Verification of the IEEE 1394.1 panic protocol with formal methods

van der Stap, C.J.G.

Award date:
2007

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Verification of the IEEE 1394.1 panic protocol with formal methods

By
C.J.G van der Stap

Supervisors:

J.M.T. Romijn
R. Kuiper
M.A. Reniers

Contents

1 Preface **5**

2 Introduction **7**

2.1 Preliminaries 8

3 PVS **10**

3.1 Introduction 10

3.2 Tame 12

3.3 NASA Langley PVS Library 12

3.4 Quirks and Tips 13

4 Panic **15**

4.1 1394.1 15

4.2 Termination Problem 16

4.3 Panic 17

4.4 Propagation of Panic 17

5 Model of Panic **21**

5.1 Assumptions and Abstractions 21
5.2 Topology 22
5.3 State 25
5.4 Actions 27
5.5 Fairness Restrictions 30

6 Correctness Specification 32
6.1 Properties 33
6.2 Invariants 37
6.3 Correlation of Properties and Invariants 42

7 Verification 43
7.1 Strategy 43
7.2 Invariants 43
7.3 Property 1: Outreach 45
7.4 Property 2: No Deadlock 45
7.5 Property 3: Termination 45
7.5.1 Fairness 46
7.6 Proof Overview 47
7.7 Constraints Arising in the Proofs 47

8 Achievements 48

9 Final Words 49

A How to Read the Proofs 53

B Specification of the Panic Model **54**

B.1 State 54

B.1.1 Topology 54

B.1.2 Portal 55

B.1.3 Overview 57

B.2 Actions 58

B.2.1 Helper Functions 58

B.2.2 Actions 58

C Detailed Proofs **63**

C.1 Lemmas 63

C.2 Invariants 63

C.3 Properties 101

C.3.1 Property 1: Correctness 101

C.3.2 Property 2: No Deadlock 101

C.3.3 Property 3: Termination 103

D PVS Proof of graph_from_edges **129**

Chapter 1

Preface

This report is the result of a master project at the Eindhoven University of Technology (TUe). The subject of the research project is the formal proof of the correctness of the panic protocol that is part of the IEEE 1394.1 FireWire bridge specification. The research was conducted between January 2006 and January 2007.

This research is based on earlier work by dr. J.M.T. Romijn, who studied the correctness of the IEEE 1394.1 specification with the aid of model checking and found a problem in the IEEE 1394.1 specification. This resulted in the panic protocol being added to the specification. Even though model checking the panic protocol was attempted, it was never fully completed due to the problem of state space explosion, which is inherent to model checking. Because of the state space explosion, it was chosen to do the verification of the panic protocol in this research project with theorem proving techniques.

There have been three occurrences of earlier work on the net update and panic protocols.
- Promela model of net update made by J.M.T. Romijn. This was also the project where the error in net update was discovered. These models were created simultaneously with the precisely document [Rom04].
- Internship of S. Vorstenbosch who worked on panic in LOTOS [ISO89] under the guidance of Romijn [Vor04].
- X. Hue continued the work of S. Vorstenbosh for his master thesis [Huo05].

Why is this research project useful? Since the entire IEEE 1394.1 standard is likely to become a widely used industrial standard, it is important to ensure the correctness of this standard. Since the panic protocol is a part of the IEEE 1394.1 standard and vital for its correct behavior, it is important to assert the

correctness of this protocol.

Furthermore, the panic protocol has not been formally proven yet. Even though model checking has been used to verify the panic protocol, it encounters the state space explosion problem for the proof of the panic protocol. Therefore it is easier to get a complete proof with theorem proving as opposed to model checking.

Also, this research project uses the tool PVS with the support library TAME. Since the proofs are done both by hand and in PVS a good understanding of the differences between making proofs in PVS and by hand will be gained.

The scientifically interesting aspects of this research project are: Using formal methods to ascertain the correctness of a complex distributed protocol (panic) and using a theorem prover (PVS) in combination with a distributed protocol (panic).

This report is written for an audience that has some mathematical and computer science background. An effort has been undertaken to make it more accessible for a wider audience.

Chapter 2

Introduction

This report details the research done on the formal verification of the panic protocol. The panic protocol is a part of the IEEE 1394.1 FireWire bridge specification. A detailed explanation of the panic protocol can be found in Chapter 4.

Because of the complex nature of the IEEE 1394.1 specification, the research scope is limited to the panic protocol. The aim of the research is to construct and verify a correctness specification for this protocol. This correctness specification is constructed from a number of useful properties. The details about the correctness specification are listed in Chapter 6.

To analyse the panic protocol and formulate the properties a mathematical model of the panic protocol is constructed. This model contains all the registers and actions that are part of the panic protocol. The mathematical model is contained in Chapter 5.

After specifying the properties mathematically these and their supporting invariants are verified both by hand and in the theorem prover PVS. More details about PVS are in Chapter 3. The explanation of the strategies used during the verification is in Chapter 7. The detailed proofs of the verification done by hand are located in Appendix C.

Chapter 8 details the achievements of this research project, and lastly in Chapter 9 some final words are added.

2.1 Preliminaries

For a better understanding of this report, some preliminary information about the formalization is needed. This information is listed below.

Conditional functions are formalized as: (*condition* ? *value*1 : *value*2). This function returns value1 when the condition holds, otherwise value2 is returned.

Automaton Panic is modeled as an automaton. An automaton consist of states and actions. Each state is a collection of values associated with the protocol to be modeled. There is a predicate characterizing the initial state(s), referred to as s_0. Actions model behavior, that is, the actual steps of the protocol. An action is declared as the following triple: a precondition, an effect and a label.

The label is the action name. Actions may be parameterized, as a shorthand for the same action description for each value in the domain of the parameter.

An action is guarded by the precondition. If the precondition holds in a state then the action may apply its effect to the current state. If in a state more than one action is enabled, one may be chosen non-deterministically. The effect of an action transforms the current state into the next state. This transformation executes all assignments of the effect while leaving the rest of the state unchanged.

A transition is the execution of an action. The transition from state s to state t through the execution of action a is formalized as: $s \xrightarrow{a} t$, where s and t are states and a is an action label. A sequence of transitions is formalized as $s \overset{\sigma}{\rightarrow}{}^{*} t$, where σ is a sequence of actions $\sigma = a_1 \dots a_n$ and $s = s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n = t$. If σ is infinite we write $s \overset{\sigma}{\rightarrow}{}^{*}$.

A reachable state is a state that can be reached from the initial state s_0. Formalized as: s is reachable if $s_0 \overset{\sigma}{\rightarrow}{}^{*} s$.

Verification In order to prove properties of an automaton we have predicates over states and action sequences.

The building blocks of our verification are invariants. In this context an invariant is a state predicate that must hold in all reachable states.

If the model contains all possible interesting behavior and can only begin in the defined initial state, then we are usually only interested in behavior from reach-

able states. Hence, all properties of the protocol are proven only for reachable states.

Fairness For some proofs fairness is needed. The fairness technique is needed because a part of the panic protocol depends on an unreliable broadcast. Fairness is used for the situation where the broadcast would fail infinitely often. For a better understanding of fairness a short preliminary is included.

There are many specification formalisms that incorporate the notion of fairness (see, for instance, [Jon94], [Lam94], [LT87], [MP92]). Fairness comes in two different flavors: weak and strong fairness. Informally the requirement of weak fairness disallows executions in which certain sets of transitions are continually enabled but not taken beyond a certain point, whereas the requirement of strong fairness disallows executions in which certain sets of transitions are enabled infinitely often but taken only finitely many times.

A criterion that any acceptable notion of fairness should satisfy is that it induces liveness properties in the sense of [AS85]: A live automaton is a pair (A, L) where A is an automaton and L is a subset of the executions of A (in our case, the fair executions), such that any finite execution of A can be extended to an execution in L.

In order to respect a strong fairness set T we only regard executions σ of the automaton such that either σ is finite and no $t \in T$ is enabled in the last state, or if σ contains infinitely many states that enable a $t \in T$ then σ contains infinitely many $t \in T$.

An execution σ that respects all fairness sets of an automaton is called a *fair* execution.

Chapter 3

PVS

In this chapter the theorem prover PVS is introduced. First, a general introduction of PVS is given. Next, information about the two PVS libraries that are used in this project is given and finally a small section with some quirks of and tips for PVS is given.

3.1 Introduction

PVS stands for Prototype Verification System. PVS is a tool that provides mechanized support for formal specification and verification. This tool consists mainly of a highly expressive specification language and an automated theorem prover. The user interface is integrated into emacs. The user can type the specification in emacs as well as control the theorem prover from inside emacs. Since PVS makes extensive use of lisp, the theorem prover accepts commands in lisp syntax.

PVS has a very good typing system. As well as providing basic types, PVS makes it possible for a user to add uninterpreted types, abstract data structures and recursive types. Furthermore, it is possible to constrain the domain of the types by means of predicate sub-typing or dependent typing.

This sub-typing system can lead to additional proof obligations. These obligations, TCCs (Type Check Conditions), are added automatically whenever needed. Most of these TCCs are discharged automatically by PVS, but some need to be proven manually. The purpose of the TCCs is to ensure that types do not introduce inconsistencies in the proofs.

When is a proof consistent? When defining types it is possible to introduce weird or unprovable statements. The TCCs point out these possible statements, for an example: A user creates a datatype D with constructors A, B and C and a function that takes a parameter of type D. Inside the function a case distinction is made for constructors A and B. Now it is possible to introduce something unprovable by using the function with C. PVS will generate a TCC for each place where the function is used and wants proof that the function is only used with A or B.

Unless all TCCs are discharged (proven) the theory is considered not complete. Hence, in PVS, a proof is not finished until all TCCs are discharged. However, there is one exception where TCCs are not generated: axioms. The axioms are used as factual information. Therefore inconsistencies can be introduced with axioms. As a simple example an axiom "false" can be created, it is obvious that anything is provable with this axiom.

Theorem proving with PVS is done in a sequent calculus. The sequent calculus has two types of sequents: antecedents and consequents. In the sequent calculus a proof will be complete when the collection of antecedents evaluates at least one consequent to true (e.a. the conjunction of antecedents implies the disjunction of the consequents).

A user can interact with the theorem prover by instructing it to perform primitive proof steps like rewriting, induction, quantifier rules and linear arithmetic. The primitive proof steps can be combined to form more powerful proof steps. The combination of proof steps is called a strategy. PVS provides several very high level and powerful strategies. Users can also develop custom strategies. The strategies are invoked in the same way as the primitive proof steps.

Some of the strategies allow the user to import lemmas as a new sequent. This makes it possible to organize proofs in smaller parts. Also, it makes them more manageable and allows for their re-use. PVS already has a prelude library that holds many theories with lemmas, which can be imported. The imported lemma can be used to prove the current branch of the proof tree.

When constructing a proof, a tree of proof steps is created and stored. The aim is to terminate every branch of the proof successfully. When all branches are proven the theorem prover accepts the proof as completed and saves the result and proof tree. The proofs are stored so they can be easily rerun when the specification is changed.

When a user finds that a proof step, or chain of steps, does not yield the expected result he can traverse back up the tree by undoing proof steps. Undoing a step can only be done if the node is a leaf. Undoing a step will not only remove the current node but also all sibling nodes, because the removed proof step created the node and its siblings.

3.2 Tame

TAME or Timed Automata Modeling Environment is a set of templates and support theories that are used in conjunction with PVS. TAME was originally invented to let users construct invariant and refinement proofs with PVS, without needing to know the ins and outs of the latter. The templates that TAME provides allow a user to make a PVS specification of a timed automaton.

The templates use the same precondition / effect style as our model and hand proofs of panic, see Chapter 5 and onwards. The precondition, effect, state and initial state are all separately declared. This enables the conversion back and forth from our manual proof to TAME see, Chapter 6.

The strategies that TAME supplies use the templates and allow invariant and refinement proofs, the latter is not used for this project. Invariant proofs also take the same form as used in our correctness hand proof of panic. Under the assumption that the invariant holds for a reachable state, it is required that the invariant is maintained after the execution of an enabled action.

An important strategy is the induct-and-simplify strategy. This strategy is the usual start of any invariant proof. It breaks up the invariant and creates a proof branch for each action of the automaton. The next step, like in a normal PVS proof, is to prove all branches.

While users prove the branches, TAME provides a couple of strategies related to the proof of invariants. One of the essential ones is the apply-specific-precond strategy. This strategy will add the precondition of the action in the current branch as a sequent so it can be used in the proof. Another strategy is apply-inv-lemma. This strategy will add an invariant to the current branch as a sequent, useful when the current invariant depends on another.

PVS and TAME require that invariants on which other invariants depend are found before the depending invariant. Otherwise the theorem prover will either say it cannot find the depending invariant or crash.

3.3 NASA Langley PVS Library

Another library used in this project is created by NASA Langley. During the modeling of the panic protocol in PVS the graph theories of this library are used. These theories provided a good foundation for the topology model, which is itself a graph.

One of the theories used was not yet completed. In this theory one TCC was not yet proven. During the course of the research project this TCC was proven, after some small adjustments to the theory file. The now completed theory is send back to NASA Langley and it will be included in the next release.

For details of this PVS proof see Appendix D.

3.4 Quirks and Tips

PVS is an evolving research prototype. This sometimes leads to some odd behavior. One of the more troublesome quirks occurs when the theorem prover starts to use rewrite rules in the wrong direction, resulting in an infinite rewrite chain. This can be resolved by hitting ctrl-c twice, followed by the (restore) command. Pressing ctrl-c will abort the current proof step, while the (restore) command will restore the proof back to the previous state.

Very high level strategies can be used to complete large steps in the proof. However, these strategies do not always make the best decisions. A good example of this is the strategy "grind". Grind will attempt to complete the proof by doing many more primitive steps. One of these steps will try to instantiate predicates with skolomized variables. Many times the wrong skolemized variables are chosen, which makes it impossible to successfully complete the proof branch.

Furthermore, high level strategies can hide or remove important sequents that are needed further down the proof tree. This again could make it impossible to successfully complete the current branch. Deletion can be prevented by adding certain parameters to the strategies. When a sequent is hidden it can be retrieved later. An example where unhiding is useful is after using the induct-and-simplify strategy. This strategy hides a copy of its uninstantiated induction hypothesis which can be useful in some proofs. This induction hypothesis can be unhidden and instantiated with a different portal, for instance the coPortal, to get more information to finish the proof.

Another oddity occurs when PVS fails the typecheck stating it cannot find a type, even though the type is declared in the file itself or an imported file. Closing emacs and PVS and then restarting them will normally fix this.

Besides the above mentioned behavior, there are certain proofs that take more effort to prove in PVS than with hand proofs on paper. This is mainly because the PVS prover wants an explicit mathematical proof, whereas on paper it is possible to make an argument that is widely accepted. For example, when for a TCC an injection needs to be constructed to prove that the union over a

finite set of doubletons results in a finite set. It is obvious that there is such an injection from the union to $[0 \dots 2N)$, where N is the cardinality of the finite set of doubletons. A doubleton is a set with exactly two different members. When making a proof by hand stating that such an injection indeed exists is acceptable, PVS however wants to see the injection either derived or constructed explicitly.

Chapter 4

Panic

As specified in [13904], panic is a part of the net update protocol. Net update is used to build and maintain a spanning tree in a network of IEEE 1394 buses as specified in [IEE96] and its amendment [IEE01]. However during earlier work a termination problem in net update was discovered. To solve this termination problem of net update the panic protocol has been created.

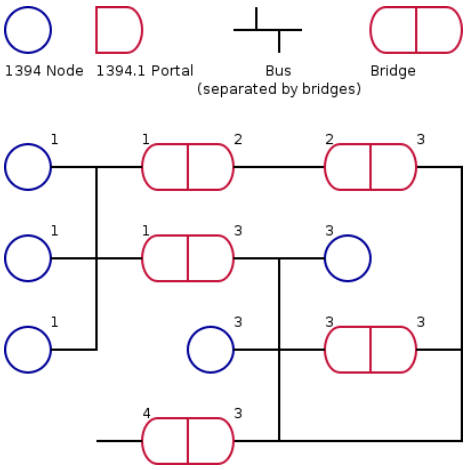
First an explanation of IEEE 1394.1 is given, followed by a description of the termination problem and panic. Lastly, the workings of the panic protocol itself are explained.

4.1 1394.1

After the standardization of the IEEE 1394 FireWire bus, the IEEE organization designed an extension. A bus is a collection of 1394 devices (like camcorders, PC's, external hard drives, etc). This extension has been published as the IEEE 1394.1-2004 standard. The 1394.1 standard defines bridges between the 1394 buses. This allows for a larger network of 1394 devices.

It was decided that the 1394.1 extension would be implemented as a distributed protocol, such that decision making is based on the information on the bus and bridge. Therefore distribution context is per bus and bridge. A device implementing the 1394.1 extension is known as a portal. A bridge between two buses exists of two of these portals. Communication between the portals of a bridge is done by an undefined synchronized communication channel.

Below is an example network with portals, buses and bridges. The buses are numbered 1 to 4. Note that all nodes and portals that are on the same bus have the same bus number.



To enable communication across the network every bridge portal needs to know how to route messages over the network. After a topology change the routing information must be updated in the portals. The protocol to update the routing information in the network is called net update. The net update protocol is triggered when a change in the topology occurs. When finished, net update will have updated all portals in the network with the new routing information.

4.2 Termination Problem

During earlier work by Romijn [vLRG03] it was discovered that after certain topology changes in the network net update would not terminate. This non-terminating behavior can occur because it is a distributed protocol. Net update uses already existing routing information in the portals. Since topology changes can happen at any time it is possible that the routing information becomes inconsistent. It turns out that these inconsistencies can cause non-termination.

Network-wide it is easy to discover when net update is caught in non-terminating behavior. unfortunately, this exact condition cannot be checked by an individual portal or bus. Since this is a requirement, because of the distributed nature of net update, a stronger variant of this condition is needed.

This stronger condition is guaranteed to hold when net update is caught in non-terminating behavior, but in some rare cases it can also trigger when net update is functioning normally. This stronger condition is henceforth referred

to as the panic condition.

4.3 Panic

After discovering the termination problem a fix was needed. The result of this fix is the panic protocol. This protocol is an addition to the net update protocol which is activated when the panic condition is detected.

The goal of panic is to remove the inconsistencies from the routing information in the portals. This is done by resetting the routing information, in each portal that panic reaches, to the initial state of net update. After panic finishes, net update will make a new attempt to update the topology information.

4.4 Propagation of Panic

For a better understanding of the panic protocol, the propagation of panic over the network is explained informally in this section.

Panic starts on a portal when the panic condition is detected. The first step of the propagation is to notify the bus it is on. This notification is done by the broadcast of a panic message over the bus. To ensure that all portals on the bus have detected panic the bus is synchronized using some lower level 1394 service. During the synchronization it is decided whether the broadcast was successful or not. If some portals did not detect panic, then the broadcast is attempted again. When all portals on the bus did detect panic, all coPortals are notified and panic propagates to the adjacent buses.

This way of propagation resembles an expanding wave. The wave is started in a portal and will expand over the adjacent buses. An important detail is that both portals on the bridge synchronize to resume normal operation at the same time. Therefore, the panic wave will not flow back into the originating bus.

Panic will stop spreading when one of the following four conditions hold:

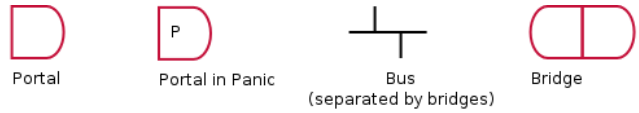
- A bus in panic is a at the edge of the network, which is when there is only one bridge connected to the bus. Panic arrives on the bus through the only bridge and has nowhere left to spread.

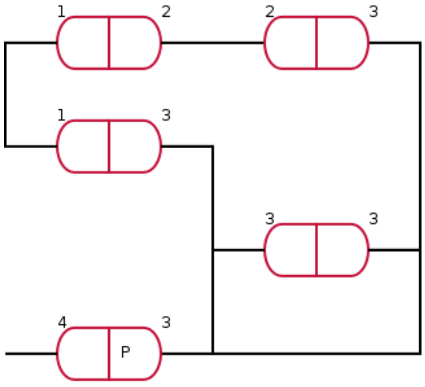
- A maximum number of hops is reached. PANIC messages carry a hops value. This hops value is increased when panic propagates across a bridge. The hops value is checked before notifying the bus. If the value hits a fixed boundary, the bus is not notified and the portal completes panic by notifying its coPortal.

- A collision of two waves occurs on a bridge. A collision occurs when both portals of a bridge have started panic individually. This situation can occur because of the loops in the network with panic propagating in both ways or when more than one portal signaled a panic condition in the network. When the waves collide, both waves will stop at the bridge because the portals resume normal operation at the same time.

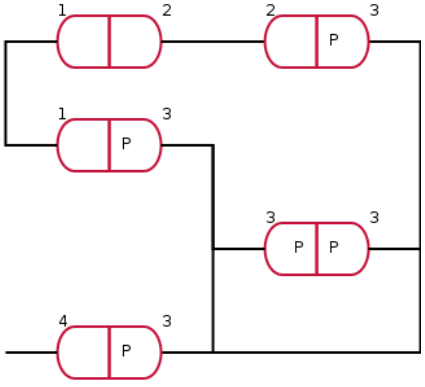
- A collision of waves occurs on a bus. When panic arrives through multiple bridges, panic waves will collide on a bus. Colliding panic waves on a bus will only spread through the remaining bridges.

Below is a visual representation of panic propagating on an example network.

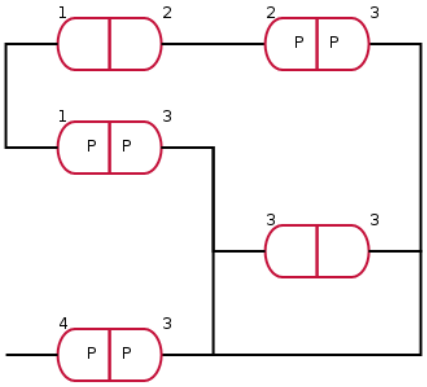




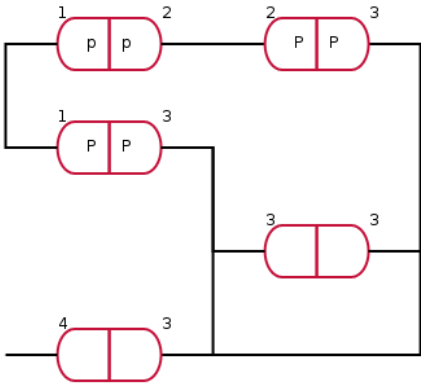
Panic starts on a portal on bus 3



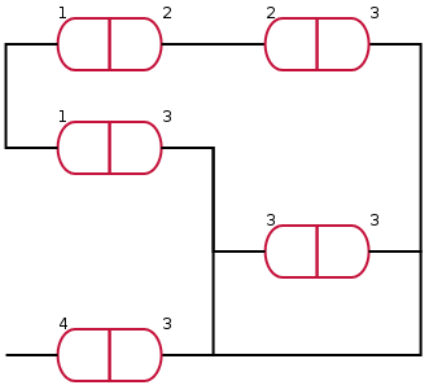
Propagation on bus 3, of panic to all portals



coPortals get notified of panic
a bridge resumes normal operaion if the buses of
both bridge portals have been notified of panic



Propagate further, on buses 1 and 2



Done

Chapter 5

Model of Panic

This chapter describes the mathematical and PVS model of the Panic protocol used in this research project. The model of panic is a state machine. It consists of three main parts: the state, the topology and the actions. The PVS model resembles the mathematical model closely.

