The verified incremental design of a distributed spanning tree algorithm

Wim H. Hesselink, whh169
email: w.h.hesselink@rug.nl, web: wimhesselink.nl
Johann Bernoulli Institute for Mathematics and Computer Science
University of Groningen,
Groningen, The Netherlands

February 24, 2015

Abstract

The paper describes an incremental mechanically–verified design of the algorithm of Gallager, Humblet, and Spira for the distributed determination of the minimum-weight spanning tree in a graph of processes. The processes communicate by means of asynchronous messages with their neighbours in the graph. A computational model for asynchrony is used that enables state based reasoning. The assumption that the message buffers are fifo is removed. The algorithm is extended with distributed termination detection. The proof of the algorithm is based on ghost variables, invariants, and a decreasing variant function. The verification is mechanized by means of the theorem prover NQTHM of Boyer and Moore. The proof obligations for the mechanical proof are discussed.

An extended abstract of this paper appeared in [Hes99]. It refers to the present paper as the full paper, but it wrongly asserts that the full paper can be obtained from an ftp-site.

Contents

1 Introduction

Given is a connected undirected graph in which all edges have different weights. It is wellknown that such a graph has a unique minimum–weight spanning tree. Now assume that the nodes of the graph are processes that can asynchronously send messages to neighbour processes, and that every process only knows the weights of its incident edges and the names of its neighbours.

In 1983, Gallager, Humblet, and Spira published an algorithm for these processes to determine the minimum–weight spanning tree, cf. [GHS83]. The algorithm is not very hard to understand and there are good informal explanations, e.g. cf. [Tel94]. There are, however, two ways to understand an algorithm: the reader can trust the author(s) and assume that missing details have been treated adequately elsewhere, or he may want a conclusive argument at every point that might go wrong. For readers of the second kind, the GHS algorithm has never been treated satisfactorily. Indeed, the level of concurrency allowed makes it very hard to verify that no undesired interference or deadlock occurs. A number of groups, cf. [ChG88, SdR94, WLL88, ZwJ93], have reported on verifications of the correctness of the algorithm (or variations of it), but handwritten proofs are almost never complete and hence not very convincing.

We have therefore undertaken the construction of a proof for a mechanical theorem prover, so that anyone who understands the language of the prover can verify the proof, i.e., can see what it is we are asserting, can let the prover verify the assertions, and can inspect any detail they want to look into. We have not proved the precise algorithm of [GHS83], but our algorithm is closely related and at least as efficient. We claim that our mechanical proof is the first complete formal proof of a GHS-type algorithm with all its optimizations.

We constructed the algorithm and its proof in a kind of reverse engineering. Knowing the algorithm of [GHS83], we performed a verified incremental design of it. Therefore, in each stage of the project, we knew the invariant properties of the algorithm at that stage. This approach by means of a formally independent design has the advantage that we can motivate or discuss most of the design decisions hidden in the algorithm.

Some earlier proofs, cf. [SdR94, ZwJ93], started with the verification of a sequential program, which is then gradually distributed in a number of program transformations. In contrast to this approach, we start with a highly nondeterministic distributed algorithm which is gradually tuned to fulfil the specification.

1.1 Overview of the work

There are several issues to be addressed: modelling assumptions, specification of the distributed algorithm, graph theory, the incremental design of the declaration of the messages, invariants for safety, elimination of deadlock, proof of termination, proof obligations for the mechanical proof.

In the remainder of this introduction we treat some issues not specific for GHS algorithm. In Chapter 2, we give the underlying abstract sequential algorithm and specify and describe the distributed algorithm. In the distributed algorithm the minimum–weight spanning tree (*MST*) is constructed as a growing "forest".

In Chapter 3, the representation of the forest is distributed over the nodes of the graph in such a way that it can grow. Chapter 4 contains the graph theory needed for the algorithm. The Chapters 5, 6, 7 describe how the growing forest fills the *MST*.

When the algorithm terminates the growing forest must have filled the *MST*. Chapter 8 deals with the proof of that. Chapter 9 deals with the proof that the algorithm indeed terminates. Since the algorithm is designed incrementally in the Chapters 3 through 8, the resulting algorithm is presented in Chapter 10. In Chapter 11, we compare the resulting algorithm with other version of the GHS algorithm.

In Chapter 12, we describe how the mechanical theorem prover NQTHM of Boyer and Moore serves as a witness for the correctness of the algorithm.

The main effort of the project was the construction of the global invariant, which is a conjunction of about 160 universal quantifications of so-called constituent invariants. It seems that none of these constituent invariants can be omitted. We also need some 80 other predicates that follow from the global invariant.

In Chapter 13, we draw some conclusions.

In Chapter 14, we present all constituent invariants and a selection of the derived invariants, both for completeness and to give an honest picture of the complexity involved. We would greatly prefer to have an easier proof or a more elegant one.

Yet the number of constituent invariants is not more than should be expected. Not counting ghost variables, the algorithm has 11 private variables, 11 messages, and 12 parameters of messages. The invariants express relationships between these 34 objects. Assuming that each object has one invariant to relate it to each other object, we would get $\binom{34}{2} = 595$ invariants, much more than the actual number 166.

1.2 Modelling asynchrony

We need to go into the modelling assumptions. Every process has a private state consisting of a number of private variables. Processes can send messages to neighbour processes. A process acts only when it accepts a message. Every message has a key word and a number of arguments. Via the declaration of the algorithm, the key word and the arguments determine the enabling condition of the message and the associated command. Acceptance of a message is defined to consist of its removal from the network together with the execution of its command. The enabling condition is the precondition for acceptance. The command can only inspect and modify private variables and send messages to neighbour processes; it always terminates. All processes concurrently execute the sequential program

while *true* **do** accept some enabled message or wait **od** .

The only fairness assumption is that, whenever the bag (multiset) of enabled messages is nonempty, one will be accepted eventually.

We distinguish a physical model and a mathematical model. In the physical model the acceptance of a message takes some time, and message acceptances of different processes may overlap. The possibility and the effect of acceptance, however, only depend on the message and the private state of the accepting process prior to acceptance. Since the acceptance is finished before the process can accept a next message and since the sending of messages only adds them to the bag of messages in transit, we may regard the acceptance of a message as a single atomic action and we may regard the atomic actions as interleaved.

In this way we arrive at the following mathematical model, a simple version of the model of [Tel94]. The *state* of the system consists of the private states of the processes together with the bag of messages that are in transit (sent, but not yet accepted by the destination process). A *transition* of the system is a step from one state to another in which a single process accepts an enabled message. An *execution* of the algorithm is a sequence of transitions that starts in some initial state. A state is called *reachable* if it occurs in an execution.

Since in every step of the system only one process is involved, this model of concurrency is simpler than the models for synchronous communication. It may even be simpler than the model with shared variables (compare [ApO91]). It is related to the I/O automata of (e.g.) [Lyn89] and to the receptive processes of [Jos92]. The model is more complex than UNITY, cf. [ChM88]. It may be regarded as a special case of UNITY, but the command associated to a message is typically much more complex than is usual in UNITY programs.

Remark. Instead of disabling, the paper [GHS83] allows processes to put the message back at the end of the message queue. In this way the order of the message queue becomes a complicating factor. We prefer to use disabling, and to eliminate the order of the messages. Since we do not assume preservation of the order of messages sent along one edge, the algorithm of [GHS83] is incorrect in our setting (in spite of what is suggested in [Tel94]). It turns out, however, that correctness can be re–established by a minor modification of the algorithm.

1.3 Invariants

An *invariant* is defined to be a predicate that holds in all reachable states. This definition is not very practical. So, we need a proof theory to verify invariants. We write $P \triangleright Q$ to denote that every atomic step of the algorithm that starts in a state where P holds, terminates in a state where Q holds. We define a predicate P to be a *strong invariant* if it holds initially and satisfies $P \triangleright P$. Note that Tel ([Tel94] page 51) uses the term *invariant* where we use the term *strong invariant*. It is easy to see that every predicate implied by a strong invariant is an invariant.

A stronger way to prove invariance of a predicate is to introduce ghost variables (auxiliary variables cf. [OwG76], p.325) and to prove that the predicate follows from a strong invariant that may use these ghost variables. Actually, we develop the algorithm in layers and decide in a final step of the design that certain variables can be regarded as ghost variables, and therefore can be eliminated from the algorithm. An invariant obtained for the algorithm with ghost variables that does not mention the ghost variables, is also valid for the algorithm without the ghost variables.

The usual way to obtain (strong) invariants is based on the following obvious result.

Theorem. Let Q be the conjunction of a family of predicates P_i with $i \in I$. Assume that Q holds initially and that $Q \triangleright P_i$ for all $i \in I$. Then Q is a strong invariant.

In such a situation, the predicates P_i follow from Q and are therefore invariants. Since they are used to construct a strong invariant, they are called *constituent invariants*. We prefer to use constituent invariants that cannot easily be expressed as conjunctions of smaller expressions (although we do not fix the language, such a predicate might be called irreducible).

For GHS we need a family of some 160 constituent invariants. The whole strong invariant is not manageable. There may be many useful conjunctions of constituent invariants. We found the only way to manage them was to list the "irreducible" ones. (Even so, we sometimes invented invariants that later turned out to be already there). Indeed, the main effort in the design was to manage this host of invariants.

Remark. So we describe the global invariant of a system as a conjunction of invariants. Instead of this, Amir Pnueli suggests to analyse the global invariant as a *disjunction* of predicates, which can then be regarded as modes of the system. For our algorithm, this approach seems not to be feasible.

1.4 Messages in transit

When we began to investigate the GHS algorithm, we had no idea what kind of invariants to use. Since messages are transient, we did not expect them in invariants. This turned out to be mistaken. For, in the end, most invariants express a property of a node when a certain message is in transit to it or from it.

For the formal description of the global state, we introduce variables $buf.q$ to hold the bag of messages in transit to process q. So, if process p sends a message with key word kw and arguments a to node $q \neq p$, according to the command $send(q, kw, a)$, this has the effect

$$buf.q := buf.q + \{(kw, a)\} \ ,$$

where $+$ denotes bag addition. Process q can accept any enabled message $m \in buf.q$. Acceptance of m has the effect that m is removed from $buf.q$, followed by execution of the command associated to m.

We use two other sending commands. Firstly, a multicast to a set S of destinations is expressed by $mcast(S, kw, a)$, which is equivalent to

$$\textbf{for all } r \in S \textbf{ do } \mathit{send}(r, \mathit{kw}, a) \textbf{ od} \; .$$

Secondly, in order to allow a finer grain of atomicity and some separation of concerns, we introduce the possibility that a process sends a message m to itself by means of the command $\mathit{delay}(m)$. The purpose of selfmessages is to postpone the execution of an action until execution is appropriate. Since the reasons to *send* and to *delay* are quite different, we take command $\mathit{send}(p, \mathit{kw}, a)$ for process p itself to be equivalent to *skip*. This makes some invariance proofs easier.

In order to discuss the messages in transit, we introduce the notations

$$\begin{array}{lcl} \mathit{kw} \textbf{ at } q & \equiv & (\exists\, a :: (\mathit{kw}, a) \in \mathit{buf}.q) \; , \\ (\mathit{kw}, j) \textbf{ at } q & \equiv & (\exists\, b :: (\mathit{kw}, j, b) \in \mathit{buf}.q) \; , \end{array}$$

which express that some message is in transit to q with key word kw (and first argument j, etc.). We write **not-at** for the negation of **at**. So u **not-at** q stands for $\neg(u \textbf{ at } q)$. If we want to discuss the number of such messages instead of the existence, the operator **at** is replaced by #. For example, $(\mathit{kw}, -, j)\#q$ is the number of messages in transit to q with key word kw and second argument j. There is no condition on first or third arguments: this is convenient since during the design the number of arguments of a message can grow.

Remark. In contrast to [GHS83], we assume point to point communication: processes send messages to neighbour processes and edges are merely pairs of neighbour nodes (the same is done in [WLL88]). In this way, we avoid channel names, but we disallow multiple edges. It follows that the identity of the sender is needed as an argument for some of the messages.

1.5 The role of the theorem prover

In the verified design of the algorithm we used the mechanical theorem prover of NQTHM of Boyer and Moore, cf. [BoM88].

We need such a tool for three reasons. Firstly, the proof of invariance of each of the constituent invariants requires meticulous case distinctions in which NQTHM is much better than human beings. Secondly, we start with the verification of a small algorithm (see Chapter 3) and extend the algorithm by gradually adding messages, private variables, and actions. For each extension, the old proof is mechanically replayed to see whether and where it needs adaptation. Thirdly, after having obtained invariance proofs for all constituent invariants, we have to make sure that the hypotheses used as preconditions in these proofs do follow from the invariants. Since, in the end, there are more than 160 constituent invariants and more than 80 auxiliary invariants, automation of this administrative task is indicated.

This host of invariants also makes the first reason more compelling. Indeed, each proof of invariance of a constituent invariant can be done by hand (and often has been done so), but after 160 proofs the accumulated probability of errors becomes threatening. It was for this reason that we started to use NQTHM. The second and third reason occurred to us during the project.

We also use NQTHM to verify the graph theory needed for the algorithm. Alternatively, one may suggest to introduce the results of the graph theory as a set of axioms but we regard this as unadvisable. It is dangerous to introduce axioms, however likely, into such a proof because of the possibility of inconsistency. Especially for NQTHM axioms are very risky: NQTHM is untyped, so it is easy to introduce inconsistencies since functions can be applied to unexpected arguments.

The tool NQTHM is called a theorem prover, not a proof checker. Indeed, it has abilities to prove lemmas that can be compared with those of a meticulous but not very gifted undergraduate student. Of course, being a tool, NQTHM is not creative or able to formalize informal arguments. Therefore, from the global point of view, NQTHM serves as a proof checker rather than as a theorem prover.

When one has performed a proof with NQTHM, one usually understands the proof much better than after a handwritten proof, since NQTHM needs assistence precisely at those points where the proof deviates from standard proof heuristics (similarly as that teaching a subject is a good way to learn it).

This paper is not intended as an introduction to the theorem prover NQTHM (we rather refer to [BoM88]), not even as an introduction to our use of NQTHM for asynchronous distributed algorithms. For that purpose we refer to [Hes97a]. In the present paper NQTHM only serves as a tool for the design of the algorithm, and as a witness for its correctness. The input to the prover is an additional source that can be inspected. It is very reliable but not very accessible. See Chapters 12 for more details.

We do not claim that NQTHM is the best tool for our purposes. We only state that NQTHM served us well. Perhaps most importantly, NQTHM is believed to be sound. Secondly, and of almost equal importance, NQTHM is able to prove the easy lemmas without user guidance. For us, NQTHM's untyped logic and its LISP-like syntax are easy enough to work with. We completely agree with the argument of [You97] 6.1, that such syntactic issues are a relatively minor concern for serious users of automated proof tools. We feel no need for a better user interface.

We regard NQTHM's lack of higher order functions as its main shortcoming. This lack is compensated, however, by on the one hand the simplicity of the logic, and on the other hand the possibilities to `constrain`, to `functionally instantiate`, and to interpret quotations of terms by means of the NQTHM function `eval$`. It may be noted that the last feature is no longer present NQTHM's successor ACL2, see [KaM97], and also that, in [You97], it is argued that in many proof projects higher order functions are not really needed and are therefore better eliminated in the specification stage.

2 Introduction to the algorithm

In this chapter we first describe the problem and an abstract sequential algorithm that solves it. We then turn to the question of distribution. In particular we give the specification of the distributed algorithm. Finally, we give a global sketch of the incremental simultaneous design of the algorithm and its proof. The graph theoretical assertions in this chapter will be discussed more extensively in Chapter 4.

2.1 The abstract algorithm

Let (V, E) be a connected undirected graph without selfloops. So V is the set of nodes (vertices) and E is the set of edges (pairs of nodes). The edges have numerical weights given by a function w on E.

Following [GHS83], we postulate that all edges have different weights. Then the graph has a unique minimum–weight spanning tree.

There are various ways to formalize minimum–weight spanning trees. For our purpose (the construction of a mechanical proof), the most convenient way is to define *MST* as the set of the edges that have no connection through lighter edges. The formal definition is given in Section 4.2. It is easy to see that every edge in a minimum–weight spanning tree belongs to this set *MST*. On the other hand, since all weights differ, the set *MST* contains no cycles, see Theorem 1 in Section 4.2. Therefore, *MST* is the minimum–weight spanning tree of the graph. It is because of this definition of *MST* that distributed determination of *MST* needs no central supervisor.

The algorithm is a distributed version of Boruvka's algorithm, cf. [Tar83]. The basic idea is the same as in the algorithms of Kruskal and Prim, see [CLR90]. In the algorithm, the elements of *MST* are determined by means of the following result. We define an outgoing edge of a set C of nodes to be an edge (x, y) with $x \in C$ and $y \notin C$. A lightest outgoing edge of C is an outgoing edge of C with the smallest weight. It is easy to verify (see Section 4.2), that

Theorem 3. A lightest outgoing edge of any set of nodes belongs to *MST*.

This result is used in the algorithm to construct *MST* as a growing forest F. So we introduce the invariant $F \subseteq MST$. Boruvka's algorithm uses Theorem 3 with for the set of nodes a connected component of forest F. This sequential algorithm is given by

```
F := ∅ ; stop := false ;
while ¬stop do
    choose v ∈ V ;
    C := {x | (v, x) ∈ F*} ;
    if possible let (x, y) be
            a lightest outgoing edge of C ;
        F := F ∪ {(x, y), (y, x)}
    else stop := true fi
od .
```

Here F^* is the reflexive transitive closure of relation F. So the assignment to C makes C the connected component of node v. Upon termination, the set C has no outgoing edges. Since C is nonempty and (V, E) is connected, it follows that C equals V, so that $(v, x) \in F^*$ for all nodes x. Since MST is a tree and $F \subseteq MST$, this implies $F = MST$ (see Corollary of Theorem 2 in Section 4.2).

Notice that we do not need the specific form of C to preserve the invariant $F \subseteq MST$. In the distributed algorithm we use two other kinds of sets C. Firstly, since the graph has no selfloops, the lightest incident edge of a node q always belongs to MST; here we use the set of nodes $\{q\}$. Secondly, we use the set $\{x \mid (v, x) \in JB^*\}$, where JB is a subset of F. Actually JB is a delayed version of F: the elements of F become elements of JB when some messages are accepted.

2.2 Distribution and specification

Now the algorithm is distributed in such a way that the nodes of the graph are processes that communicate by means of asynchronous messages to neighbours in the graph. The algorithm treats all nodes of the graph in the same way. We assume that all actions of the processes are triggered by messages. Therefore, we also assume that initially there is a *wakeup* message in transit to every node. The model does not guarantee that this *wakeup* message is the first message to be expected.

Initially, every node only knows its neighbours in the graph and the weights of the edges to these neighbours. Finally, every node knows which incident edges belong to MST.

Distribution of the algorithm means distribution of the variables F and *stop*, and distribution of the actions: the choice of a lightest outgoing edge of a component, and the corresponding modifications of F and *stop*.

Since F is to be a forest, it can be represented as a set of rooted trees. It is therefore distributed among the nodes by means of private variables that point to "parent" nodes. More precisely, we introduce for each node $q \in V$ a private variable $ib.q$ of process q and we let F be the state dependent relation on nodes given by

(F0) $$(x, y) \in F \quad \equiv \quad x \neq y \quad \wedge \quad (ib.x = y \quad \vee \quad ib.y = x) \ .$$

Variable *ib* corresponds to the variable *in-branch* of [GHS83]. We postulate that $ib.q = q$ holds initially for all nodes $q \in V$. Therefore $F = \emptyset$ holds initially.

The purpose of the algorithm is that eventually every node knows which incident edges belong to MST. Since *ib* can hold only one neighbour, we give every node q a private variable $branch.q$ with the properties

$$r \in branch.q \quad \Rightarrow \quad ib.r = q \ ,$$
$$ib.q \notin branch.q \ .$$

If we regard $ib.x$ as the father of x, then $branch.x$ is the set of known children of x.

The variable *stop* of Boruvka's algorithm is distributed by giving each node q a private variable *term.q* (for terminated) such that *term.q* implies that all nodes r know which incident edges belong to *MST*. This is expressed in the invariant

$$\text{(Goal)} \qquad \mathit{term}.q \quad \Rightarrow \quad (\ (r, s) \in \mathit{MST} \quad \equiv \quad s \in \{\mathit{ib}.r\} \cup \mathit{branch}.r\ )\ .$$

Initially, *term.q* is false for all nodes q. The objective is to preserve (Goal) and to prove that after a finite number of accepted messages there are no more messages in transit and *term.q* holds for all nodes q. The complete specification of the algorithm is as follows.

1. Predicate (Goal) is an invariant.
2. When all messages are disabled, *term.q* holds for all nodes q.
3. If *term.q* holds, all messages in transit to process q are enabled and equivalent to *skip*: process q accepts them and does nothing.
4. After a bounded number of atomic steps all messages are disabled.

Point 2 expresses local termination detection and the absence of deadlock. According to point 3, *term.q* indeed expresses termination of node q. It follows from 2 and 3 that, when all messages are disabled, there are no messages in transit anymore. Termination of the algorithm is expressed by point 4.

This concludes the specification of the distributed algorithm. It differs slightly from the description in [GHS83], where processes can wake up spontaneously or upon receiving messages from awakened neighbours. Moreover, we treat termination detection more explicitly.

2.3 Global description of the verified design

The verified simultaneous design of the algorithm and its proof mainly consists of a stepwise introduction and modification of messages with associated commands, in alternation with the introduction of invariants and proofs of invariance of predicates.

The invariants come in two flavours: constituent invariants and invariants that follow from other invariants. A priori, it is never known whether some invariant eventually will follow from other invariants. As a first guess we treat a new invariant as a constituent invariant unless we have reason to postpone the proof of invariance.

For the sake of brevity, we give in this paper only the outcome: the constituent invariants can be recognized from their names. They get names of the form (Jq0) where the J may be replaced by another capital and the 0 by another natural number. The capital serves to group the constituent invariants together in families of related predicates. These groups have no formal meaning, but only serve as a reminder of the stage (layer) where the invariant was introduced.

After the initialization, the abstract algorithm of Section 2.1 is a repetition of two actions: the determination of the lightest outgoing edge of a component and a corresponding extension of the forest. Since the forest is needed to determine components, we start in Chapter 3 with the design of a distributed data structure for the forest together with the means to modify it. For the latter purpose we introduce messages *wakeup*, *connect*, *change*, and a family of constituent invariants (Jq0) up to (Jq13). At this level, we also postulate four goal-directed constituent invariants (Iq0) up to (Iq3).

The actions of every component will be coordinated by a special pair of nodes, called the *core* of the component, see Chapter 3. We develop some special purpose graph theory in Chapter 4. In Chapter 5, we introduce private variables *ll*, *fnd*, *ci* and a message *init* to hold and distribute component information, generated at the core. Here we obtain constituent invariants in families (Kq) and (Lq), which are partly motivated by the results in Section 4.3.

In Chapter 6 we treat the task of the nodes of a component to search for the minimum weight outgoing edge. We first introduce messages *report* from the nodes of a component to its core, governed by a selfmessage *sendrep*, and a family of invariants (Mq). In order to determine the lightest outgoing edge, we introduce in Section 6.2 the possibility that a node of a component communicates with its neighbours to decide which neighbours belong to its own component. For this purpose we introduce messages *ask* and *answer*, a selfmessage *search*, and families of invariants (Nq) and (Oq).

In Chapter 7, the reports to the core are used to decide whether and where the forest must be extended, according to families (Pq), (Qq), and (Rq). The graph theory developed in Chapter 4 is used in Section 7.4 to ensure that the local decisions are globally correct. Here we prove the invariance of (Goal), our first proof obligation. We then also implement the decision at the core that the algorithm may terminate. We introduce messages *halt* to broadcast this decision, and invariants (Sq) to guarantee the correctness.

Chapter 8 is devoted to the situation where all messages are disabled. Our second proof obligation is to show that then every node q has *term.q*. Most cases of deadlock can be eliminated without modifying the algorithm, by means of a family of invariants (Tq). One specific source of deadlock, however, requires the introduction of the message *winit* with an associated family of invariants (Uq). Some more graph theory is needed together with a family (Vq) to finally settle the second proof obligation, in Section 8.7.

In Section 8.8, we perform a small program transformation to reduce some program variables to ghost variables by means of an invariant (Wq0). We use the remainder of the family (Wq) to settle the third proof obligation that every terminated node ignores all incoming messages, see Section 8.9.

For the fourth proof obligation (termination), we construct in Chapter 9 a variant function *vf* together with a family (Xq) to ensure that *vf* decreases with every step of the algorithm.

In Chapter 10, we present the resulting algorithm. It contains eleven private variables, four private ghost variables, and eleven messages: *wakeup*, *connect*, *change*, *init*, *report*, *sendrep*, *ask*, *answer*, *search*, *halt*, *winit*. Chapter 11 contains a comparison of our algorithm with the version of [GHS83] and a comparison of our approach with the one of [WLL88].

Chapter 12 describes some aspects of the mechanical proof, in particular as a witness for the correctness of the algorithm. The ideas and methods used to construct the proof are largely neglected.

An important function of the mechanical proof is book-keeping. In fact, in the proof of invariance of each of the constituent invariants, we use the hypothesis that a number of invariants holds in the precondition. With more than 160 invariants around there is the danger that one of them is used in some precondition but not proved to be invariant. In the mechanical proof we deal with this danger by constructing a predicate that is the conjunction of the universal quantification of all constituent invariants and by proving that it is, indeed, a strong invariant: it holds initially and is preserved under every step.

The algorithm as developed up to this point still contains four ghost variables. These are needed to express the global invariant, but, by inspection, are easily seen to be irrelevant for the computation. The last stage of the mechanical proof is the elimination of these ghost variables. This is described in Section 12.3.

3 Forest maintenance

In this chapter, we treat the modifications of forest F as it is represented by the pointer variables ib via formula (F0). In view of (F0), the invariant $F \subseteq MST$ mentioned above is expressed for the distributed algorithm as

(Iq0) $\qquad ib.q = q \quad \vee \quad (q, ib.q) \in MST$.

An immediate corollary of Theorem 3 is that the lightest incident edge of a node always belongs to MST. The determination of this edge requires no coordination between different nodes. The first action of each node q can therefore be to set $ib.q$ equal to its nearest neighbour. This action will be triggered by the message *wakeup*. Recall that, initially, a *wakeup* message is in transit to every node of the graph.

During the algorithm the variables ib will be modified while preserving postulate (Iq0), in such a way that eventually F as given by (F0) satisfies $F = MST$. In most cases, a modification of $ib.q$ would have the effect that one edge in F is replaced by another. Since we do not want to lose edges of F, we decide to modify $ib.q$ only under the precondition $ib.(ib.q) = q$. In fact, this is precisely the condition that no edge of F is lost.

The condition $ib.(ib.q) = q$ holds in two cases. The first case is $ib.q = q$. Then we say that node q is *sleeping*. After the acceptance by q of a first *wakeup* message, this will never be the case. In the other case, there is a pair of different nodes q and r with $ib.q = r$ and $ib.r = q$. Such a pair is called a *core*. In that case, either companion can dissolve the core by modifying its variable ib.

Using (F0) and the invariant that F is a forest, one can easily see that every component of F has a unique pair of nodes q and r with $ib.q = r$ and $ib.r = q$.

Let us assume that node q has accepted a *wakeup* message. Then we have $q \neq r$, i.e., the pair is a core. In Boruvka's algorithm, the set F is extended with the lightest outgoing edge of a component. Since we can modify relation F only at a core (or at a sleeping node), the core of a component must be moved to the lightest outgoing edge. This action will be triggered by the message *change*.

Whenever a node q modifies its variable *ib* by setting $ib.q := r$, it sends a message to r as a notification. There are two cases depending on the set variable *branch* introduced above. If $r \in branch.q$, process q knows that $ib.r = q$. So by setting $ib.q := r$ it establishes a core (q, r). Process r thus gets the possibility to modify its variable $ib.r$. In this case, process q sends a *change* message to r.

