttt{test}(l,c) message in \texttt{tarqueue}(q,r) is a protocol message for \texttt{(r,q)}.

5. The \texttt{test}(l,c) message in \texttt{tarqueue}(q,r) is the only protocol message for \texttt{(r,q)}, by TAR-C(c).
Section 4.3.3: TAR is Equitable for GC

Since the only protocol message for \((r, q)\) is removed in \(s\), component 5 of \(v_\varphi\) decreases and components 1 through 4 stay the same.

**Case 2.2.2.2:** The \(\text{TEST}(l, c)\) message in \(\text{tarqueue}(q, r)\) is not a protocol message for \((r, q)\).

6. \(lstatus((q, r)) \neq \text{rejected}\), by assumptions for Case 2.2.2.2.
7. There is a \(\text{TEST}(l', c')\) message in \(\text{tarqueue}(r, q)\) and \(lstatus((r, q)) = \text{unknown}\), by Claims 1, 2, 3, 6 and TAR-P.

But Claim 7 contradicts the assumption for Case 2.2.2.

**Case 3:** \(l > \text{level(fragment}(r)\)) and there is more than one message in \(\text{tarqueue}_r((q, r))\) in \(s'\). All \(\text{TEST}\) messages in \(\text{tarqueue}_r((q, r))\) are protocol messages for the same link, either \((q, r)\) or \((r, q)\). Since by TAR-D and TAR-C(c) there is never more than one protocol message for any link, this \(\text{TEST}(l, c)\) message is the only one. The \(\text{TEST}(l, c)\) message is put at the end of \(\text{tarqueue}_r((q, r))\) in \(s\), decreasing component 10 and not changing components 1 through 9.

- Suppose \(\pi = \text{ReceiveAccept}((q, r))\). Since \(r\) is removed from \(\text{testset(fragment}(r)\))\), component 4 of \(v_\varphi\) decreases while components 1 through 3 stay the same.

- Suppose \(\pi = \text{ReceiveReject}((q, r))\). If there are no more unknown links, then \(r\) is removed from \(\text{testset(fragment}(r)\))\), decreasing component 4 of \(v_\varphi\) and not changing components 1 through 3. Suppose there is another unknown link.

**Claims about \(s'\):**

1. \(\text{REJECT}\) is in \(\text{tarqueue}(q, r)\), by precondition.
2. There is a link \((r, t)\), \(t \neq q\), with \(lstatus((r, t)) = \text{unknown}\), by assumption.
3. \(\text{testlink}(r) = (r, q)\), by Claim 1 and TAR-D.
4. The \(\text{REJECT}\) in \(\text{tarqueue}(q, r)\) is the only protocol message for \((q, r)\), by Claim 3 and TAR-C(c).
5. \(lstatus((r, q)) \neq \text{rejected}\), by Claim 3 and TAR-K.

By Claim 5, \(lstatus((r, q)) \neq \text{rejected}\). If \(lstatus((r, q)) = \text{unknown}\) in \(s'\), it is changed to \(\text{rejected}\) in \(s\). If \(lstatus((r, q)) = \text{branch}\) in \(s'\), then it stays \text{branch} in \(s\), but there are no more protocol messages for \((r, q)\) in \(s\), by Claim 4. Thus component 5 of \(v_\varphi\) decreases while components 1 through 4 stay the same.
Section 4.3.3: TAR is Equitable for GC

- Suppose \( \pi = \text{ComputeMin}(f) \). Component 3 of \( v_\varphi \) decreases and components 1 and 2 are unchanged.

- Suppose \( \pi = \text{ChangeRoot}(f) \). Component 2 of \( v_\varphi \) decreases and component 1 is unchanged.

- Suppose \( \pi = \text{Merge}(f, g) \) or \( \text{Absorb}(f, g) \). Component 1 of \( v_\varphi \) decreases.

(c) Suppose \( (s', \pi) \notin \Psi_\varphi \), \( \psi \) is enabled in \( s' \), and \( (s', \psi) \in \Psi_\varphi \). Then \( \psi \) is still enabled in \( s \) and \( (s, \psi) \in \Psi_\varphi \), since the only possibilities are: \( \pi = \text{InTree}(l) \), \( \text{NotInTree}(l) \), or \( \text{Start}(p) \), or else \( \pi = \text{ReceiveTest}((q, r), l, c) \) and in \( s' \) \( l > \text{level(fragment}(r)) \) and there is only one message in \( \text{tarqueue}_r((q, r)) \).

iii) \( \varphi \) is \( \text{InTree}((p,q)) \). We show TAR is progressive for \( \varphi \) via \( M_3 \); Lemma 6 implies that TAR is equitable for \( \varphi \) via \( M_3 \). We simply show that if \( (p, q) = \text{minlink}(f) \), but \( \text{Istatus}((p, q)) \) is not yet branch, then eventually \( \text{ChangeRoot}(f) \) will occur.

Let \( \Psi_\varphi \) be all pairs \( (s, \psi) \) of reachable states \( s \) and actions \( \psi \) enabled in \( s \) such that one of the following is true: (Let \( f = \text{fragment}(p) \) in \( s \).)

- \( \psi = \text{InTree}((p,q)) \), or

- \( (p,q) = \text{minlink}(f) \) in \( s \), and \( \psi = \text{ChangeRoot}(f) \).

For reachable state \( s \), let \( v_\varphi(s) \) be 1 if \( (p, q) = \text{minlink}(f) \) and \( \text{ChangeRoot}(f) \) is enabled in \( s \), and 0 otherwise.

1. \( \text{awake} = \text{true} \), by precondition of \( \varphi \).
2. \( (p, q) \in \text{subtree}(f) \) or \( (p, q) = \text{minlink}(f) \), by precondition of \( \varphi \).
3. \( \text{answered}((p, q)) = \text{false} \), by precondition of \( \varphi \).
4. \( \text{Istatus}((p, q)) \neq \text{rejected} \), by Claim 2 and TAR-B.

By Claim 4, the following two cases are exhaustive.
Section 4.3.3: TAR is Equitable for GC

Case 1: \( \text{lsatus}(p, q) = \text{branch} \). Let \( \psi = \text{InTree}(p, q) \). It is enabled in \( s \) by Claims 1 and 3 and assumption for this case, and \( (s, \psi) \in \Psi_\varphi \).

Case 2: \( \text{lsatus}(p, q) = \text{unknown} \).

5. \( \text{minlink}(f) = (p, q) \), by Claim 2 and TAR-A(a).
6. \( \text{rootchanged}(f) = \text{false} \), by Claim 5 and TAR-H.

Let \( \psi = \text{ChangeRoot}(f) \). It is enabled in \( s \) by Claims 1, 5 and 6, and \( (s, \psi) \in \Psi_\varphi \).

(2) Let \( (s', \pi, s) \) be a step of TAR, where \( s' \) is reachable and is in \( E_\varphi \), \( (s', \pi) \not\in X_\varphi \), and \( s \in E_\varphi \).

(a) Suppose \( (s', \pi) \not\in \Psi_\varphi \). We show that no possibility for \( \pi \) can affect whether or not \( \text{ChangeRoot}(f) \) 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: \( \text{ChangeRoot}(f) \) is enabled in \( s' \). No action sets \textit{awake} to false. No action (other than \( \text{ChangeRoot}(f) \)) sets \textit{rootchanged}(f) to false. No action sets \( \text{minlink}(f) \) to nil. \( f \) remains in fragments because \( \pi \) is not \( \text{Absorb}(g, f) \), \( \text{Merge}(f, g) \) or \( \text{Merge}(g, f) \), for any \( g \), since \( \text{rootchanged}(f) = \text{false} \).

Case 2: \( \text{rootchanged}(f) \) is not enabled in \( s' \). By precondition of \( \varphi \), \textit{awake} is true in \( s' \). If \( \text{rootchanged}(f) = \text{true} \) in \( s' \), then the same is true in \( s \), because the only action that sets it to false is the \( \text{Merge} \) that created \( f \). If \( \text{minlink}(f) = \text{nil} \) in \( s' \), then \( (p, q) \not= \text{minlink}(f) \), so even if \( \text{minlink}(f) \) becomes nonnil (by \( \text{ComputeMin}(f) \)), \( v_\varphi \) remains 0.

(b) Suppose \( (s', \pi) \in \Psi_\varphi \). Since \( (s', \pi) \not\in X_\varphi \), \( \pi \not\in \text{InTree}(p, q) \). Thus \( \text{minlink}(f) = (p, q) \) in \( s' \) and \( \pi = \text{ChangeRoot}(f) \). Obviously \( v_\varphi \) goes from 1 to 0.

(c) Suppose \( (s', \pi) \not\in \Psi_\varphi \). \( \psi \) is enabled in \( s' \), and \( (s', \psi) \in \Psi_\varphi \). The same argument as in (2a), Case 1, applies.
iv) $\varphi$ is NotInTree((p,q)). We show that TAR is progressive for $\varphi$ via $M_3$; Lemma 6 implies that TAR is equitable for $\varphi$ via $M_3$. The goal is to show that if $q \in \text{nodes(fragment}(p))$ and $(p,q) \notin \text{subtree(fragment}(p))$, then eventually $lstatus((p,q)) = \text{rejected}$. 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 $\Psi_{\varphi}$ be $\Psi_{\text{TestNode}(p)} \cup \{(s, \text{NotInTree}(p,q)) : s \text{ reachable, NotInTree}(p,q) \text{ enabled in } s\}$. 