The state of panic consists of the values of the registers of all portals plus the topology. The topology models the connections in the network of 1394 buses, bridges and portals. The actions model the transitions between the states in the precondition / effect style. The registers and actions are derived from the description of panic in [Rom04].

5.1 Assumptions and Abstractions

To abstract from unnecessary details, a number of assumptions for the model of the panic protocol are made. Since only the panic protocol is being modeled, as little behavior as possible is included from net update and 1394a. Also certain mechanisms in 1394a are considered to work correctly.

One of the assumptions is that the 1394 reset protocol works correctly. The reset protocol resets a 1394 bus and updates all nodes on that bus with status information. This information is used in some decisions in the panic protocol and is assumed to be available.

1394a nodes without the 1394.1 extension are left out of the model. Normal 1394a nodes do not contribute to spreading panic over the network. Therefore

all nodes in this model are considered portals, hence part of a bridge.

Since panic is a distributed protocol it is possible that actions associated with different portals or the topology can happen concurrently. We abstract from this concurrency by applying the effect of one action atomically to the current state. This is justified because the communication channels between the portals of a bridge are synchronized with a semaphore. Also, 1394 processes are in place for synchronization of the communication on the bus.

For the model of panic we assume that the number of portals in the topology is finite. Also, in the 1394a standard a maximum number of portals on a bus is defined, this value is abstracted from by the *MaxNode* constant. The same holds for the maximum number of buses on a network which is defined in 1394.1 and abstracted from by the *MaxHop* constant.

Also, message types are abstracted in the model. In the 1394a and 1394.1 specification a number of message types are defined. However, there is only one message type that is used in the panic protocol, all others are ignored. This message is referred to as a PANIC message and all other message types are not modeled.

We abstract from the panic by modeling it as a register flag. This flag is called toPanic. ToPanic holds when the panic condition is true. Since this condition is detected in net update the exact detection of the panic condition is outside our scope. Therefore, we assume that toPanic is not set during panic.

5.2 Topology

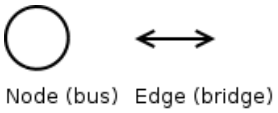
The topology reflects the current state of the network. The network consists of portals. In one network all portals can reach each other. Portals are 1394a nodes with the 1394.1 extension. It is possible for two networks to be joined together during a topology change. In a network the portals are always a member of a bus (though possibly alone) and a bridge with exactly one other portal.

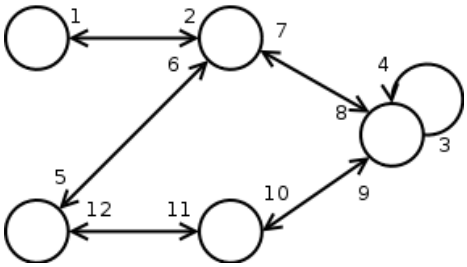
A bus is a collection of portals. Every portal belongs to exactly one bus. The portal can change from one bus to another during a topology change. The buses form a partitioning over the portals in the topology.

Bridges are one device consisting of two different portals. All portals have a coPortal. A coPortal is the other portal of the bridge to which the portal belongs. The 1394.1 standard does not allow the separation of the portals of a bridge. A bridge connects the buses that its portals are a member of. The bridge itself is not part of either bus. However, the 1394.1 standard allows for

both portals to be on the same bus. Below a graphical presentation of a bridge is shown.

The connections of bridges and buses can be visualized in a graph. Each node of the graph is a bus and each edge is a bridge. Since it is allowed for a bridge to have both portals on the same bus, self loops can exist in the graph. Below a graphical presentation of the graph.





This together with the assumptions results in the following relations for this model:

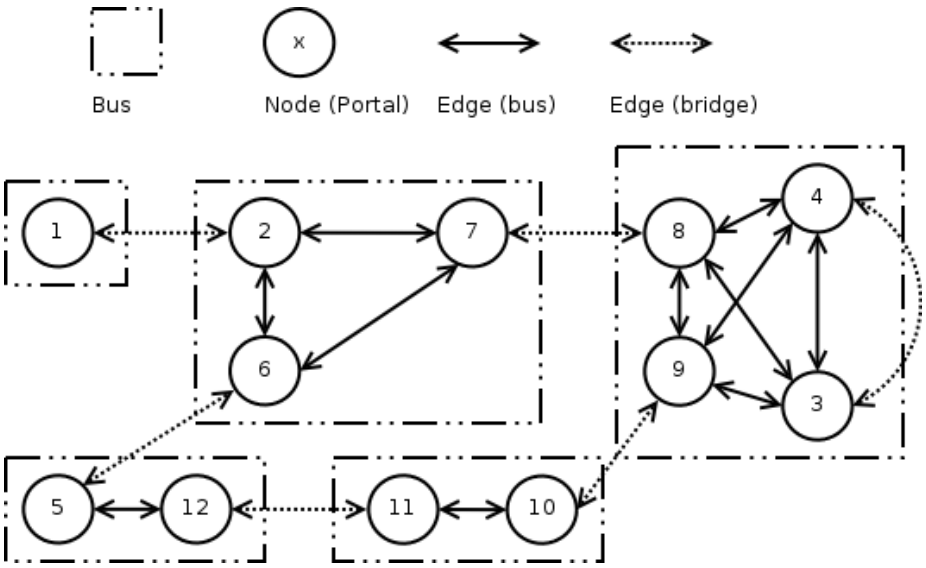
- On a network all portals can reach each other.
- Two networks can be joined together.
- Each portal belongs to exactly one bridge.
- Each portal belongs to exactly one bus.
- Each portal has a coPortal.
- A bridge consists of exactly two distinct portals.
- A bus contains at least one portal and fewer than *MaxNode* portals.
- A bridge links two buses.
- The portals belonging to a bridge do not change.
- Portals belonging to a bus can change.

In PVS the theory for the topology closely resembles the mathematical model. In the PVS model it is also necessary to give the relations explicitly. Therefore the model in PVS is more concrete than the mathematical model.

To model the topology in PVS the graphs theories from the NASA Langley library are used. The topology is modeled as one big graph where the portals are the nodes.

The edges of the graph are modeled by doubleton. A doubleton is a set of exactly two elements. The bridges are a set of edges that are not allowed to change. The buses are strongly connected subgraphs of portals that are on the same bus. The edges of these subgraphs are allowed to change.

To complete the graph both the edgeset of the bridges and the edgesets of the buses are combined. Below is a graphical representation of the graph as used in PVS. Note that the network is the same as in the above picture.



5.3 State

The state of the model consists of the topology, the registers of the portals and some variables that are added to aid in the analysis / proofs. The added variables do not influence the behavior since they are not used in any precondition ("history variables").

Portal registers are a thinned out version of the registers listed in precisely[Rom04]. For each used register an array exists, the arrays are indexed by the identification of the portal. In the same manner the extra variables that help in the analysis / proofs of panic are added.

The important registers and variables are explained below:

- Location is a variable that keeps the current location of the panic protocol in a portal. Actions move a portal from location to location (self loops are allowed). In the model there are five locations. The special location N signifies normal (portal not in panic) operation of a portal. The others, location P_0, P_1, P_3, P_4, are used when the portal is in panic. The absence of location P_2 is explained in Section 5.4
- toPanic is a flag that is set when the panic condition holds for that portal. This flag allows a portal to move from the normal location and effectively start panic.

- adjBusPanicComplete, is a register flag that keeps track of the panic status of the coPortal. This flag is set when the coPortal is done with panic.
- msgCoPortal, is a single place buffer for messages from the coPortal. This single place buffer abstracts from the behavior for sending a message between portals on a bridge. This buffer is often used in the proofs.
- msgBus, is a buffer that receives the messages from the bus. This buffer abstracts from the 1394a behavior that involves the sending of messages over the bus.

For a complete description of all registers and variables see Appendix B.1.

A short overview of all variables without their description:

Topology:

Name	**Type**	**Initial**	**Type**
portals	SetOf(portal)	All available portals	C
topology	Partition(portals)	$\{\{p\} \mid p \in Portals\}$	V
semaphore	ArrayOf boolean	$(\forall p \in portals : \neg semaphore[bridgeId(p)])$	V
busReset	ArrayOf boolean	$(\forall P \in topology : \neg busReset[P])$	V

Portal:

Name	**Type**	**Initial**	**Type**
hopsSincePanic	$\{n \mid 0 \leq n < MaxNode\}$	0	V
adjBusPanicComplete	Boolean	false	V
brdg	{0, 2, 3}	2	V
toPanic	Boolean	false	V
hasPanicked	Boolean	false	H
location	$\{N, P_i \mid 0 \leq i \leq 4\}$	N	V
msgCoPortal	Message	$\emptyset$	V
msgBus	ListOf(Message)	$\emptyset$	V

Type legend:

- C: Constant
- V: Protocol state variable
- H: History variable

In PVS the implementation is almost the same. The TAME template is used, this template defines a record where all variables are grouped together.

5.4 Actions

Actions are used to transform one state to the next state. As a basis the action descriptions from [Rom04] are used. However, there is a difference. Since our scope is limited to panic only, any behavior of net update and 1394 is abstracted.

In the precisely document a location P_2 is used to synchronize the exchange of status information between the portals on the bridge, during the bus reset. This behavior is abstracted by removing the location and integrating the effect of the actions associated with the location into the bus reset action itself.

The synchronization is done using a low level protocol of the original 1394 standard which we assume to work correctly. We prefer to exclude this behavior from our model, since it would make the proof effort unnecessarily more complex.

There are two different types of actions: topology and portal actions. All portal actions are parameterized by a portal. The topology actions are parameterized by one or more buses. A parameterized portal action is formalized in quantifications as: $a(p)$

All actions are modeled in the precondition / effect style. In this manner an action is guarded by the precondition. If the precondition holds in a state then the action may execute. If in a state more than one action is enabled one may be chosen non-deterministically. The effect of an action transforms the current state into the next state.

Below is a short description of all actions that affect only a single portal, followed by a description of the actions that can affect more than one portal. For a more detailed description of the actions see Appendix B.2.

Actions affecting a single portal:

- startPanic. Starts panic on a portal based on the panic condition. The portal switches to location P_0.
- recvPanicCoPortalA. Upon receiving a PANIC message from the coPortal if the portal is in location N, panic is started on this portal. The portal sets the adjBusPanicComplete flag to true. The portal switches from location N to location P_0.
- recvPanicCoPortalB. Upon receiving a PANIC message from the coPortal, when the portal itself is already in panic, the portal sets the adjBusPanicComplete flag to true. The portal does not switch to a new location (self loop).

- recvPanicCoPortalC. Upon receiving a PANIC message from the coPortal, the portal switches from the wait location P_4 back to location N.
- recvPanicBus. Upon receiving a PANIC message from the bus panic is started on this portal. The portal switches from location N to the wait location P_1.
- recvMsgBus. Clears messages received from the bus when te portal is in panic. The portal does not switch to a new location (self loop).
- sendPanicBus. Attempts to notify the bus of panic by a broadcast. The portal sets the busReset flag to true. The portal switches from location P_0 to the wait location P_1.
- sendPanicCoPortal. Conditional action that notifies the coPortal of panic by sending a PANIC message over the bridge, if the adjBusPanicComplete flag is set to false. The portal switches from location P_3 to wait location P_4.
- finish. Conditional action, the portal switches from location P_3 back to location N, if the adjBusPanicComplete flag is set to true and sends a PANIC message over the bridge.

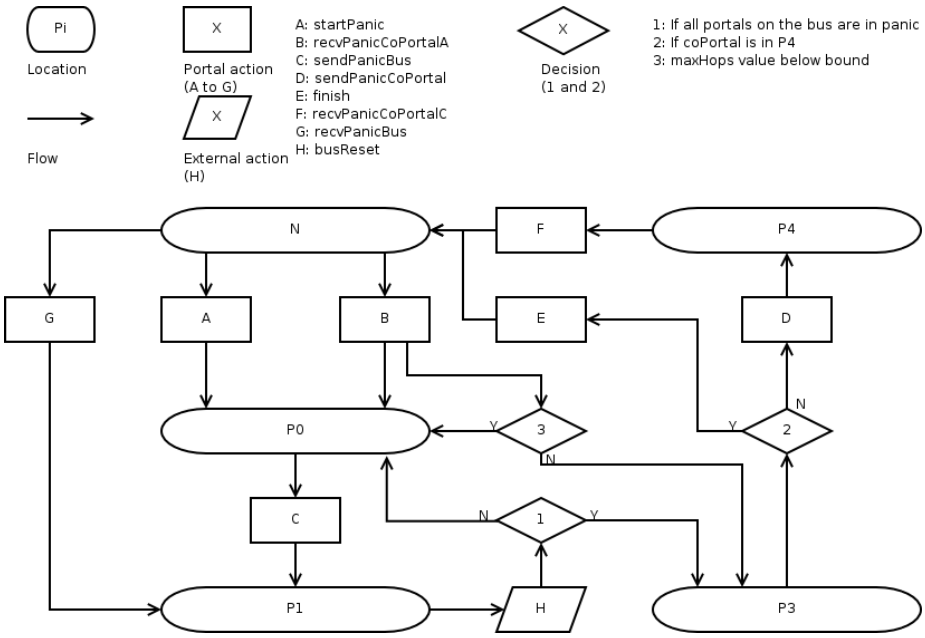
Actions that can affect more than one portal (We consider these as "environment" actions, hence not under control of an individual portal):

- busReset. Action that can happen at any time. It moves all portals on the bus, that are in location P_0 or P_1, either to location P_3 if all portals on the bus are in panic otherwise back to P_0. Furthermore, it clears the msgBus buffer of all portals on the bus. Also, the bus reset flag is cleared.
- mergeBus. A topology change action that can happen at any time. It will merge two buses from the topology into a single new bus. Afterwards it exhibits the same behavior as the busReset for the newly created bus.
- splitBus. A topology change action that can happen at any time. It will split a bus from the topology into two new buses. After which it exhibits the same behavior as the busReset action for both newly created buses.

There are two small differences between these actions and the actions from [Rom04]. The first difference is that the toPanic flag is now cleared immediately upon the execution of startPanic. The second difference is that when a portal finishes panic the adjBusPanicComplete flag is cleared as well. In both cases this does not interfere with the behavior of the panic protocol. The toPanic flag is only used in the precondition of the startPanic itself and is never set again (by the panic protocol). The adjBusPanicComplete flag is not used in actions that start panic on a portal and is normally reset upon starting panic.

These changes are introduced to make certain proofs easier without introducing new history variables. In case of the adjBusPanicComplete flag the invariants are easier to prove, directly or indirectly. In case of the toPanic flag the termination proof is easier to prove.

In PVS the actions are implemented using the TAME template. The TAME template splits up the declaration, precondition and the effect of the actions. Other than the splitup and some syntactic differences the instantiated TAME template is identical to the mathematical model.



5.5 Fairness Restrictions

We define a number of strong fairness sets with respect to a bus P, for which we define a convenient shorthand $X(s, a(p))$

$$X(s, a(p)) = p \in bus \wedge reachable(s) \wedge s \xrightarrow{a(p)} t$$

$$\begin{aligned}
&\text{SFset1}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = sendPanicBus \\
&\qquad \wedge (\exists (q : portals) : \ q \in bus : \ s.location[q] = N \\
&\qquad\quad \wedge PANIC \notin s.msgBus[q] \\
&\qquad\quad \wedge PANIC \in t.msgBus[q])\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFSet2}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = recvPanicBus\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset3}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = startPanic\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset4}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = recvPanicCoPortalA\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset5}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = recvPanicCoPortalB\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset6}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = recvPanicCoPortalC\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset7}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = sendPanicCoPortal\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset8}(P) = \\
&\quad \{s \xrightarrow{a(p)} t \mid X(s, a(p)) \wedge a = finish\}
\end{aligned}$$

$$\begin{aligned}
&\text{SFset9}(P) = \\
&\quad \{s \xrightarrow{a(P)} t \mid P \in s.topology \wedge \xrightarrow{reset(P)} t\}
\end{aligned}$$

Liveness We now show that the combination is indeed a live automaton and its fair executions are indeed live. Observe that the number of strong fairness sets is finite, hence at most countable. Then according to Proposition 1 in [AL94] the combination (A,L) is live.

An interesting detail is that if our automaton is viewed as an I/O automaton [LT89] we are not able to prove liveness and strong fairness at the same time. For an I/O automaton to be live there is an additional requirement. Its strong fairness sets are input resistant [RV96]: Input actions can not disable a strong fairness set.

If our automaton would be an I/O automaton then its environment actions would be reset(P), mergeBus(P,Q) or splitBus(P,Q). However reset(P) can disable SFset2, this is in violation with the added requirement of liveness for an I/O automaton.

We could pull SFset1 and SFset2 together and thus obtain liveness. However since both sets are now combined progress is lost since strong fairness can be obeyed even without executing actions from subset SFSet2. This is exactly the behavior that we wanted to restrict with the fairness sets.

Chapter 6

Correctness Specification

Since there was no earlier explicit correctness specification a new one is constructed. In general this specification consists of three properties concerning: deadlocks, termination and the reach of panic. Each of these properties is explained in Section 6.1.

Creating the correctness specification is complex due to the following two reasons:

1) The dynamic topology changes, that are inherent to the plug and play architecture of 1394. This is explained in Section 6.1.

2) In the specification of 1394.1 panic and net update are intertwined with each other. Also both protocols depend on each other to ensure that the network returns to a usable state. The panic condition is detected from within net update, after which panic activates.

The dynamic topology changes can result in certain parts of the network to be unreachable for the panic protocol. This situation occurs when two networks are joined together via a bridge that is finishing panic. This results in a possible collision as described in Section 4.4.

Therefor we can not guarantee that panic alone will reach the entire network. Thus in turn panic relies on net update to ensure that the other parts of the network enter panic when needed.

Due to these two complications we have weakened versions of earlier proposed properties.

To prove these functional properties we need some invariants. The rest of this chapter contains the descriptions of the properties and their supporting invariants. Lastly, the correlation between the properties and invariants is summarized graphically. Note that the actual verification process and techniques are described in Chapter 7.

6.1 Properties

Stable Topology

The dynamic topology, that is inherit to any 1394 network, allows for the topology to change at any time. Through a topology change it is possible to attain any combination of the register values existing on the buses in the net. This dynamic behavior makes it harder or perhaps impossible to prove the properties. Therefore, all properties that express something about the end result are proven under the assumption of a stable topology.

This limitation is unavoidable. After a change to the topology the net update protocol will be started and in turn, if needed, panic. Also, even though the topology can change dynamically, in real world situations it is not likely that the network will change continuously. It is not even possible to prove anything for a continuously changing topology since they express something about the end result. These properties are now proven after the last topology change has occurred.

We consider a topology stable over a sequence of actions σ, if σ does not contain mergeBus or splitBus actions. This can be formalized as:

$$stableTopology(\sigma) =$$
$$(\forall(a : action) : \ a \in \sigma : \ a \notin \{mergeBus, splitBus\})$$

Property 1: Outreach

The first property is a reachability property. Panic attempts to reset the entire network in such a way that net update can rebuild the routing. However, in the presence of topology changes this can obviously not be guaranteed. With this limitation we propose two alternatives:

Alternative 1. If there are no portals already in panic when it started, in a stable topology the entire network is reached by panic. Formalized as:

$$
\begin{array}{l}
(\forall(s,t:\ states,\ \sigma):\\
\quad(\forall(p:\ portals):\ p \in s.topology \wedge s.location[p] = N \wedge s \overset{\sigma}{\rightarrow}{}^{*}\ t \wedge t.location[p] = N\\
\qquad\wedge\ reachable(s) \wedge stableTopology(\sigma)\\
\qquad\quad\wedge\ (\exists(q:\ portals):\ q \in net(s,p):\ s.toPanic[q] \wedge t.hasPanicked[q])):\\
\quad(\forall(p:\ portals):\ t.hasPanicked[p]))
\end{array}
$$

Alternative 2. From the last topology change, panic reaches some well defined part of the network. Formalized as:

$$
\begin{array}{l}
(\forall(s,t:\ state,\ \sigma,q:portals):\\
\quad s \overset{\sigma}{\rightarrow}{}^{*}\ t \wedge reachable(s) \wedge stableTopology(\sigma):\\
\qquad(s.toPanic[q] \vee s.location[q] \in \{P_0,P_1\}) \wedge t.hasPanicked[q]:\\
\qquad\quad(\forall(p:\ portals):\ p \in subnet(s,q):\ t.location[p] = N)\\
\qquad\Rightarrow(\forall(p:\ portals):\ p \in subnet(s,q):\ t.hasPanicked[p]))
\end{array}
$$

$$
\begin{array}{l}
subnet(s:state,q:portals) =\\
\qquad\{p \in portals \mid s.hasPanicked[p] \vee s.location[p] \neq N\}\\
\quad\bigcup\\
\qquad(\bigcup\{p \in portals \mid panic\_path(s,p,q)\})
\end{array}
$$

$$
\begin{array}{l}
panic\_path(s:state,\ p,q:portals) =\\
\quad(\exists(p_0,\ldots,p_n) ::\ p_0 \cdots p_n \in paths(s,p,q))\\
\qquad\wedge\ (\forall(i:\mathbb{N}):\ 0 \leq i \leq n:\ s.location[p_i] \notin \{P_3,P_4\}
\end{array}
$$

$$
\begin{array}{l}
paths(s:state,\ p,q:portals) =\\
\quad\{p_0 \cdots p_n \mid p = p_0, \wedge q = p_n\\
\qquad\wedge\ (\forall(i:\mathbb{N}):\ 0 \leq i < n:\ p_i = coPortal(p_{i+1}) \vee p_i \in bus(s,p_{i+1}))\}
\end{array}
$$

(*hasPanicked*[p] is a history variable that is false in the initial state and is set to true by p when it finishes panic. We use this history variable to check if a portal indeed has panicked. σ is a sequence of actions.)

Property 2: No Deadlock

The second property is a no deadlock property. This property states that there are no reachable deadlock states. A deadlock state is a state that only allows topology actions and is not a final state of the protocol. Since this property is not an end result, it does not assume a stable topology.

A final state is a state where all portals are in location N, normal operation, and there is no portal left that has its toPanic flag set. This is the final state because from this state there is no ongoing panic and there is no portal left that can start panic. This can be formalized as:

$$\begin{array}{l}\text{finalState}(p,\ s) = \\ \quad (\forall (q : portals) :\ q \in net(s,p) :\ s.location[q] = N \wedge \neg s.toPanic[q])\end{array}$$

It must be shown that all non-final states must enable at least one action that is not a topology change action. Formally:

$$\begin{array}{l}(\forall (q : portals, s : state) : reachable(s) : \\ \quad (\exists (p : portals) :\ p \in net(s,q) :\ s.location[p] \neq N \vee s.toPanic[p]) \\ \qquad \Rightarrow (\exists (p : portal, a : actions) : p \in net(s,q) \wedge a \notin \{mergeBus, splitBus\} : \\ \qquad\quad enabled(s, a(p)) \vee busReset[bus(s,p)]))\end{array}$$

Property 3: Termination

The third property is the termination property. It ensures that panic will eventually finish. This property embodies one of the main reasons to add the panic protocol to net update. The panic protocol is added to solve a termination problem in the net update protocol.

To formalize the termination property a norm function is used. A norm function relates each state to a maximum number of protocol steps left. The norm function itself is a (ordered) tuple of components. An explanation of the termination argument based on the norm function is in Section 7.5

Below are all five components, with a short intuitive description.

The first component counts the number of portals that can still initiate panic.