Alternatively, process q may set $ib.q := r$ while not knowing whether $ib.r = q$. Then the action of q seems to form a new connection between trees in the forest F. Node q then sends a *connect* message to r. The three messages *wakeup*, *change*, and *connect* form the first layer of the algorithm.

3.1 The first three messages

We first treat the message *wakeup*. If a sleeping node q accepts a *wakeup* message, it sets *ib* to its nearest neighbour. This neighbour is kept in the variable *be*, which stands for *best-edge*, see [GHS83]. Since node q apparently extends forest F by setting $ib.q := r$, it sends a *connect* message to r with its own name as argument. Therefore, as a first approximation, message *wakeup* is declared by

accept $(wakeup) =$
• **if** $ib = self$ **then**
  $ib := be$;
  $send(be, connect, self)$
 fi
end .

The bullet serves to separate the enabling condition from the command, but we omit the enabling condition here since message *wakeup* is always enabled. The variables mentioned are the private variables of the accepting process and *self* is the name of the accepting process. The test $ib = self$ tests whether the accepting process is sleeping. It follows that a *wakeup* message at a nonsleeping node is ignored.

As announced above, we introduce a private variable *branch* with the intention that $branch.r$ is the set of the nodes $q \neq ib.r$ for which process r "knows" that $r = ib.q$. So $branch.r$ holds the known children of node r. This leads to the invariant

(Jq0) $\qquad q \in branch.r \quad \Rightarrow \quad ib.q = r$.

It would be nice to have equivalence instead of implication in (Jq0), but then modification of $ib.q$ would require simultaneous modification of $branch.r$ and this is impossible. As a partial inverse of (Jq0), we postulate

(Jq1) $\quad q \in \mathit{branch}.(\mathit{ib}.q) \quad \vee \quad (\mathit{connect}, q) \ \textbf{at} \ \mathit{ib}.q \quad \vee \quad \mathit{ib}.(\mathit{ib}.q) = q$.

The third alternative expresses that q is sleeping or belongs to a core. Since we do not want to regard a parent as a child, we postulate

(Jq2) $\quad \mathit{ib}.q \notin \mathit{branch}.q$.

In order to preserve (Jq2) when node q accepts *wakeup*, we postulate that a sleeping node has no known children:

(Jq3) $\quad \mathit{ib}.q = q \quad \Rightarrow \quad \mathit{branch}.q = \emptyset$.

When a *connect* message is accepted, the receiver should add the sender to its set *branch*, unless the sender equals *ib*. In view of (Jq3), node q should not accept a *connect* message when $\mathit{ib}.q = q$. As a first approximation we therefore declare

accept $(\mathit{connect}, j) =$
  enabling $\mathit{ib} \neq \mathit{self}$
• **if** $j \neq \mathit{ib}$ **then** $\mathit{branch} := \mathit{branch} \cup \{j\}$ **fi**
end .

In order to preserve (Jq0) when node q accepts a *connect* message, we postulate the invariants

(Jq4) $\quad (\mathit{connect}, q) \ \textbf{at} \ r \quad \Rightarrow \quad \mathit{ib}.q = r$,
(Jq5) $\quad (\mathit{connect}, q) \ \textbf{not-at} \ q$.

These invariants are inspired by the design decision that the first argument of a *connect* message is always the name of the sender. Predicate (Jq5) is needed to preserve (Jq4) when q accepts *wakeup*.

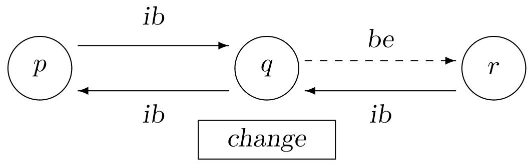
If there are no sleeping nodes, extension of forest F must happen at a core. This is accomplished by message *change*, which moves the core, or dissolves it while extending F.

accept $(\mathit{change}) =$
• **if** $\mathit{be} \in \mathit{branch}$ **then** $\mathit{send}(\mathit{be}, \mathit{change})$
  else $\mathit{send}(\mathit{be}, \mathit{connect}, \mathit{self})$ **fi** ;
  $\mathit{branch} := (\mathit{branch} \cup \{\mathit{ib}\}) \setminus \{\mathit{be}\}$;
  $\mathit{ib} := \mathit{be}$
end .

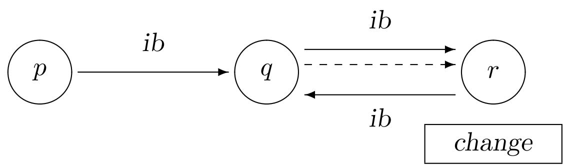
As announced above, we postulate that message *change* is always at a core, as expressed in the invariants

(Jq6) $\quad \mathit{change} \ \textbf{at} \ q \quad \Rightarrow \quad \mathit{ib}.q \neq q$,
(Jq7) $\quad \mathit{change} \ \textbf{at} \ q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.

Execution of *change* by q destroys this core by resetting $\mathit{ib}.q$. If $\mathit{be}.q \in \mathit{branch}.q$, it follows from (Jq0) that $(q, \mathit{be}.q)$ becomes a new core with a new *change* message **at** $\mathit{be}.q$. So the situation

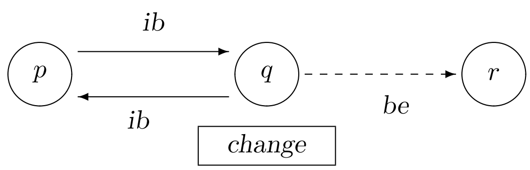


becomes

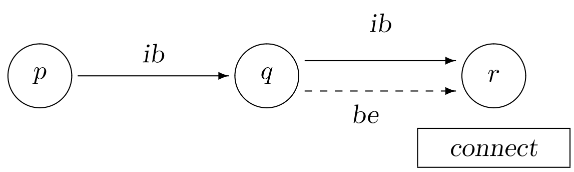


In other words, the message *change* pulls the core along the path of *be*–arrows. This process will be used in other layers to move the core to the lightest outgoing edge of the component.

On the other hand, if $be.q \notin branch.q$, then the core dissolves and a *connect* message is sent. Therefore, the situation



becomes



We could combine (Jq6) and (Jq7) into one predicate, but we don't do this, since that would have the drawback that in later applications of this invariant it would not be clear which of the two consequents is relevant.

Indeed, (Jq0) is preserved when $p \neq q$ accepts *change* because of (Jq7). It is preserved when q accepts *change* because of (Jq2), (Jq6), and (Jq7). Predicate (Jq1) is preserved by *change* because of (Jq0). Predicate (Jq2) is not threatened by *change*. Predicate (Jq3) is preserved by *change* if we postulate

(Dld0) $\quad be.q \neq q$.

Treatment of (Dld0) is postponed (delayed), since this predicate will later follow from another constituing invariant.

3.2 Some additional invariants

Predicate (Jq4) is preserved by *change* if we postulate

(Jq8) $\quad change \textbf{ at } q \quad \Rightarrow \quad (connect, q) \textbf{ not-at } r$.

Indeed, if one submits preservation of (Jq4) as a lemma to the theorem prover, the need for an invariant like (Jq8) is immediately apparent. It is up to the human designer to guess a predicate (Jq8), which can be kept invariant in the remainder of the design.

We now want to ensure that acceptance of *wakeup*, *connect*, and *change* preserves the predicates (Jq6), (Jq7), and (Jq8). Predicate (Jq6) is threatened only by *change*. It is preserved by *change* because of (Jq0) and (Dld0). Predicate (Jq7) is threatened when $p \neq q$ accepts *change* and when q accepts *wakeup* or *change*. It is preserved when q accepts *wakeup* because of (Jq6), and when q accepts *change* because of the new postulate

(Jq9) $\quad \mathit{change}\#q \leq 1$.

In fact, this implies that acceptance of *change* by q gives the postcondition that *change* is not at q. Predicate (Jq7) is preserved when $p \neq q$ accepts *change* because of (Jq0) and the new postulate

(Jq10) $\quad \mathit{change}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{change}\ \textbf{not-at}\ \mathit{ib}.q$.

Predicate (Jq8) is threatened when $p \neq q$ accepts *change* and when q accepts *wakeup* and *change*. It is preserved when q accepts *wakeup* because of (Jq6). It is preserved when q accepts *change* because of (Jq9). It is preserved when $p \neq q$ accepts *change* because of (Jq0), (Jq4), and the new postulate

(Jq11) $\quad (\mathit{connect}, r)\ \textbf{at}\ q \quad \Rightarrow \quad r \notin \mathit{branch}.q$.

We turn to the preservation of (Jq9), (Jq10), and (Jq11). Predicate (Jq9) is preserved because of (Jq0) and (Jq10), while (Jq10) is preserved because of (Jq0), (Jq7), and (Jq9). Predicate (Jq11) is preserved when $p \neq q$ accepts messages because of (Jq0), (Jq2), and (Jq7). It is preserved when q itself accepts messages because of the new postulates

(Jq12) $\quad \mathit{change}\ \textbf{at}\ q \quad \Rightarrow \quad (\mathit{connect}, \mathit{ib}.q)\ \textbf{not-at}\ q$,
(Jq13) $\quad (\mathit{connect}, r)\#q \leq 1$.

Predicate (Jq12) is preserved because of (Jq0), (Jq6), (Jq7), (Jq8), (Jq9), and (Jq10). Predicate (Jq13) is preserved because of (Jq4) and (Jq8). Details concerning these proofs can be found in the mechanical proof at [Hes@].

In this way, in the effort to prove the invariance of (Jq0), (Jq1), (Jq2), we have generated fourteen invariants, (Jq0) up to (Jq13), which form the bottom layer of the design. This bottom layer contains all modifications of the private variables *ib* and *branch*. The remainder of the algorithm is concerned with the value of *be* and the emergence of *change* messages. In the subsequent layers the declarations of *wakeup* and *change* are extended slightly, but the declaration of *connect* grows to more than half a page.

3.3 Goal directed invariants

We now come back to the main safety invariant (Iq0). It is clear that (Iq0) is only threatened when process q accepts a message *wakeup* or *change*. By inspection of the declarations of *wakeup* and *change*, one sees that predicate (Iq0) is preserved by these messages if we postulate

(Iq1) $\quad ib.q = q \quad\Rightarrow\quad (q, be.q) \in \mathit{MST}$,
(Ch-M) $\quad \mathit{change}$ **at** $q \quad\wedge\quad be.q \notin \mathit{branch}.q \quad\Rightarrow\quad (q, be.q) \in \mathit{MST}$.

Preservation of (Iq0) when $be.q \in \mathit{branch}.q$ and q accepts *change*, follows from (Jq0).

Since the purpose of a *change* message is to modify *ib*, it is erroneous when process q accepts *change* while $ib.q = be.q$. We therefore postulate

(Iq2) $\quad \mathit{change}$ **at** $q \quad\Rightarrow\quad be.q \neq ib.q$.

If process q accepts *wakeup* or *change*, it sends a message to $be.q$. We therefore need to know that there is an edge from q to $be.q$. So we postulate

(Iq3) $\quad w.(q, be.q) < \infty$.

By the convention about w, predicate (Iq3) implies (Dld0). We shall treat the invariance of (Iq1), (Iq2), and (Iq3) when modifications of *be* have been introduced. Predicate (Ch-M) will follow from other invariants.

Now the main task of the algorithm is to create enough *change* messages and (yet) to guarantee validity of (Iq1), (Iq2), (Iq3), and (Ch-M). At a later stage, we also have to prove the invariance of (Goal), to detect termination, and to create *halt* messages.

Remark. In [GHS83], the message *Change-root* (our message *change*) does not reset variable *in-branch*, which is our *ib*. Yet, on page 72 of [GHS83], it is stated that the message *Change-core* has the effect that “the inbound edge . . . is changed to correspond to *best-edge*”. This suggests that the version of [GHS83] in which *in-branch* is modified by *Initiate* is due to a late program transformation. For us, the realization that message *change*, rather than *init*, should modify *ib* was the breakthrough that enabled us to construct the layered proof.

4 Graph Theory

In this chapter we formalize the main graph theoretical concepts and results that are used in the proof of the algorithm. All theorems mentioned below have been proved mechanically, though not always in the way described here.

4.1 Reflexive transitive closures and graphs

For any binary relation R, we write R^* to denote the reflexive transitive closure. We first record a triviality, which is yet so fundamental that it is worth mentioning.

Theorem 0. Let R be a binary relation on a set V. Let φ be a boolean function on V that satisfies $\varphi.x \Rightarrow \varphi.y$ for all pairs $(x, y) \in R$. Then $\varphi.x \Rightarrow \varphi.y$ for all pairs $(x, y) \in R^*$.

It is not hard to prove this result on our theorem prover, but the prover has no heuristics to guess such a result as a subgoal for other theorems. Whenever we use this result, we do so by means of this theorem with some instantiation for R and φ.

If R is a binary relation, an R–path from q to r is defined to be a sequence $(x_0, \ldots, x_k)$ with $k \geq 0$ and $q = x_0$ and $r = x_k$, and $(x_i, x_{i+1}) \in R$ for all $0 \leq i < k$. The number k is called the length of the path. The path is called *simple* if all elements x_i differ. A pair (q, r) belongs to R^* if and only if there exists a simple R–path from q to r.

For our purposes, an undirected graph has no self-loops and no multipe edges. So it can be modeled as a pair (V, E) where V is the finite set of nodes and E is a binary relation on V with

$$(q, r) \in E \quad \Rightarrow \quad q \neq r \quad \wedge \quad (r, q) \in E \ .$$

The graph is said to be *connected* if relation E^* holds for all pairs of nodes.

A *subgraph* of graph (V, E) is a subset $F \subseteq E$ such that (V, F) is a graph. The subgraph is called *spanning* if the graph (V, F) is connected. A subgraph is called a *spanning tree* if it is spanning and has no cycles.

In a *weighted* undirected graph, every edge has an associated weight, which is a real number given by a function $w \in E \to \mathbb{R}$. Since the graph is undirected, we assume that the weight function is symmetric, i.e., that $w.(x, y) = w.(y, x)$ for all nodes x and y. In order to eliminate the set E from our considerations we extend w to a function $V \times V \to \mathbb{R} \cup \{\infty\}$ by defining

$$w.(x, y) = \infty \quad \equiv \quad (x, y) \notin E \ .$$

For all nodes x we have $w.(x, x) = \infty$, since the graph has no selfloops.

4.2 Minimum–weight spanning subtrees

The weight of a subgraph F is defined as the sum of the weights of the edges in F. Clearly, every connected graph has at least one spanning subtree of minimum weight.

If different edges may have equal weights, the graph may have more than one minimum-weight spanning subtree. In that case every algorithm for the *distributed* determination of a minimum-weight spanning tree would have some kind of consensus problem. Consider, for example, the case of four vertices in a rectangle with sides of different lengths (weights): there are two minimum-weight spanning trees, but a symmetric deterministic algorithm cannot make the choice.

Following [GHS83], we postulate that all edges have different weights, in other words that, for all nodes q, r, x, y, (A0) $\qquad w.(q,r) = w.(x,y) < \infty \quad \Rightarrow \quad (q,r) = (x,y) \quad \vee \quad (q,r) = (y,x)$.

Finally we postulate that the graph is connected, i.e., that $(x,y) \in E^*$ for all nodes x, y.

There are many ways to formalize the concept of minimum–weight spanning tree. In view of our goal to construct a mechanical proof, we have chosen to define relation *MST* as the set of the edges that have no connection through lighter edges. So, it is formally defined by

(G0) $\qquad (q,r) \in MST \quad \equiv \quad (q,r) \in E \quad \wedge \quad (q,r) \notin H.(q,r)^*$,

where $H.(q,r)^*$ is the reflexive transitive closure of relation $H.(q,r)$ given by

$$(x,y) \in H.(q,r) \quad \equiv \quad w.(x,y) < w.(q,r) \ .$$

We first prove that *MST* is a forest:

Theorem 1. Let $(q,r) \in MST$. Then every simple *MST*–path from q to r has length 1.

Proof. Suppose not. Then the subgraph *MST* contains a cycle, i.e., a path from some node to itself that consists of different edges. Since all edges in E have different weights, the cycle has a unique edge of maximal weight, say (q,r). It follows that q and r are connected by the remainder of the cycle, which consists of edges of weight less than $w.(q,r)$. This implies $(q,r) \in H.(q,r)^*$ and hence $(q,r) \notin MST$, a contradiction. □

The theorem is phrased without the concept of cycles since this concept is not used in the mechanical proof, and also since it is not useful for the application, which is the next theorem.

The result implies that *MST* is a subforest of the graph. Now it is not hard to argue that, if the graph is connected, *MST* is the unique minimum–weight spanning tree. These arguments have not been formalized, however, in the mechanical proof. We regard the determination of *MST* as it is defined here, as the goal of the algorithm.

In the proof of the algorithm, we need the following corollary.

Theorem 2. Let R be a binary relation on V with $R \subseteq MST$. Then $R^* \cap MST \subseteq R$.

Proof. Let $(q,r) \in R^* \cap MST$. Then there is a simple R–path from q to r. Since $R \subseteq MST$, this path is an *MST*–path. Theorem 1 implies that the path has length 1. This yields $(q,r) \in R$. □

Corollary of Theorem 2. Let $R \subseteq MST$ be spanning. Then $R = MST$.

Proof. This follows from Theorem 2, since all pairs belong to R^*. □

Membership of *MST* is proved by means of the following criterion

Theorem 3. Let $(q, r) \in E$ and let f be a function on V such that $f.q \neq f.r$ and that $f.x = f.y$ for all pairs x, y with $w.(x, y) < w.(q, r)$. Then $(q, r) \in MST$.

Proof. By Theorem 0, function f is constant on the components of the graph $H.(q, r)$. Therefore, $f.q \neq f.r$ implies $(q, r) \notin H.(q, r)^*$. This proves $(q, r) \in MST$. $\square$

Corollary of Theorem 3. Let q, $r \in V$ be such that r is a nearest neighbour of q (i.e. that $w.(q, r) \leq w.(q, x)$ for all $x \in V$). Then $(q, r) \in MST$.

Proof. For all pairs x, y, we have

$$w.(x, y) < w.(q, r) \quad \Rightarrow \quad x \neq q \quad \wedge \quad y \neq q \ .$$

Therefore Theorem 3 applies with $f \in V \to \mathbb{B}$ given by $f.x = (x \neq q)$. $\square$

In the proof of the algorithm we also need the following result. Let $h \in V \to V$ be a function. As usual, $h^n.x$ is defined by $h^0.x = x$ and $h^{n+1}.x = h.(h^n.x)$. So $h^n.x$ is the result of n subsequent applications of h to x.

Theorem 4. Let $v \in V$. Assume that $(h^i.v, h^{i+1}.v) \in MST$ for every number i with $h^i.v \neq h^{i+1}.v$. Then there exists a number n such that $h^{n+2}.v = h^n.v$.

Proof. Since V is a finite set, there are natural numbers $p > 0$ and n such that $h^{n+p}.v = h^n.v$. Moreover, we may assume that p is minimal, i.e., that $h^{n+i}.v \neq h^n.v$ for all i with $0 < i < p$. This assertion is sometimes called the figure–six theorem (the mechanical proof is quite an effort since it involves counting the set V in different ways).

If $p = 1$ then $h^{n+1}.v = h^n.v$ and hence $h^{n+2}.v = h^n.v$. So, assume $p > 1$. Using induction we get $h^{m+p}.v = h^m.v$ for all $m \geq n$. We also get $h^{n+i}.v \neq h^{n+j}.v$ for all i, j with $0 \leq i < j < p$. So, the vertices $a_i = h^{n+i}.v$ with $0 \leq i < p$ form a simple MST–path from a_0 to a_{p-1}. We also have $(a_0, a_{p-1}) \in MST$. Now Theorem 1 implies that $p = 2$. $\square$

4.3 Components of a forest

Distribution of the algorithm creates two problems: the global state is distributed so that local modifications are not known elsewhere, and control is distributed so that coordinated action requires some kind of consensus between processes.

According to the above analysis the subgraph F is always a forest. So the connected components of (V, F) are trees. We organize coordination in these trees by making them into directed trees. Now the choice of a root node within a tree would introduce a consensus problem if the tree has more than one node. Therefore trees with more than one node are directed towards a special edge, the core. The determination of a core also gives a consensus problem, but now the weight function on the edges can be used to break the symmetry.

Let us assume that the set V is made into a forest by means of a function $g \in V \to V$, which serves as an abstraction of (a variation of) the private variables *ib*. We regard $g.q$ as the parent of q unless $g.(g.q) = q$.

Analogously to (F0), function g induces a symmetric binary relation G given by

(G1) $\qquad (q,r) \in G \quad \equiv \quad q \neq r \quad \wedge \quad (q = g.r \quad \vee \quad r = g.q)$.

One can verify that condition (G1) implies that each connected component of G has at most one cycle.

The connected components of graph G are the equivalence classes with respect to the reflexive transitive closure G^* of G. We claim that

(G2) $\qquad (q,r) \in G^* \quad \equiv \quad (\exists\, m, n \in \mathbb{N} :: g^m.q = g^n.r)$,

where as usual $g^m.x$ is the result of m subsequent applications of function g to x. As for the claim, it is easy to see that the righthand side of (G2) defines an equivalence relation, which implies G^* and is implied by G. Since G^* is the strongest equivalence relation implied by G, this proves formula (G2).

One of the main tasks in the algorithm is that each connected component of G should determine an outgoing edge of minimal weight. So, every node should enquire whether its neighbours belong to the same component. Therefore, all nodes get a private variable *cc* to hold the current component identity. Upon creation of a new component, a new component identity will be created and broadcast through the component. In order to decide that the value of *cc* is sufficiently recent, we add a version number *ll* (for *level*). In order to indicate that a node should be active in looking for outgoing edges we add a boolean variable *fnd* (for *finding*).

In a given global state of the system, the private variables *fnd*, *ll*, and *cc* may be regarded as functions $fnd \in V \to \mathbb{B}$, $ll \in V \to \mathbb{N}$, and $cc \in V \to W$, where W is the set of component names. It is the intention that we have

(G3) $\qquad cc.q = cc.r \quad \Rightarrow \quad (q,r) \in G^*$.

The converse implication cannot be expected since it takes a number of steps to transfer the component identity from the core of the component to the outskirts. In Theorem 5 below, however, we give a kind of converse implication.

It will turn out that, roughly speaking, each node q obtains its component identity together with the level and the indication to find an outgoing edge from its parent $g.q$. This may suggest the antecedents of the following result.

Theorem 5. Assume that we have, for all nodes q :

(a) $\qquad ll.q \quad \leq \quad ll.(g.q)$,
(b) $\qquad ll.q = ll.(g.q) \quad \Rightarrow \quad cc.q = cc.(g.q)$,
(c) $\qquad fnd.q \quad \Rightarrow \quad ll.q = ll.(g.q)$,
(d) $\qquad fnd.q \quad \Rightarrow \quad fnd.(g.q) \quad \vee \quad g.(g.q) = q$.

Then it follows that, for all q, $r \in V$,

$$
\begin{aligned}
&(q,r) \in G^* \quad \wedge \quad (\mathit{fnd}.q \quad \vee \quad g.(g.q) = q) \\
&\quad \Rightarrow \quad \mathit{ll}.r \le \mathit{ll}.q \quad \wedge \quad (\mathit{ll}.r = \mathit{ll}.q \Rightarrow \mathit{cc}.r = \mathit{cc}.q) \ .
\end{aligned}
$$

Proof. We define the (lexical coupling) relation $\sqsubseteq$ by

$$x \sqsubseteq y \quad \equiv \quad \mathit{ll}.x \le \mathit{ll}.y \quad \wedge \quad (\mathit{ll}.x = \mathit{ll}.y \Rightarrow \mathit{cc}.x = \mathit{cc}.y) \ .$$

It is easy to see that relation $\sqsubseteq$ is reflexive and transitive. So it is a preorder. The formulae (a) and (b) imply that $x \sqsubseteq g.x$ for every x. We define the boolean function φ on V by $\varphi.x = (\mathit{fnd}.x \vee g.(g.x) = x)$. It follows from (a), (b), (c), and (d) that, for every x,

(*) $$
\begin{aligned}
&\varphi.x \quad \Rightarrow \quad g.x \sqsubseteq x \ , \text{ and} \\
&\varphi.x \quad \Rightarrow \quad \varphi.(g.x) \ .
\end{aligned}
$$

Now consider q and r with $\varphi.q$ and $(q,r) \in G^*$. Using induction we get $r \sqsubseteq g^n.r$ for all $n \in \mathbb{N}$. Using induction and the two formulae at (*), we also get $g^m.q \sqsubseteq q$ for all $m \in \mathbb{N}$. Since $(q,r) \in G^*$, it follows from (G2) that there exist m and n such that

$$r \sqsubseteq g^n.r = g^m.q \sqsubseteq q \ .$$

This implies $r \sqsubseteq q$, whence the assertion. □

Remark. The antecedents, in particular condition (d), are highly unintuitive. Yet this theorem is an essential ingredient of our proof of the algorithm. The four antecedents are crucial invariants that cannot be strengthened. □

The final result of this chapter is a kind of strengthening of Theorem 0 in the situation of graph G given by function g. This result will be used in Section 7.4 to show that, if the core members of a component are exchanging *report* messages with best-weight values $\mathit{bw} \ge v$, then all nodes of the component have $\mathit{bw} \ge v$.

Theorem 6. Assume that predicate φ satisfies for all nodes q:

$$g.(g.q) \notin \{q, g.q\} \quad \wedge \quad \varphi.(g.q) \quad \Rightarrow \quad \varphi.q \ .$$

Let $p \in V$ be such that

$$g.(g.p) = p \quad \wedge \quad g.p \neq p \quad \wedge \quad \varphi.p \quad \wedge \quad \varphi.(g.p) \ .$$

Then every node $q \in V$ satisfies $\quad (p,q) \in G^* \quad \Rightarrow \quad \varphi.q$.

Proof. By induction in n, and using $g.(g.p) = p$ and $g.p \neq p$, we first prove

$$
\begin{aligned}
&g^n.q \in \{p, g.p\} \quad \wedge \quad g.(g.q) = q \quad \Rightarrow \quad q \in \{p, g.p\} \ , \\
&g^n.q \in \{p, g.p\} \quad \Rightarrow \quad g.(g.q) \neq g.q \ .
\end{aligned}
$$

Using this and the assumptions about φ, we prove by induction in n that $g^n.q \in \{p, g.p\}$ implies $\varphi.q$. Then the assertion follows from (G2). □

5 Connected components of a changing graph

We come back to the distributed algorithm. According to Section 2.1, every component has the task to determine its lightest outgoing edge. For this purpose the nodes of a component must decide whether neighbours belong to the same component. The question whether nodes belong to the same component, however, can be influenced by actions of other nodes. So we need invariants to prove that the decision is taken correctly.

For this purpose we introduce private variables and state functions that satisfy the antecedents of Theorem 5 of Section 4.3. We found these conditions as the strongest predicates that we could keep invariant, but a presentation of the design where these predicates emerge as invariants before being used as antecedents in Theorem 5, was less satisfactory.

Remark. In the remainder of this paper we silently skip most of the proofs of invariance when the ingredients are invariants claimed previously. The information lacking can always be recovered from the mechanical proof: for each constituent invariant `iq`, the input for the prover contains a lemma with the name `iq-kept-valid`, see the proofs at [Hes@].

We can only offer the final input, not the input needed at a specific stage of the design. Therefore, e.g., if one inspects the proof of `jq3-kept-valid`, one sees an invariant (Lq2), which is needed later when the message *connect* is slightly modified.

5.1 Marking trees

In this section we extend the algorithm with private variables *ci*, *ll*, and *fnd*, for component information. The variables *ci* serve to define a variation of *cc* of Section 4.3. We use a new message *init* to transfer component information to the nodes of the component.