Let $v_{\varphi}(s) = v_{\text{TestNode}(p)}(s)$ for all reachable states $s$.

Let $v_{\varphi}$ be the same as for TestNode(p).

(1) Let $s$ be a reachable state of TAR in $E_{\varphi}$. We show that there exists an action $\psi$ enabled in $s$ such that $(s,\psi) \in \Psi_{\varphi}$.

$lstatus((p,q)) \neq \text{branch}$, by TAR-A(a). If $lstatus((p,q)) = \text{rejected}$, then let $\psi = \text{NotInTree}(p,q)$.

Suppose $lstatus((p,q)) = \text{unknown}$ 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 $awake = \text{true}$ by the precondition of $\varphi$.

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 $f = \text{fragment}(p)$, and $p \notin \text{testset}(f)$ (since we assume $\text{testset}(f) = \emptyset$). But since we also assume $lstatus((p,q)) = \text{unknown}$, TAR-I gives as contradiction. Thus, there is an external link of $f$, and by GC-B, $\text{accmin}(f') \neq \text{nil}$.

(2) Like TestNode(p), after noting that $\pi$ cannot be NotInTree((p,q)).

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 $M_4$. 177
Section 4.3.4: DC is Progressive for an Action of GC

Proof: By Corollary 20, \((P_{GC} \circ S_4) \land P_{DC}\) is true in every reachable state of DC. Thus, in the sequel we will use the HI, COM, GC and DC predicates.

Let \(\Psi_\psi\) be the set of all pairs \((s, \psi)\) of reachable states \(s\) of DC and actions \(\psi\) of DC such that in \(s\), a \(\text{REPORT}(w)\) is in some \(\text{dcqueue}((q, p))\) and either \(q\) is a child of \(p\), or else \(dcstatus(p) = \text{unfind}\) and \(p = \text{mw-root}(f)\); and \(\psi \in \{\text{ChannelSend}((q, p), \text{REPORT}(w)), \text{ChannelRecv}(q, p, \text{REPORT}(w)), \text{ReceiveReport}((q, p), w)\}\).

For reachable state \(s\), let \(v_\psi(s)\) be a quadruple with the following components:

1. The number of nodes \(p \in \text{nodes}(f)\) with \(dcstatus(p) = \text{find}\).
2. The number of \(\text{REPORT}\) messages in \(\text{dcqueue}_q((q, p))\), for all \((p, q) \in \text{subtree}(f)\) such that either \(q\) is a child of \(p\) or else \(p = \text{mw-root}(f)\) and \(dcstatus(p) = \text{unfind}\).
3. The number of \(\text{REPORT}\) messages in \(\text{dcqueue}_q((q, p))\) for all \((p, q) \in \text{subtree}(f)\) such that either \(q\) is a child of \(p\) or else \(p = \text{mw-root}(f)\) and \(dcstatus(p) = \text{unfind}\).
4. The number of \(\text{REPORT}\) messages in \(\text{dcqueue}_p((q, p))\) for all \((p, q) \in \text{subtree}(f)\) such that either \(q\) is a child of \(p\) or else \(p = \text{mw-root}(f)\) and \(dcstatus(p) = \text{unfind}\).

(1) Let \(s\) be a reachable state of DC in \(E_\phi\). We show that there exists an action \(\psi\) enabled in \(s\) such that \((s, \psi) \in \Psi_\psi\).

Claims about \(s\):

1. \(\text{minlink}(f) = \text{nil}\), by precondition.
2. \(\text{accmin}(f) \neq \text{nil}\), by precondition.
3. \(\text{testset}(f) = \emptyset\), by precondition.
4. There is an external link of \(f\), by Claim 2 and GC-A.
5. No \(\text{FIND}\) message is in \(\text{subtree}(f)\), by Claim 3 and DC-D(c).
6. If \(dcstatus(p) = \text{find}\), then a \(\text{REPORT}\) message is in \(\text{subtree}(p)\) headed toward \(p\), for any \(p \in \text{nodes}(f)\), by Claim 3 and DC-I(b).

Suppose a \(\text{REPORT}(w)\) is in some \(\text{dcqueue}((q, p))\) and \(q\) is a child of \(p\). By DC-B(a), \(\text{inbranch}(p) \neq \langle p, q \rangle\). Obviously, \((p, q) \neq \text{core}(f)\), so by DC-A(g), \(dcstatus(p) = \text{find}\). By Claim 5 and DC-O, the \(\text{REPORT}(w)\) is the only message in \(\text{dcqueue}((q, p))\). If it is in \(\text{dcqueue}_q((q, p))\), let \(\psi = \text{ChannelSend}((q, p), \text{REPORT}(w))\); if it is in \(\text{dcqueue}_q((q, p))\), let \(\psi = \text{ChannelRecv}(q, p, \text{REPORT}(w))\); if it is in \(\text{dcqueue}_p((q, p))\), let \(\psi = \text{ReceiveReport}(w)\). Obviously, \(\psi\) is enabled in \(s\), and \((s, \psi) \in \Psi_\psi\).

Suppose no \(\text{REPORT}\) is in any \(\text{dcqueue}((q, p))\) with \(q\) a child of \(p\). By Claim 6, \(dcstatus(p) = \text{unfind}\) for all \(p \in \text{nodes}(f)\). Then by Claims 1, 4 and 5, a \(\text{REPORT}(w)\) is
Section 4.3.4: DC is Progressive for an Action of GC

in $dcqueue((q,p))$, where $(p,q) = core(f)$ and $p = mw-root(f)$. By Claim 5 and DC-0, the $\text{REPORT}(w)$ is the only message in $dcqueue((q,p))$. If it is in $dcqueue_q((q,p))$, let $\psi = \text{ChannelSend}((q,p), \text{REPORT}(w))$; if it is in $dcqueue_{q,p}((q,p))$, let $\psi = \text{ChannelRecv}((q,p), \text{REPORT}(w))$; if it is in $dcqueue_p((q,p))$, let $\psi = \text{ReceiveReport}(w)$. Obviously, $\psi$ is enabled in $s$, and $(s, \psi) \in \Psi_\varphi$.

(2) Let $(s', \pi, s)$ be a step of $DC$, where $s'$ is reachable and is in $E_\varphi$, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$. We note the following claims about $s'$.

1. $\text{testset}(f) = \emptyset$, by precondition.
2. $\text{minlink}(f) = \text{nil}$, by precondition.
3. No $\text{FIND}$ is in $\text{subtree}(f)$, 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 $(s', \pi) \notin \Psi_\varphi$.

$\text{TestNode}(p)$ is not enabled, for $p \in \text{nodes}(f)$, by Claim 1. $\text{ChangeRoot}(f)$, $\text{Merge}(f,g)$, $\text{Merge}(g,f)$, and $\text{Absorb}(g,f)$ are not enabled, for $g \in \text{fragments}$, by Claim 2. $\text{ReceiveFind}((p,q))$, $\text{AfterMerge}(p,q)$, $\text{ChannelSend}(p,q, \text{FIND})$, and $\text{ChannelRecv}(p,q, \text{FIND})$ are not enabled, for $p \in \text{nodes}(f)$, by Claim 3. Thus $\pi$ is none of the above actions.

If $\pi = \text{ChannelSend}((q,p), \text{REPORT}(w))$ or $\text{ChannelRecv}((q,p), \text{REPORT}(w))$, for $(q,p) \in \text{subtree}(f)$, then $v_\varphi$ is unchanged, since $(s', \pi) \notin \Psi_\varphi$.

Suppose $\pi = \text{ReceiveReport}((q,p), w)$.

Case 1: $p$ is a child of $q$. By DC-A(a), $\text{inbranch}(p) = (p,q)$. By DC-B(b), $\text{dcstatus}(p) = \text{unfind}$. So the only change is the removal of the message. Since $p$ is a child of $q$, $p \neq mw-root(f)$, so $v_\varphi$ is unchanged.

Case 2: $(p,q) = core(f)$ and $p \neq mw-root(f)$. By DC-A(a), $\text{inbranch}(p) = (p,q)$. The only effect is that either the message is requeued (if $\text{dcstatus}(p) = \text{find}$), or the message is removed (if $\text{dcstatus}(p) = \text{unfind}$); in both cases, $v_\varphi$ is unchanged.

Case 3: $(p,q) = core(f)$, $p = mw-root(f)$, and $\text{dcstatus}(p) = \text{find}$. The only effect is that the message is requeued, so $v_\varphi$ is unchanged.

Suppose $\pi = \text{Merge}(g,h)$. By precondition, $\text{minlink}(g) = \text{minlink}(h) \neq \text{nil}$ in $s'$. So $f \neq g$ and $f \neq h$. Obviously $v_\varphi$ is unchanged.