Component 1:

$$\begin{array}{l}\text{comp1(s: state)} = \\ \quad (\#(q :\ portals) ::\ s.toPanic[q])\end{array}$$

The second component is a little more complex. This component gives an upper bound to the number of portals that can still be visited by panic until panic stops. The component itself is split into two parts. The first part counts the number of potential portals that can be visited when panic is communicated over a bridge. The second part does the same for when a PANIC message is communicated over a bus.

Component 2:

comp2(s: state) =
  comp2_half(s, $\{q \in \mathit{portals}\}$)
+
  comp2_pass(s, $\{q \in \mathit{portals}\}$)

comp2_half(s: state, P: setof[portals]): $\mathbb{N}$ =
  $(\sum(q : \mathit{portals}) : \ q \in P \wedge \mathit{half}(s, q) :$
    $(\mathit{MaxNode} - 1) * \mathit{nrPanics}(s.\mathit{hopsSincePanic}[q] + 1))$

comp2_pass(s: state, P: setof[portals]): $\mathbb{N}$ =
  $(\sum(q : \mathit{portals}) : \ q \in P \wedge \mathit{busInPanic}(s, q) \wedge \mathit{pass}(s, q)) :$
    $\mathit{nrPanics}((\downarrow (r \in \mathit{bus}(s, q)) : r \in \{P_0, P_1\} : s.\mathit{hopsSincePanic}[r])) - 1)$

(Note: An empty domain of the inner $\downarrow$ quantor poses no problem because in this case the domain the $\sum$ quantor is empty as well)

The definitions of half, pass and nrPanics are in the Appendix C.3, Property 3.

The third component counts the maximum number of steps needed for the portals to reach location N from a panic location.

Component 3:

comp3(s: state) =
  $(\sum(q : \mathit{portals}) :: \ \mathit{distance}(s.\mathit{location}[q]))$

The fourth component counts the number of portals that still need to detect that its coPortal has finished panic.

Component 4:

comp4(s: state) =
  $(\#(q : \mathit{portals}) : \ s.\mathit{location}[q] \neq N : \ \neg s.\mathit{adjBusPanicComplete}[q])$

The fifth and last component simply counts the number of messages in all msgBus buffers. This last component was added specifically to ensure that there is a decrease in the norm for the recvMsgBus action.

Component 5:

comp5(s: state) =
$(\#(q : portals) :: s.msgBus[p] \neq \emptyset)$

6.2 Invariants

Invariant 1

Due to the semaphore, at most one message is in transit between the portal and coPortal at any given time. The semaphore is claimed by one portal upon sending a message and released upon receiving a message by the other portal.

This is reflected by invariant 1.1 and invariant 1.2. These invariants show a connection between the semaphores and the msgCoPortal buffers. Many invariants use invariant 1 to ascertain the status of the msgCoPortal buffers from the semaphore or vice versa.

Invariant 1.1
If the semaphore of portal p is claimed then there is exactly one message in either msgCoPortal[p] or msgCoPortal[coPortal(p)].

$(\forall(s : state, p : portals) :: s.semaphore[bridgeId(p)]$
$\Rightarrow (s.msgCoPortal[p] \neq \emptyset \ XOR \ s.msgCoPortal[coPortal(p)] \neq \emptyset))$

(Note: The XOR operation yields true if and only if exactly one of the conditions hold.)

Invariant 1.2
If the semaphore of portal p is not claimed then both msgCoPortal[p] and msgCoPortal[coPortal(p)] must be empty.

$(\forall(s : states, p : portals) : \neg s.semaphore[bridgeId(p)]$
$\Rightarrow s.msgCoPortal[p] = \emptyset \wedge s.msgCoPortal[coPortal(p)] = \emptyset)$

Invariant 2

If the portal is in location N then its adjBusPanicComplete flag is not set. Useful helper invariant for the status on adjBusPanicComplete.

$$(\forall(s : states, p : portals) :: s.location[p] = N \Rightarrow \neg s.adjBusPanicComplete[p])$$

Invariant 3

If the portal is in location P_4 then its adjBusPanicComplete flag is not set. Useful helper invariant for the status on adjBusPanicComplete.

$$(\forall(s : states, p : portals) :: s.location[p] = P_4 \Rightarrow \neg s.adjBusPanicComplete[p])$$

Invariant 4

If portal p is in location P_4 and has received a PANIC message from its coPortal then the coPortal of p does not have its adjBusPanicComplete flag set.

$$(\forall(s : states, p : portals) :: s.location[p] = P_4 \wedge s.msgCoPortal[p] = PANIC \\ \Rightarrow \neg s.adjBusPanicComplete[coPortal(p)])$$

Invariant 5

Invariant 5 is originally created as a sub invariant of invariant 6. The invariant is split up into two parts around a conjunction.

A basic translation is: If portal p is not aware that coPortal(p) is in panic, then coPortal(p) is either not in location P_4 or it is finishing panic.

$$(\forall(s : states, p : portals) :: \\ (s.msgCoPortal[p] = \emptyset \wedge \neg s.adjBusPanicComplete[p] \\ \Rightarrow s.location[coPortal(p)] \neq P_4 \vee s.msgCoPortal[coPortal(p)] = PANIC) \\ \wedge (s.location[coPortal(p)] \neq P_4 \vee s.location[p] \neq P_4))$$

Invariant 6

Invariant 6 was the original starting point for the verification of the panic protocol. This invariant embodies one of the core mechanisms of the panic protocol. Therefore the completed proof of this invariant gives an adequate degree of confidence to continue the verification effort. It shows the correct signaling of the state of panic on both ends of a bridge. The correct signaling of the adjBusPanicComplete flag is vital for the correct propagation of panic through a bridge.

Invariant 6.1

Basic translation in two parts: 1) A PANIC message will only be sent once (in each direction) over the bridge in each panic cycle. 2) When a panic message is waiting to be received exactly one portal on the bridge will be in location P_4.

$$
\begin{aligned}
&(\forall(s :\ states, p :\ portals) ::\\
&\quad (s.msgCoPortal(p) = PANIC \Rightarrow (\neg s.adjBusPanicComplete(p)\\
&\qquad \wedge (s.location[coPortal(p)] \neq P_4 \Leftrightarrow s.location[p] = P_4)))\\
&\quad \wedge (\neg s.adjBusPanicComplete[coPortal(p))] \vee \neg s.adjBusPanicComplete[p]))
\end{aligned}
$$

Invariant 6.2
When the adjBusPanicCompleteflag of portal p is set, the coPortal of p must be in location P_4.

$$
\begin{aligned}
&(\forall(s :\ states, p :\ portals) ::\ s.adjBusPanicComplete[p]\\
&\quad \Rightarrow s.location[coPortal(p)] = P_4)
\end{aligned}
$$

Invariant 7

Invariant 7 was created during the construction of the proof of the termination property. It gives information about the msgCoPortal buffer when one portal of the bridge is in location N and the other in location P_4.

This situation can happen in two different cases. The first is when the portal in location N still needs to start panic. The second is when the portal in location P_4 is finishing panic.

When this situation occurs at least one end of the bridge has a PANIC message in its msgCoPortalbuffer.

$$(\forall(s : states, p : portals) :: (s.location[p] = N \wedge s.location[coPortal(p)] = P4) \\ \Rightarrow (s.msgCoPortal[p] = PANIC \vee s.msgCoPortal[coPortal(p)] = PANIC))$$

Invariant 8

Another invariant created during the construction of the proof of the termination property. It is somewhat similar to invariant 7. When both portals on the bridge are operating normally, no panic message is contained in either msgCoPortal buffer.

$$(\forall(s : states, p : portals) :: (s.location[p] = N \wedge s.location[coPortal(p)] = N) \\ \Rightarrow (s.msgCoPortal[p] = \emptyset \wedge s.msgCoPortal[coPortal(p)] = \emptyset))$$

Invariant 9

Invariant 9 shows that when there is a portal on the bus with a PANIC message in its *msgBus* then there must be a portal on the bus that is in location P_1. In other words if a PANIC message can be read from the *msgBus* buffer then there is a portal on the bus that has broadcast it.

$$(\forall(s : states, p : portals) :: PANIC \in s.msgBus[p] \\ \Rightarrow (\exists q : portals) : q \in bus(s, p) : s.location[q] = P_1))$$

Invariant 10

Invariant 10 shows the relation between the panic locations and the brdg field. This relation is needed to verify the correct working of the reset action.

$$(\forall(s : states, p : portals) :: \\ s.location[p] \neq N \Leftrightarrow s.brdg[p] = 0)$$

Invariant 11

Invariant 11 was created during the proof of the termination property. It states that an upper bound exists for the hopsSincePanic registers of the portals.

PANIC messages are transmitted over the bus with the current hopsSincePanic register value of the sending portal and since these values are later copied into the hopsSincePanic register of each receiving portal a second conjunct is needed. This second conjunct requires the same upper bound for the transmitted hopsSincePanic values.

$$
\begin{array}{l}
(\forall(s:\ states, p:\ portals) ::\ s.location[p] \in \{P_0, P_1\} \\
\quad \Rightarrow s.hopsSincePanic[p] < MaxHop) \\
\wedge \\
(\forall(s : states, p:\ portals) ::\ s.location[p] \in \{P_0, P_1\} \\
\quad (\forall m(h):\ m(h) \in s.msgBus[p] \wedge m = PANIC:\ h < MaxHop))
\end{array}
$$

Invariant 12

Invariant 12 is used in the proof of the "no deadlock" property. Since there are no actions related to the portal to leave location P_1 it is required that there is some other action enabled which does advance panic.

Therefore it is shown when a portal is in location P_1 the busReset flag is set. If the busReset flag is set the reset action will be enabled for this bus and the portals on the bus that are in location P_1 can advance in panic through the reset action.

Note that this mechanism is an abstraction in our model, see Section 5.4

$$
\begin{array}{l}
(\forall(s : states, p:\ portals) ::\ s.location[p] = P_1 \\
\quad \Rightarrow s.busReset(bus(s, p))
\end{array}
$$

Invariant 13

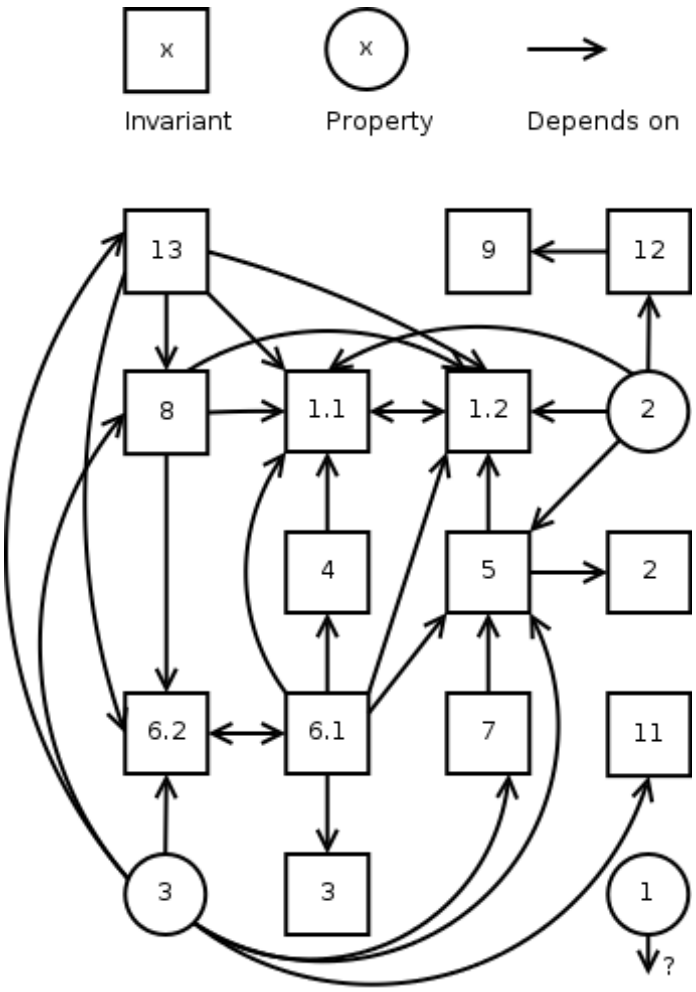
Invariant 13 states that if a portal is still in location N and its coPortal is in panic but not in location P_4, then the msgCoPortal of the coPortal must be empty.

This means that even though the coPortal started panic, anew or not, it will have an empty msgCoPortal buffer. Since, the portal did not send a new PANIC message and the coPortal will have cleared any PANIC message from an earlier panic cycle.

$$
\begin{array}{l}
(\forall(s:\ states, p:\ portals) ::\ s.location[p] = N \wedge s.location[coPortal(p)] \in \{P_0, P_1, P_3\} \\
\quad \Rightarrow s.msgCoPortal[coPortal(p)] = \emptyset)
\end{array}
$$

6.3 Correlation of Properties and Invariants

Below is a graphical representation of the correlation between the properties and invariants.



Chapter 7

Verification

This chapter describes the proof strategies used during the proving of the correctness specification. First the general strategy is explained followed by a more detailed explanation of the invariants and each of the properties. At the end of the chapter an overview of the proof status is included.

7.1 Strategy

All proofs are done both by hand and with the aid of the theorem prover PVS. With this setup it is possible to cross reference the work done with both proof techniques. A number of mistakes have been eliminated in both the hand proofs and PVS.

All proofs done by hand follow a generic proof format. This proof format is derived from some of the subjects taught at the TU/e. Most proofs need to ensure that some statement is valid under certain conditions. This can be translated to: conditions $\Rightarrow$ statement. The direct proofs are thus started by assuming the conditions and what is known from the environment and then deriving the statement from those assumptions.

7.2 Invariants

Invariants are proven by induction. Firstly, it is proven that the invariant holds in the start state s_0. Then for each reachable state s it is proven that the effect

of the actions maintain the invariant. Recall that a reachable state is a state that can be reached from the initial state s_0. Formally: s is reachable if $s_0 \xrightarrow{\sigma}{}^* s$ where σ is some sequence of actions leading to state s. This approach leads to the following proof structure:

Base: Prove invariant holds in s_0
Step: Assume, $s \xrightarrow{a} t$ and reachable(s). Assume for action a the invariant holds at the start of the action in s. Prove that the invariant is still valid after the effect of the action, in t.

This proof structure follows the format as described in the previous section. The domain of the invariant is assumed, under which the range of the invariant is proven. To complete the proof the precondition of the current action, lemmas and other invariants are used.

Translating the hand proofs to PVS with TAME is straightforward. TAME uses an almost identical way to prove the invariants as the hand proofs. The major translation effort is to get the specification into proper PVS syntax.

After this translation is completed, the invariants are proven with the theorem prover. With the aid of the TAME strategies the proofs were completed without much difficulty.

The detailed hand proofs of all invariants can be found in Appendix C.2.

7.3 Property 1: Outreach

Both the hand proof and the PVS proof are still pending.

7.4 Property 2: No Deadlock

All non-final states must enable an action which is not a topology change action. This is done by a straightforward proof without induction.

The first step was to swap the terms around the implication to get the negation of both. This was done to allow reasoning with $\forall$ quantifications which are easier to work with than their $\exists$ counterpart.

Then following the general proof format the conjunction of negated preconditions is assumed. After this the negated final state is derived under that assumption.

Unfortunately, the verification in PVS is still pending.

The detailed proof done by hand of the no deadlock property can be found in Appendix C.3, Property 2.

7.5 Property 3: Termination

The proof of the termination property starts from the state following the last topology change.

The norm function relates each state to the number of possible steps left. Hence, for panic to terminate it must be shown that the norm decreases for each action. Then if the lexicographic ordering of the norm function is well-founded, panic terminates.

Recall that the lexicographic ordering relates two tuples as follows:
$\langle n_1, \ldots, n_5 \rangle < \langle m_1, \ldots, m_5 \rangle$ iff $n_j = m_j, \wedge n_i < m_i (\exists i \in \{1, \ldots, 5\} \wedge \forall j \in \{1, \ldots, i-1\})$.

We obtain well-foundedness of the lexicographic ordering on the norm since each component gets a value from $\mathbb{N}$, and for each tuple $\langle n_1, \ldots, n_5 \rangle$ of $n_i \in \mathbb{N}$ there are only finitely many smaller tuples.

In the table below is a graphical overview that displays how the norm function decreases for each action except the Reset(P) action.

	comp1	comp2	comp3	comp4	comp5
startPanic	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$=$
recvPanicCoPortalA	$=$	$\downarrow$	$\uparrow$	$\downarrow$	$=$
recvPanicCoPortalB	$=$	$=$	$=$	$\downarrow$	$=$
recvPanicCoPortalC	$=$	$=$	$\downarrow$	$=$	$=$
recvPanicBus	$=$	$\downarrow$	$\uparrow$	$\uparrow$	$=$
recvMsgBus	$=$	$=$	$=$	$=$	$\downarrow$
sendPanicCoPortal	$=$	$=$	$\downarrow$	$=$	$=$
finish	$=$	$=$	$\downarrow$	$\uparrow$	$=$
sendPanicBus	$=$	$=$	$\downarrow$	$=$	$\uparrow$
Reset	$=$	$=$	$\downarrow \backslash \uparrow$	$=$	$\downarrow$

Table legend:

- $=$, the component remains constant for the action.
- $\uparrow$, the component increases for the action.
- $\downarrow$, the component decreases for the action.

Since the choice in the reset(P) action depends on an earlier broadcast. This broadcast is unreliable and can fail to reach a portal that still needs to receive it, hence it is not always certain if the value of comp3 increases or decreases. To solve this, a strong fairness assumption is made. For more information see Section 5.5 and for the proof see Section 7.5.1.

Unfortunately the verification in PVS is still pending.

The detailed hand proof of the termination property can be found in Appendix C.3, Property 3.

7.5.1 Fairness

As is noted above a strong fairness assumption is needed to complete the termination proof. This assumption is needed to deal with the cases where the reset(P) action does not decrease the value of comp3.

There are two actions that can prevent panic from reaching a final state. The first is sendPanicBus which fails infinitely many times to reach a portal. This will enable SFset1 infinitely many times, but transitions from SFset1 are not

taken infinitely many times. This is not allowed by strong fairness and thus leads to a contradiction.

The second is the reset action that occurs every other action. When we try to progress to the final state it is possible that the reset action will interrupt the recvPanicBus action infinitely many times. This will enable SFset2 infinitely many times, but transitions from SFset2 are not taken infinitely many times. This is not allowed by strong fairness and thus leads to a contradiction.

7.6 Proof Overview

In the table below an overview of which proofs are proven by hand and in PVS is given.

Proof	**Hand**	**PVS**
Invariant 1	√	√
Invariant 2	√	√
Invariant 3	√	√
Invariant 4	√	√
Invariant 5	√	√
Invariant 6	√	√
Invariant 7	√	√
Invariant 8	√	√
Invariant 9	√	–
Invariant 10	√	√
Invariant 11	√	–
Invariant 12	√	√
Invariant 13	√	√
Property 1	–	–
Property 2	√	–
Property 3	√	–

7.7 Constraints Arising in the Proofs

An interesting constraint that arises from the proofs is a minimum value of 2 for the *MaxNode*. This constraint is introduced because comp2 of property 3 uses this constant to define a bound. This constant has the value of 63 in the original IEEE 1394 specification and hence poses no problem with respect to the proofs.

Chapter 8

Achievements

This chapter summarizes what is achieved in the course of this research project.

First we start with the work done and deliverables. We have defined a model for the panic protocol. This model proved to be sufficient for proving the invariants and properties.

The proofs done by hand include a number of invariants, the no deadlock property and the termination property. This leaves the correctness specification proven for about two thirds.

Resulting from the hand proofs a research paper about the termination proof could be created.

In PVS a specification for panic and a model for the topology with supporting lemmas are created. With the specification and TAME all invariants are proven in PVS. Also, during the work with PVS a better understanding of the workings of PVS is gained.

As a small bonus one uncompleted theory from of the NASA Langley library was proven and submitted back. This now completed theory will be activated in the NASA Langley library upon its next release.

The added value of using theorem provers like PVS on complex distributed protocols is shown for this research project. The use of PVS allows the proofs to be made for every possible configuration of the network instead of a fixed configuration as was done earlier with model checking.

Chapter 9

Final Words

Lastly, I want to include a small chapter about my personal experiences during this project. Also I would like to thank a few people that aided me in the course of this project.

First of all, one of the things that took some time to get used to was the tool PVS. PVS is a very nice tool all in all, it has a good typing system that allows for about anything including dependent typing. However, PVS has a rather high learning curve. TAME did make it easier to start on the invariants but to make somewhat more interesting proofs proper knowledge of the PVS theorem prover is required.

Even though the high learning curve made it hard to start using PVS, having worked with it is a great experience. Especially a Q.E.D at the end of a hard proof gives a rewarding feeling. At the end of this project I think I can call myself a intermediate PVS user.

Another thing that took a fairly long time to complete was the proof of the termination property. This effort was mainly put in constructing the second component of the norm function and proving it decreased on the two actions that where needed to get the norm function to decrease for every action.

I like this branch of the computer science. Even though it was running proofs on an already existing standard a lot of creativity was needed to formulate the invariants and properties. In turn proving these properties and invariants, of a complex system as a distributed protocol such as panic, is very challenging and fun.

It must be said that the final result listed in this report does not contain all the

brainstorming, notes and other proof attempts that were undertaken to come to this result. The entire process of thinking of new possibilities to complete a part of the correctness specification and discussing them with other people is a very large part that gives rise to a deeper understanding of the problem space one is working in.

Unfortunately, even though best efforts where made to get all properties of the correctness specification proven, this was an unreachable goal. This leaves some room for future work on the panic protocol.

As simple future work the proof of property 1 could be constructed and verified in PVS. As well as verifying the results of the hand proofs of properties 2 and 3 in PVS.

A more interesting idea would be to extend the current work to include the net update protocol as well. I would estimate this extension to be roughly the same amount of work as was already undertaken for the panic protocol, but it will certainly give rise to new challenges.

Finally a special thanks goes to the following people:
- J.M.T. Romijn, for guiding me through my master project.
- The committee for being kind enough to evaluate the project.
- My girlfriend for moral support and proofreading.

Bibliography

[13904] The Institute of Electrical And Electronics Engineers, Inc. *IEEE Standard for High Performance Serial Bus Bridges*, December 2004. IEEE Std 1394.1-2004.

[AL94] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[Huo05] X. Huo. A concurrent net panic protocol for dynamic IEEE 1394.1 networks. Master's thesis, Technische Universiteit Eindhoven, 2005.

[IEE96] IEEE Computer Society. *IEEE Standard for a High Performance Serial Bus*. The Institute of Electrical And Electronics Engineers, Inc., August 1996. IEEE Std 1394-1995.

[IEE01] IEEE Computer Society. *IEEE Standard for a High Performance Serial Bus – Amendment 1*. The Institute of Electrical And Electronics Engineers, Inc., December 2001. IEEE Std 1394a-2000.

[ISO89] ISO. Information processing systems – Open Systems Interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour. ISO/IEC 8807, 1989.

[Jon94] B. Jonsson. Compositional specification and verification of distributed systems. *ACM Transactions on Programming Languages and Systems*, 16(2):259–303, March 1994.

[Lam94] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[Rom04] J.M.T. Romijn. IEEE 1394.1 net update precisely, 2004. Manuscript. Available on `http://www.win.tue.nl/~jromijn/papers.html`.

[RV96] J.M.T Romijn and F.W. Vaandrager. A note on fairness in I/O automata. *Information Processing Letters*, 59(5):245–250, September 1996.

[vLRG03] I.A. van Langevelde, J.M.T. Romijn, and N. Goga. Founding firewire bridges through promela prototyping. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS), 8th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA)*. IEEE Computer Society Press, 2003.