The command to register and transfer new component information is given by

$$\begin{array}{l}\textbf{proc}\ \ \mathit{initp}\ (v, \mathit{id}) = \\ \quad \mathit{fnd} := \mathit{true}\ ;\ \mathit{ll} := v\ ;\ \mathit{ci} := \mathit{id}\ ; \\ \quad \mathit{mcast}\ (\mathit{branch}, \mathit{init}, v, \mathit{id}) \\ \textbf{end}\ \textbf{.}\end{array}$$

If process q executes command *initp*, it sends *init* messages to all elements of *branch*.q. To allow this we need the property that $w.(q, r) < \infty$ for all nodes $r \in \mathit{branch}.q$. This property follows from (Jq0), (Jq2), (Iq0), and the symmetry of w.

A new component identity is created when process q "learns" the validity of $\mathit{ib}.(\mathit{ib}.q) = q$ by acceptance of a *connect* message from *ib*.q, compare (Jq4). Following [GHS83], we define the new component identity as the weight of the core. The new component identity is accompanied by an incremented version number *ll*. We thus redefine

$$
\begin{array}{l}
\textbf{accept}\ (\mathit{connect}, j) = \\
\quad \textbf{enabling}\ \mathit{ib} \neq \mathit{self} \\
\bullet\ \ \textbf{if}\ \ j = \mathit{ib}\ \ \textbf{then}\ \ \mathit{initp}(\mathit{ll}+1, w.(\mathit{self}, j)) \\
\quad \textbf{else}\ \ \mathit{branch} := \mathit{branch} \cup \{j\}\ \ \textbf{fi} \\
\textbf{end}\ .
\end{array}
$$

The message *init* is now defined by

$$
\begin{array}{l}
\textbf{accept}\ (\mathit{init}, v, \mathit{id}) = \\
\bullet\ \ \mathit{initp}\ (v, \mathit{id}) \\
\textbf{end}\ .
\end{array}
$$

Since the modifications only introduce the new message *init* and only modify the new variables *fnd*, *ll*, *ci*, they do not endanger any of the invariants (Jq) of Chapter 3.

The remainder of this section is devoted to assertions analogous to the antecedents of Theorem 5 of Section 4.3. For the moment we take function g to be given by *ib* and we add the assumption $\mathit{ib}.(\mathit{ib}.q) \neq q$. We first postulate

(Kq0) $\quad \mathit{ib}.(\mathit{ib}.q) \neq q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{fnd}.(\mathit{ib}.q)$,
(Kq1) $\quad \mathit{ib}.(\mathit{ib}.q) \neq q \quad \Rightarrow \quad \mathit{ll}.q \leq \mathit{ll}.(\mathit{ib}.q)$.

We proceed to show that (Kq0) is invariant. Predicate (Kq0) is threatened when q receives *connect*, *init*, *wakeup*, or *change*, and also when $p \neq q$ receives *change*. It is preserved by *connect* because of (Jq4). It is preserved by *change* if we postulate

(Kq2) $\quad \mathit{change}\ \textbf{at}\ q \quad \Rightarrow \quad \neg \mathit{fnd}.q$,
(Kq3) $\quad \mathit{change}\ \textbf{at}\ \mathit{ib}.q \quad \Rightarrow \quad \neg \mathit{fnd}.q$.

It is preserved by *init* and *wakeup* if we postulate

(Kq4) $\quad \mathit{init}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{fnd}.(\mathit{ib}.q)$,
(Kq5) $\quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) \neq \mathit{ib}.q$.

Notice that (Kq3), (Jq7), and (Kq4) together imply

(In*Ch) $\quad \mathit{init}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{change}\ \textbf{not-at}\ q$.

Now, using the invariants (Jq), one can show that the conjunction of (Kq0), (Kq2), (Kq3), (Kq4), and (Kq5) is indeed invariant.

In order to show that (Kq1) is preserved by *wakeup* and *change*, we need the postulates

(Dld1) $\quad \mathit{ib}.q = q \quad \Rightarrow \quad \mathit{ll}.q \leq \mathit{ll}.(\mathit{be}.q)$,
(Dld2) $\quad \mathit{change}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{ll}.q \leq \mathit{ll}.(\mathit{be}.q)$,
(Dld3) $\quad \mathit{change}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{ll}.(\mathit{ib}.q) \leq \mathit{ll}.q$.

In order to show that predicate (Kq1) is preserved by *init*, we need the postulates

(Kq6) $\quad (\mathit{init}, v)\ \textbf{at}\ q \quad \Rightarrow \quad v = \mathit{ll}.(\mathit{ib}.q)$,
(Kq7) $\quad \mathit{init}\ \textbf{at}\ q \quad \Rightarrow \quad \mathit{ll}.q < \mathit{ll}.(\mathit{ib}.q)$.

The treatment of the postulates (Dld1), (Dld2), and (Dld3) is postponed; (Dld1) and (Dld2) have to wait for the treatment of variable *be* in a later layer, (Dld3) must wait for a stronger invariant below.

In order to show that (Kq6) and (Kq7) are preserved, we also postulate

(Kq8) $\quad init\#q \leq 1$,
(Kq9) $\quad init$ **at** $q \quad \Rightarrow \quad \neg fnd.q$,
(Kq10) $\quad (connect, ib.q)$ **at** $q \quad \Rightarrow \quad \neg fnd.q$.

We also need the following postulates

(In*cr) $\quad init$ **at** $q \quad \Rightarrow \quad ib.(ib.q) \neq q$,
(In*CC) $\quad init$ **at** $q \quad \Rightarrow \quad (connect, ib.q)$ **not-at** q ,
(Dld4) $\quad fnd.q \quad \Rightarrow \quad ib.(ib.q) = q \quad \vee \quad q \in branch.(ib.q)$.

In fact, preservation of (Kq6) up to (Kq10) follows from the predicates postulated up to this point.

The postulates (In*cr) and (In*CC) follow from (Jq2), (Jq4), and the new postulate

(In*br) $\quad init$ **at** $q \quad \Rightarrow \quad q \in branch.(ib.q)$.

The postulates (Dld4) and (In*br) will be treated later. They may be regarded as strengthenings of (Jq1) under specific circumstances. They do not follow from (Jq1).

We are now ready to extend predicate (Kq1) with

(Kq11) $\quad (connect, q)$ **not-at** $ib.q \quad \Rightarrow \quad ll.q \leq ll.(ib.q)$.

It is easy to see that (Dld3) follows from (Jq7), (Jq12), and (Kq11), the latter with $q := ib.q$. Before treating its invariance, we notice that (Kq11) implies that the two companions of a "mutually recognized" core have the same level:

$$ib.(ib.q) = q \ \wedge \ (connect, q) \ \textbf{not-at} \ ib.q \ \wedge \ (connect, ib.q) \ \textbf{not-at} \ q$$
$$\Rightarrow \quad ll.(ib.q) = ll.q \ .$$

In view of the form of *connect*, and the invariant (Jq4), we therefore also postulate that, when one *connect* message has been accepted and the other one is still pending, the difference of the levels is one:

(Kq12) $\quad (connect, q)$ **not-at** $ib.q \quad \wedge \quad (connect, ib.q)$ **at** q
$\qquad \Rightarrow \quad ll.(ib.q) = 1 + ll.q$.

Consequently, when both *connect* messages are still pending, the levels have to be equal:

(Kq13) $\quad (connect, q)$ **at** $ib.q \quad \wedge \quad (connect, ib.q)$ **at** $q \quad \Rightarrow \quad ll.(ib.q) = ll.q$.

Remark. Here we have the first occurrence of what we regard as an asynchronous handshake: some process q sends a message (kw, q) to r and the reaction of r depends on whether there is also a similar message (kw, r) of r to q that has not yet been acknowledged. In this case $kw = connect$. The GHS algorithm contains two other asynchronous handshakes, with $kw = ask$ and $kw = report$. Since the messages are asynchronous, these handshakes always require some intricate invariants. □

It turns out to be possible to prove the invariance of (Kq11), (Kq12), (Kq13) with the present ingredients. For the proof of (Kq13), we use the observation that (Jq4), (Jq11), and (In*br) combine to imply

(In*C) $\quad init$ **at** $q \quad \Rightarrow \quad (connect, q)$ **not-at** r .

Because of condition (c) of Theorem 5 in Section 4.3, we also prove the invariance of

(Kq14) $\quad fnd.q \quad \Rightarrow \quad ll.(ib.q) \leq ll.q$.

5.2 Identifying trees

In this section we reap the fruits of the previous section. It turns out that the antecedents of Theorem 5 cannot be realized if we take function g given by ib. Instead of this, we define the state functions $jb.q$ by

$$jb.q \quad = \quad (\ \textbf{if} \ \ (connect, q) \ \textbf{at} \ ib.q \ \ \textbf{then} \ \ q \ \ \textbf{else} \ \ ib.q \ \ \textbf{fi}\ ) \ .$$

The theory of Section 4.3 is now applied with jb for function g. We write JB for the symmetric binary relation G given by (G1) with jb for g.

We now verify the conditions of Theorem 5 of Section 4.3. It is easy to see that postulate (Kq11) implies condition (a) and that (Kq11) and (Kq14) together imply condition (c). Condition (d) is expressed in

(Fn*jb) $\quad fnd.q \quad \Rightarrow \quad fnd.(jb.q) \quad \vee \quad jb.(jb.q) = q$.

This follows from the conjunction of (Kq0) and (Kq10), as is proved in

$$\begin{array}{ll}
 & fnd.q \ \ \wedge \ \ \neg fnd.(jb.q) \\
\equiv & \{\text{definition of } jb\} \\
 & fnd.q \ \ \wedge \ \ \neg fnd.(ib.q) \ \ \wedge \ \ (connect, q) \ \textbf{not-at} \ ib.q \\
\Rightarrow & \{\text{(Kq0) and (Kq10)}\} \\
 & ib.(ib.q) = q \ \ \wedge \ \ (connect, ib.q) \ \textbf{not-at} \ q \\
 & \wedge \ \ (connect, q) \ \textbf{not-at} \ ib.q \\
\Rightarrow & \{\text{definition of } jb\} \\
 & jb.(jb.q) = q \ .
\end{array}$$

Condition (G3) of Section 4.3 requires that initially all values $cc.q$ are different. It turns out that these initial values have no algorithmic impact. We therefore define a state function Ci by

$$Ci.q \quad = \quad (\ \mathbf{if}\ \ ll.q = 0\ \ \mathbf{then}\ \ \{q\}\ \ \mathbf{else}\ \ ci.q\ \ \mathbf{fi}\ ) \ .$$

Since *ci* is a number, *Ci* is a variable of a union type. The theorem prover NQTHM is untyped and, hence, handles this without problems.

We let *Ci* play the role of *cc* in Section 4.3. Now condition (b) of Theorem 5 is equivalent to the invariant

(Lq0) $\qquad (connect, q)\ \mathbf{not\text{-}at}\ ib.q \quad \wedge \quad ll.(ib.q) = ll.q \quad \Rightarrow \quad Ci.(ib.q) = Ci.q \ .$

In order to preserve (Lq0) when q accepts *init*, we postulate an analogue of (Kq6):

(Lq1) $\qquad (init, -, id)\ \mathbf{at}\ q \quad \Rightarrow \quad id = Ci.(ib.q) \ .$

Predicate (Lq0) is violated when process $p = ib.q$ accepts a *connect* message from q while $ci.p \neq ci.q$ and $ll.p = ll.q$ and $q \neq ib.p$. Following [GHS83], we therefore disable acceptance of $(connect, q)$ by process p when $q \neq ib.p$ and $ll.p \leq ll.q$. Since $ll.q$ is not known to process p, we let every *connect* message carry the level of the sender as a second argument. So the commands to send *connect* in *wakeup* and *change* are replaced by

$$send(be, connect, self, ll) \ ,$$

and acceptance of *connect* is redefined

accept $(connect, j, v) =$
  enabling $j = ib \quad \vee \quad v < ll$
• **if** $j = ib$ **then** $initp\ (ll + 1, w.(j, self))$
  else $branch := branch \cup \{j\}$ **fi**
end .

Here we eliminate the disabling condition $ib = self$. For this purpose we introduce the new invariant

(Lq2) $\qquad ib.q = q \quad \Rightarrow \quad ll.q = 0 \ .$

Remark. For the mechanical test $v < ll$ we use NQTHM's function `lessp`, which yields *false* if $ll = 0$, independently of v. Thus, silently, we also introduce the invariant that all numbers in the algorithm are ≥ 0. □

We now come back to the proof of invariance of (Lq0). The disabling of *connect* only makes sense if we also postulate

(Lq3) $\qquad (connect, q, v)\ \mathbf{at}\ ib.q \quad \Rightarrow \quad ll.q = v \quad \vee \quad ib.(ib.q) = q \ .$

The second disjunct of the consequent of (Lq3) may be somewhat disappointing, but when we delete it we cannot prove invariance. It turns out that (Lq3) as it stands is strong enough.

In order to preserve (Lq0) when q accepts a message *init* or *connect*, we postulate

(Lq4) $\quad (init, v)$ **at** $q \quad \Rightarrow \quad v > 0$,
(Lq5) $\quad ll.(ib.q) < ll.q \quad \Rightarrow \quad Ci.q = w.(ib.q, q)$.

The proofs of invariance of (Lq1) up to (Lq5) are similar to earlier proofs. In this way we get condition (b) in the intended application of Theorem 5 of Section 4.3. So, Theorem 5 is applicable. It turns out that we only need its corollary

(Thm5) $\quad (q, r) \in JB^* \quad \wedge \quad jb.(jb.q) = q$
$\Rightarrow \quad ll.r \leq ll.q \quad \wedge \quad (ll.r = ll.q \Rightarrow Ci.r = Ci.q)$.

We now want to prove the invariance of the analogue of (G3), i.e.,

(Lq6) $\quad Ci.q = Ci.r \quad \Rightarrow \quad (q, r) \in JB^*$.

We first use (Jq7) and (Jq12) to prove that, for every pair of nodes x, y, the predicate $(x, y) \in JB$ is stable (once true, it remains true). It follows that

(Stab) $\quad$ predicate $(x, y) \in JB^*$ is stable.

Because of (Stab), predicate (Lq6) is threatened only when process q or r accepts an *init* message or a *connect* message. In order to show that acceptance by p of $(connect, ib.p)$ preserves (Lq6) for $p = r$ (or $p = q$), we postulate the new invariant

(Lq7) $\quad Ci.q = w.(r, s) \quad \Rightarrow \quad (q, r) \in JB^*$.

Because of (Stab), predicate (Lq7) is threatened only when process q accepts *init* or *connect*. The critical case is acceptance by process q of $(connect, ib.q)$. Here we need that $(q, ib.q)$ is an edge of graph (V, E) by (Iq0), and that all edges of the graph have different weights, see axiom (A0) in Section 4.2. We also need the observation that, if process p accepts a *connect* message from q, the postcondition $(q, p) \in JB^*$ is established. This follows from (Jq4) and (Jq13).

Later on it will turn out to be convenient to know that equality of *Ci* implies equality of *ll*:

(Lq8) $\quad Ci.q = Ci.r \quad \Rightarrow \quad ll.q = ll.r$.

In order to show that (Lq8) is preserved by *connect*, we postulate

(Lq9) $\quad Ci.q = w.(r, ib.r) \quad \wedge \quad (connect, ib.r)$ **at** $r \quad \Rightarrow \quad ll.q = 1 + ll.r$.

In order to show that (Lq9) is preserved we also postulate

(Lq10) $\quad Ci.q = w.(r, s) \quad \Rightarrow \quad r \in branch.s \quad \vee \quad r = ib.s$,
(Lq11) $\quad Ci.q \neq \infty$.

Remark. Of course, originally, we introduced the invariants with *ci* instead of *Ci*. This gave the problem to establish (Lq6) initially. Since we did not want to use a union type in the algorithm without algorithmic necessity, we later replaced *ci* by *Ci*.

6 The investigation of outgoing edges

When a node p receives a new component identity, it gets the task to participate in the search for the minimum-weight outgoing edge of the component. This task is separated into a local task to search the neighbours and a communal task to wait for reports from the children, to compare these reports with the local result, and finally to send a report to the parent. We first treat the separation and the communication structure for the communal task. The local search is treated next. It is more difficult and contains a nasty optimization.

6.1 Collecting and sending reports

For the separation of the tasks, we extend procedure *initp* with a selfmessage *search*. We use an auxiliary boolean variable *srch* to express that the node is in the process of determining a local outgoing edge of the component. When a node p has determined the optimal outgoing edge of its subtree, it reports this fact to its parent *ib.p* by sending a *report* message.

In order to verify that a node has accepted all expected *report* messages, we introduce a variable *explist* to hold the set of nodes from which reports are expected. The two core members also send reports to each other. Treatment of these special reports is postponed: for the moment they are simply disabled.

We extend *initp* with a selfmessage *sendrep* to send the *report* message when the search is completed and all reports have been received.

Thus, procedure *initp* is redefined by

proc *initp* $(v, id) =$
  $ll := v$; $ci := id$;
  $delay\ (sendrep)$;
  $delay\ (search)$;
  $fnd := true$; $srch := true$;
  $explist := branch$;
  $mcast\ (branch, init, v, id)$
end .

We define acceptance of *sendrep* by

accept $(sendrep) =$
  enabling $\neg srch \wedge (explist = \emptyset)$
• $fnd := false$;
  $send\ (ib, report, self)$
end .

For the moment, selfmessage *search* is disabled and message *report* is defined by

accept $(report, j) =$
  enabling $j \neq ib$
• $explist := explist \setminus \{j\}$
end .

In this way, we only determine the communication flow. The contents of the communication will be treated later.

The messages introduced endanger the old invariants only by the assignment $\mathit{fnd} := \mathit{false}$ in *sendrep*. So they only endanger the old invariants that contain positive occurrences of *fnd*. These are (Kq0) and (Kq4). In order to preserve (Kq0) under *sendrep*, we postulate

(Mq0) $\quad ib.(ib.q) \neq q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad q \in \mathit{explist}.(ib.q)$,
(Mq1) $\quad \mathit{init}$ **at** $q \quad \Rightarrow \quad q \in \mathit{explist}.(ib.q)$.

In order to preserve (Mq0) and (Mq1) when $ib.q$ accepts *report*, we need the new postulates

(Mq2) $\quad \mathit{fnd}.q \quad \Rightarrow \quad (\mathit{report}, q)$ **not-at** $ib.q$,
(Mq3) $\quad \mathit{init}$ **at** $q \quad \Rightarrow \quad (\mathit{report}, q)$ **not-at** $ib.q$.

In order to preserve (Mq2) under *connect*, we postulate

(Mq4) $\quad (\mathit{connect}, r)$ **at** $q \quad \Rightarrow \quad (\mathit{report}, q)$ **not-at** r .

In order to preserve (Mq3) under *sendrep*, we postulate

(Dld5) $\quad (\mathit{report}, r)$ **at** $q \quad \Rightarrow \quad ib.q = r \quad \vee \quad \mathit{fnd}.q$,
(Mq5) $\quad \mathit{sendrep}$ **at** $q \quad \Rightarrow \quad \mathit{fnd}.q$.

Predicate (Mq5) is preserved under *sendrep* because of the new postulate

(Mq6) $\quad \mathit{sendrep}\#q \leq 1$.

We now eliminate predicate (Dld5). It is easy to see that (Dld5) follows from the new postulates

(Mq7) $\quad \mathit{fnd}.q \quad \vee \quad \mathit{explist}.q = \emptyset$,
(Mq8) $\quad (\mathit{report}, r)$ **at** $q \quad \Rightarrow \quad ib.q = r \quad \vee \quad r \in \mathit{explist}.q$.

It is easy to see that (Mq7) is preserved. In order to preserve (Mq8) when q accepts *report*, *wakeup*, or *change*, we postulate

(Mq9) $\quad (\mathit{report}, r)\#q \leq 1$,
(Mq10) $\quad (\mathit{report}, q)$ **not-at** q ,
(Dld6) $\quad (\mathit{report}, ib.q)$ **at** $q \quad \Rightarrow \quad \mathit{change}$ **not-at** q .

In order to eliminate (Dld1) and (Dld2), postulated in Section 5.1, we now postulate the invariant

(Mq11) $\quad \mathit{ll}.(be.q) < \mathit{ll}.q \quad \Rightarrow \quad be.q = ib.q$.

In fact, it is easy to see that (Dld1) follows from (Mq11), and that (Dld2) follows from (Mq11) and (Iq2). Predicate (Mq11) is preserved when q accepts *report* because of the new postulate

(Mq12) $\quad (\mathit{report}, r)$ **at** $q \quad \wedge \quad \mathit{ll}.r < \mathit{ll}.q \quad \Rightarrow \quad ib.q = r$.

In order to eliminate (In*br) and (Dld4), we postulate

(Mq13) $\quad explist.q \subseteq \ branch.q$.

Now postulate (Dld4) follows from (Mq0) and (Mq13). Similarly (In*br) follows from (Mq1) and (Mq13). The treatment of (Dld6) is postponed to another layer.

We turn to the contents of the reports. The purpose of the search is that *connect* messages must only be sent between nodes that belong to different components of graph *JB*, and moreover that such a message is sent only over the outgoing edge with least weight. In order to compare the weights of the outgoing edges, we introduce a variable *bw* (*best-weight* in [GHS83]) that is to hold the least weight obtained thus far.

We let procedure *initp* initialize the new values of *be* and *bw*. In view of the invariants (Dld0) and (Jq2), we extend *initp* with the assignments

$$be := ib \ ; \ bw := \infty \ .$$

We give *report* messages *bw* as a second argument. So we replace the sending command in *sendrep* by

$$send\ (ib, report, self, bw)\ .$$

Accepting reports is redefined by

accept $(report, j, v) =$
  enabling $j \neq ib$
• $explist := explist \setminus \{j\}$;
  if $v < bw$ **then** $be := j$; $bw := v$ **fi**
end .

At this point we can prove the invariance of (Iq1) and (Iq3), by means of the invariants obtained until now. Invariance of (Iq2) remains suspended.

6.2 The local search

The variable *be* (best edge) points by default to the father; otherwise it points to the subtree with the lightest outgoing edge, or to an outgoing edge of the node itself. Part of this is expressed by the invariant

(Nq0) $\quad be.q = ib.q \quad \vee \quad be.q \in branch.q \quad \vee \quad Ci.q \neq Ci.(be.q)$.

The main local property of *bw* is that, if the edge to *be* is an outgoing edge of the component, its weight is held by *bw* and, if q is not searching, *bw* is a lower bound of the weights of the outgoing edges of q. These two properties are expressed in

(Nq1) $\quad be.q = ib.q \quad \vee \quad be.q \in branch.q \quad \vee \quad bw.q = w.(q, be.q)$,
(Nq2) $\quad bw.q \leq w.(q, r) \quad \vee \quad srch.q \quad \vee \quad (q, r) \in JB^*$.

In order to determine a local candidate for be, we introduce a private variable te (*test-edge* in [GHS83]) to hold the neighbour that is currently being investigated. The relation of te with the graph JB is determined by the postulate

$$\text{(Nq3)} \qquad w.(q, te.q) \leq w.(q,r) \quad \vee \quad te.q = q \quad \vee \quad (q,r) \in JB^* \ .$$

Therefore, if $te.q$ is not connected to q, it is the nearest neighbour of q not connected to q.

In order to preserve (Nq3) when a new value for $te.q$ is needed, we introduce a variable $bas.q$ (related to the attribute *Basic* in [GHS83]) to hold the set of untried neighbours, according to the invariant

$$\text{(Nq4)} \qquad w.(q,r) = \infty \quad \vee \quad r \in bas.q \quad \vee \quad (q,r) \in JB^* \quad \vee \quad ib.q = r \ .$$

The final disjunct of (Nq4) is needed for the case that a *connect* message from q to r is still pending.

Since we also want that the set bas is not bigger than necessary, we extend the assignments $ib := be$ in *wakeup* and *change* by $bas := bas \setminus \{be\}$. For the same reason, the assignment to *branch* in *connect*, is extended with $bas := bas \setminus \{j\}$.

We let the search be executed by taking for te the nearest node in bas if the corresponding weight is less than bw. Otherwise the search can be called off, which is indicated by putting $srch := false$.

We use functions *lewe* and *lenb* for least weight and least neighbour of a node q with respect to a set S of neighbours. These functions are defined as follows. If there is a node $r \in S$ with $w.(q,r) < \infty$ and $w.(q,r) \leq w.(q,x)$ for all $x \in S$ then $lewe.(q,S) = w.(q,r)$ and $lenb.(q,S) = r$. Otherwise $lewe.(q,S) = \infty$ and $lenb.(q,S) = q$. Now we define

accept $(search)$ =
• **if** $lewe.(self, bas) < bw$ **then**
  $te := lenb.(self, bas)$;
  $send\ (te, ask, self, ll, ci)$
 else $srch := false$ **fi**
end .

The component identity of the neighbour is only relevant if it is recent enough, i.e., if its level is sufficiently high. Therefore, message *ask* has the level of the sender as second argument, and it is disabled as long as the receiver has a lower level. The answer is the boolean value whether or not the component identities agree. If they agree, both sender and receiver can use (Lq6) to delete the other node from its set *bas*. So message *ask* is defined by

accept (ask, j, v, id) =
 enabling $v \leq ll$
• $send\ (j, answer, id = ci)$;
 if $id = ci$ **then** $bas := bas \setminus \{j\}$ **fi**
end .

Upon a positive answer the search must be resumed. If node p receives a negative answer, it compares $w.(p, te.p)$ with its current value of bw and, if possible, adjusts be and bw. So we define

$$
\begin{array}{l}
\textbf{accept}\ (answer, b) = \\
\bullet\ \ \textbf{if}\ \ b\ \ \textbf{then} \\
\qquad bas := bas \setminus \{te\}\ ; \\
\qquad delay\ (search) \\
\quad \textbf{else} \\
\qquad srch := false\ ; \\
\qquad \textbf{if}\ \ w.(self, te) < bw\ \ \textbf{then} \\
\qquad\quad be := te\ ; \\
\qquad\quad bw := w.(self, te) \\
\qquad \textbf{fi} \\
\quad \textbf{fi}\ \ ; \\
\quad te := self \\
\textbf{end}\ .
\end{array}
$$

The new assignment to be in $answer$ endangers (Iq3). This predicate is preserved since $w.(p, te) < bw$ implies $te \neq p$ by the convention $w.(p, p) = \infty$. Predicate (Mq11) is also threatened by the assignment to be in $answer$. It is preserved if we postulate

(Nq5) $\quad answer$ **at** $q \quad \Rightarrow \quad ll.q \leq ll.(te.q)$.

Before proving the invariance of the goal directed invariants (Nq0) up to (Nq5), we first introduce some bottom-up invariants:

(Oq0) $\quad search$ **at** $q \quad \Rightarrow \quad te.q = q$,
(Oq1) $\quad search\#q \ \leq\ 1$,
(Oq2) $\quad (ask, q)$ **not-at** q ,
(Oq3) $\quad answer$ **at** $q \quad \Rightarrow \quad te.q \neq q$.

For the invariance of (Oq0) and (Oq1), we need the new postulates

(Oq4) $\quad search$ **at** $q \quad \Rightarrow \quad srch.q$,
(Oq5) $\quad srch.q \quad \Rightarrow \quad fnd.q$,
(Oq6) $\quad srch.q \quad \vee \quad te.q = q$.

Preservation of (Oq3) needs the new postulates

(An*f) $\quad answer$ **at** $q \quad \Rightarrow \quad fnd.q$,
(Oq7) $\quad answer\#q \leq 1$,
(Oq8a) $\quad (ask, q)$ **at** $r \quad \Rightarrow \quad te.q = r$.

The name (Oq8a) is chosen to indicate that this invariant will be abolished in the next section; it will be weakened to an invariant with the name (Oq8).

We note that (An*f) follows from (Oq3), (Oq5), and (Oq6). Preservation of (Oq7) and (Oq8a) follows from the new postulates

(Oq9) $\quad (\mathit{ask}, q)$ **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{answer}$ **not-at** q ,
(Oq10) $\quad (\mathit{ask}, q)\#r \leq 1$.

Preservation of (Oq9) and (Oq10) needs no new postulates.