179
Section 4.3.5: CON is Progressive for Some Actions of COM

Suppose \( \pi = \text{Absorb}(g, h) \). By precondition, \( \text{minlink}(h) \neq \text{nil} \) in \( s' \), so \( f \neq h \) by Claim 2. If \( f \neq g \), then obviously \( v_\varphi \) is unchanged. Suppose \( f = g \). As in the proof of condition (3a) in Lemma 19 for viii) \( \pi = \text{Absorb} \), Case 2, no report message is headed toward \( \text{minnode}(h) \) and \( \text{dcstatus}(r) = \text{unfind} \) for all \( r \in \text{nodes}(h) \) in \( s' \). Thus \( v_\varphi \) does not change.

The remaining actions (not mentioned above) obviously do not affect \( v_\varphi \).

(b) Suppose \((s', \pi) \in \Psi_\varphi \). We show \( v_\varphi(s) < v_\varphi(s') \). If \( \psi = \text{ChannelSend}(l, m) \), component 2 of \( v_\varphi \) decreases and component 1 is unchanged. If \( \psi = \text{ChannelRecv}(l, in) \), component 3 of \( v_\varphi \) decreases and components 1 and 2 are unchanged.

Suppose \( \psi = \text{ReceiveReport}((q, p), w) \).

Case 1: \( q \) is a child of \( p \). By DC-B(a), \( \text{inbranch}(p) \neq (p, q) \). By DC-A(g), \( \text{dcstatus}(p) = \text{find} \). If \( \text{findcount}(p) = 1 \) in \( s' \), then component 1 of \( v_\varphi \) decreases. Otherwise, component 4 decreases and components 1 through 3 are unchanged.

Case 2: \( q \) is not a child of \( p \). \( p = \text{mw-root}(f) \), and \( \text{dcstatus}(p) = \text{unfind} \). So \((p, q) = \text{core}(f) \). By DC-P, \( \text{core}(f) > \text{bestwt}(p) \). But this contradicts \((s', \pi) \notin X_\varphi \).

(c) Suppose \((s', \pi) \notin \Psi_\varphi \). \( \psi \) is enabled in \( s' \), and \((s', \psi) \in \Psi_\varphi \). We show that \( \psi \) is still enabled in \( s \) and \((s, \psi) \in \Psi_\varphi \). Since the queues are FIFO, there is no way to disable \( \psi \).

It remains to show that \((s, \psi) \) is still in \( \Psi_\varphi \).

One possible way \((s, \psi) \) could no longer be in \( \Psi_\varphi \) is if the position of \( \text{mw-root}(f) \) changes, i.e., if \( \pi = \text{Merge}(f, g) \), \( \text{Merge}(g, f) \), \( \text{Absorb}(f, g) \), or \( \text{Absorb}(g, f) \), for some fragment \( g \). But by Claim 2, \( \text{minlink}(f) = \text{nil} \). Thus \( \pi \) cannot be \( \text{Merge}(f, g) \), \( \text{Merge}(g, f) \), or \( \text{Absorb}(g, f) \). Suppose \( \pi = \text{Absorb}(f, g) \). Let \( \text{core}(f) = (p, q) \), \( p = \text{mw-root}(f) \), and \( q \) be the endpoint of \( \text{core}(f) \) closest to \( \text{target}(\text{minlink}(g)) \) in \( s' \). The minimum-weight external link of \( f \) has smaller weight than \( \text{minlink}(g) \), which by COM-A is the minimum-weight external link of \( g \). Thus \( \text{mw-root}(f) \) does not change after \( \text{Absorb}(f, g) \).

Another way is if the position of \( \text{core}(f) \) changes. This only happens if \( \pi \) is \( \text{Merge}(f, g) \), \( \text{Merge}(g, f) \) or \( \text{Absorb}(g, f) \), which we showed is impossible.

The third way is if \( \text{dcstatus}(p) \) changes from \( \text{unfind} \) to \( \text{find} \), where \( p = \text{mw-root}(f) \). This only happens if \( \pi = \text{ReceiveFind}((q, p)) \) for some \( q \). But by Claim 3, no \( \text{FIND} \) is in \( \text{subtree}(f) \), and by DC-D(\( \sim \)), no \( \text{FIND} \) can be in an external link. \( \Box \)
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 $M_6$.

Proof: By Corollary 24. $(P_{COM} \circ S_6) \land P_{CON}$ is true in every reachable state of CON. Thus, in the sequel we will use the HI, COM, and CON predicates.

1) $\varphi$ is Merge(f,g). Let $(p,q) = \text{minedge}(f)$. Let $\Psi_\varphi$ be the set of all pairs $(s,\psi)$ of reachable states $s$ of CON and actions $\psi$ of CON enabled in $s$, such that $\psi \in \{\text{ChannelSend}((q,p),\text{CONNECT}(l)), \text{ChannelRecv}((q,p),\text{CONNECT}(l)), \text{Merge}(f,g)\}$.

For reachable state $s$ of CON, let $v_\varphi(s) = (x,y)$, where $x$ is the number of messages in $\text{cqueue}_q((q,p))$ in $s$, and $y$ is the number of messages in $\text{cqueue}_{qf}((q,p))$ in $s$.

1) Suppose $s$ is a reachable state of CON in $E_\varphi$. We show that there is a $\psi$ enabled in $s$ such that $(s,\psi) \in \Psi_\varphi$.

Claims about $s$:

1. $f \neq g$, by precondition.
2. $\text{minedge}(f) = \text{minedge}(g) = (p,q)$, by precondition.
3. $\text{rootchanged}(f) = \text{true}$, by precondition.
4. $\text{rootchanged}(g) = \text{true}$, by precondition.
5. A CONNECT(l) message is in $\text{cqueue}(k)$, for some external link $k$ of $f$, by Claim 3.
6. A CONNECT(l) message is in $\text{cqueue}((p,q))$, by Claims 2, 5 and CON-D.
7. A CONNECT(m) message is in $\text{cqueue}(k)$, for some external link $k$ of $g$, by Claim 4.
8. A CONNECT(m) message is in $\text{cqueue}((q,p))$, by Claims 2, 6 and CON-D.
9. $l = \text{level}(f)$, by Claim 5 and CON-D.
10. $m = \text{level}(g)$, by Claim 7 and CON-D.
11. $\text{level}(f) \leq \text{level}(g)$, by Claim 2 and COM-A.
12. $\text{level}(g) \leq \text{level}(f)$, by Claim 2 and COM-A.
Section 4.3.5: \( CON \) is Progressive for Some Actions of \( COM \)

13. \( \text{level}(f) = \text{level}(g) \), by Claims 11 and 12.
14. \( l = m \), by Claims 9, 10 and 13.
15. No \text{CHANGEROOT} message is in \text{cqueue}((q,p)), by Claim 1 and \( CON-C \).
16. Exactly one \text{CONNECT} message is in \text{cqueue}((q,p)), by Claims 7, 8 and \( CON-D \).

If \text{CONNECT}(l) is in \text{cqueue}_p((q,p)), then let \( \psi = \text{ChannelSend}(q,p) \), \text{CONNECT}(l) \). If \text{CONNECT}(l) is in \text{cqueue}_q((q,p)), then let \( \psi = \text{ChannelRecv}(q,p) \), \text{CONNECT}(l) \). If \text{CONNECT}(l) is in \text{cqueue}_p((q,p)), then let \( \psi = \text{Merge}(f,g) \). It is easy to see in all cases that \( \psi \) is enabled in \( s \) and \( (s,\psi) \in \Psi_\varphi \).

(2) Suppose \( (s',\pi,s) \) is a step of \( CON \), \( s' \) is reachable and in \( E_\varphi, (s',\pi) \not\in X_\varphi \), and \( s \in E_\varphi \).

(a) The only actions that can increase \( v_\varphi \) are \( \text{ComputeMin}(g) \), and \( \text{ChangeRoot}(g) \). (Even though \( \text{ChannelSend}(q,p),m \) would increase \( y \), it would simultaneously decrease \( x \).) By Claim 2, \( \text{ComputeMin}(g) \) is not enabled in \( s' \). By Claim 4, \( \text{ChangeRoot}(g) \) is not enabled in \( s' \).

(b) Suppose \( (s',\pi) \in \Psi_\varphi \). Since \( (s',\pi) \not\in X_\varphi, \pi \neq \text{Merge}(f,g) \). Obviously, the other two choices for \( \psi \) decrease \( v_\varphi \).