[Vor04] S. Vorstenbosch. A global reset protocol for a dynamic IEEE 1394.1 network. Internship report, Technische Universiteit Eindhoven, 2004.

Appendix A

How to Read the Proofs

This appendix will explain the proof format used for the hand proofs.

The proof format is based on a course at the TU/e, "Logica en Verzamelingen leer". This class is taught in the first year of the computer science bachelor education. The format is chosen for its clear structure.

The structure itself consists of a numbered list of proof steps. Each proof step has an annotation preceding it in curly braces. The annotation contains the argumentation that resulted in the proof step. This rational can contain: references to preceding proof steps, lemmas and invariants and / or hints. A special hint is the "assume". All proof steps that are made under this assume hint are indented. The numbering itself does not follow any strict rules, but as a rule of thumb after an assume hint the number gets a sub index.

A seeming exception to this format is the proof of property 3. Instead of boolean formulas there are equalities and implications in between the proof steps, giving it a more mathematical appearance.

To keep the proofs manageable they are usually split first over the actions in a large case distinction. The actions are grouped together when the proof of the actions are the same.

Appendix B

Specification of the Panic Model

B.1 State

B.1.1 Topology

The topology consists of:

coPortal(p):
Function coPortal::= portal → portal.
Return the coportal of this portal.
Where $(\forall p \in portals : \ coPortal(p) \neq p \wedge coPortal(coPortal(p)) = p)$

bridgeId(p):
Function bridgeId ::= portal → Nat.
Return the unique id associated with the bridge where portal p is a part of.
Where $(\forall p, q \in portals : \ (p = coPortal(q) \Leftrightarrow bridgeId(p) = bridgeId(q))$

portals
Description: The set of all portals. This set is constant.
Type: SetOf(portal).
Value: available portals.

topology
Description: A partition over the portals.

Type: Partition(portals)
Where $(\forall P, Q \in topology : \ P \cap Q = \emptyset)$
and $\{P | P \in topology) = portals$
Initial value: $\{\{p\} \mid p \in Portals\}$

busReset
Description: A variable to indicate that a bus reset is wanted on a bus. The busReset is modelled as a virtual array indexed by the partitions of the topology.
Type: Boolean.
Initial value: $(\forall P \in topology : \ busReset[P] = true)$

semaphore
Description: A variable to indicate the state of the bridge semaphore. The semaphore is modelled as a virtual array indexed by the bridge function.
Type: Boolean.
Initial value: $(\forall p \in portals : \ semaphore[bridgeId(p)] = false)$

B.1.2 Portal

The registers of a portal important for panic are:

hopsSincePanic
Description: The total number of portals passed since start of panic.
Type: $\{n \mid 0 \leq n < MAX\_BUS\_ID\}$
Initial value: $(\forall p \in portals : \ hopsSincePanic[p] = 0)$

adjBusPanicComplete
Description: Flag that is set to true when the coPortal of this portal is done with panic.
Type: Bool.
Initial value: $(\forall p \in portals : \ adjBusPanicComplete[p] = false)$

brdg
Description: The bridge field of the portal. 0 signifies "not a bridge portal, 2 signifies "unchanged topology" and 3 signifies "changed topology".
Type: {0, 2, 3}
Initial value: $(\forall p \in portals : \ brdg[p] \in \{2, 3\})$

toPanic
Description: Flag set somewhere during normal operation of the portal indicating that this portal should start panic.
Type: Boolean
Initial value: $(\forall p \in portals : \ toPanic[p] = false)$

location
Description: The location indictes which phase / stage of the protocol is being executed. N is used to model the normal operation of the portal. P_i is used to model a location in the panic protocol.
Type: $\{N\} \cup \{P_i \mid 0 \leq i \leq 4\}$
Initial value: $(\forall p \in portals : \ location[p] = N)$

msgCoPortal
Description: Single place buffer to hold messages sent by the coPortal of this portal. Since messages send between the two portals of a bridge are synchronised with a semaphore a single place buffer is sufficient.
Type: Message.
Initial value: $(\forall p \in portals : \ msgCoPortal[p] = \emptyset)$

msgBus
Description: Buffer that holds messages received from over the bus that this portal is on.
Type: ListOf(Message).
Initial value: $(\forall p \in portals : \ msgBus[p] = \emptyset)$

History variable:

hasPanicked
Description: Flag to indicate that this portal has finished panic at least once since the initial state.
Type: Boolean.
Initial value: $(\forall p \in portals : \ hasPanicked[p] = false)$

B.1.3 Overview

A short overview of all variables without their description:

Topology:

Name	Type	Initial	Type
portals	SetOf(portal)	All available portals	C
topology	Partition(portals)	$\{\{p\} \mid p \in Portals\}$	V
semaphore	ArrayOf boolean	$(\forall p \in portals : \neg semaphore[bridgeId(p)])$	V
busReset	ArrayOf boolean	$(\forall P \in topology : \neg busReset[P])$	V

Portal:

Name	Type	Initial	Type
hopsSincePanic	$\{n \mid 0 \leq n < MaxNode\}$	0	V
adjBusPanicComplete	Boolean	false	V
brdg	{0, 2, 3}	{2, 3}	V
toPanic	Boolean	false	V
hasPanicked	Boolean	false	H
location	$\{N, P_i \mid 0 \leq i \leq 4\}$	N	V
msgCoPortal	Message	$\emptyset$	V
msgBus	ListOf(Message)	$\emptyset$	V

Type legend:

- C: Constant
- V: Protocol state variable
- H: History variable

B.2 Actions

B.2.1 Helper Functions

Helper functions used by actions:

bus(s, p):
function bus ::= state × portal → SetOf(portal)
  return $\{P \mid P \in s.topology \wedge p \in P\}$

broadcast(s, P, msg):
function broadcast ::= state × SetOf(portal) × MESSAGE → state
  s with $(\forall p \in P : \ s.msgBus[p] = s.msgBus[p] \ ++ \ [msg])$

syncBusPanicComplete(s, P, bool):
function syncBusPanicComplete ::= state × SetOf(portal) × boolean → state
s with $(\forall (p \in P) : \ (s.location[p] \in \{P_0, P_1, P_2\}) :$
    $((\exists q \in P : \ s.brdg[q] \in \{2, 3\})?s.location[p] := P_0 : s.location[p] := P_3)$
        $\wedge \ s.msgBus[p] := \emptyset)$

B.2.2 Actions

Portal actions:

startPanic(p)
Parameters: portal: p
Source: N
Destination: P_0
Precondition:
    toPanic[p] ∧
    location[p] = N
Effect:
    hopsSincePanic[p] := 0;
    adjBusPanicComplete[p] := false;
    brdg[p] := 0;
    location[p] := P_0;
    toPanic[p] := false;

recvPanicCoPortalA(p)
Parameters: portal: p
Source: N
Destination: P_0, P_3
Precondition:
msgCoPortal[p] = PANIC(h) $\wedge$
location[p] = N
Effect:
hopsSincePanic[p] := h + 1;
adjBusPanicComplete[p]:= true;
brdg[p] := 0;
msgCoPortal[p] := $\emptyset$;
semaphore[*bridgeId*(p)] := false;
if (h < *MaxNode*- 1)
location[p] := P_0
else
location[p] := P_3
fi

recvPanicCoPortalB(p)
Parameters: portal: p
Source: P_0, P_1, P_2, P_3
Destination: self loop
Precondition:
location[p] $\in \{P_0, P_1, P_2, P_3\} \wedge$
msgCoPortal[p] $\neq \emptyset$
Effect:
adjBusPanicComplete[p] := true;
msgCoPortal[p] := $\emptyset$;
semaphore[*bridgeId*(p)] := false;

recvPanicCoPortalC(p)
Parameters: portal: p
Source: P_4
Destination: N
Precondition:
msgCoPortal[p] = PANIC(h) $\wedge$
location[p] = P_4
Effect:
msgCoPortal[p] := $\emptyset$
semaphore[*bridgeId*(p)] := false;
toPanic[p] := false;
hasPanicked[p] := true;
adjBusPanicComplete[p] := false;

location[p] := N;
brdg[p] := 3;

recvPanicBus(p)
Parameters: portal: p
Source: N
Destination: P_1
Precondition:
PANIC(h) $\in$ msgBus[p] $\wedge$ location[p] = N
Effect:
hopsSincePanic[p] := h;
adjBusPanicComplete[p] := false;
brdg[p] := 0;
location[p] := P_1;
msgBus[p] := $\emptyset$

recvMsgBus(p)
Parameters: portal: p
Source: P_0, P_1, P_2, P_3, P_4
Destination: self loop
Precondition:
location[p] $\neq N \wedge$ msgBus[p] $\neq \emptyset$
Effect:
msgBus[p] := $\emptyset$

sendPanicBus(p, P)
Parameters: portal: p, SetOf(portal): P
Source: P_0
Destination: P_1
Precondition:
P $\subseteq$ (*bus*(p) - {p}) $\wedge$
location[p] = P_0
Effect:
broadcast(P, PANIC(hopsSincePanic[p]));
busReset[*bus*(p)] := true;
location[p] := P_1;

sendPanicCoPortal(p)
Parameters: portal: p
Source: P_3
Destination: P_4

Precondition:
¬semaphore[*bridgeId*(p)] ∧
¬adjBusPanicComplete[p] ∧
location[p] = P_3
Effect:
semaphore[*bridgeId*(p)] := true;
msgCoPortal[*coPortal*(p)] := PANIC(h);
location[p] := P_4;

finish(p)
Parameters: portal: p
Source: P_3
Destination: N
Precondition:
¬semaphore[*bridgeId*(p)] ∧
adjBusPanicComplete[p] ∧
location[p] = P_3
Effect:
semaphore[*bridgeId*(p)] := true;
msgCoPortal[*coPortal*(p)] := PANIC(h);
toPanic[p] := false;
hasPanicked[p] := true;
adjBusPanicComplete[p] := false;
location[p] := N;
brdg[p] := 3;

Reset action:

reset(P)
Parameters: P ∈ topology
Source:
Destination:
Precondition:
Effect:
syncBusPanicComplete(P);
busReset[P] := false;

Topology actions:

mergeBus(P, Q)
Parameters: P ∈ topology, Q ∈ topology
Source:

Destination:
Precondition:
$P \neq Q$
Effect:
topology := (topology - {P} - {Q}) $\cup$ {P $\cup$ Q};
syncBusPanicComplete(P $\cup$ Q);
busReset[P $\cup$ Q] := false;

splitBus(P, Q)
Parameters: P $\in$ topology, Q $\in$ topology
Source:
Destination:
Precondition:
(P $\cup$ Q) $\in$ topology $\wedge$
$P \cap Q = \emptyset \wedge$
$P \neq \emptyset \wedge$
$Q \neq \emptyset$
Effect:
topology := (topology - {P $\cup$ Q}) $\cup$ {P} $\cup$ {Q};
syncBusPanicComplete(P $\cup$ Q);
busReset[P] := false;
busReset[Q] := false;

Appendix C

Detailed Proofs

C.1 Lemmas

Lemma 1

$(\forall(p, q : portal) : q \in bus(p) \Rightarrow bus(p) = bus(q))$

C.2 Invariants

Invariant 1

Checked in PVS.

Invariant 1.1

$(\forall(s : \ state, p : \ portals) :: \ s.semaphore[bridgeId(p)]$
$\quad \Rightarrow (s.msgCoPortal[p] \neq \emptyset \ XOR \ s.msgCoPortal[coPortal(p)] \neq \emptyset))$

(Note: The XOR operation yields true if and only if exactly one of the conditions hold.)

Invariant 1.2

$(\forall(s:\ states, p:\ portals):\ \neg s.semaphore[bridgeId(p)]$
$\quad \Rightarrow s.msgCoPortal[p] = \emptyset \wedge s.msgCoPortal[coPortal(p)] = \emptyset)$

Initial state

{by: initial value}
1) $\neg s_0.semaphore[bridgeId(p)]$
{by: initial value}
2) $s_0.msgCoPortal[p] = \emptyset$
{by: initial value}
3) $s_0.msgCoPortal[coPortal(p)] = \emptyset$
{by: 1}
4) $s_0.Inv1.1$ for p and $coPortal$(p)
{by: 2, 3}
5) $s_0.Inv1.2$ for p and $coPortal$(p)

Actions

Prove:
$(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv1.1 \wedge s.Inv1.2:\ s'.Inv1.1 \wedge s'.Inv1.2)$

case $a \in \{startPanic(p), recvPanicBus(p), sendPanicBus(p),$
$recvMsgBus(p), reset(P), mergeBus(P,q), splitBus(P,Q)\}$
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.inv1.1 \wedge s.inv1.2$
{by: 1}
2) $s.Inv1.1 \wedge s.Inv1.2$
{by: 2, unchanged msgCoPortal[p],
unchanged msgCoPortal[$coPortal$(p)], unchanged semaphore[$bridgeID$(p)]}
3) $s'.Inv1.1 \wedge s'.Inv1.2$ for p and $coPortal$(p)
{by: 1, 3}
4) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv1.1 \wedge s.Inv1.2:\ s'.Inv1.1 \wedge s'.Inv1.2)$

case $a \in \{recvPanicCoPortalA(p), recvPanicCoPortalB(p),$
$recvPanicCoPortalC(p)\}$
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.inv1.1 \wedge s.inv1.2$
{by: 1}
2) $s.Inv1.1$

{by: precondition(a)}
3) $s.msgCoPortal(p) = PANIC$
{by: 2, 3}
4) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: effect(a)}
5) $s'.msgCoPortal[p] = \emptyset$
{by: 4, effect(a)}
6) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: effect(a)}
7) $\neg s'.semaphore[bridgeId(p)]$
{by: 7}
8) $s'.Inv1.1$ for p and $coPortal$(p)
{by: 5, 6}
9) $s'.Inv1.2$ for p and $coPortal$(p)
{by: 1, 8, 9}
10) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv1.1 \wedge s.Inv1.2 : \ s'.Inv1.1 \wedge s'.Inv1.2)$

case $a \in \{sendPanicCoPortal(p), finish(p)\}$
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.inv1.1 \wedge s.inv1.2$
{by: 1}
2) $s.Inv1.2$
{by: precondition(a)}
3) $\neg semaphore[p]$
{by: 2, 3}
4) $s.msgCoPortal[p] = \emptyset$
{by: 4, effect(a)}
5) $s'.msgCoPortal[p] = \emptyset$
{by: effect(a)}
6) $s'.msgCoPortal[coPortal(p)] \neq \emptyset$
{by: effect(a)}
7) $s'.semaphore[bridgeId(p)]$
{by: 5, 6}
8) $Inv1.1$ for p and $coPortal$(p)
{by: 7}
9) $Inv1.2$ for p and $coPortal$(p)
{by: 1, 8, 9}
10) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv1.1 \wedge s.Inv1.2 : \ s'.Inv1.1 \wedge s'.Inv1.2)$

Invariant 2

Checked in PVS.

$(\forall (s : \ states, p : \ portals) :: \ s.location[p] = N \Rightarrow \neg s.adjBusPanicComplete[p])$

Initial state

{by: initial value}
1) $\neg s_0.adjBusPanicComplete[p]$
{by: 1}
2) $s_0.Inv2$ for p and $coPortal$(p)

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2 : \ s'.Inv2)$

case $a \in \{startPanic(p), recvPanicCoPortalA(p), recvPanicCoPortalB(p), recvPanicBus(p), recvMsgBus(p), sendPanicBus(p, P), sendPanicCoPortal(p)\}$
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2$
{by: effect(a)}
2) $s'.location[p] \neq N$
{by: 2}
3) $s'.Inv2$ for p and $coPortal$(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2 : \ s'.Inv2)$

case $a \in \{recvPanicCoPortalC(p), finish(p)\}$
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2$
{by: effect(a)}
2) $\neg s'.adjBusPanicComplete[p]$
{by: 2}
3) $s'.Inv2$ for p and $coPortal$(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2 : \ s'.Inv2)$

case $a \in \{reset(P), mergeBus(P, q), splitBus(P, Q)\}$
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2$
{by: 1}
2) $s.Inv2$
{assume}
3) *case* $s.location[p] = N$

{by: 2, 3}
3.1) $\neg s.adjBusPanicComplete[p]$
{by: 3.1, effect(a)}
3.2) $\neg s'.adjBusPanicComplete[p]$
{by: 3.2}
3.3) $s'.Inv2$ for p and $coPortal$(p)
{by: 3, 3.3}
4) $s.location[p] = N \Rightarrow s'.Inv2$ for p and $coPortal$(p)
{assume}
5) *case* $s.location[p] \neq N$
{by: 5, effect(a)}
5.1) $s'.location[p] \neq N$
{by: 5.1}
5.2) $s'.Inv2$ for p and $coPortal$(p)
{by: 5, 5.2}
6) $s.location[p] \neq N \Rightarrow s'.Inv2$ for p and $coPortal$(p)
{by: 4, 6}
7) $s'.Inv2$ for p and $coPortal$(p)
{by: 1, 7}
8) $(\forall s, s', a :: reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv2 :: s'.Inv2)$

Invariant 3

$(\forall (s :: states, p :: portals) :: s.location[p] = P_4 \Rightarrow \neg s.adjBusPanicComplete[p])$

Initial state

{by: initial value}
1) $\neg s_0.adjBusPanicComplete[p]$
{by: 1}
2) $s_0.Inv3$ for p and $coPortal$(p)

Actions

Prove:
$(\forall s, s', a :: reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3 :: s'.Inv3)$

case $a \in \{startPanic(p), recvPanicCoPortalA(p), recvPanicCoPortalB(p),$
$recvPanicCoPortalC(p), recvPanicBus(p), sendPanicBus(p, P)\}$

{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3$
   {by: effect(a)}
   2) $s'.location[p] \neq P_4$
      {by: 2}
   3) $s'.Inv3$ for p and $coPortal$(p)
   {by: 1, 3}
4) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3:\ s'.Inv3)$

case a = recvMsgBus(p)
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3$
   {by: 1}
   2) $s.Inv3$
      {by: 2, unchanged s.location[p], unchanged s.adjBusPanicComplete[p]}
   3) $s'.Inv3$ for p and $coPortal$(p)
   {by: 1, 3}
3) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3:\ s'.Inv3)$

case a = sendPanicCoPortal(p)
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3$
   {by: precondition(a)}
   2) $\neg s.adjBusPanicComplete[p]$
      {by: 2, effect(a)}
   3) $\neg s'.adjBusPanicComplete[p]$
   {by: 1, 3}
4) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3:\ s'.Inv3)$

case $a \in \{reset(P), mergeBus(P, q), splitBus(P, Q)\}$
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3$
   {by: 1}
   2) $s.Inv3$
      {assume}
      3) *case* $s.location[p] = P_4$
         {by: 2, 3}
         3.1) $\neg s.adjBusPanicComplete[p]$
         {by: 3.1, effect(a)}
         3.2) $\neg s'.adjBusPanicComplete[p]$
         {by: 3.2}
         3.3) $s'.Inv3$ for p and $coPortal$(p)
         {by: 3, 3.3}
      4) $s.location[p] = P_4 \Rightarrow s'.Inv3$ for p and $coPortal$(p)
      {assume}

5) *case* $s.location[p] \neq P_4$
   {by: 5, effect(a)}
   5.1) $s'.location[p] \neq P_4$
   {by: 5.1}
   5.2) $s'.Inv3$ for p and $coPortal$(p)
   {by: 5, 5.2}
6) $s.location[p] \neq P_4 \Rightarrow s'.Inv3$ for p and $coPortal$(p)
   {by: 4, 6}
7) $s'.Inv3$ for p and $coPortal$(p)
{by: 1, 7}
8) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv3 : \ s'.Inv3)$

Invariant 4

$(\forall(s : \ states, p : \ portals) :: \ s.location[p] = P_4 \wedge s.msgCoPortal[p] = PANIC$
$\Rightarrow \neg s.adjBusPanicComplete[coPortal(p)])$

Initial state

{by: initial value}
1) $\neg s_0.adjBusPanicComplete[coPortal(p)]$
{by: initial value}
2) $\neg s_0.adjBusPanicComplete[p]$
{by: 1}
3) $s_0.Inv4$ for p
{by: 2}
3) $s_0.Inv4$ for $coPortal$(p)

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 : \ s'.Inv4)$

case $a \in$ {startPanic(p), recvMsgBus(p), finish(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
   {by: effect(a)}
   2) $s'.location[p] \neq P_4$
   {by: effect(a)}
   3) $\neg s'.adjBusPanicComplete[p]$

{by: 2}
4) $s'.Inv4$ for p
{by: 3}
5) $s'.Inv4$ for $coPortal$(p)
{by: 1, 4, 5}
6) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 : \ s'.Inv4)$

case $a \in$ {recvPanicCoPortalA(p), recvPanicCoPortalB(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
{by: effect(a)}
2) $s'.location[p] \neq P_4$
{by: pre(a)}
3) $s.msgCoPortal[p] = PANIC$
{by: 3, Inv1.1}
4) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: 4, effect(a)}
5) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 2}
6) $s'.Inv4$ for p
{by: 5}
7) $s'.Inv4$ for $coPortal$(p)
{by: 1, 6, 7}
8) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 : \ s'.Inv4)$

case $a =$ recvPanicCoPortalC(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
{by: effect(a)}
2) $s'.location[p] \neq P_4$
{by: 1}
3) $s.Inv4$
{by: precondition(a)}
4) $s.msgCoPortal[p] = PANIC$
{by: precondition(a)}
5) $s.location[p] = P_4$}
{by: 3, 4, 5}
6) $\neg s.adjBusPanicComplete[coPortal(p)]$
{by: 6, effect(a)}
7) $\neg s'.adjBusPanicComplete[coPortal(p)]$
{by: 2}
8) $s'.Inv4$ for p
{by: 7}
9) $s'.Inv4$ for $coPortal$(p)
{by: 1, 8, 9}

10) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 :\ s'.Inv4)$

case a = recvMsgBus(p)
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
{by: 1}
2) $s.Inv4$
{by: 2, unchanged location[p], unchanged adjBusPanicComplete[p]}
3) $s'.Inv4$ for p and $coPortal$(p)
{by: 1, 3}
3) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 :\ s'.Inv4)$

case a = sendPanicBus(p, P)
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
{by: effect(a)}
2) $s'.location[p] \neq P_4$
{by: 1}
3) $s.Inv4$
{by: 2}
4) $s'.Inv4$ for p
{3, unchanged adjBusPanicComplete[p]}
5) $s'.Inv4$ for $coPortal$(p)
{by: 1, 4, 5}
6) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 :\ s'.Inv4)$

case $a \in$ {reset(P), mergeBus(P, q), splitBus(P, Q)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4$
{by: 1}
2) $s.Inv4$
{assume}
3) *case* $s.location[p] = P_4$
{by: 2, 3, unchanged msgCoPortal[p],
unchanged adjBusPanicComplete[coPortal(p)]}
3.1) $s'.Inv4$ for p
{by: 3, 3.1}
4) $s'location[p] = P_4 \Rightarrow s'.Inv4$ for p
{assume}
5) *case* $s.location[p] \neq P_4$
{by: 5, effect(a)}
5.1) $s'.location[p] \neq P_4$
{by: 5.1}
5.2) $s'.Inv4$ for p
{by: 5, 5.2}

6) $s.location[p] \neq P_4 \Rightarrow s'.Inv3$ for p
{by: 4, 6}
7) $s'.Inv4$ for p
{by: 2, unchanged location[coPortal(p)] = P_4, unchanged msgCoPortal[coPortal(p)], unchanged adjBusPanicComplete[p]}
8) $s'.Inv4$ for $coPortal$(p)
{by: 1, 7, 8}
9) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv4 : \ s'.Inv4)$