We turn to the treatment of the invariants (Nq0) up to (Nq5). Preservation of (Nq0) when q accepts *report* or *answer* follows from (Mq8), (Mq13), and the new postulate

(Nq6) $\quad (\mathit{answer}, \mathit{false})$ **at** $q \quad \Rightarrow \quad \mathit{Ci}.q \neq \mathit{Ci}.(\mathit{te}.q)$.

Preservation of (Nq4) needs the new postulates

(Nq7) $\quad (\mathit{answer}, \mathit{true})$ **at** $q \quad \Rightarrow \quad (q, \mathit{te}.q) \in \mathit{JB}^*$,
(Nq8) $\quad (\mathit{ask}, -, v)$ **at** $q \quad \Rightarrow \quad v > 0$.

Preservation of (Nq5) and (Nq6) requires the new postulates

(Nq9) $\quad (\mathit{ask}, q, v)$ **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{ll}.q = v$,
(Nq10) $\quad (\mathit{ask}, q, -, \mathit{id})$ **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{Ci}.q = \mathit{id}$.

Preservation of (Nq8) needs the new postulate

(Nq11) $\quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{ll}.q > 0$.

Preservation of (Nq9) and (Nq10) only needs the new postulate

(As*f) $\quad (\mathit{ask}, q)$ **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{fnd}.q$,

which follows from (Oq2), (Oq5), and (Oq6).

At this point, the only pending postulates are (Iq2), (Dld6), and (Ch-M).

6.3 An optimization

The algorithm of [GHS83] has a nasty optimization in its response to *Test* (our message *ask*): if two nodes with the same component identity are concurrently sending *Test* messages to each other, both notice this and do not answer. This optimization is important in the worst case analysis of the number of messages needed. In fact, the authors of [GHS83] come to the upper bound $5n \log n + 2e$ where n is the number of nodes and e is the number of edges. The summand $2e$ corresponds to their estimate that every edge can only be rejected (removed from the set *bas*) once, and that every rejection only requires two messages. In the algorithm above, rejection of an edge may require four messages. So for the above version we would have to replace $2e$ by $4e$. Since the number of edges e may be quadratic in n, replacing $4e$ by $2e$ is a significant gain in efficiency (see Section 9.5). We therefore reluctantly decided to adopt the optimization.

The optimization consists of replacing *ask* by

$\textbf{accept}\ (ask, j, v, id) =$
$\quad \textbf{enabling}\ \ v \le ll$
$\bullet\ \ \textbf{if}\ \ ci \ne id\ \ \textbf{then}\ \ send\ (j, answer, false)$
$\quad \textbf{else}$
$\qquad bas := bas \setminus \{j\}\ ;$
$\qquad \textbf{if}\ \ j = te\ \ \textbf{then}$
$\qquad\quad te := self\ ;$
$\qquad\quad delay\ (search)$
$\qquad \textbf{else}\ \ send\ (j, answer, true)\ \ \textbf{fi}$
$\quad \textbf{fi}$
$\textbf{end}\ \textbf{.}$

The proof of correctness of this optimization is quite involved. Since it violates postulate (Oq8a) above, we have to replace (Oq8a) by the postulates

(Oq8) $\quad (ask, q)\ \textbf{at}\ r \quad \Rightarrow \quad te.q = r \quad \vee \quad te.r = q\ ,$
(Nq12) $\quad (ask, q)\ \textbf{at}\ r \quad \Rightarrow \quad te.q = r \quad \vee \quad (q, r) \in JB^*\ ,$
(Nq13) $\quad (ask, q, -, id)\ \textbf{at}\ r \quad \Rightarrow \quad te.q = r \quad \vee \quad Ci.r = id\ ,$
(Nq14a) $\quad (ask, q)\ \textbf{at}\ r \quad \Rightarrow \quad te.q = r \quad \vee \quad r \notin bas.q\ ,$
(Nq15) $\quad (ask, q)\ \textbf{at}\ r \quad \Rightarrow \quad te.q = r \quad \vee \quad (ask, r)\ \textbf{not-at}\ q\ .$

Moreover, we also need the postulates

(Nq16) $\quad (ask, te.q, -, Ci.q)\ \textbf{at}\ q \quad \Rightarrow \quad answer\ \textbf{not-at}\ q\ ,$
(Nq17a) $\quad (answer, true)\ \textbf{at}\ q \quad \Rightarrow \quad q \notin bas.(te.q)\ .$

One can verify that predicate (Nq16) is the only new postulate that does not hold for the non-optimized version.

Remarks. We have chosen the invariants in the non-optimized version in such a way that almost all of them could be kept in the optimization. For example, under assumption of (Oq8a), predicate (Nq9) is equivalent to

$$(ask, q, v)\ \textbf{at}\ r \quad \Rightarrow \quad ll.q = v\ ,$$

but this “version of (Nq9)” is not valid for the optimization.

In a much later stage the invariants (Nq14a) and (Nq17a) will be replaced by closely related but slightly stronger predicates. The names are chosen in such a way that comparison is easy.

7 The decision at the core

In Section 7.1, we introduce the way by which a chain of *change* messages is started and we re-establish most of the invariants that are threatened by this modification. In Section 7.2 we establish some invariants that have been claimed already but not yet proved. Section 7.3 is devoted to invariants that justify the names *best-edge* and *best-weight* for the variables *be* and *bw*. In Section 7.4, we establish the remaining pending invariant (Ch-M) introduced in Section 3.3. Finally, in Section 7.5, we treat the variables *term.q*, introduce *halt* messages, and establish the first proof obligation (Goal).

7.1 The generation of *change*

Up to this point, a *change* message is only sent upon reception of a *change* message. In this section we decide how the reports at the core generate a *change* message. The purpose of *change* messages is that they trigger the node with the lightest outgoing edge of the component to send a *connect* message over that edge. The final comparison to determine this node is performed at the core. Thus, when both core members have obtained all reports expected, they compare *bw* values. The companion with the lighter *bw* value then starts a chain of *change* messages along its *be* path.

Until now a *report* from the core companion was disabled. We weaken this enabling condition of *report* as follows. A node is allowed to accept a *report* from the companion when it has executed *sendrep*, i.e., when it satisfies $\neg \mathit{fnd}$. In that case it compares its *bw* value with the value v of the *report*. If $bw < v$, it decides to start a chain of *change* messages. It does so by means of a selfmessage *change*. We thus redefine

accept $(\mathit{report}, j, v) =$
    enabling $j \neq ib \quad \vee \quad \neg \mathit{fnd}$
• **if** $j \neq ib$ **then**
        $\mathit{explist} := \mathit{explist} \setminus \{j\}$;
        if $v < bw$ **then** $be := j$; $bw := v$ **fi**
    elsif $bw < v$ **then**
        delay (*change*)
    fi
end .

This modification clearly endangers all invariants that restrict the occurrence of *change* messages. Therefore the proofs of (Jq6), (Jq8), and (Jq9) must be adapted and, in order to preserve (Jq7) and (Jq10), we need the postulates

(Pq0) $\quad (\mathit{report}, ib.q, v)$ **at** $q \quad \wedge \quad bw.q < v \quad \Rightarrow \quad ib.(ib.q) = q$,
(Pq1) $\quad (\mathit{report}, ib.q, v)$ **at** $q \quad \wedge \quad bw.q < v \quad \Rightarrow \quad$ *change* **not-at** $ib.q$.

It turns out, however, that the modification can violate predicate (Jq12) when process q has $(\mathit{connect}, ib.q)$ in its buffer and accepts $(\mathit{report}, ib.q)$. This is a consequence of our assumption that the buffers are bags (and not FIFO).

We are dealing here with the problem that a core has been formed, but one of the core members has not yet accepted the corresponding *connect* message, while the companion and its dependent nodes have completed the testing of their surroundings. Then it is possible that the final *report* of the companion overtakes the *connect* message. This must not be allowed since the delayed core member is not yet ready for reception of the *report*.

We therefore introduce a private boolean variable *mar* (for *married*, so to speak) to indicate that the message $(connect, ib)$ has been accepted. So we redefine

```
accept (report, j, v) =
    enabling  j ≠ ib  ∨  (¬fnd ∧ mar)
•   if  j ≠ ib  then
        explist := explist \ {j} ;
        if  v < bw  then  be := j ; bw := v  fi
    else
        mar := false ;
        if  bw < v  then  delay (change)  fi
    fi
end .
```

Variable *mar* is set to *true* upon reception of $(connect, ib)$. So we redefine

```
accept (connect, j, v) =
    enabling  j = ib  ∨  v < ll
•   if j = ib then
        mar := true ;
        initp(ll + 1, w.(j, self))
    else  intobranch (j)  fi
end ,
```

where *intobranch* is defined by

```
proc  intobranch (j)  =
    branch := branch ∪ {j} ;
    bas := bas \ {j}
end .
```

We now postulate

(Pq2) $\quad mar.q \;\Rightarrow\; (connect, ib.q)$ **not-at** q .

It turns out that indeed the postulates (Pq0), (Pq1), and (Pq2) are sufficient to preserve the invariants of the families (Jq), (Kq), (Lq), (Mq), and (Nq).

We turn to the preservation of (Pq0), (Pq1), (Pq2). In order to preserve (Pq0) when q accepts *report* or *answer*, we need the new postulate

(Dld7) $\quad (report, ib.q)$ **at** $q \;\wedge\; fnd.q \;\Rightarrow\; ib.(ib.q) = q$.

In order to show that (Pq1) is preserved when $p \neq q$ accepts *report*, we postulate

(Pq3) $\quad (report, q, v)$ **at** $ib.q \quad \Rightarrow \quad bw.q = v$.

Predicate (Pq2) is preserved under *wakeup* and *change* because of the new postulates

(Pq4) $\quad mar.q \quad \wedge \quad be.(ib.q) = q \quad \Rightarrow \quad ib.(ib.q) = q$,
(Pq5) $\quad mar.q \quad \Rightarrow \quad ib.q \neq q$,
(Pq6) $\quad change$ **at** $q \quad \Rightarrow \quad \neg mar.q$.

Predicate (Pq3) is preserved under *wakeup* and *change* by postulating

(Dld8) $\quad (report, q)$ **at** $be.q \quad \Rightarrow \quad be.q = ib.q$.

In order to preserve (Pq4) when $p \neq q$ accepts *answer* or *report*, we postulate

(Dld9) $\quad mar.q \quad \Rightarrow \quad te.(ib.q) \neq q$,
(Pq7) $\quad (report, q)$ **at** $ib.q \quad \wedge \quad mar.q \quad \Rightarrow \quad ib.(ib.q) = q$.

Predicate (Pq7) is preserved when q accepts *sendrep* because of (Mq5) and the new postulate

(Pq8) $\quad mar.q \quad \wedge \quad fnd.q \quad \Rightarrow \quad ib.(ib.q) = q$.

Predicate (Pq8) can be violated when q has a pending *init* message. Indeed, it can be shown that this cannot be excluded. We therefore decide to disable reception of *init* while *mar* holds. We thus redefine

accept $(init, v, id) =$
  enabling $\neg mar$
• $initp\ (v, id)$
end .

7.2 Some pending predicates

We turn to the remaining pending proof obligations (Dld6), (Dld7), (Dld8), and (Dld9), and the invariance of (Iq2).

It is clear that (Dld6) follows from (Pq6), (Jq12), and the new postulate

(Qq0) $\quad (report, ib.q)$ **at** $q \quad \Rightarrow \quad mar.q \quad \vee \quad (connect, ib.q)$ **at** q .

In order to preserve (Qq0) when $p \neq q$ accepts *sendrep*, we postulate

(Qq1) $\quad fnd.(ib.q) \quad \wedge \quad ib.(ib.q) = q \quad \Rightarrow \quad mar.q \quad \vee \quad (connect, ib.q)$ **at** q .

Predicate (Qq1) is preserved when $p \neq q$ accepts *connect* by postulating

(Qq2) $\quad (connect, q)$ **at** $ib.q \quad \wedge \quad ib.(ib.q) = q$
$\quad \Rightarrow \quad mar.q \quad \vee \quad (connect, ib.q)$ **at** q .

It may be left to the reader to prove that predicate (Dld7) follows from (Jq4), (Pq8), and (Qq0). Predicate (Dld8) is proved as follows:

$$
\begin{array}{ll}
 & (\mathit{report}, q) \textbf{ at } r \quad \wedge \quad \mathit{ib}.q \neq r \\
\Rightarrow & \{(\text{Mq8}), (\text{Mq13})\} \\
 & (\mathit{report}, q) \textbf{ at } r \quad \wedge \quad \mathit{ib}.q \neq r \quad \wedge \quad (q = \mathit{ib}.r \vee q \in \mathit{branch}.r) \\
\Rightarrow & \{(\text{Jq0})\} \\
 & (\mathit{report}, q) \textbf{ at } r \quad \wedge \quad \mathit{ib}.q \neq r \quad \wedge \quad q = \mathit{ib}.r \\
\Rightarrow & \{(\text{Qq0})\} \\
 & (\mathit{mar}.r \vee (\mathit{connect}, q) \textbf{ at } r) \quad \wedge \quad \mathit{ib}.q \neq r \quad \wedge \quad q = \mathit{ib}.r \\
\Rightarrow & \{(\text{Jq4})\} \\
 & \mathit{mar}.r \quad \wedge \quad \mathit{ib}.q \neq r \quad \wedge \quad q = \mathit{ib}.r \\
\Rightarrow & \{(\text{Pq4})\} \\
 & \mathit{be}.q \neq r \ .
\end{array}
$$

If we now instantiate $r = \mathit{be}.q$, we get (Dld8).

We turn to the treatment of (Dld9). Since (Dld9) is concerned with the value of *te*, we first postulate the invariant

(Qq3) $\quad \mathit{te}.q = \mathit{ib}.q \quad \Rightarrow \quad \mathit{ib}.q = q \ .$

Predicate (Qq3) is preserved under acceptance of *connect*, *init*, *ask*, or *answer* because of the new postulate

(Qq4) $\quad \mathit{ib}.q \notin \mathit{bas}.q \ .$

We now observe that (Kq5), (Pq8), and (Qq3) with $q := \mathit{ib}.q$ combine and yield that

(Dld9a) $\quad \mathit{mar}.q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{te}.(\mathit{ib}.q) \neq q \ .$

Now (Dld9) follows from (Dld9a) and the additional predicate

(Qq5) $\quad \mathit{mar}.q \quad \wedge \quad \neg \mathit{fnd}.q \quad \Rightarrow \quad \mathit{te}.(\mathit{ib}.q) \neq q \ .$

Predicate (Qq5) is preserved when $p \neq q$ accepts *init*, *connect*, *ask*, or *answer* because of the new invariant

(Qq6) $\quad \mathit{mar}.q \quad \Rightarrow \quad q \notin \mathit{bas}.(\mathit{ib}.q) \ .$

We finally turn to the proof of preservation of (Iq2): if *change* is at node q then $\mathit{be}.q \neq \mathit{ib}.q$. In order to show that (Iq2) is preserved under *change* and *report*, we postulate

(Qq7) $\quad \mathit{be}.q \in \mathit{branch}.q \quad \Rightarrow \quad \mathit{be}.(\mathit{be}.q) \neq \mathit{ib}.(\mathit{be}.q) \ ,$
(Qq8) $\quad \mathit{mar}.q \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{bw}.q < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q \ .$

In order to preserve (Qq7) under *report*, *init*, and *answer*, we postulate

(Qq9) $\quad (\mathit{report}, q, v) \textbf{ at } \mathit{ib}.q \quad \wedge \quad v < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q \ .$

Predicate (Qq9) is preserved when q accepts *sendrep* because of (Mq5) and the new invariant

(Qq10) $\quad \mathit{fnd}.q \quad \wedge \quad \mathit{bw}.q < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q \ .$

7.3 Best edges

We now strive for a justification of the names *best weight* and *best edge* for the variables *bw* and *be*. This justification consists of the above invariants (Nq0) and (Nq1) in combination with the new invariants

(Rq0) $\quad be.q \in branch.q \quad \Rightarrow \quad bw.(be.q) = bw.q$,
(Rq1) $\quad r \in branch.q \quad \Rightarrow \quad bw.q \leq bw.r \quad \vee \quad r \in explist.q$.

In (Rq1) the final disjunct is included for the case that q is yet expecting a report from node r.

We first give the crucial arguments for preservation of (Rq0). For the case that $p \neq q$ accepts *report* or *answer* we use (Dld5), (An*f), and the new postulate

(Rq2) $\quad be.q \in branch.q \quad \Rightarrow \quad \neg fnd.(be.q)$.

Preservation of (Rq2) can be proved without new arguments.

Predicate (Rq0) is preserved when q itself accepts *report* because of (Pq3) and

(Re*ib) $\quad (report, q)$ **at** $r \quad \Rightarrow \quad ib.q = r \quad \vee \quad ib.r = q$,

which follows from (Jq0), (Mq8), and (Mq13).

Predicate (Rq1) is preserved when q accepts *change* because of the new postulate

(Rq3) $\quad change$ **at** $q \quad \Rightarrow \quad bw.q \leq bw.(ib.q)$.

In order to preserve (Rq1) when q accepts $(connect, j)$ with $j \neq ib.q$, we first postulate

(Rq4) $\quad (connect, q)$ **at** $r \quad \Rightarrow \quad bw.q = w.(q, r) \quad \vee \quad ib.r = q$.

Yet, including j into $branch.q$ violates (Rq1) if $w.(j, q) < bw.q$. The only way to rescue (Rq1) is to modify *connect* by also adding j to $explist.q$ in case of $w.(j, q) < bw.q$. Now *explist* is the set of neighbours to which an *init* message has been sent. We therefore redefine command *intobranch* in *connect* (cf. Section 7.1) by

```
proc intobranch (j) =
    branch := branch ∪ {j} ;
    bas := bas \ {j} ;
    if w.(j, self) < bw then
        send (j, init, ll, ci) ;
        explist := explist ∪ {j}
    fi
end .
```

This version of *intobranch* with guard $w.(j, \mathit{self}) < \mathit{bw}$ is an optimization with respect to [GHS83] where the guard *fnd* is used. This optimization does not influence the worst case behaviour.

In this way (Rq1) is saved by means of (Rq4), but the modification endangers all invariants that mention *init* or *explist*. In particular, in order to preserve (Kq4) and (Mq7), we need to ensure that the `then` part of *intobranch* is executed only when *fnd* holds. For this purpose we need the predicate

(Co*jb) $\quad (\mathit{connect}, r)$ **at** $q \quad \wedge \quad (q, r) \in \mathit{JB}^* \quad \Rightarrow \quad \mathit{jb}.q = r$.

This predicate is proved as follows. Assume that the antecedent of (Co*jb) holds, i.e., $(q, r) \in \mathit{JB}^*$ and $(\mathit{connect}, r)$ is at q. Predicate (Jq4) implies that $\mathit{ib}.r = q$. Therefore (Iq0) implies $(q, r) \in \mathit{MST}$. Since (Iq0) also yields $\mathit{JB} \subseteq \mathit{MST}$, Theorem 2 implies $(q, r) \in \mathit{JB}$. Since $(\mathit{connect}, r)$ is at q, we have $\mathit{jb}.r \neq q$. Therefore the definition of *JB* implies that $\mathit{jb}.q = r$. This proves (Co*jb). Notice that this proof needs the invariants not only at the nodes q and r but at all nodes of the graph.

At this point the only pending predicate is (Ch-M). In the next section we will show that this predicate also follows from the invariants at our disposal.

7.4 The first harvest

In this section we prove the pending predicate (Ch-M). Again we need the invariants not only locally, but in all nodes of the graph.

Predicate (Ch-M) is proved by showing that, if *change* is at q and $\mathit{be}.q \notin \mathit{branch}.q$, then $\mathit{be}.q$ is the lightest outgoing edge of the *JB*–component of q. We first prove that it is an outgoing edge and then that it is the lightest one.

Recall that, in Section 5.2, we proved

(Thm5) $\quad (q, r) \in \mathit{JB}^* \quad \wedge \quad \mathit{jb}.(\mathit{jb}.q) = q$
$\qquad\qquad \Rightarrow \quad \mathit{ll}.r \leq \mathit{ll}.q \quad \wedge \quad (\mathit{ll}.r = \mathit{ll}.q \Rightarrow \mathit{Ci}.r = \mathit{Ci}.q)$.

It follows from (Jq6), (Jq7), (Jq8), and (Jq12) that

(Ch*jb) $\quad \mathit{change}$ **at** $q \quad \Rightarrow \quad \mathit{jb}.(\mathit{jb}.q) = q \neq \mathit{jb}.q$.

Using (Thm5), (Ch*jb), (Iq2), (Mq11), and (Nq0), we then get

(Ch-out) $\quad \mathit{change}$ **at** $q \quad \wedge \quad \mathit{be}.q \notin \mathit{branch}.q \quad \Rightarrow \quad (q, \mathit{be}.q) \notin \mathit{JB}^*$.

This shows that $(q, \mathit{be}.q)$ is an outgoing edge.

We now use Theorem 6 of Section 4.3 to prove that it is the lightest outgoing edge. We first define the function $\mathit{up}.v$ by

$$\mathit{up}.v.q \quad = \quad v \leq \mathit{bw}.q \quad \wedge \quad \neg \mathit{fnd}.q \quad \wedge \quad \mathit{jb}.q \neq q \ .$$

Using (Jq1), (Mq7), (Rq1), and condition (Fn*jb) from Section 5.2, we obtain

$$\mathit{jb}.(\mathit{jb}.q) \notin \{q, \mathit{jb}.q\} \quad \wedge \quad \mathit{up}.v.(\mathit{jb}.q) \quad \Rightarrow \quad \mathit{up}.v.q \ .$$

Now Theorem 6, with *jb* for g and $\mathit{up}.v$ for φ, yields

(Thm6) $\quad jb.(jb.p) = p \neq jb.p \quad \wedge \quad up.v.p \quad \wedge \quad up.v.(jb.p)$
$\quad\Rightarrow \quad ((p,q) \in JB^* \quad \Rightarrow \quad up.v.q)$.

If we choose $v = bw.p$, the antecedents of (Thm6) are implied by (Ch*jb), together with (Kq2), (Kq3), and (Rq3). This results in

(Thm6C) change **at** $p \quad \wedge \quad (p,q) \in JB^* \quad \Rightarrow \quad \neg \mathit{fnd}.q \quad \wedge \quad bw.p \leq bw.q$.

We now use (Nq1), (Nq2), (Oq5), and (Iq2), to obtain

$$\begin{aligned} &\mathit{change} \ \mathbf{at}\ p \quad \wedge \quad be.p \notin \mathit{branch}.p \quad \wedge \quad w.(q,r) < w.(p, be.p) \\ &\Rightarrow \quad ((p,q) \in JB^* \quad \Rightarrow \quad (p,r) \in JB^*) \ . \end{aligned}$$

Together with (Ch-out), this expresses that $(p, be.p)$ is a minimum–weight outgoing edge of the JB component of p. Therefore Theorem 3 with function f given by $f.x = ((p,x) \in JB^*)$ implies predicate (Ch-M).

7.5 Termination detected

When the two core nodes observe that the component has no outgoing edges, they are allowed to terminate the algorithm. To prove this assertion, we introduce the predicate $\mathit{fincr}.p$ to express that p belongs to a final core:

$$\begin{aligned} \mathit{fincr}.p \quad \equiv \quad & jb.(jb.p) = p \neq jb.p \quad \wedge \quad \neg \mathit{fnd}.p \quad \wedge \quad \neg \mathit{fnd}.(jb.p) \\ & \wedge \quad bw.p = \infty \quad \wedge \quad bw.(jb.p) = \infty \ . \end{aligned}$$

Using (Thm6) with $v = \infty$ we get

(Thm6T) $\mathit{fincr}.p \quad \wedge \quad (p,q) \in JB^* \quad \Rightarrow \quad up.\infty.q$.

Using (Nq2), (Oq5), and the definition of the edge relation E, we then get

$$up.\infty.q \quad \wedge \quad (q,r) \in E \quad \Rightarrow \quad (q,r) \in JB^* \ .$$

Since the graph (V, E) is connected, induction over the graph with Theorem 0 then yields

(fi-JB) $\quad \mathit{fincr}.p \quad \Rightarrow \quad (p,q) \in JB^*$.

This expresses that JB is a spanning tree of the graph. Since (Iq0) implies $JB \subseteq MST$, Theorem 2 implies that $MST = JB$.

A second application of (Thm6T) together with (fi-JB) yields

(fi-up) $\quad \mathit{fincr}.p \quad \Rightarrow \quad up.\infty.q$,

which assertion will be useful to show that, when $\mathit{fincr}.p$ holds, the algorithm is in the process of termination.

It is now important that a node p that satisfies $\mathit{fincr}.p$ can observe this. Indeed, when a node accepts a *report* from its companion and observes that both core nodes have $bw = \infty$, it may conclude that fincr holds:

(Re-fi) $(report, ib.p, \infty)$ **at** $p \quad \wedge \quad bw.p = \infty \quad \wedge \quad \neg fnd.p \quad \wedge \quad mar.p$
$\Rightarrow \quad fincr.p$,

which assertion follows from (Mq2), (Mq4), (Pq2), (Pq3), (Pq5), and the new postulate

(Sq0) $(report, ib.q, \infty)$ **at** $q \quad \wedge \quad mar.q \quad \Rightarrow \quad ib.(ib.q) = q$.

The invariance of (Sq0) requires the new postulate

(Sq1) $be.q = ib.q \quad \vee \quad bw.q < \infty$.

The invariance of this predicate is easy.

In view of (Re-fi), a node p that accepts $(report, ib.p, \infty)$ in a state with $bw.p = \infty$, can broadcast *halt* messages. So we replace the final command of *report* by

if $bw < v$ **then** *delay*(*change*)
elsif $v = \infty$ **then** *delay*(*halt*) **fi** .

Here *halt* is a new message, declared by

accept (*halt*) =
• *terminated* := *true* ;
mcast (*branch*, *halt*)
end .

It is easy to see that this modification preserves all invariants. We postulate the new invariants

(Sq2) *halt* **at** $q \quad \Rightarrow \quad (r, s) \in JB^*$,
(Sq3) $term.q \quad \Rightarrow \quad (r, s) \in JB^*$.

Preservation of (Sq2) follows from (fi-JB) and (Re-fi). Predicate (Sq3) is preserved because of (Sq2). According to (Sq3), validity of *term.q* implies that *JB* spans the graph.

Since we later want to show that, when *term.q* holds, the algorithm is in the process of termination, we also postulate

(Sq4) *halt* **at** $q \quad \Rightarrow \quad up.\infty.r$,
(Sq5) $term.q \quad \Rightarrow \quad up.\infty.r$.

In fact, we have that $up.\infty.r$ is equivalent to

$ib.r \neq r \quad \wedge \quad (connect, r)$ **not-at** $ib.r \quad \wedge \quad bw.r = \infty \quad \wedge \quad \neg fnd.r$.

This implies that *wakeup* is ignored and that there are no messages *connect*, *change*, *init*, *ask*, *answer*, *sendrep*, *search* in transit. Therefore preservation of (Sq4) follows from (fi-up) and (Re-fi). Predicate (Sq5) is preserved because of (Sq4).

Using (Sq3), (Sq5), (Iq0), (Jq0), and (Jq1), we then get our first goal

(Goal) $term.q \quad \Rightarrow \quad (\,(r, s) \in MST \quad \equiv \quad s \in \{ib.r\} \cup branch.r\ )$.

8 Upon termination

In this Chapter we deal with the second proof obligation, which is to prove that, when all messages are disabled, all processes q have *term.q*. So we analyse the state under the assumption that all messages in transit are disabled.

Below, we shall first determine the conditions under which a node is disabled. We then proceed to eliminate the possible messages pending at such a node.

Up to now, most invariants remain valid if arbitrary messages are thrown away. Let us say that an invariant is a *progress invariant* of message type *kw* if it can be invalidated by throwing away a message with name *kw*. The only progress invariants mentioned as yet are (Jq1), (Kq11), (Kq12), (Lq0), (Qq0), (Qq1), and (Qq2). All these predicates are progress invariants only of *connect*. In this section we shall encounter progress invariants for the other messages.