(c) Suppose \( (s',\pi) \not\in \Psi_\varphi, \psi \) is enabled in \( s' \) and \( (s',\psi) \in \Psi_\varphi \). We show \( \psi \) is enabled in \( s \) and \( (s,\psi) \in \Psi_\varphi \). If \( \psi = \text{ChannelSend} \) or \( \text{ChannelRecv} \), then it can only be disabled by occurring. If \( \psi = \text{Merge}(f,g) \), then since \( s \in E_\varphi \), \( \psi \) is still enabled in \( s \) (by the argument in part (1)). In all cases, \( (s,\psi) \in \Psi_\varphi \).

ii) \( \varphi \) is Absorb\((f,g)\). Let \( (q,p) = \text{minlink}(g) \). Let \( \Psi_\varphi \) be the set of all pairs \( (s,\psi) \) of reachable states \( s \) of \( CON \) and actions \( \psi \) of \( CON \) enabled in \( s \) such that \( \psi \in \{ \text{ChannelSend}(q,p), \text{CONNECT}(l) \}, \text{ChannelRecv}(q,p), \text{CONNECT}(l) \} \). \( \text{Absorb}(f,g) \).\}

For reachable state \( s \) of \( CON \), let \( v_\varphi(s) = (x,y) \), where \( x \) is the number of messages in \text{cqueue}_q((q,p)) in \( s \), and \( y \) is the number of messages in \text{cqueue}_p((q,p)) in \( s \).

(1) Suppose \( s \) is a reachable state of \( CON \) in \( E_\varphi \). We show that there is a \( \psi \) enabled in \( s \) such that \( (s,\psi) \in \Psi_\varphi \).

Claims about \( s \):

1. \( \text{level}(g) < \text{level}(f) \), by precondition.
Section 4.3.5: CON is Progressive for Some Actions of COM

2. \((q, p) = \text{minlink}(g)\), by assumption.

3. \(f = \text{fragment}(p)\), by precondition.

4. \(\text{rootchanged}(g) = \text{true}\), by precondition.

5. A \(\text{CONNECT}(l)\) message is in \(c\text{queue}(k)\), where \(k\) is an external link of \(g\), by Claim 4.

6. A \(\text{CONNECT}(l)\) message is in \(c\text{queue}((q, p))\), by Claims 2, 5 and CON-D.

7. No \(\text{CHANGEROOT}\) message is in \(c\text{queue}((q, p))\), by Claims 5 and 6 and CON-C.

If \(\text{CONNECT}(l)\) is in \(c\text{queue}_q((q, p))\), then let \(\psi = \text{ChannelSend}((q, p), \text{CONNECT}(l))\). If \(\text{CONNECT}(l)\) is in \(c\text{queue}_p((q, p))\), then let \(\psi = \text{ChannelRecv}((q, p), \text{CONNECT}(l))\). If \(\text{CONNECT}(l)\) is in \(c\text{queue}_p((q, p))\), then let \(\psi = \text{Absorb}(f, g)\). In all cases, it is easy to see that \(\psi\) is enabled in \(s\) and \((s, \psi) \in \Psi_\varphi\).

2. Suppose \((s', \pi, s)\) is a step of \(\text{CON}\), \(s'\) is reachable and in \(E_\varphi\), \((s', \pi) \notin X_\varphi\), and \(s \in E_\varphi\).

(a) The only actions that can increase \(v_\varphi\) are \(\text{ComputeMin}(g)\), and \(\text{ChangeRoot}(g)\). (Even though \(\text{ChannelSend}((q, p), m)\) would increase \(y\), it would simultaneously decrease \(x\).) By Claim 2, \(\text{ComputeMin}(g)\) is not enabled in \(s'\). By Claim 4, \(\text{ChangeRoot}(g)\) is not enabled in \(s'\).

(b) Suppose \((s', \pi) \in \Psi_\varphi\). Since \((s', \pi) \notin X_\varphi\), \(\pi \neq \text{Absorb}(f, g)\). Obviously, the other two choices for \(\psi\) decrease \(v_\varphi\).

(c) Suppose \((s', \pi) \notin \Psi_\varphi\), \(\psi\) is enabled in \(s'\) and \((s', \psi) \in \Psi_\varphi\). We show \(\psi\) is enabled in \(s\) and \((s, \psi) \in \Psi_\varphi\). If \(\psi = \text{ChannelSend}\) or \(\text{ChannelRecv}\), then it can only be disabled by occurring. If \(\psi = \text{Absorb}(f, g)\), then since \(s \in E_\varphi\), \(\psi\) is still enabled in \(s\) (by the argument in part (1)). In all cases, \((s, \psi) \in \Psi_\varphi\) by definition.

iii) \(\varphi\) is \(\text{ChangeRoot}(f)\). Let \(\Psi_\varphi\) be the set of all pairs \((s, \psi)\) of reachable states \(s\) of \(\text{CON}\) and actions \(\psi\) of \(\text{CON}\) enabled in \(s\), such that \(\psi \in \{\text{ReceiveChangeRoot}((q, p)), \text{ChannelSend}((q, p), \text{CHANGEROOT}), \text{ChannelRecv}((q, p), \text{CHANGEROOT}) : p \in \text{nodes}(f)\} \cup \{\text{ChangeRoot}(f)\}\).

For reachable state \(s\) of \(\text{CON}\), let \(v_\varphi(s)\) be a triple defined as follows. If there is no \(\text{CHANGEROOT}\) message in \(\text{subtree}(f)\) in \(s\), then \(v_\varphi(s)\) is \((0, 0, 0)\). Suppose, in \(s\), there is a \(\text{CHANGEROOT}\) message in \(c\text{queue}((q, p))\), where \(p \in \text{nodes}(f)\). Then \(v_\varphi(s)\) is:

1. the number of nodes in the path in \(\text{subtree}(f)\) from \(p\) to \(\text{minnode}(f)\) in \(s\) (counting the endpoints \(p\) and \(\text{minnode}(f)\));
Section 4.3.5: CON is Progressive for Some Actions of COM

2. the number of CHANGEROOT messages in \( cqueue_r((r,t)) \), for all \( t \in nodes(f) \) in \( s \); and
3. the number of CHANGEROOT messages in \( cqueue_l((r,t)) \), 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 \( E_\varphi \), then there is a \( \psi \) enabled in \( s \) such that \( (s, \psi) \in \Psi_\varphi \).

**Claims about \( s \):**

1. \( \text{rootchanged}(f) = \text{false} \), by precondition of \( \varphi \).
2. \( \text{minlink}(f) \neq \text{nil} \), by precondition of \( \varphi \).

If \( |nodes(f)| = 1 \) (i.e., \( subtree(f) = \{p\} \), for some \( p \)), then let \( \psi = \text{ChangeRoot}(f) \). Obviously, \( \psi \) is enabled in \( s \) and \( (s, \psi) \in \Psi_\varphi \). Now suppose \( |nodes(f)| > 1 \).

3. \( \text{minnode}(f) \neq \text{root}(f) \), by Claims 1 and 2 and CON-B.
4. Exactly one CHANGEROOT message is in \( cqueue((q,p)) \), for some \( (p,q) \in subtree(f) \), by Claims 1 and 2 and CON-B.
5. \( (q,p) \neq \text{core}(f) \), by Claim 4 and CON-C.
6. No CONNECT message is in \( cqueue((q,p)) \), by Claim 5 and CON-E.

If the CHANGEROOT message is in \( cqueue_q((q,p)) \), then let \( \psi = \text{ChannelSend}((q,p), \text{CHANGEROOT}) \). If the CHANGEROOT message is in \( cqueue_p((q,p)) \), then let \( \psi = \text{ChannelRecv}((q,p), \text{CHANGEROOT}) \). If the CHANGEROOT message is in \( cqueue_p((q,p)) \), then let \( \psi = \text{ReceiveChangeRoot}((q,p)) \). In all three cases, \( \psi \) is enabled in \( s \) because of Claims 4 and 6. By definition, \( (s, \psi) \in \Psi_\varphi \).

(2) Suppose \( (s', \pi, s) \) is a step of \( CON \) such that \( s' \) is reachable and in \( E_\varphi \). \( (s', \pi) \notin X_\varphi \), 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 \( \text{minlink}(f) \neq \text{nil} \) in \( s' \), \( \pi \neq \text{ComputeMin}(f) \). Since \( \text{rootchanged}(f) = \text{false} \) in \( s' \), \( \pi \neq \text{Merge}(f,g), \text{Merge}(g,f) \), or \( \text{Absorb}(g,f) \) for any \( g \).
Section 4.3.6: GHS is Equitable for TAR

Suppose $\pi = \text{Absorb}(f, g)$. First we show that $\text{minnode}(f)$ is unchanged. By COM-A, $\text{level}(h) \geq \text{level}(f)$, where $h = \text{fragment}(\text{target}(\text{minlink}(f)))$; by precondition of $\text{Absorb}(f, g)$, $h \neq g$, and thus $\text{wt}(\text{minlink}(f)) < \text{wt}(\text{minlink}(g))$. Also by COM-A, $\text{minlink}(g)$ is the minimum-weight external link of $g$. Thus $\text{minlink}(f)$ does not change. Second, we show that no changeroot message is in $\text{subtree}(g)$. By precondition of $\text{Absorb}(f, g)$, $\text{rootchanged}(g) = \text{true}$. Then by CON-C, no changeroot message is in $\text{subtree}(g)$.

No other value of $\pi$, such that $(s', \pi) \not\in \Psi_\varphi$, affects $v_\varphi$.