Invariant 5

$(\forall(s : \ states, p : \ portals) ::$
$(s.msgCoPortal[p] = \emptyset \wedge \neg s.adjBusPanicComplete[p]$
$\Rightarrow s.location[coPortal(p)] \neq P_4 \vee s.msgCoPortal[coPortal(p)] = PANIC)$
$\wedge (s.location[coPortal(p)] \neq P_4 \vee s.location[p] \neq P_4))$

Initial state

{by: initial value}
1) $\neg s_0.location[coPortal(p)]$
{by: initial value}
2) $\neg s_0.location[p]$
{by: 1}
3) $s_0.Inv5$ for p
{by: 2}
3) $s_0.Inv5$ for $coPortal$(p)

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5 : \ s'.Inv5)$

case $a \in$ {startPanic(p), recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: 1}
2) $s.Inv5$
{by: effect(a)}
3) $s.location[p] = N$

{by: 3, Inv2}
4) $s.\neg adjBusPanicComplete[p]$
{by: effect(a)}
5) $s'.\neg adjBusPanicComplete[p]$
{by: effect(a)}
6) $s'.location[p] \neq P_4$
{by: 2, 4, 5, 6, unchanged msgCoPortal[p], unchanged location[coPortal(p)], unchanged msgCoPortal[coPortal(p)]}
7) $s'.Inv5$ for p
{by: 6}
8) $s'.Inv5$ for $coPortal$(p)
{by: 1, 7, 8}
9) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5 : \ s'.Inv5)$

case a = recvPanicCoPortalA(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: effect(a)}
2) $s'.adjBusPanicComplete[p]$
{by: effect(a)}
3) $s'.location[p] \neq P_4$
{by: 2, 3}
4) $s'.Inv5$ for p
{by: 3}
5) $s'.Inv5$ for $coPortal$(p)
{by: 1, 4, 5}
6) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5 : \ s'.Inv5)$

case a = recvPanicCoPortalB(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: effect(a)}
2) $s'.adjBusPanicComplete[p]$
{by: precondition(a)}
3) $s.location[p] \neq P_4$
{by: 3, effect(a)}
4) $s'.location[p] \neq P_4$
{by: 2, 4}
5) $s'.Inv5$ for p
{by: 4}
6) $s'.Inv5$ for $coPortal$(p)
{by: 1, 5, 6}
7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5 : \ s'.Inv5)$

case a = recvPanicCoPortalC(p)

{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: 1}
2) $s.Inv5$
{by: precondition(a)}
3) $s.location[p] = P_4$
{by: 2, 3}
4) $s.location[coPortal(p)] \neq P_4$
{by: effect(a)}
5) $s'.location[p] \neq P_4$
{by: 4, effect(a)}
6) $s'.location[coPortal(p)] \neq P_4$
{by: 6}
7) $s'.Inv5$ for p
{by: 5}
8) $s'.Inv5$ for *coPortal*(p)
{by: 1, 7, 8}
9) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5:\ s'.Inv5)$

case a = recvMsgBus(p)
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: 1}
2) $s.Inv5$
{by: 2, unchanged location[p], unchanged location[coPortal(p)],
unchanged adjBusPanicComplete[p], unchanged adjBusPanicComplete[coPortal(p)],
unchanged msgCoPortal[p], unchanged msgCoPortal[coPortal(p)]}
3) $s'.Inv5$ for p and *coPortal*(p)
{by: 1, 3}
3) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Invr54:\ s'.Invr54)$

case $a \in$ {sendPanicBus(p, P), reset(P), mergeBus(P, q), splitBus(P, Q)}
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
{by: 1}
2) $s.Inv5$
{by: 2, unchanged location[p] $\neq P_4$, unchanged location[coPortal(p)] $\neq P_4$,
unchanged adjBusPanicComplete[p], unchanged adjBusPanicComplete[coPortal(p)],
unchanged msgCoPortal[p], unchanged msgCoPortal[coPortal(p)]}
3) $s'.Inv5$ for p and *coPortal*(p)
{by: 1, 3}
3) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5:\ s'.Inv5)$

case a = sendPanicCoPortal(p)
{assume}

1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
   {by: precondition(a)}
   2) $\neg s.semaphore[p]$
   {by: 2, Inv1.2}
   3) $s.msgCoPortal[p] = \emptyset$
   {by: 2, Inv1.2}
   4) $s.msgCoPortal[coPortal(p)] = \emptyset$
   {by: precondition(a)}
   5) $\neg s.adjBusPanicComplete[p]$
   {by: 1}
   6) $s.Inv5$
   {by: 3, 4, 5, 6}
   7) $s.location[coPortal(p)] \neq P_4$
   {by: effect(a)}
   8) $s'.msgCoPortal[coPortal(p)] = PANIC$
   {by: 7, effect(a)}
   9) $s'.location[coPortal(p)] \neq P_4$
   {by: 9}
   10) $s'.Inv5$ for p
   {by: 8, 9}
   11) $s'.Inv5$ for $coPortal$(p)
   {by: 1, 10, 11}
12) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5:\ s'.Inv5)$

case a = finish
   {assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5$
   {by: effect(a)}
   2) $s'.location[p] \neq P_4$
   {by: effect(a)}
   3) $s'.msgCoPortal[coPortal(p)]$
   {by: 3}
   4) $s'.Inv5$ for p
   {by: 2}
   5) $s'.Inv5$ for $coPortal$(p)
   {by: 1, 4, 5}
6) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv5:\ s'.Inv5)$

Invariant 6

Invariant 6.1

$(\forall (s:\ states, p:\ portals) ::$

$(s.msgCoPortal(p) = PANIC \Rightarrow (\neg s.adjBusPanicComplete(p)$
$\quad \wedge (s.location[coPortal(p)] \neq P_4 \Leftrightarrow s.location[p] = P_4)))$
$\wedge (\neg s.adjBusPanicComplete[coPortal(p))] \vee \neg s.adjBusPanicComplete[p]))$

Invariant 6.2

$(\forall(s :\ states, p :\ portals) ::\ s.adjBusPanicComplete[p]$
$\quad \Rightarrow s.location[coPortal(p)] = P_4)$

Initial state

{by: initial value}
1) $\neg s_0.msgCoPortal[p]$
{by: initial value}
2) $\neg s_0.msgCoPortal[coPortal(p)]$
{by: initial value}
3) $\neg s_0.location[p]$
{by: initial value}
4) $\neg s_0.adjBusPanicComplete[p]$
{by: initial value}
5) $\neg s_0.adjBusPanicComplete[coPortal(p)]$
{by: 1, 3}
6) $s_0.Inv6.1$ for p
{by: 2, 3}
7) $s_0.Inv6.1$ for $coPortal$(p)
{by: 4}
8) $s_0.Inv6.2$ for p
{by: 5}
9) $s_0.Inv6.2$ for $coPortal$(p)

Actions

Prove:
$(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 :\ s'.Inv6.1 \wedge s'.Inv6.2)$

case $a \in$ {startPanic(p), recvPanicBus(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1$

{by: precondition(a)}
3) $\neg s.adjBusPanicComplete[p]$
{by: effect(a)}
4) $\neg s'.adjBusPanicComplete[p]$
{by: effect(a)}
5) $s'.location[p] \neq P_4$
{assume}
6) *case* $s.msgCoPortal[p] = PANIC$
{by: 6, Inv1.1}
6.1) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: 2, 3, 6, 6.1}
6.2) $s.location[coPortal(p)] = P_4$
{by: 6.1, effect(a)}
6.3) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 6.2, effect(a)}
6.4) $s'.location[coPortal(p)] = P_4$
{by: 4, 5, 6.4}
6.5) $s'.Inv$6.1 for p
{by: 4, 6.3}
6.6) $s'.Inv$6.1 for *coPortal*(p)
{by: 6, 6.5, 6.6}
7) $s.msgCoPortal[p] = PANIC \Rightarrow s'.Inv$6.1 for p and *coPortal*(p)
{assume}
8) *case* $s.msgCoPortal[p] = \emptyset$
{by: 8, effect(a)}
8.2) $s'.msgCoPortal[p] = \emptyset$
{by: 4, 8}
8.3) $s'.Inv$6.1 for p
{by: 2, unchanged msgCoPortal[coPortal(p)],
unchanged adjBusPanicComplete[coPortal(p)],
unchanged location[coPortal(p)]}
8.4) $s'.Inv$6.1 for *coPortal*(p)
{by: 8, 8.3, 8.4}
9) $s'msgCoPortal[p] = \emptyset \Rightarrow s'.Inv$6.1 for p and *coPortal*(p)
{by: 7, 9}
10) $s.Inv$6.1 for p and *coPortal*(p)
{by: 4}
11) $s'.Inv$6.2 for p
{by: 1}
12) $s.Inv$6.2
{by: precondition(a)}
13) $s.loc[p] \neq P_4$
{by: 12, 13}
14) $\neg s.adjBusPanicComplete[coPortal(p)]$
{by: 14, effect(a)}
15) $\neg s'.adjBusPanicComplete[coPortal(p)]$

{by: 15}
16) $s'.Inv6.2$ for $coPortal$(p)
{by: 1, 10, 11, 16}
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case $a \in$ {recvPanicCoPortalA(p), recvPanicCoPortalB(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1$
{by: precondition(a)}
3) $s.msgCoPortal[p] = PANIC$
{by: 3, Inv1.1}
4) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: precondition(a)}
5) $s.location[p] \neq P_4$
{by: 2, 3, 5}
6) $s.location[coPortal(p)] = P_4$
{by: 6, Inv3}
7) $\neg s.adjBusPanicComplete[coPortal(p)]$
{effect(a)}
8) $s'.msgCoPortal[p] = \emptyset$
{4, effect(a)}
9) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{7, effect(a)}
10) $\neg s'.adjBusPanicComplete[coPortal(p)]$
{by: 8, 10}
11) $s'.Inv6.1$ for p
{by: 9, 10}
12) $s'.Inv6.1$ for $coPortal$(p)
{by: 6, effect(a)}
13) $s'.location[coPortal(p)] = P_4$
{by: 13}
14) $s'.Inv6.2$ for p
{by: 10}
15) $s'.Inv6.2$ for $coPortal$(p)
{by: 1, 11, 12, 14, 15}
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case $a =$ recvPanicCoPortalC(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: precondition(a)}
2) $s.msgCoPortal[p] = PANIC$
{by: precondition(a)}

3) $s.location[p] = P_4$
{by: 2, 3, Inv4}
4) $\neg s.adjBusPanicComplete[coPortal(p)]$
{by: 4, effect(a)}
5) $\neg s'.adjBusPanicComplete[coPortal(p)]$
{by: effect(a)}
6) $\neg s'.adjBusPanicComplete[p]$
{by: 2, Inv1.1}
7) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: effect(a)}
8) $s'.msgCoPortal[p] = \emptyset$
{by: 7, effect(a)}
9) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 6, 8}
10) $s'.Inv6.1$ for p
{by: 5, 9}
11) $s'.Inv6.1$ for $coPortal$(p)
{by: 6}
12) $s'.Inv6.2$ for p
{by: 5}
13) $s'.Inv6.2$ for $coPortal$(p)
{by: 1, 10, 11, 12, 13}

14) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case a = recvMsgBus(p)
{assume}

1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1 \wedge s.Inv6.2$
{by: 2, unchanged location[p], unchanged location[coPortal(p)],
unchanged adjBusPanicComplete[p], unchanged adjBusPanicComplete[coPortal(p)],
unchanged msgCoPortal[p], unchanged msgCoPortal[coPortal(p)]}
3) $s'.Inv6.1 \wedge s'.Inv6.2$ for p and $coPortal$(p)
{by: 1, 3}

3) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case a = sendPanicCoPortal(p)
{assume}

1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1$
{by: 1}
3) $s.Inv6.2$
{by: precondition(a)}
4) $\neg s.semaphore[bridgeId(p)]$

{by: 4, Inv1.1}
5) $s.msgCoPortal[p] = \emptyset$
{by: 4, Inv1.2}
6) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: precondition(a)}
7) $\neg s.adjBusPanicComplete[p]$
{by: 5, 6, 7, Inv5}
8) $s.location[coPortal(p)] \neq P_4$
{by: precondition(a)}
9) $s.location[p] \neq P_4$
{by: 3, 9}
10) $\neg s.adjBusPanicComplete[coPortal(p)]$
{by: 5, effect(a)}
11) $s'.msgCoPortal[p] = \emptyset$
{by: 7, effect(a)}
12) $\neg s'.adjBusPanicComplete[p]$
{by: 10, effect(a)}
13) $\neg s'.adjBusPanicComplete[coPortal(p)]$
{by: effect(a)}
14) $s'.location[p] = P_4$
{by: 8, effect(a)}
15) $s'.location[coPortal(p)] \neq P_4$
{by: 11, 12}
16) $s'.Inv6.1$ for p
{by: 13, 14, 15}
17) $s'.Inv6.1$ for $coPortal$(p)
{by: 12}
18) $s'.Inv6.2$ for p
{by: 13}
19) $s'.Inv6.2$ for $coPortal$(p)
{by: 1, 16, 17, 18, 19}
20) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case a = finish(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1$
{by: 1}
3) $s.Inv6.2$
{by: precondition(a)}
4) $s.adjBusPanicComplete[p]$
{by: 2, 4}
5) $\neg s.adjBusPanicComplete[coPortal(p)]$
{by: 3, 5}

6) $s.location[coPortal(p)] = P_4$
{by: precondition(a)}
7) $\neg s.semaphore[p]$
{by: 7, Inv1.1}
8) $s.msgCoPortal[p] = \emptyset$
{by: effect(a)}
9) $\neg s'.adjBusPanicComplete[p]$
{by: 5, effect(a)}
10) $\neg s'.adjBusPanicComplete[coPortal(p)]$
{by: effect(a)}
11) $s'.location[p] \neq P_4$
{by: 6, effect(a)}
12) $s'.location[coPortal(p)] = P_4$
{by: 8, effect(a)}
13) $s'.msgCoPortal[p] = \emptyset$
{by: 9, 13}
14) $s'.Inv6.1$ for p
{by: 10, 11, 12}
15) $s'.Inv6.1$ for $coPortal$(p)
{by: 9}
16) $s'.Inv6.2$ for p
{by: 10}
17) $s'.Inv6.3$ for $coPortal$(p)
{by: 1, 14, 15, 16, 17}

18) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

case $a \in$ {sendPanicBus(p, P), reset(P), mergeBus(P, Q), splitBus(P, Q)}
{assume}

1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2$
{by: 1}
2) $s.Inv6.1 \wedge s.Inv6.2$
{by: 2, unchanged location[p] $\neq P_4$, unchanged location[coPortal(p)] $\neq P_4$, unchanged adjBusPanicComplete[p], unchanged adjBusPanicComplete[coPortal(p)], unchanged msgCoPortal[p], unchanged msgCoPortal[coPortal(p)]}
3) $s'.Inv6.1 \wedge s'.Inv6.2$ for p and $coPortal$(p)
{by: 1, 3}

3) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv6.1 \wedge s.Inv6.2 : \ s'.Inv6.1 \wedge s'.Inv6.2)$

Invariant 7

$(\forall (s : states, p : \ portals) :: \ (s.location[p] = N \wedge s.location[coPortal(p)] = P4)$
$\Rightarrow (s.msgCoPortal[p] = PANIC \vee s.msgCoPortal[coPortal(p)] = PANIC))$

Initial state

{by: Initial value}
1) $s_0.location[coPortal(p)] = N$
{by: Initial value}
2) $s_0.location[p] = N$
{by: 1}
3) $s_0.Inv7$ for p
{by: 2}
4) $s_0.Inv7$ for coPortal(p)

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {startPanic(p), recvPanicCoPortalA(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: effect(a)}
2) $s'.location[p] = P_0$
{by: 2}
3) $s'.inv7$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {recvPanicCoPortalB(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: precondition(a)}
2) $s.location[p] \in \{P_0, P_1, P_3\}$
{by: unchanged s.location[p]}
3) $s'.location[p] \in \{P_0, P_1, P_3\}$
{by: 3}
4) $s'.inv7$ for p and coPortal(p)
{1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {recvPanicCoPortalC(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: precondition(a)}
2) $s.location[p] = P_4$

{by: 2, Inv5}
3) $s.location[coPortal(p)] \neq P_4$
{by: effect(a)}
4) $s'.location[p] = N$
{by: unchanged location[coPortal(p)]}
5) $s'.location[coPortal(p)] \neq P_4$
{by: 5}
6) $Inv7$ for p
{by: 4}
7) $Inv7$ for coPortal(p)
{by: 1, 5, 6}
8) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {recvPanicBus(p), sendPanicBus(p, P)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: effect(a)}
2) $s'.location[p] = P_1$
{by: 2}
3) $s'.Inv7$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a =$ recvMsgBus(p)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: precondition(a)}
2) $s.location[p] \in \{P_0, P_1, P_3, P_4\}$
{by: 2, unchanged location[p]}
3) $s'.location[p] \in \{P_0, P_1, P_3, P_4\}$
{assume}
4) *case* $s'.location[p] \neq P_4$
{by: 3, 4}
4.1) $Inv7$ for p and coPortal(p)
{by: 4, 4.1}
5) $s'.location[p] \neq P_4 \Rightarrow s'.Inv7$
for p and coPortal(p)
{assume}
6) *case* $s'.location[p] = P_4$
{by: 6, Inv5}
6.1) $s'.location[coPortal(p)] \neq P_4$
{by: 6.1}
6.2) $s'.Inv7$ for p and coPortal(p)
{by: 6, 6.2}
7) $s'.location[p] = P_4 \Rightarrow s'.Inv7$

for p and coPortal(p)
{by: 5, 7}
8) $s'.Inv7$ for p and coPortal(p)
{by: 1, 8}
9) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {sendPanicCoPortal(p), finish(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{by: effect(a)}
2) $s.msgCoPortal[coPortal(p)] = PANIC$
{by: 2}
3) $s'.Inv7$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

case $a \in$ {reset(P), mergeBus(P, Q), splitBus(P, Q)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7$
{assume}
2) *case* $s.location[p] \in \{P_0, P_1, P3\}$
{by: 1}
2.1) $s.Inv7$
{by: 2, 2.1, unchanged location[p] $\in \{P_0, P_1, P_3\}$}
2.2) $s'.Inv7$ for p and coPortal(p)
{by: 2, 2.2}
3) $s.location[p] \in \{p_0, P_1, P_3\} \Rightarrow s'.Inv7$ for p and coPortal(p)
{assume}
4) *case* $s.location[p] = N$
{by: 1}
4.1) $s.Inv7$
{by: 4.1, unchanged location[p], unchanged location[coPortal(p)]}
4.2) $s'.Inv7$ for p and coPortal(p)
{by: 4, 4.2}
5) $s.location[p] = N \Rightarrow s'.Inv7$ for p and coPortal(p)
{assume}
6) *case* $s.location[p] = P_4$
{by: 6, Inv5}
6.1) $s.location[coPortal(p)] \neq P_4$
{by: 6.1, unchanged location[p], unchanged location[coPortal(p)]}
6.2) $s'.Inv7$ for p and coPortal(p)
{by: 6, 6.2}
7) $s.location[p] = P_4 \Rightarrow s'.Inv7$ for p and coPortal(p)
{by: 3, 5, 7}
8) $s'.Inv7$ for p and coPortal(p)

{by: 1, 8}
9) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv7 : \ s'.Inv7)$

Invariant 8

$(\forall(s : \ states, p : \ portals) :: \ (s.location[p] = N \wedge s.location[coPortal(p)] = N)$
$\Rightarrow (s.msgCoPortal[p] = \emptyset \wedge s.msgCoPortal[coPortal(p)] = \emptyset))$

Initial state

{by: Initial value}
1) $s_0.msgCoPortal[p] = \emptyset$
{by: Initial value}
2) $s_0.msgCoPortal[coPortal(p)] = \emptyset$
{by: 1, 2}
3) $s_0.Inv8$ for p and coPortal(p)

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {startPanic(p), recvPanicCoPortalA(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: effect(a)}
2) $s'.loc[p] = P_0$
{by: 2}
3) $s'.Inv8$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {recvPanicCoPortalB(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: precondition(a)}
2) $s.location[p] \in \{P_0, P_1, P_3\}$
{by: unchanged location[p]}

3) $s'.location[p] \in \{P_0, P_1, P_3\}$
{by: 3}
4) $s'.Inv8$ for p and coPortal(p)
{by: 1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {recvPanicCoPortalC(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: precondition(a)}
2) $s.msgCoPortal[p] = PANIC$
{by: 2, Inv1}
3) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: effect(a)}
4) $s'.msgCoPortal[p] = \emptyset$
{by: unchanged msgCoPortal[coPortal(p)]}
5) $s'.msgCoPortal[coPortal(p)]$
{by: 4, 5}
6) $s'.Inv8$ for p and coPortal(p)
{by: 1, 6}
7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: effect(a)}
2) $s'.location[p] = P_1$
{by: 2}
3) $s'.Inv8$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {recvMsgBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: precondition(a)}
2) $s.location[p] \neq N$
{by: 2, unchanged location[p]}
3) $s'.location[p] \neq N$
{by: 3}
4) $s'.Inv8$ for p and coPortal(p)
{by: 1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

case $a \in$ {sendPanicCoPortal(p)}

{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: effect(a)}
2) $s'.location[p] = P_4$
{by: 2}
3) $s'.Inv8$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8:\ s'.Inv8)$

case $a \in$ {sendPanicBus(p, P)}
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: effect(a)}
2) $s'.location[p] = P_1$
{by: 2}
3) $s'.Inv8$ for p and coPortal(p)
{by: 1, 3}
4) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8:\ s'.Inv8)$

case $a \in$ {finish(p)}
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{by: precondition(a)}
2) $s.adjBusPanicComplete[p]$
{by: 2, Inv6.2}
3) $s.location[coPortal(p)] = P_4$
{by: unchanged location[coPortal(p)]}
4) $s'.location[coPortal(p)] = P_4$
{by: 4}
5) $s'.Inv8$ for p and coPortal(p)
{by: 1, 5}
6) $(\forall s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8:\ s'.Inv8)$

case $a \in$ {reset(P), mergeBus(P, Q), splitBus(P, Q)}
{assume}
1) $s, s', a:\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8$
{assume}
2) *case* $s.location[p] = N$
{by: 1}
2.1) $s.Inv8$
{by: 2, 2.1, unchanged location[p]}
2.2) $s'.Inv8$ for p and coPortal(p)
{by: 2, 2.2}
3) $s.location[p] = N \Rightarrow s'.Inv8$ for p and coPortal(p)
{assume}

4) *case* $s.location[p] \neq N$
   {by: effect(a)}
   4.1) $s'.location[p] \neq N$
   {by: 4.1}
   4.2) $s'.Inv8$ for p and coPortal(p)
   {by: 4, 4.2}
5) $s.location[p] \neq N \Rightarrow s'.Inv8$ for p and coPortal(p)
   {by: 3, 5}
6) $s'.Inv8$ for p and coPortal(p)
{by: 1, 6}

7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv8 : \ s'.Inv8)$

Invariant 9

$(\forall (s : states, p : \ portals) :: \ PANIC \in s.msgBus[p]$
$\Rightarrow (\exists q : \ portals) : \ q \in bus(s, p) : \ s.location[q] = P_1))$

Initial state

{by: initial value}

1) $s_0.msgBus[p] = \emptyset$
   {by: 1}
2) $s_0.Inv9$

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9 : \ s'.Inv9)$

case $a \in$ {startPanic(p), recvPanicCoPortalA(p), recvPanicCoPortalB(p), recvPanicCoPortalC(p), sendPanicCoPortal(p)}
{assume}

1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9$
   {by: 1}
   2) $s.Inv9$
   {by: 2, unchanged $msgBus$[p]}
   3) $t.Inv9$
   {by: 1, 3}

4) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9 :\ s'.Inv9)$

case $a \in$ {recvPanicBus(p), recvMsgBus(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9$
{by: effect(a)}
2) $s'.msgBus[p] = \emptyset$
{by: 2}
3) $s'.Inv9$
{by: 1, 3}
4) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9 :\ s'.Inv9)$

case $a \in$ {reset(P), splitBus(P, Q), mergeBus(P, Q)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9$
{by: effect(a)}
2) $s'.msgBus[p] = \emptyset, \forall p \in bus(p)$
{by: 2}
3) $s'.Inv9, \forall p \in bus(p)$
{by: 1, 3}
4) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9 :\ s'.Inv9)$

case $a \in$ {sendPanicBus(p, P)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9$
{by: effect(a)}
2) $s'.location[p] = P_1$
{by: definition of bus}
3) $p \in bus(p)$
{2, 3, predicate calculus}
4) $(\exists p :\ p \in bus(p) :\ s'.location[p] = P_1)$
{by: 4, dummy transformation on p to q}
5) $s'.Inv9$
{by: 1, 5}
6) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv9 :\ s'.Inv9)$

Invariant 10

Checked in PVS.