The Sections 8.1, 8.2 deal with the simpler cases of deadlock (i.e., of disabled messages). The most crucial disabled messages are *connect* and *ask* messages. The disabling conditions for these messages hinge on the difference between the levels of sender and receiver. In Section 8.4, we therefore investigate the set *Low* of the nodes of minimal level. In Section 8.6 we introduce messages *winit* to ensure that, if a node q is disabled and is element of *Low*, then its neighbour *ib.q* also belongs to *Low*. Using this we prove that *Low* contains a core.

Section 8.7 contains the second harvest. Here we prove that, when all messages in transit are disabled, all nodes q have *term.q* (the second proof obligation).

In Section 8.8, we introduce an integer variable *fc* to eliminate the variables *explist*, *srch*, and *fnd*. In Section 8.9, we prove that all terminated nodes are idle (the third proof obligation).

The aim of this Section is to show that, if all messages at all nodes are disabled, all nodes q have *term.q*. This is expressed by

$$\mathit{DIS} \quad \Rightarrow \quad \mathit{term.q}\ .$$

Here disabledness of the system is defined by $\mathit{DIS}:\ (\forall\ q :: \mathit{Dis.q})$, where *Dis.q* expresses that all messages at node q are disabled. It follows from the declaration of the messages that *Dis.q* is equivalent to the conjunction of the following six predicates

$$\begin{array}{ll}
\mathit{Dfr.q}: & \mathit{kw}\ \mathbf{at}\ q \quad\Rightarrow\quad \mathit{kw} \notin \{\mathit{wakeup}, \mathit{change}, \mathit{search}, \mathit{answer}, \mathit{halt}\}\ , \\
\mathit{Dco.q}: & (\mathit{connect}, j, v)\ \mathbf{at}\ q \quad\Rightarrow\quad j \neq \mathit{ib.q} \quad\wedge\quad \mathit{ll.q} \leq v\ , \\
\mathit{Dse.q}: & \mathit{sendrep}\ \mathbf{at}\ q \quad\Rightarrow\quad \mathit{srch.q} \quad\vee\quad \mathit{explist.q} \neq \emptyset\ , \\
\mathit{Das.q}: & (\mathit{ask}, j, v)\ \mathbf{at}\ q \quad\Rightarrow\quad \mathit{ll.q} < v\ , \\
\mathit{Din.q}: & \mathit{init}\ \mathbf{at}\ q \quad\Rightarrow\quad \mathit{mar.q}\ , \\
\mathit{Dre.q}: & (\mathit{report}, j)\ \mathbf{at}\ q \quad\Rightarrow\quad j = \mathit{ib.q} \quad\wedge\quad (\mathit{mar.q} \Rightarrow \mathit{fnd.q})\ .
\end{array}$$

We now treat these disabling conditions one by one.

8.1 Progress invariants for *report*

In order to show that a disabled process has no pending *init* message, we claim that

(In*Re) $\quad init\ \mathbf{at}\ q \quad\wedge\quad mar.q \quad\Rightarrow\quad (report, ib.q)\ \mathbf{at}\ q\ .$

This predicate follows from (In*cr) and the new postulate

(Tq0) $\quad mar.q \quad\Rightarrow\quad ib.(ib.q) = q \quad\vee\quad (report, ib.q)\ \mathbf{at}\ q\ .$

In order to show that (Tq0) is preserved when *ib.q* accepts *change*, we need the new postulate

(Tq1) $\quad mar.q \quad\Rightarrow\quad (connect, q)\ \mathbf{at}\ ib.q \ \vee\ fnd.(ib.q) \ \vee\ (report, ib.q)\ \mathbf{at}\ q\ .$

Predicate (Tq1) is preserved when *q* accepts *connect* because of the new invariant

(Tq2) $\quad (connect, ib.q)\ \mathbf{at}\ q$
$\qquad\Rightarrow\quad (connect, q)\ \mathbf{at}\ ib.q \quad\vee\quad fnd.(ib.q) \quad\vee\quad (report, ib.q)\ \mathbf{at}\ q\ .$

It follows from *Dfr*, *Din*, *Dre*, (In*Re), and (Kq9) that we have

(Dkws) $\quad Dis.q \quad\wedge\quad kw\ \mathbf{at}\ q$
$\qquad\Rightarrow\quad kw \notin \{init, wakeup, change, search, answer, halt\}\ .$

So, at a disabled node, the only pending messages can be *connect*, *sendrep*, *ask*, or *report*.

8.2 Sending *wakeup* and *halt*

At present the algorithm has two obvious cases in which deadlock can occur. The first case is when a process wakes up and sends a *connect* message to a node that has no pending *wakeup* message. The second case is that a process sends an *ask* message to a node without a pending *wakeup* message. It would be possible, in the commands of *wakeup* and *search*, to add the sending of a *wakeup* message to the nodes *be* and *te*, respectively. This would require more *wakeup* messages, however, than an ordinary broadcast of *wakeup* messages. We therefore prefer to postulate that, initially, *wakeup* messages are in transit to all processes. Using (Dld0), we then obtain the invariant

(Tq3) $\quad wakeup\ \mathbf{at}\ q \quad\vee\quad ib.q \neq q\ .$

Using (Tq3), (Lq3), (Jq4), and (Qq0), we conclude from *Dco.q* and *Dre.q* that

(DisCo) $\quad Dis.q \ \wedge\ (connect, r)\ \mathbf{at}\ q \quad\Rightarrow\quad ib.q \neq q \ \wedge\ ib.q \neq r \ \wedge\ ll.q \leq ll.r\ ,$
(DisRe) $\quad Dis.q \quad\wedge\quad (report, r)\ \mathbf{at}\ q \quad\Rightarrow\quad ib.q = r \quad\wedge\quad fnd.q\ .$

It follows from (Tq0), (Tq1), (DisCo), and (DisRe) that

(Dis-ma) $\quad Dis.q \quad\wedge\quad Dis.(ib.q) \quad\wedge\quad mar.q \quad\Rightarrow\quad fnd.q \quad\vee\quad fnd.(ib.q)\ .$

The main progress invariant for *halt* is

(Tq4) $\quad term.(ib.q) \quad\Rightarrow\quad ib.(ib.q) = q \quad\vee\quad halt\ \mathbf{at}\ q \quad\vee\quad term.q\ .$

8.3 Deadlock in search

We postulate two progress invariants for the selfmessages *search* and *sendrep*:

(Tq5) $\quad srch.q \;\wedge\; te.q = q \;\Rightarrow\; search$ **at** q ,
(Tq6) $\quad fnd.q \;\Rightarrow\; sendrep$ **at** q .

The invariance of these predicates is easily verified. Using $Dfr.q$ and $Dse.q$, we obtain from these predicates

(DisSe) $\quad Dis.q \;\wedge\; fnd.q \;\wedge\; te.q = q \;\Rightarrow\; explist.q \neq \emptyset$.

The name *explist* is intended to suggest that $q \in explist.r$ means that r is expecting a *report* message from q. This is formalized in the invariant

(Tq7) $\quad q \in explist.r \;\Rightarrow\; init$ **at** $q \;\vee\; fnd.q \;\vee\; (report, q)$ **at** r .

Using (Tq7), (DisRe), (Dkws), (Mq13), and (Jq2), we obtain

(Dis-ex) $\quad Dis.q \;\wedge\; Dis.r \;\wedge\; q \in explist.r \;\Rightarrow\; fnd.q$.

The conjunction of (DisSe) and (Dis-ex) will be used to show that in a state where all nodes of a component are disabled and some of them satisfy *fnd* then some of them have $te.q \neq q$.

We now show that, if $te.q \neq q$, there is a pending *ask* or *answer* message. Here the optimization described in subsection 6.3 causes a major complication. We postulate

(Tq8) $\quad te.q = q \;\vee\; (ask, q)$ **at** $te.q \;\vee\; answer$ **at** q
$\qquad\vee\; (ask, te.q, ll.q, ci.q)$ **at** q .

In order to show that (Tq8) is preserved when $p \neq q$ accepts *ask* we postulate

(Tq9) $\quad (ask, q)$ **at** $te.q \;\wedge\; te.(te.q) = q \;\wedge\; ci.(te.q) = ci.q$
$\qquad\Rightarrow\; (ask, te.q)$ **at** q .

In order to show that (Tq9) is preserved when q accepts *search*, we postulate

(Tq10) $\quad te.q \neq q \wedge ci.(te.q) = ci.q \wedge q \in bas.(te.q) \;\Rightarrow\; (ask, q)$ **at** $te.q$.

Predicate (Tq10) is preserved because of some old invariants together with the new invariant

(Tq11) $\quad ll.(te.q) < ll.q \;\Rightarrow\; (ask, q)$ **at** $te.q$.

It follows from $Das.q$, (Dkws), and (Tq8), and from $Das.(te.q)$ and (Nq9), that we have

(Dis-te) $\quad Dis.q \;\wedge\; te.q \neq q \;\Rightarrow\; (ask, q)$ **at** $te.q$,
(DisAsk) $\quad Dis.(te.q) \;\wedge\; (ask, q)$ **at** $te.q \;\Rightarrow\; ll.(te.q) < ll.q$.

8.4 The low level region

Inspired by condition (DisAsk), we introduce the set *Low* of the nodes of minimal level:

$$q \in Low \quad \equiv \quad (\forall\, x :: ll.q \leq ll.x) \ .$$

The aim is to show that property *DIS* implies that $\neg fnd$ holds at the nodes in *Low*. Notice that set *Low* depends on the state of the system.

It follows from (Dis-te), (DisAsk), that we have

$$DIS \quad \wedge \quad te.q \neq q \quad \Rightarrow \quad ll.(te.q) < ll.q \ .$$

This implies

(Low-te) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad te.q = q \ .$

On the other hand it follows from (Jq0), (Jq2), and (Kq1), that

(Low-br) $q \in Low \quad \wedge \quad r \in branch.q \quad \Rightarrow \quad r \in Low \ .$

Now we have all ingredients to prove

(Low-fn) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad \neg fnd.q \ .$

In fact, in order to exploit the invariants (DisSe) and (Dis-ex), we choose a function h (in a nondeterministic way, and dependent on the state), that satisfies $h.q \in explist.q$ if $explist.q$ is nonempty, and $h.q = q$ otherwise. By (Mq13), (Jq0), (Jq2), and (Iq0), we have $(q, h.q) \in MST$ for all q with $h.q \neq q$. By Theorem 4 in Section 4.2, this implies that

$$(\forall\, q :: (\exists\, n :: h^{n+2}.q = h^n.q)) \ .$$

We also observe that

$$\begin{array}{ll} & h.q \neq q \quad \wedge \quad h.(h.q) = q \\ \Rightarrow & \{\text{definition of } h\} \\ & h.q \in explist.q \quad \wedge \quad q \in explist.(h.q) \\ \Rightarrow & \{\text{(Mq13) and (Jq0)}\} \\ & ib.(h.q) \in branch.(h.q) \\ \Rightarrow & \{\text{(Jq2)}\} \\ & false \ . \end{array}$$

We thus get the stronger result

$(\forall\, q :: (\exists\, n :: h^{n+1}.q = h^n.q))$, and hence by (Mq13), (Jq0), and (Jq3):
$(\forall\, q :: (\exists\, n :: explist.(h^n.q) = \emptyset))$.

By induction, it follows from (Low-br) and (Mq13), that

$$q \in Low \quad \Rightarrow \quad h^k.q \in Low \qquad \text{for all } k.$$

Using (Low-te) and (DisSe), we then get

$$DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad (\exists\, n :: \neg fnd.(h^n.q)) \ .$$

Finally, (Dis-ex) implies that the dummy n is zero. This proves (Low-fn).

Below we shall establish the crucial property that in a terminal state the forest constructed contains a core. For this purpose, we shall prove that

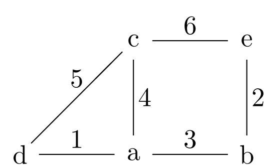
$$Dis.q \quad \wedge \quad q \in Low \quad \Rightarrow \quad ib.q \in Low \ .$$

Unfortunately, this is not yet an invariant of the algorithm. Indeed, the present version of the algorithm is incorrect and may lead to deadlock. The point is that in the present version the receiver of *connect* does not always send its level to a new dependent node.

8.5 An operational intermezzo

Let us first give an example to show that the present version of the algorithm may lead to deadlock. The situation is as follows. There is a component A of graph G, which sends a *connect* message to a component B. Component B is still searching its surroundings and, in particular, it is waiting for an *answer* from component C. Component C has a level *ll* lower than the levels of components A and B. It sends a *connect* message to component A and is absorbed into A without *init* messages sent back. Therefore the level of C remains low and the *ask* message from B remains pending.

One of the smallest graphs in which this can occur has five nodes, say a, b, c, d, and e. It has six edges, say in order of increasing weights: $\{a, d\}$, $\{b, e\}$, $\{a, b\}$, $\{a, c\}$, $\{c, d\}$, and $\{c, e\}$. In the picture, we have given the edges the weights 1 up to 6.



a small graph

The scenario goes as follows. First, a, b, d, e accept *wakeup* messages and send and accept *connect* messages to and from the nearest neighbour. In this way two components are formed $A = \{a, d\}$ and $B = \{b, e\}$, both with level $ll = 1$. These components only consist of cores. So, no *init* messages are needed. Now *ask* messages are being sent: between a and b, and from d and e to c. The first two *ask* messages evoke the *answer* no, while the *ask* messages to c remain pending since $ll.c = 0$.

The nodes a and b send *report* messages to d and e, respectively. Finally node c accepts *wakeup* and sends a *connect* message to a. Since $ll.c < ll.a$, node a accepts the *connect* message and puts $branch := \{c\}$. Node a does not send an *init* message to c. Therefore $ll.c$ remains less than one and the *ask* messages from d and e are never answered. The system deadlocks in a state where the forest has two components $\{a, c, d\}$ and $\{b, e\}$.

8.6 Absorption into a nonprobing component

The above analysis suggests that the **else**–part of procedure *intobranch* must be extended with a weak version of the *init* message that only distributes the values of *ll* and *ci*. In [GHS83], this version is differentiated from *init* by means of an additional (esssentially) boolean parameter. Since we do not want to modify all invariants related to *init*, we introduce a new message *winit* (for *weak init*). So the final command in *intobranch* becomes

$$\begin{array}{l}
\textbf{if}\;\; w.(j, \mathit{self}) < bw \;\; \textbf{then} \\
\quad \mathit{send}\ (j, \mathit{init}, \mathit{ll}, \mathit{ci})\ ; \\
\quad \mathit{explist} := \mathit{explist} \cup \{j\} \\
\textbf{else}\;\; \mathit{send}\ (j, \mathit{winit}, \mathit{ll}, \mathit{ci})\;\; \textbf{fi}\;\; .
\end{array}$$

Since our buffers are not required to be FIFO buffers, we must reckon with the possibility that a *winit* message is overtaken by an *init* message. In that case a subsequent reception of *winit* might be harmful. We therefore decide that outdated *winit* messages are ignored. In this way we arrive at the declaration

$$\begin{array}{l}
\textbf{accept}\ (\mathit{winit}, v, \mathit{id}) = \\
\bullet\;\; \textbf{if}\;\; \mathit{ll} < v \;\; \textbf{then} \\
\qquad \mathit{ll} := v\ ;\ \mathit{ci} := \mathit{id}\ ;\ \mathit{be} := \mathit{ib}\ ; \\
\qquad \mathit{mcast}\ (\mathit{branch}, \mathit{winit}, v, \mathit{id}) \\
\quad\;\; \textbf{fi} \\
\textbf{end}\ .
\end{array}$$

Here, variable *be* is reset to *ib* in order not to endanger predicate (Mq11).

The introduction of *winit* endangers all invariants that mention *ll*, *ci*, or *be*. In order to preserve them we need a number of invariants that are more or less analogous to invariants for *init*. Since outdated messages are being ignored, we may include the additional assumption that the message is not outdated (i.e., that the guard of the command of *winit* holds). The first invariant that we claim is an analogue of (In*br):

(Uq0) $\quad (\mathit{winit}, u)\ \textbf{at}\ q \;\;\wedge\;\; \mathit{ll}.q < u \;\;\Rightarrow\;\; q \in \mathit{branch}.(\mathit{ib}.q)\ .$

Together with (Jq2), (Jq4), (Jq7), (Jq11) this implies

(Wi*) $\quad (\mathit{winit}, u)\ \textbf{at}\ q \;\;\wedge\;\; \mathit{ll}.q < u$
$\qquad\qquad \Rightarrow\;\; \mathit{ib}.(\mathit{ib}.q) \neq q \;\;\wedge\;\; (\mathit{connect}, \mathit{ib}.q)\ \textbf{not-at}\ q$
$\qquad\qquad \wedge\;\; (\mathit{connect}, q)\ \textbf{not-at}\ r \;\;\wedge\;\; \mathit{change}\ \textbf{not-at}\ q\ .$

None of the invariants of family (Jq) is threatened. In order to preserve (Kq1), (Kq6), and (Kq7), we postulate the invariants

(Uq1) $\quad (\mathit{winit}, u)\ \textbf{at}\ q \;\;\Rightarrow\;\; u \leq \mathit{ll}.(\mathit{ib}.q)\ ,$
(Uq2) $\quad (\mathit{winit}, u)\ \textbf{at}\ q \;\;\wedge\;\; \mathit{ll}.q < u \;\;\Rightarrow\;\; \neg \mathit{fnd}.q\ ,$
(Uq3) $\quad (\mathit{winit}, u)\ \textbf{at}\ q \;\;\wedge\;\; \mathit{init}\ \textbf{at}\ q \;\;\Rightarrow\;\; u < \mathit{ll}.(\mathit{ib}.q)\ .$

Remark. The invariants (Uq1) and (Uq3) are somewhat stronger than necessary. It is possible to extend the antecedents of (Uq1) and (Uq3) with the conjunct $ll.q < u$. We have removed these conjuncts in a late stage of the design for convenience in the proof of termination. □

In order to preserve (Lq0), we postulate the following analogue of (Lq1):

$$\text{(Uq4)} \qquad (\mathit{winit}, \mathit{ll}.(\mathit{ib}.q), \mathit{id}) \textbf{ at } q \quad \wedge \quad \mathit{ll}.q < \mathit{ll}.(\mathit{ib}.q) \quad \Rightarrow \quad \mathit{id} = \mathit{ci}.(\mathit{ib}.q) \ .$$

The predicates (Lq6), (Lq7), (Lq8), (Lq9), (Lq10), (Lq11) are preserved when we postulate

$$\begin{array}{ll}
\text{(Uq5)} & (\mathit{winit}, -, \mathit{ci}.r) \textbf{ at } q \quad \Rightarrow \quad (q, r) \in \mathit{JB}^* \ , \\
\text{(Uq6)} & (\mathit{winit}, -, w.(r, s)) \textbf{ at } q \quad \Rightarrow \quad (q, r) \in \mathit{JB}^* \ , \\
\text{(Uq7)} & (\mathit{winit}, u, \mathit{ci}.r) \textbf{ at } q \quad \Rightarrow \quad u = \mathit{ll}.r \ , \\
\text{(Uq8)} & (\mathit{connect}, \mathit{ib}.r) \textbf{ at } r \quad \wedge \quad (\mathit{winit}, u, w.(r, \mathit{ib}.r)) \textbf{ at } q \quad \Rightarrow \quad u = 1 + \mathit{ll}.r \ , \\
\text{(Uq9)} & (\mathit{winit}, -, w.(r, s)) \textbf{ at } q \quad \Rightarrow \quad r \in \mathit{branch}.s \quad \vee \quad \mathit{ib}.s = r \ , \\
\text{(Uq10)} & (\mathit{winit}, -, \infty) \textbf{ not-at } q \ .
\end{array}$$

Predicate (Qq7) requires the new postulate

$$\text{(Uq11)} \qquad (\mathit{winit}, u) \textbf{ at } q \quad \wedge \quad \mathit{ll}.q < u \quad \wedge \quad \mathit{be}.(\mathit{ib}.q) = q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \ .$$

The antecedent of (Uq11) may seem unlikely, but it can occur when some node accepts *winit* and then sends *winit* to q.

Predicate (Qq9) needs (analogously to (Mq3)) the new postulate

$$\text{(Uq12)} \qquad (\mathit{winit}, u) \textbf{ at } q \quad \wedge \quad \mathit{ll}.q < u \quad \Rightarrow \quad (\mathit{report}, q) \textbf{ not-at } \mathit{ib}.q \ .$$

We now have to show that the predicates (Uq) are invariant. In order to preserve (Uq5) and (Uq7) under *winit* we need the new invariants

$$\begin{array}{ll}
\text{(Uq13)} & (\mathit{winit}, -, \mathit{id}) \textbf{ at } q \quad \wedge \quad (\mathit{winit}, -, \mathit{id}) \textbf{ at } r \quad \Rightarrow \quad (q, r) \in \mathit{JB}^* \ , \\
\text{(Uq14)} & (\mathit{winit}, v, \mathit{id}) \textbf{ at } q \quad \wedge \quad (\mathit{winit}, w, \mathit{id}) \textbf{ at } r \quad \Rightarrow \quad v = w \ .
\end{array}$$

In this way it is proved that the introduction of message *winit* preserves all the invariants introduced up to now. It remains to show that *winit* serves some goal.

The sole purpose of the introduction of message *winit* is captured in the progress invariant

$$\begin{array}{ll}
\text{(Uq15)} & \mathit{ll}.q < \mathit{ll}.(\mathit{ib}.q) \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) \neq q \\
& \Rightarrow \quad (\mathit{connect}, q) \textbf{ at } \mathit{ib}.q \quad \vee \quad \mathit{init} \textbf{ at } q \quad \vee \quad (\mathit{winit}, \mathit{ll}.(\mathit{ib}.q)) \textbf{ at } q \ .
\end{array}$$

The proof of invariance of (Uq15) is delicate but needs no new invariants.

Now, finally, we get the invariant announced as the motivation for this section. Indeed, the invariants (Uq15), (Dkws), (DisCo), and (Kq11) (the latter one applied to $\mathit{ib}.q$) together imply that $\mathit{ll}.(\mathit{ib}.q) \leq \mathit{ll}.q$ follows from $\mathit{Dis}.q$ and $\mathit{Dis}.(\mathit{ib}.q)$. We therefore have, as announced at the end of Section 8.4, the invariant

(Low-ib) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad ib.q \in Low$.

We now apply Theorem 4 of Section 4.2 to obtain

$$DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad (\exists\, n :: \; ib^n.q \in Low \quad \wedge \quad ib^{n+2}.q = ib^n.q) \; .$$

At this point, we need that Low is nonempty. For this purpose it suffices to postulate that V is nonempty. Now using (Low-ib) and the definition of Low, we obtain the existence of a core in Low:

(Low-cr) $DIS \quad \Rightarrow \quad (\exists\, p \in Low :: \; ib.p \neq p \; \wedge \; ib.p \in Low \; \wedge \; ib.(ib.p) = p)$.

8.7 Analysis of the ultimate core

In the previous Section, we proved that in the terminal graph the region Low contains a core. It remains to show that the tree connected to this core is the minimum–weight spanning tree of the graph. For this purpose we need to analyse the co-ordination of the two core members.

One critical moment in the execution of the algorithm is when two core members have determined the bw values and send reports to each other in order to decide which of the two is to execute *change*. It is crucial that either the two bw values differ or that the system is allowed to terminate. In the latter case the bw values both must be equal to ∞. This property indeed follows from the invariants we have collected, but the proof is delicate. So we propose to prove

Theorem. The invariants obtained imply

(Re*cr) $(report, ib.p)\ \mathbf{at}\ p \quad \wedge \quad ib.(ib.p) = p \quad \wedge \quad mar.p \quad \wedge \quad bw.p < \infty$
$\qquad \Rightarrow \quad bw.p \neq bw.(ib.p)$.

Proof. Let p be a node that satisfies the antecedent and yet has $bw.p = bw.(ib.p)$. We derive a contradiction.

The idea is to use axiom (A0) of Section 4.2, which says that all finite weights differ, in combination with the invariants

(Nq1) $\quad be.q = ib.q \quad \vee \quad be.q \in branch.q \quad \vee \quad bw.q = w.(q, be.q)$,
(Rq0) $\quad be.q \in branch.q \quad \Rightarrow \quad bw.q = bw.(be.q)$.

This combination may suggest to replace q repeatedly by $be.q$, as long as $be.q \in branch.q$. For this purpose, we introduce the function ben given by

$$ben.q \quad = \quad (\ \mathbf{if} \;\; be.q \in branch.q \;\; \mathbf{then} \;\; be.q \;\; \mathbf{else} \;\; q \;\; \mathbf{fi}\ ) \; .$$

It follows from (Jq0), (Jq2), and (Iq0), that we have

$$ben.q \neq q \quad \Rightarrow \quad ben.(ben.q) \neq q \quad \wedge \quad (q, ben.q) \in MST \; .$$

Therefore, Theorem 4 implies that repeated application of ben leads to a fixpoint. So there are natural numbers a and b and nodes $p1 = ben^a.p$ and $p2 = ben^b.(ib.p)$ with $ben.p1 = p1$ and $ben.p2 = p2$. Moreover, (Rq0) implies that $bw.p1 = bw.p = bw.(ib.p) = bw.p2$.

We now want to apply the third alternative of (Nq1). In order to eliminate the first alternative, we introduce a function $recent$ given by $recent.q \equiv (be.q \neq ib.q)$. Then (Nq1) reduces to

$$recent.q \quad \wedge \quad ben.q = q \quad \Rightarrow \quad bw.q = w.(q, be.q) \ .$$

Using (Qq8), (Qq9), and (Pq3), we get $recent.p$ and $recent.(ib.p)$. On the other hand, (Qq7) yields

$$recent.q \quad \Rightarrow \quad recent.(ben.q) \ .$$

We thus get $recent.p1$ and $recent.p2$, and hence $w.(p1, be.p1) = w.(p2, be.p2)$. Now axiom (A0) implies $p2 = p1$ or $p2 = be.p1$. We use component identities to eliminate the second possibility. In fact, invariant (Nq0) implies $Ci.p1 \neq Ci.(be.p1)$. In order to prove that p and $ib.p$ have equal Ci values, we observe that (Lq0) and (Kq11) together imply

$$\begin{array}{l} (connect, q) \ \textbf{not-at} \ ib.q \ \wedge \ (connect, ib.q) \ \textbf{not-at} \ q \ \wedge \ ib.(ib.q) = q \\ \Rightarrow \quad Ci.q = Ci.(ib.q) \ . \end{array}$$

Together with (Mq4) and (Pq2), this implies that $Ci.p = Ci.(ib.p)$. Function ben preserves Ci because of

$$be.q \in branch.q \quad \Rightarrow \quad Ci.q = Ci.(be.q),$$

which property is proved in

$$\begin{array}{ll} & be.q = r \quad \wedge \quad r \in branch.q \\ \Rightarrow & \{\text{(Jq0), (Jq2), (Jq11), (Mq11)}\} \\ & ib.r = q \quad \wedge \quad (connect, r) \ \textbf{not-at} \ q \quad \wedge \quad r \neq ib.q \quad \wedge \quad ll.q \leq ll.r \\ \Rightarrow & \{\text{(Kq1)}\} \\ & ib.r = q \quad \wedge \quad (connect, r) \ \textbf{not-at} \ q \quad \wedge \quad ll.q = ll.r \\ \Rightarrow & \{\text{(Lq0)}\} \\ & Ci.q = Ci.r \ . \end{array}$$

We thus obtain $Ci.p1 = Ci.p2$. Therefore axiom (A0) implies that $p1 = p2$.