(b) Suppose $(s', \pi) \in \Psi_\varphi$. We show $v_\varphi(s) < v_\varphi(s')$.

If $\pi = \text{ChannelSend}((q, p), \text{CHANGEROOT})$, then the second component of $v_\varphi$ decreases while the first remains the same. If $\pi = \text{ChannelRecv}((q, p), \text{CHANGEROOT})$, then the third component of $v_\varphi$ decreases while the first two remain the same.

Suppose $\pi = \text{ReceiveChangeRoot}((q, p))$. By CON-C and CON-B there is exactly one changeroot message in $\text{subtree}(f)$. Since $(s, \pi) \not\in X_\varphi$, $p \neq \text{minnode}(f)$. Thus, the first component of $v_\varphi(s')$ is at least 1. The first component of $v_\varphi$ decreases by 1 in $s$, by definition of $\text{tominlink}(p)$. Thus $v_\varphi(s) < v_\varphi(s')$.

(c) Suppose $(s', \pi) \not\in \Psi_\varphi$, $\psi$ is enabled in $s'$, and $(s', \psi) \in \Psi_\varphi$. We show $\psi$ is enabled in $s$, and $(s, \psi) \in \Psi_\varphi$.

Suppose $\psi = \text{ChangeRoot}(f)$.

Claims about $s'$:

1. $\text{rootchanged}(f) = \text{false}$, by precondition of $\psi$.
2. $\text{minlink}(f) \neq \text{nil}$, by precondition of $\psi$.
3. $\text{subtree}(f) = \{p\}$, by precondition of $\psi$.
4. No changeroot message is in $\text{queue}((q, p))$ for any $q$, by Claim 3 and CON-C.
5. $\text{ComputeMin}(f)$ is not enabled, by Claim 2.
6. $\text{Merge}(f, g)$, $\text{Merge}(g, f)$, and $\text{Absorb}(g, f)$ are not enabled for any $g$, by Claim 1.
7. $\text{ReceiveChangeRoot}((q, p))$ is not enabled for any $q$, by Claim 4.

By Claims 5. 6 and 7, $\pi$ is no action that can disable $\psi$; hence, $\psi$ is enabled in $s$. By definition, $(s, \psi) \in \Psi_\varphi$. 

185
Suppose \( \psi = \text{ReceiveChangeRoot}(q,p), \text{ChannelSend}(q,p), \text{ChannelRecv}(q,p) \). The only action that can disable \( \psi \) is \( \psi \) itself. Thus, \( \psi \) 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 \( \text{SendTest}(p) \), and for \( \text{ChangeRoot}(f) \) when \( \text{subtree}(f) \) is a singleton node. For \( \text{SendTest}(p) \), we show that an \( \text{INITIATE}-\text{find} \) message eventually reaches \( p \). The big effort is for the \( \text{ChangeRoot}(f) \). We must show that eventually every node will be awakened, either by a \( \text{Start} \) action, or by the receipt of a \( \text{CONNECT} \) or \( \text{TEST} \) message. This requires a global argument about the entire graph. This is another place in which the state component of \( \Psi \) 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 \( \mathcal{M}_{\text{TAR}} \).

**Proof:** We show that GHS is equitable for each locally-controlled action \( \varphi \) of TAR via \( \mathcal{M}_{\text{TAR}} \). First, a point of notation: let \( \text{Receive}(q,p,m) \) be a synonym for \( \text{ReceiveConnect}(q,p,l) \) if \( m = \text{CONNECT}(l) \), a synonym for \( \text{ReceiveInitiate}(q,p,l,c,st) \) if \( m = \text{INITIATE}(l,c,st) \), etc.

By Corollary 26, \( P'_{\text{GHS}} \) 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) \( \varphi \) is \( \text{InTree}(l) \) or \( \text{NotInTree}(l) \). By Lemma 5, we are done.

ii) \( \varphi \) is \( \text{ChannelSend}(q,p,m) \). We show that GHS is progressive for \( \varphi \) via \( \mathcal{M}_{\text{TAR}} \). Lemma 6 gives the result.

Let \( \Psi_\varphi \) be the set of all pairs \((s, \psi)\) of reachable states \( s \) of GHS and actions \( \psi \) of GHS enabled in \( s \) such that \( m' \) is the message at the head of \( \text{queue}_q((q,p)) \) in \( s \), and \( \psi = \text{ChannelSend}(q,p,m') \).

For reachable state \( s \), let \( v_\varphi(s) \) be the number of messages in \( \text{queue}_q((q,p)) \) ahead of the message at the head of \( \text{tarqueue}_q((q,p)) \).

Verifying the progressive conditions is straightforward.

iii) \( \varphi \) is \( \text{ChannelRecv}(q,p,m) \). We show that GHS is progressive for \( \varphi \) via \( \mathcal{M}_{\text{TAR}} \). Lemma 6 gives the result.
Section 4.3.6: GHS is Equitable for TAR

Let $\Psi_\varphi$ be the set of all pairs $(s, \psi)$ of reachable states $s$ of $GHS$ and actions $\psi$ of $GHS$ enabled in $s$ such that $m'$ is the message at the head of $queue_{qp}(q,p)$ in $s$, and $\psi = ChannelRec(q,p,m')$.

For reachable state $s$, let $v_\varphi(s)$ be the number of messages in $queue_{qp}(q,p)$ ahead of the message at the head of $tarqueue_{qp}(q,p)$.

Verifying the progressive conditions is straightforward.

iv) $\varphi$ is $ReceiveTest((q,p),l,c)$, $ReceiveAccept((q,p))$, or $ReceiveReject((q,p))$. We show that $GHS$ is progressive for $\varphi$ via $\mathcal{MTAR}$. Lemma 6 gives the result.

Let $\Psi_\varphi$ be the set of all pairs $(s, \psi)$ of reachable states $s$ of $GHS$ and actions $\psi$ of $GHS$ enabled in $s$ such that $m'$ is the message at the head of $queue_p(q,p)$ in $s$, and $\psi = Receive((q,p),m)$.

For reachable state $s$, let $v_\varphi(s)$ be the number of messages in $queue_p(q,p)$ ahead of the message at the head of $tarqueue_p(q,p)$.

Verifying the progressive conditions is straightforward.

v) $\varphi$ is $SendTest(p)$. We show that $GHS$ is progressive for $\varphi$ via $\mathcal{MTAR}$. Lemma 6 gives the result.

Let $\Psi_\varphi$ be the set of all pairs $(s, \pi)$ of reachable states $s$ of $GHS$ and actions $\pi$ of $GHS$ enabled in $s$ such that one of the following is true: (Let $f = fragment(p)$.)

- $\text{CONNECT}(l)$ is in $queue((q,r))$, where $(q,r) = core(f)$ and $p \in subtree(q)$, $m$ is any message in $queue((q,r))$ that is not behind the $\text{CONNECT}(l)$ in $s$, and $\psi \in \{ChannelSend((q,r),m), ChannelRec((q,r),m), Receive((q,r),m)\}$.

- An $\text{INITIAL}(l,c,\text{find})$ message in $queue((t,u))$ is headed toward $p$ and $m$ is any message in $queue((t,u))$ that is not behind the $\text{INITIAL}(l,c,\text{find})$ in $s$, and $\psi \in \{ChannelSend((t,u),m), ChannelRec((t,u),m), Receive((t,u),m)\}$.

For reachable state $s$, $v_\varphi(s)$ is a 7-tuple with the following components.

If no $\text{CONNECT}$ is in $queue((q,r))$, 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 $\text{CONNECT}$ message in $queue((q,r))$. 

187
Section 4.3.6: GHS is Equitable for TAR

1. The number of messages in $\text{queue}_q((q, r))$ that are not behind the CONNECT.
2. The number of messages in $\text{queue}_{qr}((q, r))$ that are not behind the CONNECT.
3. The number of messages in $\text{queue}_r((q, r))$ that are not behind the CONNECT.

If no $\text{INITIATE}(l, c, \text{find})$ 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 $\text{queue}((t, u))$.

4. The number of nodes on the path in $\text{subtree}(f)$ from $u$ to $p$, including the endpoints.
5. The number of messages in $\text{queue}_l((t, u))$ that are not behind the $\text{INITIATE}(l, c, \text{find})$.
6. The number of messages in $\text{queue}_{tl}((t, u))$ that are not behind the $\text{INITIATE}(l, c, \text{find})$.
7. The number of messages in $\text{queue}_{u}((t, u))$ that are not behind the $\text{INITIATE}(l, c, \text{find})$.

(1) Let $s$ be a reachable state of GHS in $E_\varphi$. Thus, $p \in \text{testset}(f)$ and $\text{testlink}(p) = \text{nil}$. By the definition of $\text{testset}(f)$, either a FIND message is headed toward $p$ in some $\text{queue}((q, r))$, or a CONNECT message is in $\text{queue}((q, r))$, where $(q, r) = \text{core}(f)$ and $p \in \text{subtree}(q)$. In either case, let $m$ be the message at the head of $\text{queue}((t, u))$. Let $\psi$ be $\text{ChannelSend}((q, r), m)$ if $m$ is in $\text{queue}_q((q, r))$; let $\psi$ be $\text{ChannelRecv}((q, r), m)$ if $m$ is in $\text{queue}_{qr}((q, r))$; let $\psi$ be $\text{Receive}((q, r), m)$ if $m$ is in $\text{queue}_r((q, r))$. Obviously, $\psi$ is enabled in $s$ and $(s, \psi) \in \Psi_\varphi$.