$(\forall (s :\ states, p :\ portals) ::$
$s.location[p] \neq N \Leftrightarrow s.brdg[p] = 0)$

Initial state

{by: initial value}
1) $s_0.location[p] = N$
{by: 1}
2) $s_0.Inv10$

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

case $a \in$ {startPanic(p), recvPanicCoPortalA(p), recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10$
{by: effect(a)}
2) $s'.location[p] \neq N$
{by: effect(a)}
3) $s'.brdg[p] = 0$
{by: 2 and 3}
4) $s'.Inv10$
{by: 1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

case $a \in$ {recvPanicCoPortalB(p), sendPanicBus(p), sendPanicCoPortal(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10$
{by: precondition(a)}
2) $s.location[p] \neq N$
{by: 1}
3) $s.Inv10$
{by: 2, 3}
4) $s.brdg[p] = 0$
{by: unchanged location[p], unchanged brdg[p]}
5) $s'.Inv10$
{by: 1, 4}
6) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

case $a \in$ {recvPanicCoPortalC(p), finish(p) }
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10$
{by: effect(a)}

2) $s'.location[p] = N$
{by: effect(a)}
3) $s'.brdg[p] \neq 0$
{by: 2 and 3}
4) $s'.Inv10$
{by: 1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

case a = reset(P)
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10$
{assume}
2) $p \in P$
{by: 1}
3) $s.Inv10$
{by: 3, unchanged location[p] = N, unchanged brdg[p]}
4) $s'.Inv10$
{by: 2, 4}
5) $(\forall (p : \ portals) : \ p \in P : \ s'.Inv10)$
{by: 1, 5}
6) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

case $a \in$ {mergeBus(P, Q), splitBus(P, Q)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10$
{assume}
2) $p \in (P \cup Q)$
{by: 1}
3) $s.Inv10$
{by: 3, unchanged location[p] = N, unchanged brdg[p]}
4) $s'.Inv10$
{by: 2, 4}
5) $(\forall (p : \ portals) : \ p \in (P \cup Q) : \ s'.Inv10)$
{by: 1, 5}
6) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv10 : \ s'.Inv10)$

Invariant 11

$(\forall (s : \ states, p : \ portals) :: \ s.location[p] \in \{P_0, P_1\}$
$\quad \Rightarrow s.hopsSincePanic[p] < MaxHop)$
$\wedge$
$(\forall (s : states, p : \ portals) :: \ s.location[p] \in \{P_0, P_1\}$

$(\forall m(h) :\ m(h) \in s.msgBus[p] \wedge m = PANIC :\ h < MaxHop))$

Initial state

{by: initial value}
1) $s_0.location[p] = N$
{by: 1}
2) $s_0.Inv11$ for p

Actions

Prove:
$(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 :\ s'.Inv11)$

case $a \in$ {startPanic(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: 1}
2) $s.Inv11$
{by: effect(a)}
3) $s'.hopsSincePanic[p] = 0$
{by: 2, unchanged msgCoPortal[p]}
4) $(\forall m(h) :\ m(h) \in s'.msgBus[p] \wedge m = PANIC :\ h < MaxHop)$
{by: 3, 4, $MaxHop > 0$}
5) $s'.Inv11$
{by: 1, 5}
6) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 :\ s'.Inv11)$

case $a \in$ {recvPanicCoPortalA(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: 1}
2) $s.Inv11$
{by: 1, unchanged msgCoPortal[p]}
3) $(\forall m(h) :\ m(h) \in s'.msgBus[p] \wedge m = PANIC :\ h < MaxHop)$
{assume}
4) *case* $s.msgCoPortal[p](h) < MaxHop - 1$
{by: effect(a)}
4.1) $s'.hopsSincePanic[p] < MaxHop$
{by: 3, 4.1}
4.2) $s'.Inv11$

{by: 4, 4.2}
5) $s.msgCoPortal[p](h) < MaxHop - 1 \Rightarrow s'.Inv11$
{assume}
6) *case* $s.msgCoPortal[p](h) \geq MaxHop - 1$
{by: effect(a)}
6.1) $s'.location[p] = P_3$
{by: 3, 6.1}
6.2) $s'.Inv11$
{by: 6, 6.2}
7) $s.msgCoPortal[p](h) \geq MaxHop - 1 \Rightarrow s'.Inv11$
{by: 1, 7}
8) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

case $a \in$ {recvPanicCoPortalB(p), recvMsgBus(p), finish(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: 1}
2) $s.Inv11$
{by: 2, unchanges hopsSincePanic[p], unchanged msgCoPortal[p]}
3) $s'.Inv11$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

case $a \in$ {recvPanicCoPortalC(p), sendPanicCoPortal(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: effect(a)}
2) $s.location[p] \notin \{P_0, P_1\}$
{by: 2}
3) $s'.Inv11$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

case $a \in$ {recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{effect(a)}
2) $s'.msgBus(p) = \emptyset$
{by: 1}
3) $s.Inv11$
{by: 3}
4) $(\forall m(h) : \ m(h) \in s.msgBus[p] \wedge m = PANIC : \ h < MaxHop))$
{by: 4, effect(a)}
5) $s'.hopsSincePanic[p] < MaxHop$
{by: 2, 5}

6) $s'.Inv11$
{by: 1, 6}
7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

case $a \in$ {sendPanicBus(p, P)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: 1}
2) $s.Inv11$
{by: 2}
3) $s.hopsSincePanic[p] < MaxHop$
{by: 3, effect(a)}
4) $(\forall q : \ q \in P : \ s'.msgBus[q] < MaxHop)$
{by: 2, unchanged hopsSincePanic[q] $\forall q \in P$}
5) $s'.inv11 \forall q \in P$
{by: 1, unchanged hopsSincePanic[p], unchanged $msgBus$[p]}
6) $s' Inv11$ for p
{by: 1, 5, 6}
7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

case $a \in$ {reset(P), mergeBus(P, Q), splitBus(P, Q)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11$
{by: 1}
2) $s.Inv11$
{by: effect(a)}
3) $s'.msgBus[p] = \emptyset, \forall p \in bus(p)$
{by: 2, 3, unchanged hopsSincePanic[p] $\forall p \in bus(p)$}
4) $s'.Inv11 \forall p \in bus(p)$
{by: 1, 4}
5) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv11 : \ s'.Inv11)$

Invariant 12

$(\forall (s : states, p : \ portals) :: \ s.location[p] = P_1$
$\Rightarrow s.busReset(bus(s, p))$

Initial state

{by: initial value}
1) $s_0.location[p] \neq P_1$

{by: 1}
2) $s_0.Inv12$

Actions

Prove:
$(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

case $a \in$ {startPanic(p), recvPanicCoPortalA(p), recvPanicCoPortalC(p), sendPanicCoPortal(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12$
{by: effect(a)}
2) $s'.location[p] \neq P_1$
{by: 2}
3) $s'.Inv12$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

case $a \in$ {recvPanicCoPortalB(p), recvMsgBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12$
{by: 1}
2) $s.Inv12$
{by: 2, unchanged location[p]}
3) $s'.Inv12$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

case $a \in$ {recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12$
{by: 1}
2) $s.Inv12$
{by: precondition(a)}
3) $PANIC \in s.msgBus[p]$
{by: 3, Inv9}
4) $(\exists q : \ q \in bus(p) : \ s.location[q] = P_1)$
{by: 2, 4}
5) $s.busReset(bus(p))$
{by: 5, unchanged busReset(bus(p))}
6) $s'.busReset(bus(p))$

{by: 6}
7) $s'.Inv12$
{by: 1, 7}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

case $a \in$ {sendPanicBus(p, P)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12$
{by: effect(a)}
2) $s'.busReset(bus(p))$
{by: 2}
3) $s'.Inv12$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

case $a \in$ {reset(P), splitBus(P, Q), mergeBus(P, Q)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12$
{by: effect(a)}
2) $s'.location[p] \neq P_1, \forall p \in bus(p)$
{by: 2}
3) $s'.Inv12, \forall p \in bus(p)$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv12 : \ s'.Inv12)$

Invariant 13

$(\forall (s : \ states, p : \ portals) :: \ s.location[p] = N \wedge s.location[coPortal(p)] \in \{P_0, P_1, P_3\}$
$\Rightarrow s.msgCoPortal[coPortal(p)] = \emptyset)$

Initial state

{by: initial value}
1) $s_0.location[p] = N$
{by: initial value}
2) $s_0.location[coPortal(p)] = N$
{by: 2}
3) $s_0.Inv13$ for p
{by: 1}
3) $s_0.Inv13$ for coPortal(p)

Actions

Prove:
$(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 :\ s'.Inv13)$

case $a \in$ {startPanic(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: 1}
2) $s.Inv13$
{by: effect(a)}
3) $s'.location[p] = P_0$
{by: 3}
4) $s'.Inv13$ for p
{assume}
5) *case* $s'.location[coPortal(p)] = N$
{by: 5, Inv1.2}
5.1) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 5.1}
5.2) $s'.Inv13$ for coPortal(p)
{by: 5, 5.2}
6) $s'.location[coPortal(p)] = N \Rightarrow s'.Inv13$ for coPortal(p)
{assume}
7) *case* $s'.location[coPortal(p)] \neq N$
{by: 7}
7.1) $s'.Inv13$ for coPortal(p)
{by: 7, 7.1}
8) $s'.location[coPortal(p)] \neq N \Rightarrow s'.Inv13$ for coPortal(p)
{by: 1, 4, 6, 8}
9) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 :\ s'.Inv13)$

case $a \in$ {recvPanicCoPortalA(p), recvPanicCoPortalB(p),
recvPanicCoPortalC(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: precondition(a)}
2) $s.msgCoPortal[p] = PANIC$
{by: 2, Inv1.1}
3) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: effect(a)}
4) $s'.msgCoPortal[p] = \emptyset$
{by: 3, unchanged msgCoPortal[coPortal(p)]}
5) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 5}
6) $s'.Inv13$ for p

{by: 4}
7) $s'.Inv13$ for coPortal(p)
{by: 1, 6, 7}
8) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 : \ s'.Inv13)$

case $a \in$ {recvPanicBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: precondition(a)}
2) $s.location[p] = N$
{by: effect(a)}
3) $s'.location[p] = P_1$
{by: 3}
4) $s'.Inv13$ for p
{assume}
5) *case* $s.location[coPortal(p)] = N$
{by: 2, 5, Inv8}
5.1) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: 5.1, unchanged msgCoPortal[coPortal(p)]}
5.2) $s'.msgCoPortal[coPortal(p)] = \emptyset$
{by: 5.2}
5.3) $s'.Inv13$ for coPortal(p)
{by: 5, 5.3}
6) $s.location[coPortal(p)] = N \Rightarrow s'.Inv13$ for coPortal(p)
{assume}
7) *case* $s.location[coPortal(p)] \neq N$
{by: unchanged location[coPortal(p)]}
7.1) $s'.location[coPortal(p)] \neq N$
{by: 7.1}
7.2) $s'.Inv13$ for coPortal(p)
{by: 7, 7.2}
8) $s.location[coPortal(p)] \neq N \Rightarrow s'.Inv13$ for coPortal(p)
{by: 1, 4, 6, 8}
9) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 : \ s'.Inv13)$

case $a \in$ {recvMsgBus(p)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: 1}
2) $s.Inv13$
{by: 2, unchanged location[p], unchanged location[coPortal(p)]}
3) $s'.Inv13$
{by: 1, 3}
4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 : \ s'.Inv13)$

case $a \in$ {sendPanicCoPortal(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: precondition(a)}
2) $\neg s.semaphore[bridgeId(p)]$
{by: 2, Inv1.2}
3) $s.msgCoPortal[p] = \emptyset$
{by: effect(a)}
4) $s'.location[p] = P_4$
{by: 3, unchanged msgCoPortal[p]}
5) $s'.msgCoPortal[p] = \emptyset$
{by: 4}
6) $s'.Inv13$ for p
{by: 5}
7) $s'.Inv13$ for coPortal(p)
{by: 1, 6, 7}
8) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 :\ s'.Inv13)$

case $a \in$ {sendPanicBus(p, P)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: 1}
2) $s.Inv13$
{by: 2, unchanged location[p] $\in \{P_0, P_1, P_3\}$,
unchanged location[coPortal(p)]}
3) $s'.Inv13$
{by: 1, 3}
4) $(\forall s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 :\ s'.Inv13)$

case $a \in$ {finish(p)}
{assume}
1) $s, s', a :\ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: precondition(p)}
2) $s.adjBusPanicComplete[p]$
{by: 2, Inv6.2}
3) $s.location[coPortal(p)] = P_4$
{by: 3, unchanged location[coPortal(p)]}
4) $s'.location[coPortal(p)] = P_4$
{by: effect(a)}
5) $s'.location[p] = N$
{by: 4}
6) $s'.Inv13$ for p
{by: 5}
7) $s'.Inv13$ for coPortal(p)
{by: 1, 6, 7}

4) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 : \ s'.Inv13)$

case $a \in$ {reset(P), mergeBus(P, Q), splitBus(P, Q)}
{assume}
1) $s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13$
{by: 1}
2) $s.Inv13$
{assume}
3) $case\ s.location[p] \in \{P_0, P_1, P_3\}, \forall p \in bus(p)$
{by: 2, unchanged location[coPortal(p)],
unchanged location[p] $\in \{P_0, P_1, P_3\} \forall p \in bus(p)$}
3.1) $s'.Inv13, \forall p \in bus(p) \wedge coPortal(p)$
{by: 3, 3.1}
4) $s.location[p] \in \{P_0, P_1, P_3\} \Rightarrow s'.Inv13,$
$\forall p \in bus(p) \wedge coPortal(p)$
{assume}
5) $case\ s.location[p] \in \{N, P_4\}, \forall p \in bus(p)$
{by: 2, unchanged location[coPortal(p)],
unchanged location[coPortal(p)] $\in \{N, P_4\}$
5.1) $s'.Inv13, \forall p \in bus(p) \wedge coPortal(p)$
{by: 5, 5.1}
6) $s.location[p] \in \{N, P_4\} \Rightarrow s'.Inv13,$
$\forall p \in bus(p) \wedge coPortal(p)$
{by: 1, 4, 6}
7) $(\forall s, s', a : \ reachable(s) \wedge s \xrightarrow{a} s' \wedge s.Inv13 : \ s'.Inv13)$

C.3 Properties

C.3.1 Property 1: Correctness

Unfortunately there was not enough time to construct a proof for this property.

C.3.2 Property 2: No Deadlock

proof:
$(\forall(q : portals, s : state) : reachable(s) :$
$\quad(\exists(p : portal) : \ p \in net(s,q) : \ s.location[p] \neq N \vee s.toPanic[p])$
$\qquad\Rightarrow (\exists(p : portal, a : actions) : p \in net(s,q) \wedge a \notin \{mergeBus, splitBus, reset\} :$
$\qquad\quad enabled(s, a(p)) \vee busReset[bus(s,p)]))$
$\equiv$ {by: Contraposition, deMorgan}
$(\forall(q : portals, s : state) :$
$\quad(\forall(p : portals, a : actions) : \ p \in net(s,q) \wedge a \notin \{mergeBus, splitBus, reset\} :$
$\qquad\neg enabled(s, a(p)) \wedge \neg busReset[bus(s,p)])) :$
$\quad(\forall(p : portals) : \ p \in net(s,q) : \ s.location[p] = N \wedge \neg s.toPanic[p]))$

$\quad$ {assume}
1) $q, s :$
$\qquad(\forall(p : portals, a : actions) : p \in net(s,q) \wedge a \notin \{mergeBus, splitBus, reset\} :$
$\qquad\quad\neg enabled(a(p)) \wedge \neg busReset[bus(s,p)])$
$\qquad$ {assume}
$\quad$1.1) $p : \ p \in net(s,q)$
$\qquad\qquad$ {by: 1, 1.1}
$\qquad\quad$2) $(\forall(a : action) : \ a \notin \{mergeBus, splitBus, reset\} :$
$\qquad\quad\neg enabled(a(p)) \wedge \neg busReset[bus(s,p)])$
$\qquad\quad$ {by: 1, 1.1}
$\qquad\quad$3) $(\forall(a : action) : \ a \notin \{mergeBus, splitBus, reset\} :$
$\qquad\quad\neg enabled(a(coPortal(p))) \wedge \neg busReset[bus(s, coPortal(p))])$
$\qquad\quad$ {by: 2, expand enabled(a)}
$\qquad\quad$ {a = startPanic[p]}
$\qquad\quad$4.1) $s.toPanic[p] \Rightarrow s.loc[p] \neq N$
$\qquad\quad$ {a = recvPanicCoPortalA[p]}
$\qquad\quad$4.2) $s.msgCoPortal[p] \neq \emptyset \Rightarrow s.location[p] \neq N$
$\qquad\quad$ {a = recvPanicCoPortalB[p]}
$\qquad\quad$4.3) $s.msgCoPortal[p] \neq \emptyset \Rightarrow s.location[p] \in \{N, P_4\}$
$\qquad\quad$ {a = recvPanicCoPortalC[p]}
$\qquad\quad$4.4) $s.msgCoPortal[p] \neq \emptyset \Rightarrow s.location[p] \neq P_4$
$\qquad\quad$ {a = sendPanicBus(p)}
$\qquad\quad$4.5) $s.location[p] \neq P_0$

{a = sendPanicCoPortal(p)}
4.6) $\neg s.semaphore[bridgeId(p)] \Rightarrow (\neg s.adjBusPanicComplete[p] \Rightarrow s.location[p] \neq P_3))$
{a = finish(p)}
4.7) $\neg s.semaphore[bridgeId(p)] \Rightarrow (s.adjBusPanicComplete[p] \Rightarrow s.location[p] \neq P_3))$
4.8) $\neg busReset[bus(p)]$
{by: 3, expand enabled(a)}
{recvPanicCoPortalA(coPortal(p))}
5.1) $s.msgCoPortal[coPortal(p)] \neq \emptyset \Rightarrow s.location[coPortal(p)] \neq N$
{recvPanicCoPortalB(coPortal(p))}
5.2) $s.msgCoPortal[coPortal(p)] \neq \emptyset \Rightarrow s.location[coPortal(p)] \in \{N, P_4\}$
{recvPanicCoPortalC(coPortal(p))}
5.3) $s.msgCoPortal[coPortal(p)] \neq \emptyset \Rightarrow s.location[coPortal(p)] \neq P_4$
{by: 4.8, Inv12}
6) $s.location[p] \neq P_1$
{assume}
7) $s.msgCoPortal[p] \neq \emptyset$
{by: 7, 4.2}
7.1) $s.location[p] \neq N$
{by: 7, 4.4}
7.2) $s.location[p] \neq P_4$
{by: 7, 4.3}
7.3) $s.location[p] \in \{N, P_4\}$
{by: 7.1, 7.2, 7.3}
7.4) *False*
{by: 7, 7.4}
8) $s.msgCoPortal[p] = \emptyset$
{by: 5.1, 5.2, 5.3, 7, 8, symmetry}
9) $s.msgCoPortal[coPortal(p)] = \emptyset$
{by: 8, 9, Inv1}
10) $\neg s.semaphore[bridgeId(p)]$
{by: 10, 4.6, 4.7}
11) $s.location[p] \neq P_3$
{assume}
12) $s.location[p] \neq N$
{by: 4.5, 6, 11}
12.1) $s.location[p] = P_4$
{by: 12.1, Inv3}
12.2) $\neg s.adjBusPanicComplete[p]$
{8, 12.2, Inv5}
12.3) $s.location[coPortal(p)] = P_4 \vee s.msgCoPortal[coPortal(p)] = PANIC$
{by: 12.3}
12.4) $s.msgCoPortal[coPortal(p)] = \emptyset \Rightarrow s.location[coPortal(p)] = P_4$
{by: 9, 12.4}
12.5) $s.location[coPortal(p)] = P_4$
{by: 12.5, Inv5}
12.6) *False*

13) $s.location[p] = N$
{by: 4.1, 13}
14) $\neg s.toPanic[p]$
{by: 1.1, 13, 14}
1.2) $(\forall(p : portal) : \ p \in net(s,q) : \ s.location[p] = N \wedge \neg s.toPanic[p]))$
{by: 1, 1.2}
1.3)$(\forall(q : portals, s : state) :$
$(\forall(p : portals, a : actions) : \ p \in net(s,q) \wedge a \notin \{mergeBus, splitBus, reset\} :$
$\neg enabled(s, a(p)) \wedge \neg busReset[bus(s,p)]) :$
$(\forall(p : portals) : \ p \in net(s,q) : \ s.location[p] = N \wedge \neg s.toPanic[p]))$

C.3.3 Property 3: Termination

	comp1	comp2	comp3	comp4	comp5
startPanic	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$=$
recvPanicCoPortalA	$=$	$\downarrow$	$\uparrow$	$\downarrow$	$=$
recvPanicCoPortalB	$=$	$=$	$=$	$\downarrow$	$=$
recvPanicCoPortalC	$=$	$=$	$\downarrow$	$=$	$=$
recvPanicBus	$=$	$\downarrow$	$\uparrow$	$\uparrow$	$=$
recvMsgBus	$=$	$=$	$=$	$=$	$\downarrow$
sendPanicCoPortal	$=$	$=$	$\downarrow$	$=$	$=$
finish	$=$	$=$	$\downarrow$	$\uparrow$	$=$
sendPanicBus	$=$	$=$	$\downarrow$	$=$	$\uparrow$
Reset	$=$	$=$	$\downarrow \setminus \uparrow$	$=$	$\downarrow$

Table legend:

- $=$, the component remains constant for the action.
- $\uparrow$, the component increases for the action.
- $\downarrow$, the component decreases for the action.

Each component of property 3 is proven individually. For each component it must be shown, for each actions that $\downarrow$or $=$, that the component either decreases or remains constant respectively, unless there is a higher ordered component that is already decreasing (ordering in the table: high to low is from left to right).

The proof of each component requires certain lemmas with respect to the set of portals P that is not involved in the effect of an action a. The lemmas prove: if a executes then the contribution of the portals in P to the component is the same before and after the execution of a.

The proof of each component is split over the actions for which a proof is required to show that the component decreases or remains constant. At the start of each action a short summary of the situation pertaining that action is given.

The strategy to prove the L > R is to apply worst case assumptions. L is made as small as possible and R is made as large as possible.