In order to derive that $p = ib.p$, we introduce a kind of inverse of ben. Let function ibn be defined by

$$ibn.q \quad = \quad (\ \textbf{if} \ \ ib.(ib.q) \neq q \ \ \textbf{then} \ \ ib.q \ \ \textbf{else} \ \ q \ \ \textbf{fi} \ ) \ .$$

Using (Jq0) and (Jq2), we get

$$ben.q \neq q \quad \Rightarrow \quad ibn.(ben.q) = q \ .$$

We now assume that the natural numbers a and b are minimal. If $a \leq b$, then $ib.p = ibn^b.(ben^b.(ib.p)) = ibn^b.(ben^a.p) = ibn^{b-a}.p = p$. If $a > b$, a similar calculation also yields $ib.p = p$. So, we have $ib.p = p$. This however contradicts (Pq5). $\Box$

This theorem is crucial for the proof of invariance of

$$
\begin{array}{ll}
\text{(Vq0)} & \mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{ib}.q \neq q \quad \wedge \quad \neg \mathit{mar}.q \quad \wedge \quad \mathit{bw}.q \leq \mathit{bw}.(\mathit{ib}.q) \\
& \Rightarrow \quad (\mathit{connect}, \mathit{ib}.q) \ \mathbf{at}\ q \quad \vee \quad \mathit{change}\ \mathbf{at}\ q \quad \vee \quad \mathit{change}\ \mathbf{at}\ \mathit{ib}.q \\
& \ \vee \quad \mathit{halt}\ \mathbf{at}\ q \quad \vee \quad \mathit{term}.q\ .
\end{array}
$$

Notice that this is the only progress invariant for *change* messages. In order to show that (Vq0) is preserved when q accepts *answer* or (report, j) with $j \neq \mathit{ib}.q$, we use (An*f), (Dld5) and the new invariant

$$
\text{(Vq1)} \qquad \mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{mar}.q\ .
$$

We now fulfil our second main proof obligation:

Theorem. Assume that graph (V, E) is connected. Then, for every node q, we have the invariant

$$
\mathit{DIS} \quad \Rightarrow \quad \mathit{term}.q.
$$

Proof. Using (Tq3), (Vq0), (DisCo), and (Dkws), we obtain

$$
\begin{array}{l}
\mathit{DIS} \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{bw}.q \leq \mathit{bw}.(\mathit{ib}.q) \\
\Rightarrow \quad \mathit{mar}.q \quad \vee \quad \mathit{term}.q\ .
\end{array}
$$

To eliminate $\mathit{mar}.q$, we observe that (Dis-ma), (Low-ib), and (Low-fn) imply

$$
\mathit{DIS} \quad \wedge \quad q \in \mathit{Low} \quad \Rightarrow \quad \neg \mathit{mar}.q\ .
$$

This implies

$$
\mathit{DIS} \ \wedge \ q \in \mathit{Low} \ \wedge \ \mathit{ib}.(\mathit{ib}.q) = q \ \wedge \ \mathit{bw}.q \leq \mathit{bw}.(\mathit{ib}.q) \quad \Rightarrow \quad \mathit{term}.q\ .
$$

Now, using (Sq5) and symmetry, we get

$$
\begin{array}{l}
\mathit{DIS} \quad \wedge \quad q \in \mathit{Low} \quad \wedge \quad \mathit{ib}.q \in \mathit{Low} \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) = q \\
\Rightarrow \quad \mathit{term}.q \quad \wedge \quad \mathit{term}.(\mathit{ib}.q)\ .
\end{array}
$$

Therefore, both members of the final core of (Low-cr) satisfy *term*.

We now use the invariant (Tq4) which implies that

$$
\mathit{DIS} \quad \wedge \quad \mathit{term}.(\mathit{ib}.q) \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad \mathit{term}.q\ .
$$

Using Theorem 6 of Section 4.3 we then get $\mathit{term}.q$ for all nodes q connected to the core. Since all nodes are connected because of (Sq3), it follows that all nodes q have $\mathit{term}.q$. □

8.8 The last program transformation

We now reduce the variables *srch*, *fnd*, *explist* to ghost variables by introducing an integer variable *fc* (for *find-count*, see [GHS83]), related to the other variables by the invariant

(Wq0) $\quad fc.q \quad = \quad \#fnd.q + \#srch.q + \#explist.q \ .$

Recall that, for P boolean, $\#P$ denotes 0 or 1 when P is *false* or *true*, respectively.

In order to preserve postulate (Wq0), we extend the assignment to *explist* in *initp* with $fc := 2 + \#explist$ and we extend the assignments $te := self$ in *search* and *answer*, the assignment to *fnd* in *sendrep*, and the assignment to *explist* in *report* with $fc := fc - 1$. In order to eliminate *explist* and *fnd*, we observe that from (Wq0), (Mq5), (Mq7), and (Oq5) we get

$$\begin{array}{l} fnd.q \quad \equiv \quad fc.q \neq 0 \ , \\ sendrep \ \mathbf{at} \ q \quad \Rightarrow \quad (\neg srch.q \ \wedge \ explist.q = \emptyset \quad \equiv \quad fc = 1) \ . \end{array}$$

We therefore can eliminate the variables *explist* and *fnd* from the guards of *report* and *sendrep*. See Chapter 10 for the concrete modifications in the algorithm.

8.9 Terminated nodes are idle

In this section we deal with the third proof obligation of Section 2.2, that every terminated process is idle. As byproducts we also prove that, as soon as some process has terminated, all messages are enabled and all levels are equal.

Our proof obligation is captured in the invariant

(tm-idl) $\quad term.q \quad \Rightarrow \quad idle.q \ ,$

where *idle.q* is defined to mean that every message at q is enabled and such that acceptance is equivalent to *skip*, i.e., that the only resulting state change is the removal of the message.

As a step in the proof, we define the predicate *open.q* to mean that every message at node q is enabled and we claim

(tm-opn) $\quad term.p \quad \Rightarrow \quad open.q \ .$

Using (Sq5) and some other invariants we first show

(tm-ms) $\quad term.p \quad \wedge \quad kw \ \mathbf{at} \ q \quad \Rightarrow \quad kw \in \{report, wakeup, halt, winit\} \ .$

Of these four remaining messages, *report* is the only one that can be disabled. So, for (tm-opn), it suffices to observe that (Sq5) and (Qq0) imply

$$term.p \quad \wedge \quad (report, ib.q) \ \mathbf{at} \ q \quad \Rightarrow \quad \neg fnd.q \quad \wedge \quad mar.q \ .$$

For predicate (tm-idl), we treat the four messages one by one. Message *report* is never equivalent to *skip* since it always modifies the private variables *fc* or *mar*. So we need to prove (tm-Re) $\quad term.q \quad \Rightarrow \quad report \ \mathbf{not\text{-}at} \ q$.

In order to do so, we first prove the invariance of the new postulates

$$
\begin{array}{ll}
\text{(Wq1)} & mar.q \quad \Rightarrow \quad ib.(ib.q) = q \quad \vee \quad bw.(ib.q) < \infty \quad \vee \quad init \ \mathbf{at} \ q \ , \\
\text{(Wq2)} & mar.q \quad \Rightarrow \quad ib.(ib.q) = q \quad \vee \quad q \in branch.(ib.q) \ .
\end{array}
$$

Predicate (Wq2) is needed for (Wq1) when $ib.q$ accepts *init* or *connect*. Predicate (tm-Re) follows from (Wq1), (Sq5), (Pq3), (Qq0), and some other invariants.

Message *halt* does not violate (tm-idl) because of the invariant

(Wq3) $\quad term.q \quad \Rightarrow \quad halt \ \mathbf{not\text{-}at} \ q$.

Preservation of (Wq3) needs the new postulates

$$
\begin{array}{ll}
\text{(Wq4)} & halt \ \mathbf{at} \ q \quad \vee \quad term.q \quad \Rightarrow \quad ib.(ib.q) = q \quad \vee \quad term.(ib.q) \ , \\
\text{(Wq5)} & halt\#q \leq 1 \ , \\
\text{(Wq6)} & halt \ \mathbf{at} \ q \quad \vee \quad term.q \quad \Rightarrow \quad (report, ib.q, \infty) \ \mathbf{not\text{-}at} \ q \ .
\end{array}
$$

Message *wakeup* is equivalent to *skip* at node q if $ib.q \neq q$. Therefore *wakeup* does not violate (tm-idl) because (Sq5) implies

$$term.p \quad \Rightarrow \quad ib.q \neq q \ .$$

Message $(winit, u)$ is equivalent to *skip* at node q if $u \leq ll.q$. For *winit*, it therefore suffices to prove

(tm-Wi) $\quad term.p \quad \wedge \quad (winit, u) \ \mathbf{at} \ q \quad \Rightarrow \quad u \leq ll.q$.

We prove this predicate by constructing a state function $LLBW$ such that

$$
\begin{array}{ll}
\text{(tm-hi)} & term.p \quad \Rightarrow \quad ll.q = LLBW \ , \\
\text{(Wq7)} & (winit, u) \ \mathbf{at} \ q \quad \Rightarrow \quad u < LLBW \ .
\end{array}
$$

Function $LLBW$ is defined by

$$
\begin{array}{lcl}
llbw.q & = & ll.q + \#(bw.q < \infty) \ , \\
LLBW & = & (\text{MAX } x \in V :: llbw.x) \ .
\end{array}
$$

It follows from (Kq6) and (Kq7) that $llbw.q$ never decreases. Consequently, $LLBW$ never decreases. Therefore, predicate (Wq7) is threatened only when some process p generates a *winit* message while accepting a message of the form $(connect, j)$. In that case it has $bw.p \leq w.(j, p) < \infty$ and, hence, it sends $(winit, u)$ with

$$u = ll.p < llbw.p \leq LLBW \ .$$

This proves that (Wq7) is an invariant.

As for (tm-hi), we first use (Kq11), (Uq15), (Wq7), (Kq4), and (Sq5) to prove that

$$term.p \quad \wedge \quad ll.q = LLBW \quad \wedge \quad (q, r) \in JB \quad \Rightarrow \quad ll.r = LLBW \ .$$

On the other hand, predicate (Sq5) implies

$$term.p \quad \Rightarrow \quad (\exists\, x \in V :: ll.x = LLBW)\ .$$

Finally, predicate (tm-hi) follows from (Sq3) and Theorem 0. This concludes the proof of (tm-Wi), and hence of (tm-idl).

Remarks. If (tm-hi) holds, predicate (Wq7) is stronger than necessary to prove (tm-Wi). We need the strength of (Wq7), however, to prove (tm-hi). The invariants (tm-opn) and (tm-hi) express that, as soon as some process terminates, all messages are enabled and all levels are equal.

9 Towards termination

In this Chapter we prove termination of the algorithm. We do this by constructing a state function *vf* with values in the natural numbers which decreases whenever some process accepts a message. We do this carefully, so as also to obtain the estimate on the message complexity given in [GHS83]. This illustrates Hehner's thesis "Termination is timing", cf. [Heh89].

9.1 An upper bound for the levels

The first point is to prove that the levels *ll* of the nodes have an upper bound that is logarithmic in the size of the graph. Here we use one of the few invariants claimed in [GHS83], see p. 72, namely

(Xq0) $\quad 2^{ll.q} \leq (\# \, r :: (q, r) \in JB^*)$.

Indeed, using (Co*jb), (Kq12), (Kq13), and some other invariants, one can prove that (Xq0) is invariant. Now let $n = \#V$ be the number of nodes of the graph and let $L = \log_2 n$ be the number of binary digits of n. It then follows from (Xq0) that we have the invariant $ll.q \leq L - 1$. We can therefore define $coll.q = L - 1 - ll.q$ with the invariant

$$coll.q \geq 0 \ .$$

Using (Kq6) and (Kq7) for *init*, one can easily verify that *coll.q* never increases, and that it decreases whenever process q accepts *init* or $(connect, ib.q)$.

Remark. Reference [GHS83] claims (Xq0), but has no clear definition of relation JB^*. □

9.2 Bounding *ask* and *answer*

We would like to use the number of elements of *bas* as a variant function to bound the number of accepted *ask* and *answer* messages. Unfortunately, the algorithm sometimes deletes elements from the set *bas* for other reasons. We therefore introduce a ghost variable *bash*, closely related to *bas*, which is initially equal to *bas* and inherits the modifications of *bas* in *ask* and *answer*, but which is not modified in *wakeup*, *change*, and *connect*. See Chapter 10 for the concrete modifications.

It is clear that *bash.q* is never enlarged. The critical property of *bash* that we need, is that *bash.q* becomes smaller whenever process q receives the answer *true* or receives an *ask* message that need not be answered. More precisely, the first property is

(An*B0) $\quad (answer, true)$ **at** $q \quad \Rightarrow \quad te.q \in bash.q$.

This property follows from (Oq3) and the new postulate

(Xq1) $\quad te.q = q \quad \vee \quad te.q \in bash.q$.

In order to show that (Xq1) is preserved when q executes *search* we postulate

$$\text{(Xq2)} \qquad bas.q \subseteq bash.q \ .$$

Since deletion from *bash* is always accompanied by the same deletion from *bas*, predicate (Xq2) is invariant. Since *te* is set to *self* whenever *te* is deleted from *bash*, predicate (Xq1) is also invariant. The second property of *bash* that we need is

$$\text{(Xq3)} \qquad (ask, r) \ \mathbf{at} \ q \quad \Rightarrow \quad r \in bash.q \ .$$

Predicate (Xq3) is preserved under *answer* and *search* because of the new postulates

$$\begin{array}{ll} \text{(Xq4)} & (ask, te.q) \ \mathbf{at} \ q \quad \Rightarrow \quad (answer, true) \ \textbf{not-at} \ q \ , \\ \text{(Xq5)} & q \in bash.r \quad \Rightarrow \quad r \in bash.q \quad \vee \quad te.r = q \ . \end{array}$$

In order to show that (Xq5) is preserved when q accepts *ask* or *answer*, we have to postulate the following strengthenings of (Nq14a) and (Nq17a).

$$\begin{array}{ll} \text{(Nq14)} & (ask, q) \ \mathbf{at} \ r \quad \Rightarrow \quad te.q = r \quad \vee \quad r \notin bash.q \ , \\ \text{(Nq17)} & (answer, true) \ \mathbf{at} \ q \quad \Rightarrow \quad q \notin bash.(te.q) \ . \end{array}$$

It is preserved when $p \neq q$ accepts *answer* because of

$$\text{(An*B1)} \quad (answer, false) \ \mathbf{at} \ q \quad \Rightarrow \quad q \in bash.(te.q) \ .$$

This predicate follows from (Nq6), (Nq16), (Oq7), and the new postulate

$$\begin{array}{ll} \text{(Xq6)} & te.q = q \quad \vee \quad q \in bash.(te.q) \quad \vee \quad (answer, true) \ \mathbf{at} \ q \\ & \vee \quad (ask, te.q, -, Ci.q) \ \mathbf{at} \ q \ . \end{array}$$

It is clear that (Nq14a) and (Nq17a) follow from (Nq14), (Nq17), and (Xq2). Preservation of (Nq14) and (Nq17) is proved in the same way as for (Nq14a) and (Nq17a). Actually, we removed the invariance proofs of (Nq14a) and (Nq17a) after the introduction of the stronger invariants (Nq14) and (Nq17).

To summarize, we have that $\#bash.q$ never increases and that it decreases whenever process q accepts the *answer true* or an *ask* message with third argument equal to $Ci.q$.

We cover the remaining cases for the messages *answer* and *ask* by introducing the functions

$$\begin{array}{lll} vfanswer.q & = & \#srch.q + coll.q, \\ vfask.q & = & \#(srch.q \wedge (answer, false) \ \textbf{not-at} \ q) + coll.q \ . \end{array}$$

These functions never increase. Function *vfanswer* decreases when q accepts the *answer false*. Function *vfask* decreases when $p \neq q$ accepts $(ask, q, -, id)$ with $id \neq ci.p$. Combining these three functions, we obtain

$$vfaa.q \quad = \quad \#bash.q + vfanswer.q + vfask.q \ .$$

This function never increases. It decreases when q accepts a message *answer* or $(ask, -, -, ci.q)$. It also decreases when $p \neq q$ accepts $(ask, q, -, id)$ with $id \neq ci.p$.

9.3 Other local parts of the variant function

In order to give bounds for the number of accepted messages *search*, *sendrep*, *wakeup*, *change*, *halt*, we define

$$\begin{aligned}
\mathit{vfsearch}.q &= \#(\mathit{srch}.q \wedge \mathit{te}.q = q) + \#\mathit{bash}.q + \mathit{coll}.q \ , \\
\mathit{vfsendrep}.q &= \#\mathit{fnd}.q + \mathit{coll}.q \ , \\
\mathit{vfwakeup}.q &= \mathit{wakeup}\#q \ , \\
\mathit{vfchange}.q &= \#(\mathit{ib}.q \neq \mathit{be}.q \ \vee \ \mathit{fnd}.q) + \mathit{coll}.q \ , \\
\mathit{vfhalt}.q &= \#(\neg \mathit{term}.q) \ .
\end{aligned}$$

Each of these functions never increases, and actually decreases when q accepts a message *search*, *sendrep*, *wakeup*, *change*, *halt*, respectively. Note that we still allow more than one *wakeup* message in transit to one node, and that $\mathit{vfchange}.q$ also decreases when q accepts *wakeup* and $\mathit{ib}.q = q$. For the proof that *vfhalt* decreases when q accepts *halt*, we use (Wq3).

We use the sum $\#\mathit{explist}.q + \#\mathit{mar}.q$ to deal with the number of *report* messages. In fact, by (Mq8), this sum decreases whenever process q accepts *report*. It may increase, however, when q accepts *init* or *connect*. The assignment $\mathit{explist} := \mathit{branch}$ suggests to look for an upper bound of $\#\mathit{branch}$. By (Iq0), (Jq0), and (Jq2), we have $\#\mathit{branch}.q < \mathit{tdeg}.q$, where $\mathit{tdeg}.q$ is the degree of node q in the minimum spanning tree MST. We therefore define

$$\mathit{vfreport}.q = \#\mathit{explist}.q + \#\mathit{mar}.q + (\mathit{tdeg}.q - 1) \times \mathit{coll}.q \ .$$

This function can only increase when process q accepts *connect*, in which case it increases with at most 1. It decreases whenever q accepts *report*.

The message *connect* now needs a variant function that may subtract two. We therefore define

$$\mathit{vfconnect}.q = \mathit{tdeg}.q - 1 - \mathit{branch}.q + \#((\mathit{connect}, q) \ \mathbf{at} \ \mathit{ib}.q) \ .$$

This function is nonnegative. It only increases, with 1, when q accepts *wakeup* and $\mathit{ib}.q = q$. It decreases when q accepts $(\mathit{connect}, r)$ with $r \neq \mathit{ib}.q$, and also when $\mathit{ib}.q$ accepts $(\mathit{connect}, q)$. Note that the sum of *vfconnect* and *vfchange* never increases.

The messages *winit* are the hardest to deal with, because acceptance of *winit* need not modify the private state. On the other hand, we cannot just include the number of *winit* messages in the variant function since these messages are generated during the algorithm in a rather uncontroled way. After consideration of some alternatives, we came to the following solution. We count the number of values $u \leq \mathit{ll}.q$ for which (winit, u) is not at q. So we introduce the function

$$\mathit{cwin}.q = (\# \ u :: 0 < u \leq \mathit{ll}.q \ \ \wedge \ \ (\mathit{winit}, u) \ \textbf{not-at} \ q) \ .$$

It turns out that $\mathit{cwin}.q$ only changes when process q accepts some message *init*, *winit*, or *connect*. In order to treat these modifications, we have to verify the following two new invariants

(Xq7) $\quad (winit, u)\#q \leq 1$,
(Xq8) $\quad (winit, u)$ **at** $q \quad \Rightarrow \quad u > 0$.

We use these invariants to show that $cwin.q$ increases when q accepts a *winit* message. Using (Kq6), (Kq7), and (Uq3), we show that $cwin.q$ increases when q accepts an *init* message. It follows from (Wi*) that $cwin.q$ increases when q accepts a message $(connect, ib.q)$.

It is easy to see that $cwin.q \leq ll.q$ and hence that $cwin.q \leq L - 1$. We therefore define

$$vfwin.q \quad = \quad L - 1 - cwin.q \ .$$

This function is nonnegative, it never increases, and decreases whenever q accepts *winit*, *init*, or $(connect, ib.q)$.

9.4 Construction of the variant function

We combine the various functions constructed above into one local variant function

$$\begin{aligned} vfloc.q \quad = \quad & vfaa.q + vfsearch.q + vfsendrep.q \\ & + vfwakeup.q + vfchange.q + vfhalt.q \\ & + vfreport.q + vfconnect.q + vfwin.q \ . \end{aligned}$$

Function $vfloc.q$ never increases. It decreases whenever q accepts a message different from *connect* and $(ask, -, -, id)$ with $id \neq ci.q$. It also decreases when some process p accepts $(connect, q)$ or $(ask, q, -, id)$ with $id \neq ci.p$. So, if p accepts a message, there is precisely one process q such that $vfloc.q$ decreases and all other functions $vfloc.r$ do not increase.

We therefore combine the local variant functions into

$$vf \quad = \quad (\sum q \in V :: vfloc.q) \ .$$

This function decreases whenever some process p accepts a message. By construction vf takes values in the natural numbers. This proves that the number of messages that can be accepted during execution of the algorithm is bounded by the initial value of vf.

9.5 The message complexity of the algorithm

We now calculate the initial value of vf to determine the message complexity. For simplicity we assume that initially there is precisely one *wakeup* message to every node. Initially, $coll.q = L - 1$, and $bash.q$ is the set $Nhb.q$ of neighbour nodes of q. Careful calculation yields that initially

$$vfloc.q \quad = \quad 2 \times \#Nhb.q + tdeg.q \times L + 5 \times L - 3 \ .$$

Now let $n = \#V$ be the number of nodes and $e = \#E$ the number of edges. Summing over all nodes, we have $\sum \#Nhb.q = 2 \times e$ and $\sum tdeg.q = 2 \times (n-1)$. It follows that, initially,

$$vf \quad = \quad 4 \times e - 3 \times n + (7 \times n - 2) \times L \ .$$

At first sight, this may be disappointing, since [GHS83] has the upper bound $2 \times e + 5 \times n \times L$. The selfmessages *search* and *sendrep*, however, should not be included in the message complexity, as they can be handled locally or even be eliminated. They contribute $\#Nhb.q + 2 \times (L - 1)$ to *vfloc*, and hence $2 \times e - 2 \times n + 2 \times n \times L$ to vf. It follows that the total number of external messages is bounded by $2 \times e - n + (5 \times n - 2) \times L$. This confirms the estimate of [GHS83].

For example, for the algorithm without *winit* messages, the scenario described in section 8.5 deadlocks after 20 steps. If the algorithm is extended with *winit* messages, the scenario properly terminates in 63 steps, whereas *vf* has the initial value 108.

One should notice the difference between message complexity and time complexity. Message complexity is the maximum number of messages sent during any execution. Time complexity is the worst case execution time assuming that all processes act concurrently, that each message takes at most one time unit to reach its destination, and that computation time is negligible. We refer to [SiB95] for a minimum spanning tree algorithm with a better *time* complexity than ours and [GHS83].

10 The algorithm

In this chapter we present the resulting algorithm. The ghost variables *fnd*, *srch*, *explist*, *bash*, and the actions upon them are treated between parentheses. Each process has the private variables

$$
\begin{array}{l}
ib, be, te : node \; ; \\
term, mar \,\{\, , fnd, srch \,\} \; : boolean \; ; \\
branch, bas \,\{\, , explist, bash \,\} \; : \textbf{set of } node \; ; \\
ll, bw, fc, ci : number \quad .
\end{array}
$$

We use functions *lewe* and *lenb* for least weight and least neighbour of a node q with respect to a set of nodes S. If there is a node $r \in S$ with $w.(q,r) < \infty$ and $w.(q,r) \leq w.(q,x)$ for all $x \in S$ then $lewe.(q,S) = w.(q,r)$ and $lenb.(q,S) = r$. Otherwise $lewe.(q,S) = \infty$ and $lenb.(q,S) = q$.

Initial conditions:

$$
\begin{array}{l}
\quad ib.q = q \quad \wedge \quad te.q = q \quad \wedge \quad branch.q = \emptyset \quad \wedge \quad ll.q = 0 \\
\wedge \quad \neg term.q \quad \wedge \quad \neg mar.q \quad \wedge \quad fc.q = 0 \\
\wedge \quad bas.q = \{r \mid w.(q,r) < \infty\} \quad \wedge \quad buf.q = \{(wakeup)\} \\
\{ \; \wedge \quad \neg fnd.q \quad \wedge \quad \neg srch.q \quad \wedge \quad explist.q = \emptyset \quad \wedge \quad bash.q = bas.q \; \} \\
\wedge \quad bw.q = lewe.(q,V) \quad \wedge \quad bw.q < \infty \quad \wedge \quad be.q = lenb.(q,V) \; .
\end{array}
$$

We first give procedure *initp*, which occurs in the messages *connect* and *init*.

$$
\begin{array}{l}
\textbf{proc} \;\; initp \; (v, id) = \\
\quad ll := v \; ; \; ci := id \; ; \\
\quad be := ib \; ; \; bw := \infty \; ; \\
\quad fc := \#branch + 2 \; ; \\
\quad \{ \; explist := \; branch \; ; \; fnd := true \; ; \; srch := true \; \} \\
\quad delay \; (sendrep) \; ; \\
\quad delay \; (search) \; ; \\
\quad mcast \; (branch, init, v, id) \\
\textbf{end} \; .
\end{array}
$$

The eleven messages are declared by

$$
\begin{array}{l}
\textbf{accept} \; (wakeup) = \\
\bullet \; \textbf{if} \;\; ib = self \;\; \textbf{then} \\
\quad\quad ib := be \; ; \\
\quad\quad bas := bas \setminus \{be\} \; ; \\
\quad\quad send(be, connect, self, ll) \\
\quad \textbf{fi} \\
\textbf{end} \; .
\end{array}
$$

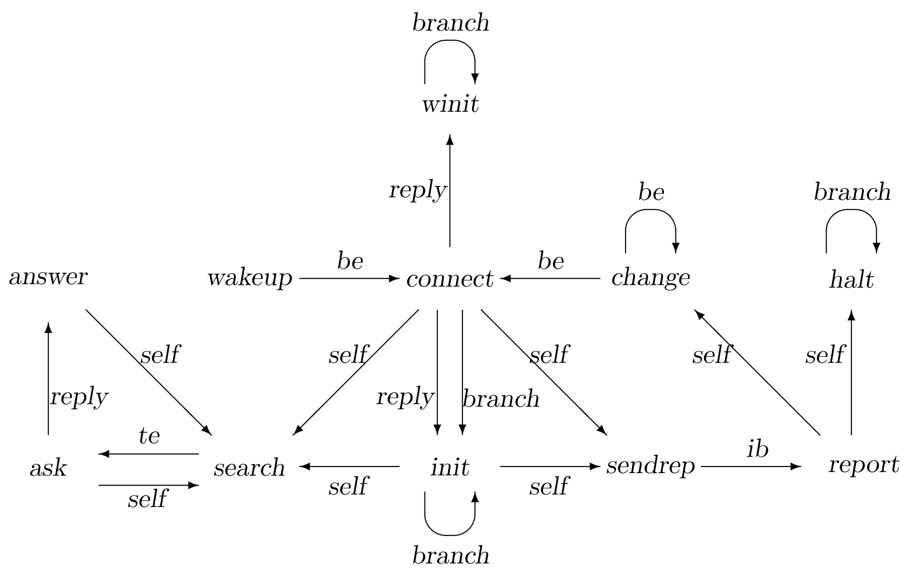
$$
\begin{array}{l}
\textbf{accept} \; (change) = \\
\bullet \; \textbf{if} \;\; be \in branch \; \textbf{then} \\
\quad\quad send(be, change) \\
\quad \textbf{else} \;\; send(be, connect, self, ll) \;\; \textbf{fi} \;\; ; \\
\quad branch := (branch \cup \{ib\}) \setminus \{be\} \; ; \\
\quad bas := bas \setminus \{be\} \; ; \\
\quad ib := be \\
\textbf{end} \; .
\end{array}
$$

```
accept (connect, j, v) =
  enabling  j = ib  ∨  v < ll
• if j = ib then
    mar := true ;
    initp(ll + 1, w.(self, j))
  else
    branch := branch ∪ {j} ;
    bas := bas \ {j} ;
    if  w.(j, self) < bw  then
      send (j, init, ll, ci) ;
      { explist := explist ∪ {j} }
      fc := fc + 1
    else  send (j, winit, ll, ci)  fi
  fi
end .

accept (init, v, id) =
  enabling ¬mar
• initp (v, id)
end .

accept (sendrep) =
  enabling fc = 1
• fc := 0 ; { fnd := false }
  send (ib, report, self, bw)
end .

accept (report, j, v) =
  enabling  j ≠ ib ∨ (mar ∧ fc = 0)
• if  j ≠ ib  then
    fc := fc − 1 ;
    { explist := explist \ {j} }
    if  v < bw  then
      be := j ; bw := v
    fi
  else
    mar := false ;
    if bw < v then delay (change)
    elsif v = ∞ then delay (halt) fi
  fi
end .

accept (halt) =
• term := true ;
  mcast(branch, halt)
end .
```

```
accept (search) =
• if  lewe.(self, bas) < bw  then
    te := lenb.(self, bas) ;
    send (te, ask, self, ll, ci)
  else
    { srch := false }
    fc := fc − 1
  fi
end .

accept (ask, j, v, id) =
  enabling v ≤ ll
• if  ci ≠ id  then
    send (j, answer, false)
  else
    bas := bas \ {j} ;
    { bash := bash \ {j} }
    if  j = te  then
      te := self ;
      delay (search)
    else  send (j, answer, true)  fi
  fi
end .

accept (answer, b) =
• if  b  then
    bas := bas \ {te} ;
    { bash := bash \ {te} }
    delay (search)
  else
    fc := fc − 1 ;
    { srch := false }
    if  w.(self, te) < bw  then
      be := te ; bw := w.(self, te)
    fi
  fi  ;
  te := self ;
end .

accept (winit, v, id) =
• if  ll < v  then
    ll := v ; ci := id ; be := ib ;
    mcast (branch, winit, v, id)
  fi
end .
```

The invariants of the above algorithm are listed in the Appendix.