(2) Let $(s', \pi, s)$ be a step of GHS, $s'$ be reachable and in $E_\varphi$, $(s', \pi) \notin X_\varphi$, 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, this is enough. We consider all the ways that $v_\varphi$ could change.

Can a CONNECT be added to $\text{queue}((q, r))$, with $(q, r) = \text{core}(f)$ by $\pi$? By COM-F, $(p, q) \in \text{subtree}(f)$, so by TAR-A(b), $\text{lstatus}((q, r)) = \text{branch}$. 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 $(s', \pi) \notin \Psi_\varphi$, no CONNECT can be removed from the relevant queue.

For a given fragment $f$, core$(f)$ never changes.

188
Section 4.3.6: GHS is Equitable for TAR

Can the identity of \textit{fragment}(p) change? Since \( p \in \text{testset}(f) \) by the precondition of \( \varphi \), \( \minlink(f) = \text{nil} \) in \( s' \) by GC-C. Thus no \textit{Absorb}(g,f), \textit{Merge}(f,g) or \textit{Merge}(g,f) is enabled in \( s' \).

The number of messages in the same queue as the relevant \textit{CONNECT} message but not behind it cannot change, because the queues are FIFO (and \( (s', \pi) \notin \Psi_\varphi \)).

Can a relevant \textit{INITIATE} message be added? The only way it can is if either a \textit{CONNECT} message in \textit{queue}((q,r)) with \( (q,r) = \text{core}(f) \) and \( p \in \text{subtree}(q) \) is received, or if the same \textit{INITIATE} message headed toward \( p \) is received. Since \( (s', \pi) \notin \Psi_\varphi, \pi \) is neither of these actions.

Can the path from \( u \) to \( p \) change, where an \textit{INITIATE}(l,c,\text{find}) is in \textit{queue}((t,u)) 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 \textit{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 \( \psi \) such that \( (s', \psi) \in \Psi_\varphi \) can become disabled in \( s \) without occurring, since the queues are FIFO.

vi) \( \varphi \) is \text{ComputeMin}(f). We show that the hypotheses of Lemma 7 are satisfied to get the result.

Let \( A = \text{GHS}, B = \text{TAR}, C = \text{DC}, D = \text{GC}, \) and \( \rho = \text{ComputeMin}(f) \) in the hypotheses of Lemma 7.

(1) If \( e \) is an execution of \( \text{GHS} \), then by Lemmas 1 and 25, \( \mathcal{M}_{\text{DC}}(e) \) is an execution of \( \text{DC} \).

(2) Let \( s \) be a reachable state of \( \text{TAR} \). If \( \varphi \) is enabled in \( \mathcal{S}_{\text{TAR}}(s) \), then as argued in Section 4.2.3 (\( \text{TAR} \) to \( \text{GC} \)), \( \varphi \) is enabled in \( \mathcal{S}_3(\mathcal{S}_{\text{TAR}}(s)) \). By the way the \( S \)'s are defined, \( \mathcal{S}_3(\mathcal{S}_{\text{TAR}}(s)) = \mathcal{S}_4(\mathcal{S}_{\text{DC}}(s)) \), so \( \rho = \varphi \) is enabled in \( \mathcal{S}_4(\mathcal{S}_{\text{DC}}(s)) \).

(3) Suppose \( (s', \pi, s) \) is a step of \( \text{GHS} \) and \( s' \) is reachable. If \( \varphi \) is not in \( \mathcal{A}_{\text{TAR}}(s', \pi) \), then \( \rho \) is not in \( \mathcal{M}_4(\mathcal{M}_{\text{DC}}(s', \pi)) \) by inspection.
Section 4.3.6: GHS is Equitable for TAR

(4) DC is progressive for \( \rho \) via \( \mathcal{M}_4 \), using \( \Psi_\rho \) and \( v_\rho \), by Lemma 30.

(5) Let \( \psi \) be such that \( (t, \psi) \in \Psi_\rho \) for some \( t \). Possible values of \( \psi \) are ChannelSend(1, REPORT(w)), ChannelRecv(1, REPORT(w)), and ReceiveReport(1, w). Essentially the same arguments as in ii), iii) and iv) show that GHS is progressive for \( \psi \).

vii) \( \varphi \) 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 \( \varphi \) is enabled in \( S_{TAR}(s) \). As argued in Section 4.2.3 (TAR to GC), \( \varphi \) is enabled in \( S_2(S_{TAR}(s)) \). As argued in Section 4.2.2 (GC to COM), \( \varphi \) is enabled in \( S_3(S_{TAR}(s)) \). By the way the \( S \)'s are defined, \( S_2(S_3(S_{TAR}(s))) = S_6(S_{CON}(s)) \), so \( \rho = \varphi \) is enabled in \( S_6(S_{CON}(s)) \).

(3) Suppose \( (s', \pi, s) \) is a step of GHS and \( s' \) is reachable. If \( \varphi \) is not in \( A_{TAR}(s', \pi) \), then \( \rho \) is not in \( M_6(M_{CON}(s'\pi s)) \) by inspection.

(4) CON is progressive for \( \rho \) via \( M_6 \), using \( \Psi_\rho \) and \( v_\rho \), by Lemma 31.

(5) Let \( \psi \) be such that \( (t, \psi) \in \Psi_\rho \) for some \( t \). Possible values of \( \psi \) are ChannelSend(1, CHANGEROOT), ChannelRecv(1, CHANGEROOT), and ReceiveChangeRoot(1). Essentially the same arguments as in ii), iii) and iv) show that GHS is progressive for \( \psi \).

viii) \( \varphi \) is ChangeRoot(f), subtree(f) is \( \{ p \} \) for some \( p \). We show that GHS is progressive for \( \varphi \) via \( M_{TAR} \). Lemma 6 gives the result.

Let \( \Psi_\varphi \) be the set of all pairs \((s, \psi)\) of reachable states \( s \) of GHS and internal actions \( \psi \) of GHS enabled in \( s \) such that none of the following is true:

- \( \psi = ReceiveConnect((q, r), l) \) for some \( q, r \) and \( l \), and in \( s \), \( nstatus(r) \neq \) sleeping, \( l \geq nlevel(r) \), \( lstatus((r, q)) \neq \) unknown, and only one message is in \( queue,( (q, r)) \).

- \( \psi = ReceiveTest((q, r), l, c) \) for some \( q, r, l \) and \( c \), and in \( s \), \( nstatus(r) \neq \) sleeping, \( l > nlevel(c) \), and only one message is in \( queue,( (q, r)) \).
Section 4.3.6: GHS is Equitable for TAR

- $\psi = \text{ReceiveReport}((q, r), w)$ for some $q, r$ and $w$, and in $s$, $\text{inbranch}(r) = (q, r)$, $\text{nstatus}(r) = \text{find}$, and only one message is in $\text{queue}_r((q, r))$.

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 $\text{rootchanged}(g) = \text{false}$ in $s$.
3. The number of fragments $g$ with $\text{minlink}(g) = \text{nil}$ in $s$.
4. The number of nodes $q \in V(G)$ such that $q \in \text{testset}((\text{fragment}(q)))$.
5. The summation over all $q \in V(G)$ of $\text{level}((\text{fragment}(q))) - \text{nlevel}(q)$.
6. The summation over all $q \in V(G)$ of $\text{findcount}(q)$.
7. The number of links $(q, r)$ such that either $\text{Istatus}((q, r)) = \text{unknown}$, or $\text{Istatus}((q, r)) = \text{branch}$ and there is a protocol message for $(q, r)$.
8. The number of links $(q, r)$ such that no $\text{ACCEPT}$ or $\text{REJECT}$ is in $\text{queue}((q, r))$.
9. The summation over all fragments $g$ such that a $\text{CHANGEROOT}$ is in some $\text{subtree}(g)$ of the number of nodes in the path in $\text{subtree}(g)$ from $r$ to $\text{minnode}(g)$.
10. The number of fragments $g$ such that $\text{AfterMerge}(q, r)$ for $\text{DC}$ is enabled for some $q \in \text{nodes}(g)$.
11. The number of messages in $\text{queue}_q((q, r))$, for all $(q, r) \in L(G)$.
12. The number of messages in $\text{queue}_{qr}((q, r))$, for all $(q, r) \in L(G)$.
13. The number of messages in $\text{queue}_r((q, r))$, for all $(q, r) \in L(G)$.
14. The number of messages in $\text{queue}_r((q, r))$ that are behind a $\text{CONNECT}$ or $\text{TEST}$, for all $(q, r) \in L(G)$.

(1) Let $s$ be a reachable state of $GHS$ in $E_\varphi$. We now demonstrate that some action $\psi$ is enabled in $s$ with $(s, \psi) \in \Psi_\varphi$.