Component 1

comp1(s: state) =
  $(\#(q : \ portals) :: \ s.toPanic[q])$

Lemma comp1.1

$(\forall(p : portals, s, t : state, a : action) : s \overset{a(p)}{\rightarrow} t :$
    $(\#(q : \ portals) : \ q \neq p : \ s.toPanic[q])$
  $=$
    $(\#(q : \ portals) : \ q \neq p : \ t.toPanic[q])$

Proof of Component 1 According to the table, we have to prove that:

- Component 1 decreases for action $a =$ startPanic
- Component 1 remains constant for action $a \neq$ startPanic

Action $a =$ startPanic(p), prove decreasing
situation:
    $s.toPanic[p]$
  $\neg$ $t.toPanic[p]$

proof:
      comp1(s) > comp1(t)
  $\equiv$ {Expand comp1}
      $(\#(q : \ portals) :: \ s.toPanic[q])$
    $>$
      $(\#(q : \ portals) :: \ t.toPanic[q])$
  $\Leftarrow$ {Domain split on p, use situation}
      $(\#(q : \ portals) : \ q \neq p : \ s.toPanic[q]) + 1$
    $>$
      $(\#(q : \ portals) : \ q \neq p : \ t.toPanic[q]) + 0$

$\Leftarrow$\{Lemma comp1.1, predicate calculus\}
$1 > 0$
$\Leftarrow$\{Predicate calculus\}
True

Action $\neq$ startPanic(p), prove constant
situation:
$s.toPanic[p] = t.toPanic[p]$

rewrites:
$(s.toPanic[p]\ ?\ 1 : 0) = x$

proof:
comp1(s, p) $\geq$ comp1(t, p)
$\equiv$ {Expand comp1}
$(\#(q :\ portals) ::\ s.toPanic[q])$
$\geq$
$(\#(q :\ portals) ::\ t.toPanic[q])$
$\Leftarrow$\{Domain split on p, use situation and rewrites\}
$(\#(q :\ portals)) :\ q \neq p :\ s.toPanic[q]) + x$
$\geq$
$(\#(q \in portals) :\ q \neq p :\ t.toPanic[q]) + x$
$\Leftarrow$\{Lemma comp1.1, predicate calculus\}
True

Component 2

Functions

half(s: state, p: portals): boolean =
$(s.location[coPortal(p)] = N \wedge s.msgCoPortal[p] = \emptyset \wedge s.location[p] \neq N)$
$\vee\ (s.location[p] \notin \{N, P_4\} \wedge s.msgCoPortal[coPortal(p)] = PANIC)$

pass(s: state, p: portals): boolean =
$(s.location[p] = N) \wedge (s.location[coPortal(p)] = N$
$\vee(s.location[coPortal(p)] = P_4 \wedge s.msgCoPortal[coPortal(p)] = PANIC)))$
$\vee\ (s.location[p] = P_3)$
$\vee\ (s.location[p] = P_4 \wedge (s.location[coPortal(p)] = N \vee s.msgCoPortal[p] = \emptyset))$

busInPanic(s: state, p: portals): boolean =
$(\exists(q :\ portals) :\ q \in bus(s, p) :\ s.location[q] \in \{P_0, P_1\})$

nrPanics(h: nat): nat =
  $MaxNode^{(MaxHop-h)}$

Component 2

comp2(s: state) =
    comp2_half(s, $\{q \in portals\}$)
  +
    comp2_pass(s, $\{q \in portals\}$)

comp2_half(s: state, P: setof[portals]): $\mathbb{N}$ =
  $(\sum(q : portals) : \ q \in P \wedge half(s, q) :$
    $(MaxNode - 1) * nrPanics(s.hopsSincePanic[q] + 1))$

comp2_pass(s: state, P: setof[portals]): $\mathbb{N}$ =
    $(\sum(q : portals) : \ q \in P \wedge busInPanic(s, q) \wedge pass(s, q)) :$
      $nrPanics((\downarrow (r \in bus(s, q)) : r \in \{P_0, P_1\} : s.hopsSincePanic[r])) - 1)$

(Note: An empty domain of the inner $\downarrow$ quantor poses no problem because in this case the domain the $\sum$ quantor is empty as well)

Lemma comp2.1

$(\forall(p : portals, \ s, t : state, \ a : action) : s \overset{a(p)}{\rightarrow} t :$
    $comp2\_half(s, \{q \in portals \mid q \neq p \wedge q \neq coPortal(p)\})$
  =
    $comp2\_half(t, \{q \in portals \mid q \neq p \wedge q \neq coPortal(p)\}))$

Lemma comp2.2
Note: If we exclude the bus of the portal p involved in the transition, then the value of comp2_pass is constant for that transition. The domain of the $\downarrow$ in the definition of comp2_pass over $bus(s, q)$ still includes p and coPortal(p).

$(\forall(p : portals, \ s, t : state, \ a : action) : \ s \overset{a(p)}{\rightarrow} t :$
    $comp2\_pass(s, \{q \in portals \mid q \notin bus(s, p) \wedge q \neq coPortal(p)\})$
  =
    $comp2\_pass(t, \{q \in portals \mid q \notin bus(t, p) \wedge q \neq coPortal(p)\}))$

Lemma comp2.3

$$
\begin{array}{l}
(\forall(p : \mathit{portals},\ s,t : \mathit{state},\ a : \mathit{action}) :\ s \xrightarrow{a(p)} t : \\
\quad \{q : \mathit{portals} \mid q \in \mathit{bus}(s,p) \wedge s.\mathit{location}[q] \in \{P_0, P_1\}\} = \{q : \mathit{portals} \mid q \in \mathit{bus}(t,p) \wedge t.\mathit{location}[q] \in \{P_0, P_1\}\} \\
\quad \Rightarrow \\
\qquad \mathit{comp2\_pass}(s, \{q \in \mathit{portals} \mid q \in \mathit{bus}(s,p) \wedge q \neq p\}) \\
\quad = \\
\qquad \mathit{comp2\_pass}(t, \{q \in \mathit{portals} \mid q \in \mathit{bus}(t,p) \wedge q \neq p\}))
\end{array}
$$

Lemma comp2.4

$$
\begin{array}{l}
\qquad \text{comp2\_pass}(s,\ \{p\}) + \text{comp2\_pass}(s,\ \{\text{coPortal}(p)\}) \\
\quad > \\
\qquad \text{comp2\_half}(t,\ \{p\})
\end{array}
$$

Situation of lemma comp2.4:
pass(s, p)
pass(s, $\mathit{coPortal}(p)$)
half(s, p)
busInPanic(s, p)
busInPanic(t,p)
busInPanic(s,$\mathit{coPortal}(p)$) = busInPanic(t, $\mathit{coPortal}(p)$)

Rewrites of lemma comp2.4:
$(\downarrow (r : \mathit{portals}) :\ r \in \mathit{bus}(s,p) \wedge s.\mathit{location}[r] \in \{P_0, P_1\} :$
$\quad s.\mathit{hopsSincePanic}[r]) = \text{h}$
$(\downarrow (r : \mathit{portals}) :\ r \in \mathit{bus}(t,p) \wedge t.\mathit{location}[r] \in \{P_0, P_1\} :$
$\quad t.\mathit{hopsSincePanic}[r]) = \text{h}$
$\quad$ {by: inv11, worst case}
$(\downarrow (r : \mathit{portals}) :\ r \in \mathit{bus}(s, \mathit{coPortal}(p)) \wedge s.\mathit{location}[r] \in \{P_0, P_1\} :$
$\quad s.\mathit{hopsSincePanic}[r]) = \mathit{MaxHop} - 1$
$t.\mathit{hopsSincePanic}[p] = \text{h}$

Proof of lemma comp2.4:

$$
\begin{array}{l}
\qquad \text{comp2\_pass}(s,\ \{p\}) + \text{comp2\_pass}(s,\ \{\text{coPortal}(p)\}) \\
\quad > \\
\qquad \text{comp2\_half}(t,\ \{p\}) \\
\equiv \{\text{Expand comp2\_pass and comp2\_half}\} \\
\qquad \mathit{nrPanics}((\downarrow (r : \mathit{portals}) :\ r \in \mathit{bus}(s,p) \wedge s.\mathit{location}[r] \in \{P_0, P_1\} : \\
\qquad\quad s.\mathit{hopsSincePanic}[r])) - 1 \\
\quad + \\
\qquad \mathit{nrPanics}((\downarrow (r : \mathit{portals}) :\ r \in \mathit{bus}(\mathit{coPortal}(s,p)) \wedge s.\mathit{location}[r] \in \{P_0, P_1\} : \\
\qquad\quad s.\mathit{hopsSincePanic}[r])) - 1
\end{array}
$$

$$
\begin{array}{ll}
& > \\
& \quad (MaxNode - 1) * nrPanics(t.hopsSincePanic[p] + 1) \\
\equiv & \{\text{Use rewrites}\} \\
& \quad nrPanics(h) - 1 \\
& + \\
& \quad nrPanics(MaxHop - 1) - 1 \\
& > \\
& \quad (MaxNode - 1) * nrPanics(h + 1) \\
\equiv & \{\text{Math, expand } nrPanics\} \\
& \quad MaxNode^{(MaxHop-h)} - 1 \\
& + \\
& \quad MaxNode - 1 \\
& > \\
& \quad (MaxNode - 1) * MaxNode^{MaxHop-h-1} \\
\equiv & \{\text{Math}\} \\
& \quad MaxNode^{(MaxHop-h)} - 1 \\
& + \\
& \quad MaxNode - 1 \\
& > \\
& \quad MaxNode^{(MaxHop-h)} - MaxNode^{MaxHop-h-1} \\
\equiv & \{\text{Math, subtract } MaxNode^{MaxHop-h} \text{ from both sides}\} \\
& \quad MaxNode - 2 \\
& > \\
& \quad -MaxNode^{(MaxHop-h-1)} \\
\equiv & \{\text{Math}\} \\
& \quad MaxNode + MaxNode^{(MaxHop-h-1)} \\
& > \\
& \quad 2 \\
\equiv & \{\text{Math, worst case } h = MaxHop - 1\} \\
& \quad MaxNode + 1 \\
& > \\
& \quad 2 \\
\Leftarrow & \{\text{Predicate calculus, Using } MaxNode \geq 2\} \\
& \quad True
\end{array}
$$

Lemma comp2.5a

$$
\begin{array}{l}
comp2\_pass(s, \{q : portals\}) = \\
\quad comp2\_pass(s, \{q : portals \mid q \notin bus(s, p) \wedge q \neq coPortal(p)\}) \\
+ \\
\quad comp2\_pass(s, \{q : portals \mid q \in bus(s, p) \vee q = coPortal(p)\})
\end{array}
$$

Lemma comp2.5b

$$
\begin{array}{l}
comp2\_half(s, \{q : portals\}) = \\
\quad comp2\_half(s, \{q : portals \mid q \notin bus(s,p) \wedge q \neq coPortal(p)\}) \\
+ \\
\quad comp2\_half(s, \{q : portals \mid q \in bus(s,p) \vee q = coPortal(p)\})
\end{array}
$$

Lemma comp2.6

$$
\begin{array}{l}
(\forall (P : setof[portals],\ s,t : state) :\ P \in s.topology \wedge s \overset{reset(P))}{\rightarrow} t : \\
\quad comp2\_half(s, \{q \in portals \mid q \notin P\}) \\
\quad = \\
\quad comp2\_half(t, \{q \in portals \mid q \notin P\}))
\end{array}
$$

Lemma comp2.7

$$
\begin{array}{l}
(\forall (P : setof[portals],\ s,t : state) :\ P \in s.topology \wedge s \overset{reset(P))}{\rightarrow} t : \\
\quad comp2\_pass(s, \{q \in portals \mid q \notin P\}) \\
\quad = \\
\quad comp2\_pass(t, \{q \in portals \mid q \notin P\})
\end{array}
$$

Lemma comp2.8

$$
\begin{array}{l}
(\forall (q : portals,\ P : setof[portals],\ s,t : state) : \\
\quad s \overset{reset(P)}{\rightarrow} t \wedge P \in s.topology \wedge q \in P \wedge s.location[q] \in \{P_0, P_1, P_3\} \\
\quad\quad \wedge\ t.location[q] \in \{P_0, P_1, P_3\} : \\
\quad half(s,q) = half(t,q))\}
\end{array}
$$

Lemma comp2.9

comp2(s) $\geq$ comp2(t)

Situation of lemma comp2.9:

$s \overset{a(p)}{\rightarrow} t$

half(t, p) = half(s, p)
half(t, $coPortal(p)$) = half(s, $coPortal(p)$)

pass(s, p) = pass(t, p)
pass(t, $coPortal(p)$) = pass(s, $coPortal(p)$)

Proof of lemma comp2.9:
comp2(s) $\geq$ comp2(t)
$\equiv$ {Expand comp2}
comp2_half(s, $\{q \mid q \in portals\}$) + comp2_pass(s, $\{q \mid q \in portals\}$)
$\geq$
comp2_half(t, $\{q \mid q \in portals\}$) + comp2_pass(t, $\{q \mid q \in portals\}$)
$\Leftarrow$ {Apply lemma comp2.1 and lemma comp2.2}
comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
+
comp2_pass(s, $bus(s,p)$) + comp2_pass(s, {coPortal(p)})
$\geq$
comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, $bus(t,p)$) + comp2_pass(t, {coPortal(p)})
$\Leftarrow$ {Math, use situation and domain split on p}
comp2_pass(s, ($bus(s,p)$ - $\{p\}$))
$\geq$
comp2_pass(t, ($bus(t,p)$ - $\{p\}$))
$\Leftarrow$ {Use lemma comp2.3}
True

proof of Component 2

According to the table, we have to prove that:

- Component 2 decreases for action $a \in$ {recvPanicCoPortalA, recvPanicBus}
- Component 2 remains constant for $a \in$ {recvPanicCoPortalB, recvPanicCoPortalC, recvMsgBus, sendPanicCoPortal, finish, sendPanicBus, Reset}

Action a = recvPanicCoPortalA(p), prove decreasing

Situation:
$s.location[p] = N$
$t.location[p] = P_0$
$s.location[coPortal(p)] = t.location[coPortal(p)]$
{by: effect and Inv6.2}

$t.location[coPortal(p)] = P_4$

$s.msgCoPortal[p] = PANIC$
$t.msgCoPortal[p] = \emptyset$
$s.msgCoPortal[coPortal(p)] = \emptyset$
$t.msgCoPortal[coPortal(p)] = \emptyset$

$\neg$ half(s, p)
$\neg$ half(t, p)
  half(s, coPortal(p))
$\neg$ half(t, coPortal(p))

$\neg$ pass(s, p)
$\neg$ pass(t, p)
  pass(s, coPortal(p))
  pass(t, coPortal(p))

$(\forall(q : portals) :\ q \in bus(p) :\ busInPanic(t, q))$

Rewrites:
$t.hopsSincePanic[p] = h + 1$
$s.hopsSincePanic[coPortal(p)] = h$
$t.hopsSincePanic[coPortal(p)] = h$
$(\downarrow (r : portals) :\ r \in (bus(p) - \{p\})) :$
  $r \in \{P_0, P_1\} : t.hopsSincePanic[r]) = k$

Proof:
    comp2(s) > comp2(t)
≡ {Expand comp2}
    comp2_half(s, $\{q \mid q \in portals\}$) + comp2_pass(s, $\{q \mid q \in portals\}$)
  >
    comp2_half(t, $\{q \mid q \in portals\}$) + comp2_pass(t, $\{q \mid q \in portals\}$)
⇐{Apply lemmas comp2.1, comp2.2, comp2.5a and comp2.5b}
    comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
  +
    comp2_pass(s, $bus(s, p)$) + comp2_pass(s, {coPortal(p)})
  >
    comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
  +
    comp2_pass(t, $bus(t, p)$) + comp2_pass(t, {coPortal(p)})
⇐{Use situation and domain split on p,
  worst case $(\forall(q : portals) :\ q \in (bus(s, p) - \{p\}) :\ s.location[q] \notin \{P_0, P_1\})$}
    comp2_half(s, {coPortal(p)})
  +
    comp2_pass(s, ($bus(s, p)$ - $\{p\}$)) + comp2_pass(s, $\{p\}$)
  >

comp2_pass(t, ($bus(t,p)$ - $\{p\}$)) + comp2_pass(t, $\{p\}$)
⇐{Use situation, math}
comp2_half(s, $\{\text{coPortal}(p)\}$)
>
comp2_pass(t, ($bus(t,p)$ - $\{p\}$))
≡ {Expand comp2_half and comp2_pass, use rewrites}
$(MaxNode - 1) * nrPanics(h + 1)$
>
$(\sum(q \in (bus(p) - \{p\})) : busInPanic(t,q) \wedge pass(t,q)) :$
$nrPanics((\downarrow (r : portals) : \ r \in bus(t,q))$
$\wedge\ t.location[r] \in \{P_0, P_1\} : t.hopsSincePanic[r])) - 1)$
⇐{Use math, lemma 1, domain split on p, worst case assumption on the domain of $\sum$}
$(MaxNode - 1) * nrPanics(h + 1)$
>
$(MaxNode - 1) * nrPanics($
$(\downarrow (r : \ portals) : \ r \in bus(t,p) \wedge r \neq p \wedge t.location[r] \in \{P_0, P_1\} :$
$t.hopsSincePanic[r]) \downarrow t.hopsSincePanic[p]) - 1$
≡ {Use math, rewrites}
$nrPanics(h + 1)$
>
$nrPanics(k \downarrow (h + 1)) - 1$
⇐{Use $(h + 1) \geq (k \downarrow (h + 1))$}
$nrPanics(h + 1)$
>
$nrPanics(h + 1) - 1$
⇐{Predicate calculus}
True

Action a = recvPanicCoPortalB(p), prove constant

Situation:
$s.location[p] \in \{P_0, P_1, P_3\}$
$t.location[p] = s.location[p]$
$s.location[coPortal(p)] = t.location[coPortal(p)]$
{by: effect and Inv6.2}
$t.location[coPortal(p)] = P_4$

$s.msgCoPortal[p] = PANIC$
$t.msgCoPortal[p] = \emptyset$
$s.msgCoPortal[coPortal(p)] = \emptyset$
$t.msgCoPortal[coPortal(p)] = \emptyset$

¬ half(s, p)
¬ half(t, p)
¬ half(s, coPortal(p))

¬ half(t, coPortal(p))

¬ pass(s, p)
¬ pass(t, p)
pass(s, coPortal(p))
pass(t, coPortal(p))

$(\forall(q : \mathit{portals}) : \ q \in \mathit{bus}(s,p) : \ \mathit{busInPanic}(s,q))$
$(\forall(q : \mathit{portals}) : \ q \in \mathit{bus}(t,p) : \ \mathit{busInPanic}(t,q))$

Proof:
comp2(s) $\geq$ comp2(t)
≡{Expand comp2}
comp2_half(s, $\{q \mid q \in \mathit{portals}\}$) + comp2_pass($s$, $\{q \mid q \in \mathit{portals}\}$)
$\geq$
comp2_half(t, $\{q \mid q \in \mathit{portals}\}$) + comp2_pass($t$, $\{q \mid q \in \mathit{portals}\}$)
⇐{Apply lemma comp2.1 and lemma comp2.2}
comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
+
comp2_pass(s, $\mathit{bus}(s,p)$) + comp2_pass(s, {coPortal(p)})
$\geq$
comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, $\mathit{bus}(t,p)$) + comp2_pass(t, {coPortal(p)})
⇐{Use situation, domain split on p}
comp2_pass(s, ($\mathit{bus}(s,p)$ - $\{p\}$))
$\geq$
comp2_pass(t, ($\mathit{bus}(t,p)$ - $\{p\}$))
⇐{Predicate calculus, use lemma comp2.3}
True

Action a = recvPanicCoPortalC(p), prove constant

Situation:
$s.location[p] = P_4$
$t.location[p] = N$
{by: precondition and Inv5}
$s.location[coPortal(p)] \neq P_4$
$t.location[coPortal(p)] = s.location[coPortal(p)]$

$s.msgCoPortal[p] = PANIC$
$t.msgCoPortal[p] = \emptyset$
$s.msgCoPortal[coPortal(p)] = \emptyset$
$t.msgCoPortal[coPortal(p)] = \emptyset$

¬ half(s, p)
¬ half(t, p)
half(s, coPortal(p)) = half(t, coPortal(p))

pass(s, p) = pass(t, p)
pass(s, coPortal(p)) = pass(t, coPortal(p))

$(\forall(q : portals) :\ q \in bus(s,p) :\ busInPanic(s,q))$

Rewrites:
$s.hopsSincePanic[p] = h$
$(\downarrow (r : portals) :\ r \in (bus(q) - \{p\}) \wedge s.location[r] \in \{P_0, P_1\} :$
$s.hopsSincePanic[r])$ = k

Proof:
comp2(s) $\geq$ comp2(t)
≡{Expand comp2}
comp2_half(s, $\{q \mid q \in portals\}$) + comp2_pass(s, $\{q \mid q \in portals\}$)
$\geq$
comp2_half(t, $\{q \mid q \in portals\}$) + comp2_pass(t, $\{q \mid q \in portals\}$)
⇐{Apply lemma comp2.1 and lemma comp2.2}
comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
+
comp2_pass(s, $bus(s,p)$) + comp2_pass(s, {coPortal(p)})
$\geq$
comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, $bus(s,p)$) + comp2_pass(t, {coPortal(p)})
⇐{Use situation, domain split on p}
comp2_pass(s, ($bus(s,p)$ - $\{p\}$))
$\geq$
comp2_pass(t, ($bus(s,p)$ - $\{p\}$))
⇐{Predicate calculus, use busInPanic(t, q)}
True

Action a = recvPanicBus(p), prove decreasing

Situation:
$s.location[p] = N$
$t.location[p] = P_1$
$s.location[coPortal(p)] = t.location[coPortal(p)]$

$s.msgCoPortal[p] = t.msgCoPortal[p]$

$s.msgCoPortal[coPortal(p)] = t.msgCoPortal[coPortal(p)]$

$t.hopsSincePanic[p] = (\downarrow\ (r : portals) :\ r \in bus(p) \wedge t.location[r] \in \{P_0, P_1\} :$
$t.hopsSincePanic[r])$

$\neg$ half(s, p)
half(t, p) $\equiv (t.location[coPortal(p)] = N \wedge t.msgCoPortal[p] = \emptyset)$
$\vee\ (t.msgCoPortal[coPortal(p)] = PANIC)$
half(s, coPortal(p)) $\equiv (s.location[p] = N \wedge s.msgCoPortal[coPortal(p)] = \emptyset)$
$\vee\ (s.location[coPortal(p)] \notin \{N, P_4\} \wedge s.msgCoPortal[p] = PANIC)$
half(t, coPortal(p)) $\equiv t.location[coPortal(p)] \notin \{N, P_4\}$
$\wedge\ t.msgCoPortal[p] = PANIC$

pass(s, p) $\equiv s.location[coPortal(p)] = N$
$\vee\ (s.location[coPortal(p)] = P_4 \wedge s.msgCoPortal[coPortal(p)] = PANIC)$
$\neg$ pass(t, p)
pass(s, coPortal(p)) $\equiv s.location[coPortal(p)] \in \{N, P_3, P_4\}$
pass(t, coPortal(p)) $\equiv t.location[coPortal(p)] = P_3$
$\vee(t.location[coPortal(p)] = P_4 \wedge t.msgCoPortal[coPortal(p)] = \emptyset)$

$(\forall(q : portals) :\ q \in (bus(s, p) - \{p\}) :\ busInPanic(s, q))$
$(\forall(q : portals) :\ q \in (bus(t, p) - \{p\}) :\ busInPanic(t, q))$