In the diagram below we give the calling relations between these eleven messages. An arrow from message *kw0* to message *kw1* indicates that the acceptance of *kw0* can result in the sending of a message *kw1*. The destination of the message is indicated at the arrow. The destination *branch* means that the message is sent to all elements of *branch* and *reply* means that the message is sent back to the sender of *kw0*; the names *be* and *te* stand for best-edge and test-edge, respectively.



11 Comparisons

If we compare the above algorithm with the version of [GHS83], we get some differences that have yet to be mentioned or emphasized.

0. In [GHS83], the variable *in-branch*, which is our *ib*, is reset by *Initiate* (our message *init*), and not by *Change-root* (our message *change*) as in our version. Note however that on page 72 of [GHS83] a message *Change-core* is said to have the effect that "the inbound edge ... is changed to correspond to *best-edge*". Secondly, the handshake that forms a new core requires two *Initiate* messages not needed in our version. These two deviations from [GHS83] make it hard to adapt our proof to the version of [GHS83].

1. In [GHS83], the private variables get their initial values when executing *wakeup*. If one wants to formulate invariants of the algorithm, it is more convenient that the initial values are really initial, cf. [WLL88].

2. We have unified the messages *Accept* and *Reject* of [GHS83] into one message *answer*. We have split the message *Initiate* into two messages *init* and *winit*, since they need different motivations and use different invariants. The latter fact is mainly due to our decision to eliminate the order of the messages. Our message *init* can be disabled. This is also because of message reordering.

3. Our selfmessage *search* is an optimized version of procedure *test* of [GHS83]: a node p only sends an *ask* message to neighbour q when the weight of the edge is less than $bw.p$. This applies when node p has obtained a small value of *bw* by a *report* from one of its children. We have a similar optimization in the **else** part of *connect*, where an *init* message is only sent if the relevant edge has a weight less than $bw.p$. A third optimization in our version is that the **then** part of message *connect* does not send an *init* message to the companion as in [GHS83, WLL88], but itself executes procedure *initp*. This optimization is related to our decision that variable *ib* must be modified by *change* rather than *init*. All these modifications however do not influence the estimates for the worst case complexity.

4. At three points the algorithm of [GHS83] needs fifo channels, although Tel ([Tel94], pp. 67, 244) suggests otherwise. The first point is that, when a new core is formed, the *Initiate* message must not be passed by the *Report* message that may dissolve the core. Secondly, such a dissolving *Report* message must not be passed by a new *Initiate* message.

Since we do not require fifo channels, we have to avert these dangers by other means. This is done by means of disabling with the boolean variable *mar*. In fact, the invariants

(Qq0) $\quad (report, ib.q)$ **at** $q \quad \wedge \quad \neg mar.q \quad \Rightarrow \quad (connect, ib.q)$ **at** q and
(In*Re) $\quad init$ **at** $q \quad \wedge \quad mar.q \quad \Rightarrow \quad (report, ib.q)$ **at** q

imply that the disabled message at the lefthand side is being sent after the enabling message at the right.

The third point where the version of [GHS83] requires fifo channels, is that different *Initiate* messages must not pass each other. We have solved this point by the condition $ll < v$ in message *winit*. All three points were found in our proof effort.

5. Our set variables *branch* and *bas* replace the status of edge variable *SE* of [GHS83]. The status of node variable *SN* of [GHS83] is replaced by our ghost variable *fnd*. The variable *find-count* is replaced by our variable *fc*, which however counts more than the number of elements of *explist*, as in [GHS83].

Let us conclude with a comparison of our approach to the one of [WLL88]. At the level of the final proof, our layered design is merely an informal concept, whereas the proof of [WLL88] is based on a formal theory of lattices of automata. On the other hand, our model of concurrency is more abstract than the one of [WLL88]: we only need one bag for all messages in transit to a given node, where [WLL88] uses three FIFO buffers for each edge of the graph. The final algorithm of [WLL88] is much closer to [GHS83] than ours is. In fact, our version is the result of a formally independent design that was strongly inspired by [GHS83], but we did not aim at an exact copy in the irrelevant details.

12 Hearing the witness NQTHM

The theorem prover NQTHM serves as our witness for the correctness of the algorithm. So we have to deal with two questions: is the witness reliable and what does it say. For the first question, we can only state that the soundness of NQTHM has never been disputed.

In this Chapter we deal with the second point, the testimony of the witness. As always, the answers of the witness depend on the questions posed. In our case a number of definitions is presented to the prover and (after many sessions of cross examinations) the prover testifies and proves a short list of final theorems. The sessions with cross examinations are represented in an abstract way by the preceding chapters of this paper. Here we focus on the definitions that are needed to understand and evaluate the final testimony.

For the input to the theorem prover we refer to our WWW pages, [Hes@]. This input consists of a number of files with the extension `events`. They can be consulted for every detail the reader wants to go into. In this Chapter we mainly describe the first file `ghsAB` and the last one `ghsZ`.

12.1 The witness learns an asynchronous algorithm

The starting point is the mathematical model of asynchrony as presented in Section 1.2. So, we have to argue about a global state that consists of private states of the processes together with the bag of messages in transit.

We organize the private states as association lists, i.e., lists of key–value pairs that are inspected by the standard function `assoc`. The buffer of a node is at key `'buffer`. The global state is then an association list with a private state for each node.

We construct a function `step` that yields a new global state, given a current global state `x`, dependent on a declaration `d`, when process `p` tries to accept the first message `m` in its buffer.

```
(defn step (p d x)
    (let ((m (message p x)))
         (if (enabledm m p d x)
             (exe p
                   (findcmd m d)
                   (parameters m p d)
                   (popbuffer p x) )
             x ) ) )
```

The new state equals `x` if the message is disabled. Otherwise the command of `m` is executed with the parameters of the message, and the message is removed from the buffer.

Function `step` lets process `p` try and execute the first message in its buffer. The model, however, is more nondeterministic. It only requires that, whenever the bag of enabled messages is nonempty, one will be accepted eventually. Since disabled messages do not modify the global state, we have to construct a function in which an arbitrary enabled message is accepted.

We therefore construct a function `genstep` in which an arbitrary enabled process from the list of processes `plist` accepts an arbitrary enabled message, if one exists.

```
(defn genstep (oracle plist d x)
    (if (enabledany plist d x)
        (let ((p (favproc (car oracle) plist d x)))
             (step p d
                   (swapbufena (cdr oracle) p d x) ) )
        x ) )
```

Here `favproc` is a function that chooses an enabled process from `plist` and `swapbufena` permutes the messages in the buffer of `p` in such a way that its head is an enabled message, if possible. The nondeterminacy of function `genstep` is guided by the argument `oracle`. It can be verified, that `oracle` is always a free variable, never subject to additional constraints.

Finally, to model a number of nondeterminate steps we introduce the function

```
(defn execution (n ora plist d x)
    (if (zerop n) x
        (execution (sub1 n) (cdr ora) plist d
                   (genstep (car ora) plist d x) ) ) )
```

Here variable `ora` serves as a list of independent oracles.

So much for the model of asynchrony. The same introductory definitions can be used for any asynchronous algorithm. We used them for instance to treat Segall's PIF algorithm, cf. [Hes97a].

In order to avoid the dichotomy between edges and nodes, we have chosen to treat edges as pairs of nodes. So a graph is represented as an association list that assigns weights to pairs of nodes. We use the value `f`, i.e., `(false)`, to represent the value ∞. Here we exploit the untyped nature of NQTHM. We introduce a weight function `w` such that `(w g x y)` is the weight $w.(\mathtt{x},\mathtt{y})$ of the edge between nodes x and y in graph encoded by `g`.

We now define execution of one step of the algorithm by

```
(defn stepghs (g p x)
    (step p (dcl-ghs g) x) )
```

where `(dcl-ghs g)` is the declaration of the messages according to Chapter 10, with respect to graph `g`. Execution of `n` subsequent nondeterminate steps of the algorithm is defined by

```
(defn ghs (n ora g x)
    (execution n ora (nodes g) (dcl-ghs g) x) )
```

We also formulate the conditions on the graph

```
(defn goodgraph (g)
    (and (connectedgraph g)
         (lessp 1 (card-of (nodes g)))
         (listp (car g))
         (is-set (weightlist g)) ) )
```

So, the graph must be connected and have at least two nodes. The third condition is needed for the syntactic manipulations used in the elimination of the ghost variables. The fourth condition forces that all weights differ.

This concludes the discussion of the definitions needed to evaluate the testimony of the theorem prover for the algorithm with the ghost variables. This list is contained in the first 600 lines of the events file `ghsAB`. The remainder of `ghsAB` is concerned with the removal of ghost variables, but we first turn to the proof obligations for the algorithm with ghost variables.

12.2 The final testimony

We now skip many lines of input to the prover and come to the last events file `ghsZ`. Here we get the final testimony. That is, if NQTHM proves all lemmas in the input files, the lemmas in `ghsZ` contain the proof obligations presented in Section 2.2.

It is convenient to separate the initial state of the algorithm from the final conditions by means of an invariant `globalinv`. This invariant is constructed as the conjunction of the universal quantifications of the invariants introduced in the previous Chapters of this paper, but for the present purposes the form of the invariant is irrelevant. Only its constructive existence matters, and the fact that it is an invariant: it holds initially and it is preserved. These assertions are captured in

```
(lemma globalinv-is-initialized (rewrite)
    (implies (goodgraph g)
             (globalinv g (initstate g ora)) ) )

(lemma ghs-preserves-globalinv (rewrite)
    (implies (globalinv g x)
             (globalinv g (ghs n ora g x)) ) )
```

Now the four proof obligations of Section 2.2 are proved one by one. Firstly, predicate (Goal) is an invariant, since it follows from `globalinv`:

```
(lemma goal-1 (rewrite)
    (implies (and (member p (nodes g))
                  (member q (nodes g))
                  (member r (nodes g))
                  (terminated p x)
                  (globalinv g x) )
             (iff (mintree g q r)
                  (member r (branch+ q x)) ) ) )
```

Secondly, when all messages are disabled, *term.q* holds for all nodes *q*:

```
(lemma all-nodes-know-termination (rewrite)
    (implies (and (member q (nodes g))
                  (not (enabledany-ghs g x))
                  (globalinv g x) )
             (terminated q x) ) )
```

Thirdly, if *term.q* holds, all messages in transit to process *q* are enabled and equivalent to *skip*: process *q* accepts them and does nothing.

```
(lemma terminated-skip (rewrite)
    (implies (and (member q (nodes g))
                  (terminated q x)
                  (globalinv g x)
                  (listp (buffer q x)) )
             (equal (stepghs g q x)
                    (popbuffer q x) ) ) )
```

The condition `(listp (buffer q x))` expresses that *buf.q* is nonempty. The term `(popbuffer q x)` stands for the global state obtained when process *q* removes the first message from *buf.q*.

Finally, after a bounded number of atomic steps all messages are disabled. This fact is expressed in a more positive way: if after `n` steps some process is (still) enabled, than `n` is less than some bound that only depends on the graph and the initial state. Here we use the variant function *vf*, constructed in Section 9.

```
(functionally-instantiate ghs-terminates (rewrite)
    (implies (and (enabledany-ghs g (ghs n ora g x))
                  (globalinv g x) )
             (lessp n (vf g (nodes g) x)) )
    gstep*-terminates
    ( ... ) )
```

Here we instantiate an axiomatic theory that proves that, if every step of an algorithm decrements a function `vf`, the algorithm terminates in at most `(vf x)` steps. See [Hes97b] for the use of axiomatic theories in NQTHM.

This concludes the proof obligations for the algorithm with ghost variables.

12.3 The final removal of ghost variables

At this point the verified algorithm has the ghost variables *fnd*, *srch*, *explist*, *bash*. These variables are never inspected and do not occur in the specification. So they can be eliminated, cf. [OwG76], (3.7).

The algorithm without the ghost variables is defined in the second part of the events file `ghsAB`, by means of a general function `unghost-dcl` that removes ghost variables, and a constant `varset-ghs` that indicates which variables must be retained.

```
(defn dcl-ghs0 (g)
    (unghost-dcl (varset-ghs) (dcl-ghs g)) )
```

Here the reader can choose to go through the definitions of `unghost-dcl` and `varset-ghs`, or to ask NQTHM's execution environment for the meaning of `(dcl-ghs0 'g)`.

Since it uses the ghost variables, the global invariant is no longer available. We therefore define function `ghs0` to yield the global state after `n` steps from the initial state.

```
(defn ghs0 (n ora g)
    (execution n (cdr ora) (nodes g)
               (dcl-ghs0 g)
               (initstate0 g (car ora)) ) )
```

Here `initstate0` is the projection of the initial state `initstate` to the set of variables retained.

The main theorems are rather similar to the ones proved in the previous Section. In three of the four main theorems below, we use `x` to stand for this global state, via NQTHM's `let` construct.

```
(lemma goal (rewrite)
    (let ((x (ghs0 n ora g)))
         (implies (and (member p (nodes g))
                       (member q (nodes g))
                       (member r (nodes g))
                       (goodgraph g)
                       (terminated p x) )
             (iff (mintree g q r)
                  (member r (branch+ q x)) ) ) ) )

(lemma all-nodes-know-termination-again (rewrite)
    (let ((x (ghs0 n ora g)))
         (implies (and (member q (nodes g))
                       (not (enabledany-ghs0 g x))
                       (goodgraph g) )
                  (terminated q x) ) ) )

(lemma terminated-skip-again (rewrite)
    (let ((x (ghs0 n ora g)))
         (implies (and (member q (nodes g))
                       (terminated q x)
                       (goodgraph g)
                       (listp (buffer q x)) )
                  (equal (step q (dcl-ghs0 g) x)
                         (popbuffer q x) ) ) ) )
```

```
(lemma ghs0-terminates (rewrite)
    (implies (and (enabledany-ghs0 g (ghs0 n ora g))
                  (goodgraph g) )
             (lessp n (vfstart g (nodes g))) ) )
```

In the last lemma, function `vfstart` gives a bound independent of the global state. We can do this since `ghs0` starts at the initial state.

12.4 Overview of the events files

The size of the input files to the prover for any project greatly depends on the style and the proficiency of the user. Yet such numbers give an indication of the amount of work to be done. In the table below we list the nine events files, each with number of lines, number of events, and an indication of the contents. An event is a definition, a lemma, or a disable or enable event.

file	# lines	# events	contents
`ghsAB`	827	111	main definitions
`ghsC`	350	58	auxiliary definitions
`ghsD`	894	114	semantic lemmas
`ghsE`	2178	328	graph theory
`ghsF`	574	75	elimination of ghost variables
`ghsJR`	11805	1588	invariants for safety
`ghsSY`	15335	1747	invariants and variant functions
`ghsZ`	183	12	proof obligations
total	32146	4033	

The event files can be obtained from [Hes@]. The theorem prover NQTHM can be obtained (also for free) by ftp from Computational Logic Inc. Information is available at `nqthm-request@cli.com`.

As explained above, the reader who wants to judge whether the algorithm is proved, need only read the files `ghsAB` and `ghsZ`, and then submit all event files in order to NQTHM by means of NQTHM's command `prove-file`. This should result in a file `ghsZ.proved` where NQTHM certifies that it proved the files, and that no nondefinitional axioms were assumed.

13 Conclusions

Redesign of the algorithm provided motivation for almost all design decisions of [GHS83]. We were able to add some minor optimizations, without making the proof more complex. The grain of atomicity has been made somewhat finer by the introduction of the selfmessages *search*, *sendrep*, and at one point *change* and *halt*.

Early in the design we decided that fifo channels should not be needed for the algorithm and would complicate the proof unnecessarily. This guess turned out to be justified. Although the original version of [GHS83] needs fifo channels, the fifo assumption has been removed rather easily, see Section 11.

The proof techniques used are completely classical: ghost variables, invariants, and variant functions for termination. They were combined with the use of a powerful first-order theorem prover for book-keeping. The proof required much work, more or less quadratic in the number of invariants. For, with every extension, we had to go through all previous invariants. In many cases, the theorem prover decided that no new arguments were needed, but usually there was a fraction that needed additional arguments.

14 Appendix: list of invariants

The algorithm exposed in Chapter 10 has the invariants listed below. In these invariants we use the private variables and the ghost variables mentioned in Chapter 10, and also the following derived variables:

$$Ci.q = \textbf{if}\ \ ll.q > 0\ \ \textbf{then}\ \ ci.q\ \ \textbf{else}\ \ q\ \ \textbf{fi}\ .$$
$$jb.q = \textbf{if}\ \ (connect, q)\ \textbf{not-at}\ ib.q\ \ \textbf{then}\ \ ib.q\ \ \textbf{else}\ \ q\ \ \textbf{fi}\ .$$

JB^* is the reflexive transitive closure of relation JB given by

$$(q, r) \in JB \quad \equiv \quad q \neq r \quad \wedge \quad (jb.q = r \quad \vee \quad jb.r = q)\ .$$

The list of 166 constituent invariants

(Iq0) $ib.q = q \quad \vee \quad (q, ib.q) \in MST$.
(Iq1) $ib.q = q \quad \Rightarrow \quad (q, be.q) \in MST$.
(Iq2) $change$ **at** $q \quad \Rightarrow \quad be.q \neq ib.q$.
(Iq3) $w.(q, be.q) < \infty$.

(Jq0) $q \in branch.r \quad \Rightarrow \quad ib.q = r$.
(Jq1) $q \in branch.(ib.q) \quad \vee \quad (connect, q)$ **at** $ib.q \quad \vee \quad ib.(ib.q) = q$.
(Jq2) $ib.q \notin branch.q$.
(Jq3) $ib.q = q \quad \Rightarrow \quad branch.q = \emptyset$.
(Jq4) $(connect, q)$ **at** $r \quad \Rightarrow \quad ib.q = r$.
(Jq5) $(connect, q)$ **not-at** q .
(Jq6) $change$ **at** $q \quad \Rightarrow \quad ib.q \neq q$.
(Jq7) $change$ **at** $q \quad \Rightarrow \quad ib.(ib.q) = q$.
(Jq8) $change$ **at** $q \quad \Rightarrow \quad (connect, q)$ **not-at** r .
(Jq9) $change\#q \leq 1$.

(Jq10) $\quad$ *change* **at** $q \quad \Rightarrow \quad$ *change* **not-at** $ib.q$.
(Jq11) $\quad$ $(connect, r)$ **at** $q \quad \Rightarrow \quad r \notin branch.q$.
(Jq12) $\quad$ *change* **at** $q \quad \Rightarrow \quad (connect, ib.q)$ **not-at** q .
(Jq13) $\quad$ $(connect, r)\#q \leq 1$.

(Kq0) $\quad$ $ib.(ib.q) \neq q \quad \wedge \quad fnd.q \quad \Rightarrow \quad fnd.(ib.q)$.
(Kq1) $\quad$ $ib.(ib.q) \neq q \quad \Rightarrow \quad ll.q \leq ll.(ib.q)$.
(Kq2) $\quad$ *change* **at** $q \quad \Rightarrow \quad \neg fnd.q$.
(Kq3) $\quad$ *change* **at** $ib.q \quad \Rightarrow \quad \neg fnd.q$.
(Kq4) $\quad$ *init* **at** $q \quad \Rightarrow \quad fnd.(ib.q)$.
(Kq5) $\quad$ $fnd.q \quad \Rightarrow \quad ib.(ib.q) \neq ib.q$.
(Kq6) $\quad$ $(init, v)$ **at** $q \quad \Rightarrow \quad v = ll.(ib.q)$.
(Kq7) $\quad$ *init* **at** $q \quad \Rightarrow \quad ll.q < ll.(ib.q)$.
(Kq8) $\quad$ $init\#q \leq 1$.
(Kq9) $\quad$ *init* **at** $q \quad \Rightarrow \quad \neg fnd.q$.
(Kq10) $\quad$ $(connect, ib.q)$ **at** $q \quad \Rightarrow \quad \neg fnd.q$.
(Kq11) $\quad$ $(connect, q)$ **not-at** $ib.q \quad \Rightarrow \quad ll.q \leq ll.(ib.q)$.
(Kq12) $\quad$ $(connect, q)$ **not-at** $ib.q \quad \wedge \quad (connect, ib.q)$ **at** q
$\Rightarrow \quad ll.(ib.q) = 1 + ll.q$.
(Kq13) $\quad$ $(connect, q)$ **at** $ib.q \quad \wedge \quad (connect, ib.q)$ **at** $q \quad \Rightarrow \quad ll.(ib.q) = ll.q$.
(Kq14) $\quad$ $fnd.q \quad \Rightarrow \quad ll.(ib.q) \leq ll.q$.

(Lq0) $\quad$ $(connect, q)$ **not-at** $ib.q \ \wedge \ ll.(ib.q) = ll.q \quad \Rightarrow \quad Ci.(ib.q) = Ci.q$.
(Lq1) $\quad$ $(init, -, id)$ **at** $q \quad \Rightarrow \quad id = Ci.(ib.q)$.
(Lq2) $\quad$ $ib.q = q \quad \Rightarrow \quad ll.q = 0$.
(Lq3) $\quad$ $(connect, q, v)$ **at** $ib.q \quad \Rightarrow \quad ll.q = v \quad \vee \quad ib.(ib.q) = q$.
(Lq4) $\quad$ $(init, v)$ **at** $q \quad \Rightarrow \quad v > 0$.
(Lq5) $\quad$ $ll.(ib.q) < ll.q \quad \Rightarrow \quad Ci.q = w.(ib.q, q)$.
(Lq6) $\quad$ $Ci.q = Ci.r \quad \Rightarrow \quad (q, r) \in JB^*$.
(Lq7) $\quad$ $Ci.q = w.(r, s) \quad \Rightarrow \quad (q, r) \in JB^*$.
(Lq8) $\quad$ $Ci.q = Ci.r \quad \Rightarrow \quad ll.q = ll.r$.
(Lq9) $\quad$ $Ci.q = w.(r, ib.r) \quad \wedge \quad (connect, ib.r)$ **at** $r \quad \Rightarrow \quad ll.q = 1 + ll.r$.
(Lq10) $\quad$ $Ci.q = w.(r, s) \quad \Rightarrow \quad r \in branch.s \quad \vee \quad r = ib.s$.
(Lq11) $\quad$ $Ci.q \neq \infty$.

(Mq0) $\quad$ $ib.(ib.q) \neq q \quad \wedge \quad fnd.q \quad \Rightarrow \quad q \in explist.(ib.q)$.
(Mq1) $\quad$ *init* **at** $q \quad \Rightarrow \quad q \in explist.(ib.q)$.
(Mq2) $\quad$ $fnd.q \quad \Rightarrow \quad (report, q)$ **not-at** $ib.q$.
(Mq3) $\quad$ *init* **at** $q \quad \Rightarrow \quad (report, q)$ **not-at** $ib.q$.
(Mq4) $\quad$ $(connect, r)$ **at** $q \quad \Rightarrow \quad (report, q)$ **not-at** r .
(Mq5) $\quad$ *sendrep* **at** $q \quad \Rightarrow \quad fnd.q$.
(Mq6) $\quad$ $sendrep\#q \leq 1$.
(Mq7) $\quad$ $fnd.q \quad \vee \quad explist.q = \emptyset$.
(Mq8) $\quad$ $(report, r)$ **at** $q \quad \Rightarrow \quad ib.q = r \quad \vee \quad r \in explist.q$.
(Mq9) $\quad$ $(report, r)\#q \leq 1$.
(Mq10) $\quad$ $(report, q)$ **not-at** q .
(Mq11) $\quad$ $ll.(be.q) < ll.q \quad \Rightarrow \quad be.q = ib.q$.

(Mq12) (report, r) **at** $q \quad \wedge \quad \mathit{ll}.r < \mathit{ll}.q \quad \Rightarrow \quad \mathit{ib}.q = r$.
(Mq13) $\mathit{explist}.q \subseteq \ \mathit{branch}.q$.

(Nq0) $\mathit{be}.q = \mathit{ib}.q \quad \vee \quad \mathit{be}.q \in \mathit{branch}.q \quad \vee \quad \mathit{Ci}.q \neq \mathit{Ci}.(\mathit{be}.q)$.
(Nq1) $\mathit{be}.q = \mathit{ib}.q \quad \vee \quad \mathit{be}.q \in \mathit{branch}.q \quad \vee \quad \mathit{bw}.q = w.(q, \mathit{be}.q)$.
(Nq2) $\mathit{bw}.q \leq w.(q,r) \quad \vee \quad \mathit{srch}.q \quad \vee \quad (q,r) \in \mathit{JB}^*$.
(Nq3) $w.(q, \mathit{te}.q) \leq w.(q,r) \quad \vee \quad \mathit{te}.q = q \quad \vee \quad (q,r) \in \mathit{JB}^*$.
(Nq4) $w.(q,r) = \infty \quad \vee \quad r \in \mathit{bas}.q \quad \vee \quad (q,r) \in \mathit{JB}^* \quad \vee \quad \mathit{ib}.q = r$.
(Nq5) answer **at** $q \quad \Rightarrow \quad \mathit{ll}.q \leq \mathit{ll}.(\mathit{te}.q)$.
(Nq6) $(\mathit{answer}, \mathit{false})$ **at** $q \quad \Rightarrow \quad \mathit{Ci}.q \neq \mathit{Ci}.(\mathit{te}.q)$.
(Nq7) $(\mathit{answer}, \mathit{true})$ **at** $q \quad \Rightarrow \quad (q, \mathit{te}.q) \in \mathit{JB}^*$.
(Nq8) $(\mathit{ask}, -, v)$ **at** $q \quad \Rightarrow \quad v > 0$.
(Nq9) (ask, q, v) **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{ll}.q = v$.
(Nq10) $(\mathit{ask}, q, -, \mathit{id})$ **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{Ci}.q = \mathit{id}$.
(Nq11) $\mathit{fnd}.q \quad \Rightarrow \quad \mathit{ll}.q > 0$.
(Nq12) (ask, q) **at** $r \quad \Rightarrow \quad \mathit{te}.q = r \quad \vee \quad (q,r) \in \mathit{JB}^*$.
(Nq13) $(\mathit{ask}, q, -, \mathit{id})$ **at** $r \quad \Rightarrow \quad \mathit{te}.q = r \quad \vee \quad \mathit{Ci}.r = \mathit{id}$.
(Nq14) (ask, q) **at** $r \quad \Rightarrow \quad \mathit{te}.q = r \quad \vee \quad r \notin \mathit{bash}.q$.
(Nq15) (ask, q) **at** $r \quad \Rightarrow \quad \mathit{te}.q = r \quad \vee \quad (\mathit{ask}, r)$ **not-at** q .
(Nq16) $(\mathit{ask}, \mathit{te}.q, -, \mathit{Ci}.q)$ **at** $q \quad \Rightarrow \quad \mathit{answer}$ **not-at** q .
(Nq17) $(\mathit{answer}, \mathit{true})$ **at** $q \quad \Rightarrow \quad q \notin \mathit{bash}.(\mathit{te}.q)$.