By preconditions of $\varphi$, $\text{awake} = \text{true}$, $\text{minlink}(f) \neq \text{nil}$ and $\text{rootchanged}(f) = \text{false}$ in $s$. By GHS-K, $\text{nstatus}(p) = \text{true}$ in $s$. But since $\text{awake} = \text{true}$, there is some node $q$ such that $\text{nstatus}(q) \neq \text{sleeping}$. Thus $A$, the set of all fragments $g$ such that $\text{nstatus}(q) \neq \text{sleeping}$ for some $q \in \text{nodes}(g)$, is non-empty. Let $l$ be the minimum level of all fragments in $A$, and let $A_l = \{q \in A : \text{level}(g) = l\}$.

The strategy is to use a case analysis as follows. For each case, we show that there is some $\text{queue}((q, r))$ with some message $m$ in it in $s$. Let $\psi$ be chosen as follows. If some message $m'$ is at the head of $\text{queue}_q((q, r))$, let $\psi = \text{ChannelSend}((q, r), m')$. If no message is in $\text{queue}_q((q, r))$ and some message $m'$ is at the head of $\text{queue}_{qr}((q, r))$, let $\psi = \text{ChannelSend}((q, r), m')$. If no message is in $\text{queue}_q((q, r))$ or $\text{queue}_{qr}((q, r))$, then at least one message, namely $m$.
Section 4.3.6: GHS is Equitable for TAR

is in \textit{queue}_r((q, r)); let \( \psi = \text{Receive}((q, p), m') \), where \( m' \) is the message at the head of \textit{queue}_r((q, r)).

For each choice, \( \psi \) is obviously enabled in \( s \). There are two methods to verify that \( (s, \psi) \in \Psi_\omega \). Method 1 is to show that \( m \) is not \textit{CONNECT}, \textit{TEST} or \textit{REPORT}. Then, if \( \psi = \text{Receive}((q, r), m') \) and \( m' \) is \textit{CONNECT}, \textit{TEST} or \textit{REPORT}, there is more than one message in \textit{queue}_r((q, r)). Method 2 is to show that some variable in \( s \) has a value such that even if \( \psi = \text{Receive}((q, r), m') \), where \( m' \) is \textit{CONNECT}, \textit{TEST} or \textit{REPORT}, we have that \( (s, \psi) \in \Psi_\omega \).

Case 1: There is a fragment \( g \in \mathcal{A}_l \) with \textit{testset}(g) \( \neq \emptyset \). Let \( q \) be some element of \textit{testset}(g). By definition of \textit{testset}(g), Cases 1.1, 1.2 and 1.3 are exhaustive.

Case 1.1: A \textit{CONNECT}(l) message is in \textit{queue}(r, t), where \((r, t) = \text{core}(g) \) and \( q \in \text{subtree}(r) \) in \( s \). We use Method 2. By \textit{COM-F}, \((r, t) \in \text{subtree}(g) \), so by \textit{TAR-A(b)}, \textit{lstatus}((t, r)) = \textit{branch}.

Case 1.2: An \textit{INITIATE}(l, c, \text{find}) message is in some \textit{queue}((r, t)) headed toward \( q \) in \( s \). By Method 1, we are done.

Case 1.3: \textit{testlink}(q) \( \neq \text{nil} \) in \( s \). By \textit{TAR-C(a)}, \textit{testlink}(q) = (q, r) for some \( r \). By \textit{TAR-C(c)}, there is a protocol message for \((q, r)\).

Case 1.3.1: The protocol message is an \textit{ACCEPT} or \textit{REJECT} in \textit{queue}((r, q)). By Method 1, we are done.

Case 1.3.2: The protocol message is \textit{TEST}(l', c) in \textit{queue}((q, r)). Thus \textit{lstatus}((q, r)) \( \neq \text{rejected} \). By \textit{TAR-E(b)}, \( l' = l \). If \textit{nstatus}(r) = \textit{sleeping} or \( l \leq \text{level}(r) \), we are done, by Method 2. Suppose \textit{nstatus}(r) \( \neq \text{sleeping} \) and \( l > \text{level}(r) \). By definition of \( \mathcal{A}_l \), \( l \leq \text{level}(\text{fragment}(r)) \), and thus \( \text{level}(r) < \text{level}(\text{fragment}(r)) \). By \textit{NOT-G}, either a \textit{NOTIFY}(\text{level}(\text{fragment}(r))) message is in some \textit{queue}((t, u)) headed toward \( r \), in which case we are done by Method 1, or \textit{AfterMerge}(t, u) is enabled for \textit{NOT}, with \( r \in \text{subtree}(u) \). In the latter case, by \textit{GHS-L}, a \textit{CONNECT} is at the head of \textit{queue}((u, t)); the same argument as in Case 1.1 gives the result.

Case 2: \textit{testset}(g) = \emptyset for all \( g \in \mathcal{A}_l \).

Case 2.1: There is a fragment \( g \) in \( \mathcal{A}_l \) with \textit{minlink}(g) = \text{nil}. Since \( g \neq f \) and \( G \) is connected, there is an external link of \( g \). Since \textit{testset}(g) = \emptyset, by \textit{DC-D(c)} no \textit{FIND} message is in \textit{subtree}(g).
Section 4.3.6: GHS is Equitable for TAR

Suppose $dcstatus(q) = \text{unfind}$ for all $q \in \text{nodes}(g)$. By definition of $\text{minlink}(g)$, a REPORT message is in some $\text{queue}(q, r)$ headed toward $\text{mw-root}(g)$. We are done by Method 2.

Suppose $dcstatus(q) = \text{find}$ for some $q \in \text{nodes}(g)$. By DC-I(b), since $\text{testset}(g) = \emptyset$, a REPORT message is in some $\text{queue}(r, t)$ in $\text{subtree}(q)$ headed toward $q$. By DC-B(a), $\text{inbranch}(t) \neq (t, r)$. We are done by Method 2.

Case 2.2: $\text{minlink}(g) \neq \text{nil}$ for all $g \in A_l$.

Case 2.2.1: There is a fragment $g$ in $A_l$ with $\text{rootchanged}(g) = \text{false}$. By GHS-K, if $\text{subtree}(g) = \{q\}$ for some $q$, then $v.status(q) = \text{sleeping}$. By definition of $A_l$, $\text{subtree}(g) \neq \{q\}$ for any $q$. By CON-B, a CHANGEROOT message is in some $\text{queue}(q, r)$ in $\text{subtree}(g)$. We are done by Method 1.

Case 2.2.2: $\text{rootchanged}(g) = \text{true}$ for all $g \in A_l$. By CON-D, a CONNECT message is in $\text{queue}(\text{minlink}(g))$ for all $g \in A_l$.

Case 2.2.2.1: There is a fragment $g$ in $A_l$ with $\text{minlink}(g) = (q, r)$ and $\text{level}(\text{fragment}(r)) > 1$.

If $\text{nlevel}(r) > 1$, we are done by Method 2. Suppose $\text{nlevel}(r) \leq 1$. Essentially the same argument as in Case 1.3(b) gives the result.

Case 2.2.2.2: For all fragments $g$ in $A_l$, $\text{level}(\text{fragment}(\text{target}(\text{minlink}(g)))) \leq 1$. By COM-A, $\text{level}(\text{fragment}(\text{target}(\text{minlink}(g)))) = 1$ for all $g \in A_l$.

Case 2.2.2.2.1: There is a fragment $g$ in $A_l$ such that $\text{minlink}(g) = (q, r)$, and $\text{fragment}(r) \notin A_l$. By definition of $A_l$, $\text{nstatus}(r) = \text{sleeping}$, and we are done by Method 2.

Case 2.2.2.2.2: For all fragments $g$ in $A_l$, $\text{fragment}(\text{target}(\text{minlink}(g))) \in A_l$. As argued in Lemma 27, Case 2.2.2 of verifying (1) for $\varphi = \text{Combine}$, there are two fragments $g$ and $h$ in $A_l$ such that $\text{minedge}(g) = \text{minedge}(h) = (q, r)$. By TAR-H, $lstatus((r, q)) = lstatus((q, r)) = \text{branch}$. By Method 2, we are done.

(2) Let $(s', \pi, s)$ be a step of $GHS$, where $s'$ is reachable and in $E_\varphi$, $(s', \pi) \notin X_\varphi$, and $s \in E_\varphi$.
Section 4.3.6: GHS is Equitable for TAR

(a) We show that if \((s', \pi) \not\in \Psi_\phi\), then \(v_\phi(s) = v_\phi(s')\); together with part (b) below, this gives the result. \(\Psi_\phi\) is defined to include all the state-action pairs that change the state. Thus, if \((s', \pi) \not\in \Psi_\phi\), then \(s = s'\), and obviously \(v_\phi(s) = v_\phi(s')\).

(b) Suppose \((s, \pi) \in \Psi_\phi\). 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 \(v_\phi\) decreases in going from \(s'\) to \(s\), and components 1 through 11 are unchanged.