Rewrites:
$(\downarrow\ (r : portals) :\ r \in bus(s, p) \wedge s.location[r] \in \{P_0, P_1\} :\ s.hopsSincePanic[r]) = \mathrm{h}$
$(\downarrow\ (r : portals) :\ r \in bus(t, p) \wedge s.location[r] \in \{P_0, P_1\} :\ t.hopsSincePanic[r]) = \mathrm{h}$

Proof:
comp2(s) > comp2(t)
$\equiv$ {Expand comp2}
comp2_half(s, $\{q \mid q \in portals\}$) + comp2_pass(s, $\{q \mid q \in portals\}$)
>
comp2_half(t, $\{q \mid q \in portals\}$) + comp2_pass(t, $\{q \mid q \in portals\}$)
$\Leftarrow$ {Apply lemma comp2.1 and lemma copm2.2, domain split on p}
comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
+
comp2_pass(s, ($bus(s, p)$ - $\{p\}$)) + comp2_pass(s, $\{p\}$)
+
comp2_pass(s, {coPortal(p)})
>
comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, ($bus(t, p)$ - $\{p\}$)) + comp2_pass(t, $\{p\}$)
+
comp2_pass(t, {coPortal(p)})
$\Leftarrow$ {Use situation, lemma comp2.3}

comp2_half(s, {coPortal(p)})
+
comp2_pass(s, {p}) + comp2_pass(s, {coPortal(p)})
>
comp2_half(t, {p}) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, {coPortal(p)})
⇐{Case distinctions 1 to 5, using situation and Inv6.1, Inv7, Inv8, Inv13}
1) case $s.msgCoPortal[coPortal(p)] = PANIC$, by inv8 $s.location[coPortal(p)] = P_4$
{half_vector = {F,T,F,F}, pass_vector = {T,F,T,F}}
comp2_pass(s, {p}) + comp2_pass(s, {coPortal(p)})
>
comp2_half(t, {p})

2) case $s.msgCoPortal[coPortal(p)] = \emptyset \wedge s.location[coPortal(p)] = N$
{half_vector = {F,T,T,F}, pass_vector = {T,F,T,F}}
comp2_pass(s, {p}) + comp2_pass(s, {coPortal(p)})
>
comp2_half(t, {p})

3) case $s.msgCoPortal[coPortal(p)] = \emptyset \wedge s.location[coPortal(p)] = P_4$
{half_vector = {F,F,T,F}, pass_vector = {F,F,T,T}}
comp2_half(s, {coPortal(p)})
+
comp2_pass(s, {coPortal(p)})
>
comp2_pass(t, {coPortal(p)})

4) case by inv13 $s.msgCoPortal[coPortal(p)] = \emptyset$, $s.location[coPortal(p)] \in \{P_0, P_1\}$,
{half_vector = {F,F,T,F}, pass_vector = {F,F,F,F}, MaxNode ≥ 2}
comp2_half(s, {coPortal(p)})
>
0

5) case by inv13 $s.msgCoPortal[coPortal(p)] = \emptyset$, $s.location[coPortal(p)] = P_3$,
by inv6.1 $s.msgCoPortal[p] = \emptyset$
{half_vector = {F,F,T,F}, pass_vector = {F,F,T,T}, MaxNode ≥ 2}
comp2_half(s, {coPortal(p)})
+
comp2_pass(s, {coPortal(p)})
>
comp2_pass(t, {coPortal(p)})
⇐{Use math, predicate calculus}
2) case $s.location[coPortal(p)] = N \wedge s.msgCoPortal[coPortal(p)] = \emptyset$
{half_vector = {F,T,T,F}, pass_vector = {T,F,T,F}}
comp2_pass(s, {p}) + comp2_pass(s, {coPortal(p)})

$>$
comp2_half(t, $\{p\}$)

3) case $s.location[coPortal(p)] \in \{N, P_4\} \wedge s.msgCoPortal[coPortal(p)] = PANIC$
{half_vector = {F,T,F,F}, pass_vector = {T,F,T,F}}
comp2_pass(s, $\{p\}$) + comp2_pass(s, {coPortal(p)})
$>$
comp2_half(t, $\{p\}$)
$\Leftarrow$ {Use lemma comp2.4}
True

Action a = recvMsgBus(p), prove constant

Situation:
$s.location[p] \in \{P_0, P_1, P_3, P_4\}$
$t.location[p] = s.location[p]$
$t.location[coPortal(p)] = s.location[coPortal(p)]$

$s.msgCoPortal[p] = t.msgCoPortal[p]$
$s.msgCoPortal[coPortal(p)] = t.msgCoPortal[coPortal(p)]$

half(t, p) = half(s, p)
half(t, $coPortal(p)$) = half(s, $coPortal(p)$)

pass(s, p) = pass(t, p)
pass(t, $coPortal(p)$) = pass(s, $coPortal(p)$)

Proof:
comp2(s) $\geq$ comp2(t)
$\Leftarrow$ {Use lemma comp2.9}
True

Action a = sendPanicBus(p), prove constant

Situation:
$s.location[p] = P_0$
$t.location[p] = P_1$
$t.location[coPortal(p)] = s.location[coPortal(p)]$

$s.msgCoPortal[p] = t.msgCoPortal[p]$
$s.msgCoPortal[coPortal(p)] = t.msgCoPortal[coPortal(p)]$

half(t, p) = half(s, p)

half(t, coPortal(p)) = half(s, coPortal(p))

pass(s, p) = pass(t, p)
pass(t, coPortal(p)) = pass(s, coPortal(p))

Proof:
comp2(s) $\geq$ comp2(t)
$\Leftarrow$ {Use lemma comp2.9}
True

Action a = sendPanicCoPortal(p), prove constant

Situation:
$s.location[p] = P_3$
$t.location[p] = P_4$
$s.location[coPortal(p)] = t.location[coPortal(p)]$
{by: effect and Inv5}
$s.location[coPortal(p)] \neq P_4$

$s.msgCoPortal[p] = \emptyset$
$t.msgCoPortal[p] = \emptyset$
$s.msgCoPortal[coPortal(p)] = \emptyset$
$t.msgCoPortal[coPortal(p)] = PANIC$

half(t, p) = half(s, p)
half(t, coPortal(p)) = half(s, coPortal(p))

pass(s, p) = pass(t, p)
pass(t, coPortal(p)) = pass(s, coPortal(p))

Proof:
comp2(s) $\geq$ comp2(t)
$\Leftarrow$ {Use lemma comp2.9}
True

Action a = finish(p), prove constant

Situation:
$s.location[p] = P_3$
$t.location[p] = N$
{by: precondition and Inv6.2}
$s.location[coPortal(p)] = P_4$

$t.location[coPortal(p)] = t.location[coPortal(p)]$

$s.msgCoPortal[p] = \emptyset$
$t.msgCoPortal[p] = \emptyset$
$s.msgCoPortal[coPortal(p)] = \emptyset$
$t.msgCoPortal[coPortal(p)] = PANIC$

$\neg$ half(s, p)
$\neg$ half(t, p)
$\neg$ half(s, coPortal(p))
$\neg$ half(t, coPortal(p))

pass(s, p)
$\neg$ pass(t, p)
pass(s, coPortal(p))
pass(t, coPortal(p))

Proof:

comp2(s) $\geq$ comp2(t)
$\equiv$ {Expand comp2}
comp2_half(s, $\{q \mid q \in portals\}$) + comp2_pass(s, $\{q \mid q \in portals\}$)
$\geq$
comp2_half(t, $\{q \mid q \in portals\}$) + comp2_pass(t, $\{q \mid q \in portals\}$)
$\Leftarrow$ {Apply lemma comp2.1, lemma comp2.2 and domain split on p}
comp2_half(s, $\{p\}$) + comp2_half(s, {coPortal(p)})
+
comp2_pass(s, ($bus(s,p)$ - $\{p\}$)) + comp2_pass(s, $\{p\}$) + comp2_pass(s, {coPortal(p)})
$\geq$
comp2_half(t, $\{p\}$) + comp2_half(t, {coPortal(p)})
+
comp2_pass(t, ($bus(t,p)$ - $\{p\}$)) + comp2_pass(t, $\{p\}$) + comp2_pass(t, {coPortal(p)})
$\Leftarrow$ {Use situation and domain split on p}
comp2_pass(s, ($bus(s,p)$ - $\{p\}$)) + comp2_pass(s, $\{p\}$)
$\geq$
comp2_pass(t, ($bus(t,p)$ - $\{p\}$))
$\Leftarrow$ {Use lemma comp2.3, assume busInPanic(t, p)}
comp2_pass(s, $\{p\}$)
$\geq$
0
$\Leftarrow$ {Predicate calculus}
True

Action = reset(P), prove constant

Situation:

$(\forall(p:\ portals):\ p \in P \wedge s.location[p] \in \{P_0, P_1\}:$
$\quad t.location[p] = ((\exists(q:\ portals):\ q \in P:\ s.location[q] = N)\ ?\ P_0:\ P_3))$

$(\forall(p:\ portals):\ p \in P \wedge s.location[p] \notin \{P_0, P_1\}:$
$\quad t.location[p] = s.location[p])$

Proof:

$\quad$ comp2$(s) \geq$ comp2(t)
$\equiv$ {Expand comp2}
$\quad$ comp2_half$(s, \{q \mid q \in portals\})$ + comp2_pass$(s, \{q \mid q \in portals\})$
$\geq$
$\quad$ comp2_half$(t, \{q \mid q \in portals\})$ + comp2_pass$(t, \{q \mid q \in portals\})$
$\Leftarrow$ {Domain split on bus P, use lemma comp2.6 and comp2.7}
$\quad$ comp2_half(s, P) + comp2_pass(s, P)
$\geq$
$\quad$ comp2_half(t, P) + comp2_pass(t, P)
$\Leftarrow$ {Use situation and lemma comp2.8}
$\quad$ comp2_pass(s, P)
$\geq$
$\quad$ comp2_pass(t, P)
$\Leftarrow$ {Domain split on portal locations}
$\quad$ comp2_pass$(s, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$+$
$\quad$ comp2_pass$(s, \{q \mid q \in P \wedge s.location[q] \notin \{P_0, P_1\}\})$
$\geq$
$\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$+$
$\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \notin \{P_0, P_1\}\})$
$\Leftarrow$ {Use situation}
$\quad$ comp2_pass$(s, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$\geq$
$\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$\Leftarrow$ {Case distinction on $(\exists(q : portals):\ q \in P:\ s.location[q] = N)$}
$\quad$ {case1: $(\exists(q : portals):\ q \in P:\ s.location[q] = N)$}
$\quad\quad$ comp2_pass$(s, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$\quad\quad \geq$
$\quad\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\} \wedge t.location[q] = P_0\})$

$\quad$ {case2: $\{(\forall(q : portals):\ q \in P:\ s.location[q] \neq N)\}$}
$\quad\quad$ comp2_pass$(s, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\}\})$
$\quad\quad \geq$
$\quad\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\} \wedge t.location[q] = P_3\})$
$\Leftarrow$ {Use $s.location[q] \in \{P_0, P_1\} \Rightarrow \neg pass(s, q)$}
$\quad$ 0
$\geq$
$\quad$ comp2_pass$(t, \{q \mid q \in P \wedge s.location[q] \in \{P_0, P_1\} \wedge t.location[q] = P_3\})$

$\Leftarrow$\{Use $t.location[q] = P_3 \Rightarrow \neg busInPanic(t, q)$\}
  0
$\geq$
  0
$\Leftarrow$\{Predicate calculus\}
  True

Component 3

comp3(s: state) =
  $(\sum(q : portals) ::\ distance(s.location[q]))$

Functions

distance(s: state, p: portal)
$(\forall i :\ 0 \leq i \leq 4 :$
  $(s.location[p] = P_i?(5 - i) : 0))$

Lemma comp3.1

$(\forall(p : portal, s, t : state, a : action) : s \overset{a(p)}{\rightarrow} t :$
  $(\sum(q : portals) :\ q \neq p :\ distance(s, q))$
=
  $(\sum(q : portals) :\ q \neq p :\ distance(t, q))$

Proof of Component 3

According to the table, we have to prove that:

- Component 3 decreases for action $a \in$ {recvPanicCoPortalC, sendPanicCoPortal, finish, sendPanicBus, reset}
- Component 3 remains constant for action $a \in$ {recvPanicCoPortalB, recvMsgBus}

Note that reset is decreasing under the assumption of strong fairness, see Section 5.5

Action $a \in$ {recvPanicCoPortalB(p), recvMsgBus(p)}, prove constant

situation:
$s.location[p] = t.location[p]$

proof:
comp3(s) $\geq$ comp3(t)
$\equiv$ {Expand comp3}
$(\sum(q : portals) :: distance(s, q))$
$\geq$
$(\sum(q : portals) :: distance(t, q))$
$\Rightarrow$ {Domain split on p}
$(\sum(q : portals) : q \neq p : distance(s, q))$
$+$
$distance(s, p)$
$\geq$
$(\sum(q : portals) : q \neq p : distance(t, q))$
$+$
$distance(t, p)$
$\Rightarrow$ {Use lemma comp3.1, situation}
True

Action $a =$ recvPanicCoPortalC(p), prove decreasing

situation:
$s.location[p] = P_4$
$t.location[p] = N$

proof:
comp3(s) $>$ comp3(t)
$\equiv$ {Expand comp3}
$(\sum(q : portals) :: distance(s, q))$
$>$
$(\sum(q : portals) :: distance(t, q))$
$\Rightarrow$ {Domain split on p}
$(\sum(q : portals) : q \neq p : distance(s, q))$
$+$
$distance(s, p)$
$>$
$(\sum(q : portals) : q \neq p : distance(t, q))$
$+$
$distance(t, p)$
$\Rightarrow$ {Use lemma comp3.1, math}
$distance(s, p)$
$>$

$distance(t, p)$
$\equiv$ {Expand distance using situation}
$1 > 0$
$\Rightarrow$ {Predicate calculus}
True

Action a = sendPanicCoPortal(p), prove decreasing

situation:
$s.location[p] = P_3$
$t.location[p] = P_4$

proof:
comp3(s) > comp3(t)
$\equiv$ {Expand comp3}
$(\sum(q : portals) :: \ distance(s, q))$
$>$
$(\sum(q : portals) :: \ distance(s, q))$
$\Rightarrow$ {Domain split on p}
$(\sum(q : portals) : \ q \neq p : \ distance(s, q))$
$+$
$distance(s, p)$
$>$
$(\sum(q : portals) : \ q \neq p : \ distance(t, q))$
$+$
$distance(t, p)$
$\Rightarrow$ {Use lemma comp3.1, math}
$distance(s, p)$
$>$
$distance(t, p)$
$\equiv$ {Expand distance using situation}
$2 > 1$
$\Rightarrow$ {Predicate calculus}
True

Action a = finish(p), prove decreasing

situation:
$s.location[p] = P_3$
$t.location[p] = N$

proof:
comp3(s) > comp3(t)
$\equiv$ {Expand comp3}

$(\sum(q : portals) :: distance(s, q))$
$>$
$(\sum(q : portals) :: distance(t, q))$
$\Rightarrow$ {Domain split on p}
$(\sum(q : portals) : q \neq p : distance(s, q))$
$+$
$distance(s, p)$
$>$
$(\sum(q : portals) : q \neq p : distance(t, q))$
$+$
$distance(t, p)$
$\Rightarrow$ {Use lemma comp3.1, math}
$distance(s, p)$
$>$
$distance(t, p)$
$\equiv$ {Expand distance using situation}
$2 > 0$
$\Rightarrow$ {Predicate calculus}
True

Action a = sendPanicBus(p, P), prove decreasing

situation:
$s.location[p] = P_0$
$t.location[p] = P_1$

proof:
comp3(s) $>$ comp3(t)
$\equiv$ {Expand comp3}
$(\sum(q : portals) :: distance(s, q))$
$>$
$(\sum(q : portals) :: distance(t, q))$
$\Rightarrow$ {Domain split on p}
$(\sum(q : portals) : q \neq p : distance(s, q))$
$+$
$distance(s, p)$
$>$
$(\sum(q : portals) : q \neq p : distance(t, q))$
$+$
$distance(t, p)$
$\Rightarrow$ {Use lemma comp3.1, math}
$distance(s, p)$
$>$
$distance(t, p)$
$\equiv$ {Expand distance using situation}

$5 > 4$
$\Rightarrow$ {Predicate calculus}
True

Action = reset(P)

Decreasing by strong fairness:
case 1:
$(\exists(p : portals) : p \in P : s.location[p] \in \{P_0, P_1\}) \wedge (\exists(p : portals) :$
$p \in P : s.location[p] = N)$
$\Rightarrow comp3(t) = comp3(s) + (\#(p : portals) : p \in P : s.loc[p] = P_1)$
case 2:
$(\exists(p : portals) : p \in P : s.location[p] \in \{P_0, P_1\}) \wedge (\forall(p : portals) : p \in P :$
$s.location[p] \neq N)$
$\Rightarrow comp3(t) < com3(s)$
case 3:
$(\forall(p : portals) : p \in P : s.location[p] \notin \{P_0, P_1\})$
$\Rightarrow comp3(t) = comp3(s)$

Note that in case 1 and 3 the norm does not decrease. In case 1 either SFset1 or SFset2 was enabled in s before the reset(P) action occurred and in case 3 either comp3(s,P) = 0 or another action is enabled (see also the proof for Property 2 (no deadlock C.3.2). The above case distinction is used in the termination proof of the norm, based on strong fairness assumptions, which is in Chapter 7.5.1

Component 4

comp4(s: state) =
$(\#(q : portals) : s.location[q] \neq N : \neg s.adjBusPanicComplete[q])$

Lemma comp4.1

$(\forall(p : portal, s, t : state, a : action) : s \overset{a(p)}{\rightarrow} t :$
$(\#(q : portals) : q \neq p \wedge s.location[q] \neq N : \neg s.adjBusPanicComplete[q])$
$=$
$(\#(q : portals) : q \neq p \wedge t.location[q] \neq N : \neg t.adjBusPanicComplete[q])$

Proof of Component 4

According to the table, we have to prove that:

- Component 4 decreases for action a = recvPanicCoPortalb
- Component 4 remains constant for action a = recvMsgBus

Action a = recvMsgBus(p), prove constant

situation:
$s.location[p] \neq N$
$t.location[p] = s.location[p]$

$s.adjBusPanicComplete[p] = t.adjBusPanicComplete[p]$

rewrites:
$(s.adjBusPanicComplete[p]?1 : 0) = x$
$(t.adjBusPanicComplete[p]?1 : 0) = x$

proof:
comp4(s) $\geq$ comp4(t)
$\equiv$ {Expand comp4}
$(\#(q : portals) : \ s.location[q] \neq N : \ \neg s.adjBusPanicComplete[q])$
$\geq$
$(\#(q : portals) : \ t.location[q] \neq N : \ \neg t.adjBusPanicComplete[q])$
$\Rightarrow$ {Domain split on p, use rewrites and situation}
$(\#(q : portals) : \ q \neq p \wedge s.location[q] \neq N : \neg s.adjBusPanicComplete[q])$
$+$
x
$\geq$
$(\#(q : portals) : \ q \neq p \wedge t.location[q] \neq N : \neg t.adjBusPanicComplete[q])$
$+$
x
$\Rightarrow$ {Use lemma comp4.1, predicate calculus}
True

Action = recvPanicCoPortalB(p, P), prove decreasing

situation:
$s.location[p] \neq N$
$t.location[p] = s.location[p]$

$\neg$ $s.adjBusPanicComplete[p]$
$t.adjBusPanicComplete[p]$

proof:
comp4(s) $>$ comp4(t)
$\equiv$ {Expand comp4}

$$
\begin{array}{l}
\quad (\#(q : portals) :\ s.location[q] \neq N :\ \neg s.adjBusPanicComplete[q]) \\
> \\
\quad (\#(q : portals) :\ t.location[q] \neq N :\ \neg t.adjBusPanicComplete[q]) \\
\Rightarrow \{\text{Domain split on } p\} \\
\quad (\#(q : portals) :\ q \neq p \wedge s.location[q] \neq N : \neg s.adjBusPanicComplete[q]) \\
+ \\
\quad 1 \\
> \\
\quad (\#(q : portals) :\ q \neq p \wedge t.location[q] \neq N : \neg t.adjBusPanicComplete[q]) \\
+ \\
\quad 0 \\
\Rightarrow \{\text{Use lemma comp4.1, predicate calculus}\} \\
\quad True
\end{array}
$$

Component 5

$$
\begin{array}{l}
\text{comp5(s: state)} = \\
\quad (\#(q : portals) ::\ s.msgBus[p] \neq \emptyset)
\end{array}
$$

Lemma comp5.1

$$
\begin{array}{l}
(\forall (p : portal, s, t : state, a : action) : s \overset{a(p)}{\rightarrow} t : \\
\quad (\#(q : portals) :\ q \neq p :\ s.msgBus[p] \neq \emptyset) \\
= \\
\quad (\#(q : portals) :\ q \neq p :\ s.msgBus[p] \neq \emptyset)
\end{array}
$$

Proof of Component 5

According to the table, we have to prove that:

- Component 5 decreases for action a = recvMsgBus

Action a = recvMsgBus(p)

situation:
$s.msgBus[p] \neq \emptyset$
$t.msgBus[p] = \emptyset$

proof:
comp5(s) > comp5(t)

$\equiv$ {Expand comp5}
$(\#(q : portals) :: s.msgBus[p] \neq \emptyset)$
$>$
$(\#(q : portals) :: t.msgBus[p] \neq \emptyset)$
$\Rightarrow$ {Domain split on p}
$(\#(q : portals) : q \neq p : s.msgBus[p] \neq \emptyset) + 1$
$>$
$(\#(q : portals) : q \neq p : t.msgBus[p] \neq \emptyset) + 0$
$\Rightarrow$ {Use lemma comp5.1, predicate calculus}
True

Appendix D

PVS Proof of graph_from_edges

This appendix details the work done on the graph_from_edges PVS theory from NASA Langley. This theory is part of the larger graph library. The theory, as its name suggests, creates a graph from a given set of edges. A graph consists of a node set and an edge set pair. Both sets are of the type finiteset.

This theory was very interesting for the construction of the topology theory. The topology graph consisted of two edge sets, one for the bridges and one for the buses. The total graph could be constructed by taking the union of both edge sets and using this union on the graph_from_edges theory.

Though, this theory looked promising it was not active in the graph library. Upon further inspection the graph_from_edges theory had one unproved TCC. This TCC was generated because the node set was retrieved with a predicate subtype. Since the node set was declared as a finite PVS, generated a TCC to verify that the predicate subtype resulted in a finite set.

As a sketch, since there are only a finite number N of edges there can only be a maximum of 2N nodes. This argument in a hand proof would normally be enough to convince a reader that the resulting set is finite. In PVS however, it is a rather difficult proof.

A finite set in the PVS prelude is defined as an injection from a set to a certain maximum natural number. The TCC generated had only one consequent that stated that the predicate subtype had to be a finite set. With just this consequent and the seaming impossibility to extract the predicate from the sub-

type the only possibility left to complete the proof in its current form was to construct a new injective function.

Since there was no information nor experts available on how to actually construct such an injective function, a different approach was taken. The actual predicate subtype looked a lot like the big union of sets. The edges are a set of exactly two items; doubletons. Using the big union to build the set of nodes has the advantage that a theory for proving the finiteness of the result of the big union is available.

After making the change it became clear why a predicate subtype was chosen. A new TCC was generated to prove a relation between the big union and the original predicate. This time however, the predicate was not contained in a subtype and the TCC was provable.