(Oq0) search **at** $q \quad \Rightarrow \quad \mathit{te}.q = q$.
(Oq1) $\mathit{search}\#q \ \leq \ 1$.
(Oq2) (ask, q) **not-at** q .
(Oq3) answer **at** $q \quad \Rightarrow \quad \mathit{te}.q \neq q$.
(Oq4) search **at** $q \quad \Rightarrow \quad \mathit{srch}.q$.
(Oq5) $\mathit{srch}.q \quad \Rightarrow \quad \mathit{fnd}.q$.
(Oq6) $\mathit{srch}.q \quad \vee \quad \mathit{te}.q = q$.
(Oq7) $\mathit{answer}\#q \leq 1$.
(Oq8) (ask, q) **at** $r \quad \Rightarrow \quad \mathit{te}.q = r \quad \vee \quad \mathit{te}.r = q$.
(Oq9) (ask, q) **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{answer}$ **not-at** q .
(Oq10) $(\mathit{ask}, q)\#r \leq 1$.

(Pq0) $(\mathit{report}, \mathit{ib}.q, v)$ **at** $q \quad \wedge \quad \mathit{bw}.q < v \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.
(Pq1) $(\mathit{report}, \mathit{ib}.q, v)$ **at** $q \quad \wedge \quad \mathit{bw}.q < v \quad \Rightarrow \quad \mathit{change}$ **not-at** $\mathit{ib}.q$.
(Pq2) $\mathit{mar}.q \quad \Rightarrow \quad (\mathit{connect}, \mathit{ib}.q)$ **not-at** q .
(Pq3) (report, q, v) **at** $\mathit{ib}.q \quad \Rightarrow \quad \mathit{bw}.q = v$.
(Pq4) $\mathit{mar}.q \quad \wedge \quad \mathit{be}.(\mathit{ib}.q) = q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.
(Pq5) $\mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.q \neq q$.
(Pq6) change **at** $q \quad \Rightarrow \quad \neg \mathit{mar}.q$.
(Pq7) (report, q) **at** $\mathit{ib}.q \quad \wedge \quad \mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.
(Pq8) $\mathit{mar}.q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.

(Qq0) $(\mathit{report}, \mathit{ib}.q)$ **at** $q \quad \Rightarrow \quad \mathit{mar}.q \quad \vee \quad (\mathit{connect}, \mathit{ib}.q)$ **at** q .
(Qq1) $\mathit{fnd}.(\mathit{ib}.q) \ \wedge \ \mathit{ib}.(\mathit{ib}.q) = q \quad \Rightarrow \quad \mathit{mar}.q \ \vee \ (\mathit{connect}, \mathit{ib}.q)$ **at** q .
(Qq2) $(\mathit{connect}, q)$ **at** $\mathit{ib}.q \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) = q$

$\Rightarrow \quad \mathit{mar}.q \quad \vee \quad (\mathit{connect}, \mathit{ib}.q)$ **at** q .

(Qq3) $\mathit{te}.q = \mathit{ib}.q \quad \Rightarrow \quad \mathit{ib}.q = q$.

(Qq4) $\mathit{ib}.q \notin \mathit{bas}.q$.

(Qq5) $\mathit{mar}.q \quad \wedge \quad \neg\mathit{fnd}.q \quad \Rightarrow \quad \mathit{te}.(\mathit{ib}.q) \neq q$.

(Qq6) $\mathit{mar}.q \quad \Rightarrow \quad q \notin \mathit{bas}.(\mathit{ib}.q)$.

(Qq7) $\mathit{be}.q \in \mathit{branch}.q \quad \Rightarrow \quad \mathit{be}.(\mathit{be}.q) \neq \mathit{ib}.(\mathit{be}.q)$.

(Qq8) $\mathit{mar}.q \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{bw}.q < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q$.

(Qq9) (report, q, v) **at** $\mathit{ib}.q \quad \wedge \quad v < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q$.

(Qq10) $\mathit{fnd}.q \quad \wedge \quad \mathit{bw}.q < \infty \quad \Rightarrow \quad \mathit{be}.q \neq \mathit{ib}.q$.

(Rq0) $\mathit{be}.q \in \mathit{branch}.q \quad \Rightarrow \quad \mathit{bw}.(\mathit{be}.q) = \mathit{bw}.q$.

(Rq1) $r \in \mathit{branch}.q \quad \Rightarrow \quad \mathit{bw}.q \leq \mathit{bw}.r \quad \vee \quad r \in \mathit{explist}.q$.

(Rq2) $\mathit{be}.q \in \mathit{branch}.q \quad \Rightarrow \quad \neg\mathit{fnd}.(\mathit{be}.q)$.

(Rq3) change **at** $q \quad \Rightarrow \quad \mathit{bw}.q \leq \mathit{bw}.(\mathit{ib}.q)$.

(Rq4) $(\mathit{connect}, q)$ **at** $r \quad \Rightarrow \quad \mathit{bw}.q = w.(q, r) \quad \vee \quad \mathit{ib}.r = q$.

(Sq0) $(\mathit{report}, \mathit{ib}.q, \infty)$ **at** $q \quad \wedge \quad \mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.

(Sq1) $\mathit{be}.q = \mathit{ib}.q \quad \vee \quad \mathit{bw}.q < \infty$.

(Sq2) halt **at** $q \quad \Rightarrow \quad (r, s) \in \mathit{JB}^*$.

(Sq3) $\mathit{term}.q \quad \Rightarrow \quad (r, s) \in \mathit{JB}^*$.

(Sq4) halt **at** $q \quad \Rightarrow \quad \mathit{up}.\infty.r$, where
$\mathit{up}.v.q \quad = \quad \neg\mathit{fnd}.q \quad \wedge \quad \mathit{jb}.q \neq q \quad \wedge \quad v \leq \mathit{bw}.q$.

(Sq5) $\mathit{term}.q \quad \Rightarrow \quad \mathit{up}.\infty.r$.

(Tq0) $\mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad (\mathit{report}, \mathit{ib}.q)$ **at** q .

(Tq1) $\mathit{mar}.q \quad \Rightarrow \quad (\mathit{connect}, q)$ **at** $\mathit{ib}.q \vee \mathit{fnd}.(\mathit{ib}.q) \vee (\mathit{report}, \mathit{ib}.q)$ **at** q .

(Tq2) $(\mathit{connect}, \mathit{ib}.q)$ **at** q
$\Rightarrow \quad (\mathit{connect}, q)$ **at** $\mathit{ib}.q \quad \vee \quad \mathit{fnd}.(\mathit{ib}.q) \quad \vee \quad (\mathit{report}, \mathit{ib}.q)$ **at** q .

(Tq3) wakeup **at** $q \quad \vee \quad \mathit{ib}.q \neq q$.

(Tq4) $\mathit{term}.(\mathit{ib}.q) \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad \mathit{halt}$ **at** $q \quad \vee \quad \mathit{term}.q$.

(Tq5) $\mathit{srch}.q \quad \wedge \quad \mathit{te}.q = q \quad \Rightarrow \quad \mathit{search}$ **at** q .

(Tq6) $\mathit{fnd}.q \quad \Rightarrow \quad \mathit{sendrep}$ **at** q .

(Tq7) $q \in \mathit{explist}.r \quad \Rightarrow \quad \mathit{init}$ **at** $q \quad \vee \quad \mathit{fnd}.q \quad \vee \quad (\mathit{report}, q)$ **at** r .

(Tq8) $\mathit{te}.q = q \quad \vee \quad (\mathit{ask}, q)$ **at** $\mathit{te}.q \quad \vee \quad \mathit{answer}$ **at** q
$\vee \quad (\mathit{ask}, \mathit{te}.q, \mathit{ll}.q, \mathit{Ci}.q)$ **at** q .

(Tq9) (ask, q) **at** $\mathit{te}.q \quad \wedge \quad \mathit{te}.(\mathit{te}.q) = q \quad \wedge \quad \mathit{Ci}.(\mathit{te}.q) = \mathit{Ci}.q$
$\Rightarrow \quad (\mathit{ask}, \mathit{te}.q)$ **at** q .

(Tq10) $\mathit{te}.q \neq q \ \wedge \ \mathit{ci}.(\mathit{te}.q) = \mathit{ci}.q \ \wedge \ q \in \mathit{bas}.(\mathit{te}.q) \quad \Rightarrow \quad (\mathit{ask}, q)$ **at** $\mathit{te}.q$.

(Tq11) $\mathit{ll}.(\mathit{te}.q) < \mathit{ll}.q \quad \Rightarrow \quad (\mathit{ask}, q)$ **at** $\mathit{te}.q$.

(Uq0) (winit, u) **at** $q \quad \wedge \quad \mathit{ll}.q < u \quad \Rightarrow \quad q \in \mathit{branch}.(\mathit{ib}.q)$.

(Uq1) (winit, u) **at** $q \quad \Rightarrow \quad u \leq \mathit{ll}.(\mathit{ib}.q)$.

(Uq2) (winit, u) **at** $q \quad \wedge \quad \mathit{ll}.q < u \quad \Rightarrow \quad \neg\mathit{fnd}.q$.

(Uq3) (winit, u) **at** $q \quad \wedge \quad \mathit{init}$ **at** $q \quad \Rightarrow \quad u < \mathit{ll}.(\mathit{ib}.q)$.

(Uq4) $(\mathit{winit}, \mathit{ll}.(\mathit{ib}.q), \mathit{id})$ **at** $q \quad \wedge \quad \mathit{ll}.q < \mathit{ll}.(\mathit{ib}.q) \quad \Rightarrow \quad \mathit{id} = \mathit{Ci}.(\mathit{ib}.q)$.

(Uq5) $(\mathit{winit}, -, \mathit{Ci}.r)$ **at** $q \quad \Rightarrow \quad (q, r) \in \mathit{JB}^*$.

(Uq6) $(\mathit{winit}, -, w.(r, s))$ **at** $q \quad \Rightarrow \quad (q, r) \in \mathit{JB}^*$.

(Uq7) $(\mathit{winit}, u, \mathit{Ci}.r)$ **at** $q \quad \Rightarrow \quad u = \mathit{ll}.r$.
(Uq8) $(\mathit{connect}, \mathit{ib}.r)$ **at** $r \ \wedge \ (\mathit{winit}, u, w.(r, \mathit{ib}.r))$ **at** $q \quad \Rightarrow \quad u = 1 + \mathit{ll}.r$.
(Uq9) $(\mathit{winit}, -, w.(r, s))$ **at** $q \quad \Rightarrow \quad r \in \mathit{branch}.s \quad \vee \quad \mathit{ib}.s = r$.
(Uq10) $(\mathit{winit}, -, \infty)$ **not-at** q .
(Uq11) (winit, u) **at** $q \quad \wedge \quad \mathit{ll}.q < u \quad \wedge \quad \mathit{be}.(\mathit{ib}.q) = q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q$.
(Uq12) (winit, u) **at** $q \quad \wedge \quad \mathit{ll}.q < u \quad \Rightarrow \quad (\mathit{report}, q)$ **not-at** $\mathit{ib}.q$.
(Uq13) $(\mathit{winit}, -, \mathit{id})$ **at** $q \quad \wedge \quad (\mathit{winit}, -, \mathit{id})$ **at** $r \quad \Rightarrow \quad (q, r) \in \mathit{JB}^*$.
(Uq14) $(\mathit{winit}, v, \mathit{id})$ **at** $q \quad \wedge \quad (\mathit{winit}, w, \mathit{id})$ **at** $r \quad \Rightarrow \quad v = w$.
(Uq15) $\mathit{ll}.q < \mathit{ll}.(\mathit{ib}.q) \quad \wedge \quad \mathit{ib}.(\mathit{ib}.q) \neq q$
$\Rightarrow \quad (\mathit{connect}, q)$ **at** $\mathit{ib}.q \quad \vee \quad \mathit{init}$ **at** $q \quad \vee \quad (\mathit{winit}, \mathit{ll}.(\mathit{ib}.q))$ **at** q .

(Vq0) $\mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{ib}.q \neq q \quad \wedge \quad \neg \mathit{mar}.q \quad \wedge \quad \mathit{bw}.q \leq \mathit{bw}.(\mathit{ib}.q)$
$\Rightarrow \quad (\mathit{connect}, \mathit{ib}.q)$ **at** $q \quad \vee \quad \mathit{change}$ **at** $q \quad \vee \quad \mathit{change}$ **at** $\mathit{ib}.q$
$\vee \quad \mathit{halt}$ **at** $q \quad \vee \quad \mathit{term}.q$.
(Vq1) $\mathit{ib}.(\mathit{ib}.q) = q \quad \wedge \quad \mathit{fnd}.q \quad \Rightarrow \quad \mathit{mar}.q$.

(Wq0) $\mathit{fc}.q \quad = \quad \#\mathit{fnd}.q + \#(\mathit{te}.q \neq q) + \#\mathit{explist}.q$.
(Wq1) $\mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad \mathit{bw}.(\mathit{ib}.q) < \infty \quad \vee \quad \mathit{init}$ **at** q .
(Wq2) $\mathit{mar}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad q \in \mathit{branch}.(\mathit{ib}.q)$.
(Wq3) $\mathit{term}.q \quad \Rightarrow \quad \mathit{halt}$ **not-at** q .
(Wq4) halt **at** $q \quad \vee \quad \mathit{term}.q \quad \Rightarrow \quad \mathit{ib}.(\mathit{ib}.q) = q \quad \vee \quad \mathit{term}.(\mathit{ib}.q)$.
(Wq5) $\mathit{halt}\#q \leq 1$.
(Wq6) halt **at** $q \quad \vee \quad \mathit{term}.q \quad \Rightarrow \quad (\mathit{report}, \mathit{ib}.q, \infty)$ **not-at** q .
(Wq7) (winit, u) **at** $q \quad \Rightarrow \quad u < \mathit{LLBW}$.

(Xq0) $2^{\mathit{ll}.q} \leq (\#\ r :: (q, r) \in \mathit{JB}^*)$.
(Xq1) $\mathit{te}.q = q \quad \vee \quad \mathit{te}.q \in \mathit{bash}.q$.
(Xq2) $\mathit{bas}.q \subseteq \mathit{bash}.q$.
(Xq3) (ask, r) **at** $q \quad \Rightarrow \quad r \in \mathit{bash}.q$.
(Xq4) $(\mathit{ask}, \mathit{te}.q)$ **at** $q \quad \Rightarrow \quad (\mathit{answer}, \mathit{true})$ **not-at** q .
(Xq5) $q \in \mathit{bash}.r \quad \Rightarrow \quad r \in \mathit{bash}.q \quad \vee \quad \mathit{te}.r = q$.
(Xq6) $\mathit{te}.q = q \quad \vee \quad q \in \mathit{bash}.(\mathit{te}.q) \quad \vee \quad (\mathit{answer}, \mathit{true})$ **at** q
$\vee \quad (\mathit{ask}, \mathit{te}.q, -, \mathit{Ci}.q)$ **at** q .
(Xq7) $(\mathit{winit}, u)\#q \leq 1$.
(Xq8) (winit, u) **at** $q \quad \Rightarrow \quad u > 0$.

The derived invariants named in the text

(An*B0) $(\mathit{answer}, \mathit{true})$ **at** $q \quad \Rightarrow \quad \mathit{te}.q \in \mathit{bash}.q$.
(An*B1) $(\mathit{answer}, \mathit{false})$ **at** $q \quad \Rightarrow \quad q \in \mathit{bash}.(\mathit{te}.q)$.
(An*f) answer **at** $q \quad \Rightarrow \quad \mathit{fnd}.q$.
(As*f) (ask, q) **at** $\mathit{te}.q \quad \Rightarrow \quad \mathit{fnd}.q$.
(Ch*jb) change **at** $q \quad \Rightarrow \quad \mathit{jb}.(\mathit{jb}.q) = q \quad \wedge \quad \mathit{jb}.q \neq q$.
change **at** $p \quad \wedge \quad \mathit{be}.p \notin \mathit{branch}.p \quad \wedge \quad w.(q, r) < w.(p, \mathit{be}.p)$
$\Rightarrow \quad ((p, q) \in \mathit{JB}^* \quad \Rightarrow \quad (p, r) \in \mathit{JB}^*)$.
(Ch-M) change **at** $q \quad \wedge \quad \mathit{be}.q \notin \mathit{branch}.q \quad \Rightarrow \quad (q, \mathit{be}.q) \in \mathit{MST}$.
(Ch-out) change **at** $q \quad \wedge \quad \mathit{be}.q \notin \mathit{branch}.q$

$\Rightarrow \quad (q, be.q) \notin JB^*$.

(Co*jb) $(connect, r)$ **at** $q \quad \wedge \quad (q,r) \in JB^* \quad \Rightarrow \quad jb.q = r$.

(Dld0) $be.q \neq q$.

(Dld1) $ib.q = q \quad \Rightarrow \quad ll.q \leq ll.(be.q)$.

(Dld2) $change$ **at** $q \quad \Rightarrow \quad ll.q \leq ll.(be.q)$.

(Dld3) $change$ **at** $q \quad \Rightarrow \quad ll.(ib.q) \leq ll.q$.

(Dld4) $fnd.q \quad \Rightarrow \quad ib.(ib.q) = q \quad \vee \quad q \in branch.(ib.q)$.

(Dld5) $(report, r)$ **at** $q \quad \Rightarrow \quad ib.q = r \quad \vee \quad fnd.q$.

(Dld6) $(report, ib.q)$ **at** $q \quad \Rightarrow \quad change$ **not-at** q .

(Dld7) $(report, ib.q)$ **at** $q \quad \wedge \quad fnd.q \quad \Rightarrow \quad ib.(ib.q) = q$.

(Dld8) $(report, q)$ **at** $be.q \quad \Rightarrow \quad be.q = ib.q$.

(Dld9) $mar.q \quad \Rightarrow \quad te.(ib.q) \neq q$.

(Dld9a) $mar.q \quad \wedge \quad fnd.q \quad \Rightarrow \quad te.(ib.q) \neq q$.

(DisAsk) $Dis.(te.q) \quad \wedge \quad (ask, q)$ **at** $te.q \quad \Rightarrow \quad ll.(te.q) < ll.q$.

(DisCo) $Dis.q \quad \wedge \quad (connect, r)$ **at** q

$\Rightarrow \quad ib.q \neq q \quad \wedge \quad ib.q \neq r \quad \wedge \quad ll.q \leq ll.r$.

(Dis-ex) $Dis.q \quad \wedge \quad Dis.r \quad \wedge \quad q \in explist.r \quad \Rightarrow \quad fnd.q$.

(Dis-ma) $Dis.q \quad \wedge \quad Dis.(ib.q) \quad \wedge \quad mar.q \quad \Rightarrow \quad fnd.q \quad \vee \quad fnd.(ib.q)$.

(DisRe) $Dis.q \quad \wedge \quad (report, r)$ **at** $q \quad \Rightarrow \quad ib.q = r \quad \wedge \quad fnd.q$.

(DisSe) $Dis.q \quad \wedge \quad fnd.q \quad \wedge \quad te.q = q \quad \Rightarrow \quad explist.q \neq \emptyset$.

(Dis-te) $Dis.q \quad \wedge \quad te.q \neq q \quad \Rightarrow \quad (ask, q)$ **at** $te.q$.

(Dkws) $Dis.q \quad \wedge \quad kw$ **at** $q \quad \Rightarrow \quad kw \notin \{init, wakeup, change, answer\}$.

(Fn*jb) $fnd.q \quad \Rightarrow \quad fnd.(jb.q) \quad \vee \quad jb.(jb.q) = q$.

(fi-JB) $fincr.p \quad \Rightarrow \quad (p, q) \in JB^*$.

(fi-up) $fincr.p \quad \Rightarrow \quad up.\infty.q$.

(Goal) $term.q \quad \Rightarrow \quad (\ (r, s) \in MST \quad \equiv \quad s \in \{ib.r\} \cup branch.r\ )$.

(In*br) $init$ **at** $q \quad \Rightarrow \quad q \in branch.(ib.q)$.

(In*C) $init$ **at** $q \quad \Rightarrow \quad (connect, q)$ **not-at** r .

(In*CC) $init$ **at** $q \quad \Rightarrow \quad (connect, ib.q)$ **not-at** q .

(In*Ch) $init$ **at** $q \quad \Rightarrow \quad change$ **not-at** q .

(In*cr) $init$ **at** $q \quad \Rightarrow \quad ib.(ib.q) \neq q$.

(In*Re) $init$ **at** $q \quad \wedge \quad mar.q \quad \Rightarrow \quad (report, ib.q)$ **at** q .

(Low-br) $q \in Low \quad \wedge \quad r \in branch.q \quad \Rightarrow \quad r \in Low$.

(Low-cr) $DIS \quad \Rightarrow \quad (\exists\, p \in Low :: \ ib.p \in Low \ \wedge \ ib.(ib.p) = p \ \wedge \ (ib.p \neq p)$.

(Low-fn) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad \neg fnd.q$.

(Low-ib) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad ib.q \in Low$.

(Low-te) $DIS \quad \wedge \quad q \in Low \quad \Rightarrow \quad te.q = q$.

(Re*cr) $(report, ib.q)$ **at** $q \quad \wedge \quad ib.(ib.q) = q \quad \wedge \quad mar.q \quad \wedge \quad bw.q < \infty$

$\Rightarrow \quad bw.q \neq bw.(ib.q)$.

$(report, r)$ **at** $q \quad \Rightarrow \quad w.(q, r) < \infty$.

(Re-fi) $(report, ib.p, \infty)$ **at** $p \quad \wedge \quad bw.p = \infty \quad \wedge \quad \neg fnd.p \quad \wedge \quad mar.p$

$\Rightarrow \quad fincr.p$.

(Re*ib) $(report, q)$ **at** $r \quad \Rightarrow \quad ib.q = r \quad \vee \quad ib.r = q$.

(Stab) predicate $(x, y) \in JB^*$ is stable.

(Thm5) $(q, r) \in JB^* \quad \wedge \quad jb.(jb.q) = q$

$\Rightarrow \quad ll.r \leq ll.q \quad \wedge \quad (ll.r = ll.q \Rightarrow Ci.r = Ci.q)$.

(Thm6) $jb.(jb.p) = p \neq jb.p \quad \wedge \quad up.v.p \quad \wedge \quad up.v.(jb.p)$
$\Rightarrow \quad ((p,q) \in JB^* \quad \Rightarrow \quad up.v.q)$.
(Thm6C) $change$ **at** $p \quad \wedge \quad (p,q) \in JB^* \quad \Rightarrow \quad \neg fnd.q \quad \wedge \quad bw.p \leq bw.q$.
(Thm6T) $fincr.p \quad \wedge \quad (p,q) \in JB^* \quad \Rightarrow \quad up.\infty.q$.
(tm-hi) $term.p \quad \Rightarrow \quad ll.q = LLBW$.
(tm-idl) $term.q \quad \Rightarrow \quad idle.q$.
(tm-ms) $term.p \quad \wedge \quad kw$ **at** $q \quad \Rightarrow \quad kw \in \{report, wakeup, halt, winit\}$.
(tm-opn) $term.p \quad \Rightarrow \quad open.q$.
(tm-Re) $term.q \quad \Rightarrow \quad report$ **not-at** q .
(tm-Wi) $term.p \quad \wedge \quad (winit, u)$ **at** $q \quad \Rightarrow \quad u \leq ll.q$.
(Wi*) $(winit, u)$ **at** $q \quad \wedge \quad ll.q < u$
$\Rightarrow \quad ib.(ib.q) \neq q \quad \wedge \quad (connect, ib.q)$ **not-at** q
$\wedge \quad (connect, q)$ **not-at** $r \quad \wedge \quad change$ **not-at** q .

References

[ApO91] K.R. Apt, E.-R. Olderog: Verification of Sequential and Concurrent Programs. Springer V. 1991.

[BoM88] R.S. Boyer, J Moore: A Computational Logic Handbook. Academic Press, Boston etc., 1988.

[ChM88] K.M. Chandy, J. Misra: Parallel Program Design, A Foundation (Addison-Wesley, 1988)

[ChG88] C. Chou and E. Gafni: Understanding and verifying distributed algorithms using stratified decomposition. In *Proceedings 7th ACM Symposium on Principles of Distributed Computing*, 1988.

[CLR90] T.H. Cormen, C.E. Leiserson, R.L. Rivest: Introduction to algorithms. The MIT Press, 1990.

[GHS83] R.G. Gallager, P.A. Humblet, P.M. Spira: A distributed algorithm for minimum–weight spanning trees. ACM Trans. on Programming Languages and Systems **5** (1983) 66–77.

[Heh89] E.C.R. Hehner: Termination is timing.In *J.L.A. van de Snepscheut (ed.): Mathematics of Program Construction.* Springer, 1989 (LNCS 375), pp. 36–47.

[Hes97a] W.H. Hesselink: A mechanical proof of Segall's PIF algorithm. Formal Aspects of Computing **9** (1997) 208–226.

[Hes97b] W.H. Hesselink: Theories for mechanical proofs of imperative programs. Formal Aspects of Computing **9** (1997) 448–468.

[Hes99] W.H. Hesselink: The verified incremental design of a distributed spanning tree algorithm: extended abstract. Formal Aspects of Computing **11** (1999) 45–55.

[Hes@] W.H. Hesselink: Web site: `http://wimhesselink.nl/mechver/ghs`.

[Jos92] M.B. Josephs. Receptive process theory. Acta Informatica **29** (1992) 17–31.

[KaM97] M. Kaufmann, J S. Moore: An industrial strength theorem prover for a logic based on Common Lisp. IEEE Transactions on Software Engineering bf 23 (1997) 203–213.

[Lyn89] N.A. Lynch: Multivalued possibilities mappings. In *J.W. de Bakker, W.-P. de Roever, G. Rozenberg (Eds.): Stepwise Refinement of Distributed Systems.* LNCS 430, Springer V., 1990, pp. 519–543.

[OwG76] S. Owicki, D. Gries: An axiomatic proof technique for parallel programs. Acta Informatica **6** (1976) 319–340.

[SiB95] G. Singh, A.J. Bernstein: A highly asynchronous minimum spanning tree protocol. Distrib. Comput. **8** (1995) 151–161.

[SdR94] F.A. Stomp, W.-P. de Roever: Principles for sequential reasoning about distributed algorithms. Formal Aspects of Computing, 6(E), pp 1–70 (1994). Can be retrieved by downloading the file FACj-6E-p1.ps.Z in directory pub/fac of ftp.cs.man.ac.uk.

[Tar83] R.E. Tarjan: Data structures and network algorithms. Society for Industrial and Applied Mathematics 1983.

[Tel94] G. Tel. Distributed Algorithms. Cambridge University Press, 1994.

[WLL88] J. Welch, L. Lamport, and N. Lynch: A lattice-structured proof technique applied to a minimum weight spanning tree algorithm. In *Proceedings 7th ACM Symposium on Principles of Distributed Computing*, 1988.

[You97] W.D. Young: Comparing verification systems: interactive consistency in ACL2. IEEE Transactions on Software Engineering **23** (1997) 214–223.

[ZwJ93] J. Zwiers and W. Janssen: Partial order based design of concurrent systems. In J. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX School/Symposium "A decade of concurrency", Noordwijkerhout, 1993, LNCS 803*, Springer Verlag, 1994, pp. 622–684.