- \(\pi = \text{ChannelSend}(\langle q, r \rangle, m)\). Component 11.

- \(\pi = \text{ChannelRecv}(\langle q, r \rangle, m)\). Component 12.

- \(\pi = \text{ReceiveConnect}(\langle q, r \rangle, l)\).

Case 1: nstatus\((r) = \text{sleeping} in s'\). If \((q, r)\) is not the minimum-weight external link of \(r\), then: component 2. Otherwise, component 1.

Case 2: nstatus\((r) \neq \text{sleeping}, l = nlevel(r)\) and no CONNECT is in queue\((r, q)\) in \(s'\).

Suppose lstatus\((\langle r, q \rangle) = \text{unknown}\). Since \((s', \pi) \in \Psi_\phi\), another message is in queue\((q, r)\). By CON-D, CON-E and GHS-C, the other message is not a CONNECT or TEST. Component 14.

Suppose lstatus\((\langle r, q \rangle) \neq \text{unknown}\). Since DC simulates AfterMerge\((r, q)\), neither AfterMerge\((r, q)\) nor AfterMerge\((q, r)\) is enabled in \(s\). Component 10.

Case 3: nstatus\((r) \neq \text{sleeping}, l = nlevel(r)\), and CONNECT is in queue\((r, q)\) in \(s'\). Component 1.

Case 4: nstatus\((r) \neq \text{sleeping} and l < nlevel(r) in s'\). Component 1.

- \(\pi = \text{ReceiveInitiate}(\langle q, r \rangle, l, c, st)\). By NOT-H(a), \(l > nlevel(r)\). Component 5.

- \(\pi = \text{ReceiveTest}(\langle q, r \rangle, l, c)\). Let \(g = \text{fragment}(r)\).

Case 1: nstatus\((r) = \text{sleeping} in s'\). Component 2.

Case 2: nstatus\((r) \neq \text{sleeping} in s'\).
Section 4.3.6: GHS is Equitable for TAR

**Case 2.1:** \( l \leq \text{level}(g) \), and either \( c \neq \text{core}(g) \) or \( \text{testlink}(r) \neq (r,q) \) in \( s' \). If an \text{ACCEPT} is added, then component 8. If a \text{REJECT} is added, then either component 7 or component 8.

**Case 2.2:** \( l \leq \text{level}(g) \), \( c = \text{core}(g) \), and \( \text{testlink}(r) = (r,q) \) in \( s' \). If there is no link \((r,t), t \neq q, \) with \( \text{lstatus}(r,t) = \) unknown, then component 4. If there is such a link, then component 7.

**Case 2.3:** \( l > \text{level}(g) \) in \( s' \). Since \((s,\pi) \in \Psi_\phi \), there is another message in \( \text{queue}_r((q,r)) \). By TAR-C(c) and GHS-C, the other message is not \text{CONNECT} or \text{TEST}. Component 14.

- \( \pi = \text{ReceiveAccept}(\langle langleq, r \rangle) \). Component 4.
- \( \pi = \text{ReceiveReject}(\langle q, r \rangle) \). If there is no link \((r,t), t \neq q, \) with \( \text{lstatus}(r,t) = \) unknown, then component 4. If there is such a link, then component 7.
- \( \pi = \text{ReceiveReport}(\langle q, r \rangle, w) \).

**Case 1:** \((q,r) = \text{core}(g), \text{nstatus}(r) \neq \text{find} \) and \( w > \text{bestwt}(r) \) in \( s' \). If \( \text{lstatus}(\text{bestlink}(r)) = \text{branch} \), then component 3. Otherwise, component 2.

**Case 2a:** \((q,r) \neq \text{core}(g) \) in \( s' \). If \( \text{inbranch}(r) = (r,q) \), then component 13. Otherwise, component 6.

**Case 2b:** \((q,r) = \text{core}(g) \) and \( \text{nstatus}(r) = \text{find} \) in \( s' \). The only change is that the \text{REPORT} message is requeued. We show that there is no other message in \( \text{queue}(\langle q, r \rangle) \), and thus \((s',\pi) \notin \Psi_\phi \). First note that by COM-F, \((q,r) \in \text{subtree}(g) \). By GHS-B, no \text{CONNECT} is in the queue. By DC-O, no \text{INITIATE}(*,*,\text{found}) is in the queue. By GHS-E, no \text{INITIATE}(*,*,\text{find}) is in the queue. By TAR-E(a), no \text{TEST} or \text{REJECT} is in the queue. By DC-O, no other \text{REPORT} is in the queue. By TAR-F, no \text{ACCEPT} is in the queue. By CON-C, no \text{CHANGEROOT} is in the queue.

**Case 2c:** \((q,r) = \text{core}(g) \), \( \text{nstatus}(r) = \text{unfind} \), and \( w \leq \text{bestwt}(p) \). Component 13.

- \( \pi = \text{ReceiveChangeRoot}(\langle q, r \rangle) \). If \( \text{lstatus}(\text{bestlink}(r)) \neq \text{branch} \), then component 2. Otherwise, component 9.

---

(c) Suppose \((s',\pi) \notin \Psi_\phi \), \( \psi \) is enabled in \( s' \), and \((s',\psi) \in \Psi_\phi \). Since \((s',\pi) \notin \Psi_\phi \), \( s = s' \). Obviously, \( \psi \) is enabled in \( s \) and \((s,\psi) \in \Psi_\phi \).
Section 4.4: Satisfaction

ix) \( \varphi \) is \textit{Merge}(f,g). We use Lemma 7. The same argument as in \textit{vii}), with 
\( \rho = \text{Merge}(f,g) \) and (3) as below, gives the result.

\( \text{(3)} \) Let \( \psi \) be such that \( (t,\psi) \in \Psi_{\rho} \) for some \( t \). Possible values of \( \psi \) are 
\text{ChannelSend}(k,\text{CONNECT}(l)), \text{ChannelRecv}(k,\text{CONNECT}(l)), \) and \text{Merge}(f,g). Essentially the same arguments as in \textit{ii}), \textit{iii}) and \textit{iv}) show that \textit{GHS} is progressive for \( \psi \).

x) \( \varphi \) is \textit{Absorb}(f,g). We use Lemma 7. The same argument as in \textit{vii}), with 
\( \rho = \text{Absorb}(f,g) \) and (3) as below, gives the result.

\( \text{(3)} \) Let \( \psi \) be such that \( (t,\psi) \in \Psi_{\rho} \) for some \( t \). Possible values of \( \psi \) are 
\text{ChannelSend}(k,\text{CONNECT}(l)), \text{ChannelRecv}(k,\text{CONNECT}(l)), \) and \text{Absorb}(f,g). Essentially the same arguments as in \textit{ii}), \textit{iii}) and \textit{iv}) show that \textit{GHS} is progressive for \( \psi \).

4.4 Satisfaction

Theorem 33: \textit{GHS} solves \textit{MST}(G).

Proof: By Theorem 12, \textit{HI} solves \textit{MST}(G). By Lemmas 13 and 27 and Theorem 8, \textit{COM} satisfies \textit{HI}. By Lemmas 15 and 28 and Theorem 8, \textit{GC} satisfies \textit{COM}. By Lemmas 17 and 29 and Theorem 8, \textit{TAR} satisfies \textit{GC}. By Lemmas 25 and 32 and Theorem 9, \textit{GHS} satisfies \textit{TAR}. Thus, since “satisfies” and “solves” are defined using subsets of schedules, \textit{GHS} solves \textit{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


Section 4.4: Satisfaction


Appendix


Appendix

In this Appendix, we review the aspects of the model from [LT] that are relevant to this paper.

An input-output automaton 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
Appendix

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 s₀π₁s₁... of alternating states and actions such that s₀ is a start state, (sᵢ₋₁, πᵢ, sᵢ) 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') ⊆ 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
Appendix

composition is the union of the partitions of the component automata.

Given an automaton $A$ and a subset $\Pi$ of its actions, we define the automaton $\text{Hide}_\Pi(A)$ to be the automaton $A'$ differing from $A$ only in that each action in $\Pi$ 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 $\text{part}(A)$, 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 $\text{Fairbeh}(A)$, is the external schedule module with the input and output actions of $A$, and with the set of schedules $\{\alpha|\text{ext}(A) : \alpha$ is the schedule of a fair execution of $A\}$. A problem is (specified by) an external schedule module. Automaton $A$ solves the problem $P$ if $\text{Fairbeh}(A)$ 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 $\text{Fairbeh}(A)$ is a sub-schedule module of $\text{Fairbeh}(B)$.

---

1 If $\alpha$ is a sequence from a set $S$ and $T$ is a subset of $S$, then $\alpha|T$ is defined to be the subsequence of $\alpha$ consisting of elements in $T$. 

200
OFFICIAL DISTRIBUTION LIST

Director
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. R. Grafton, Code 433

Director, Code 2627
Naval Research Laboratory
Washington, DC 20375

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314

National Science Foundation
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

Dr. E.B. Royce, Code 38
Head, Research Department
Naval Weapons Center
China Lake, CA 93555