Report Number: WUCS-92-44

1992-12-01

Formal Specifications and Design of a Message Router

Christian Creveull and Gruia-Catalin Roman

Formal derivation refers to a family of design techniques that entail the development of programs which are guaranteed to be correct by construction. This paper investigates the possible application of one such technique-- UNITY-style specification refinement-- to industrial-grade problems. The formal specification and design of a message router illustrates the derivation process and helps identify those methodological elements that are likely to contribute to successful use of this technique in industrial environment. Although, the message router cannot be characterized as being industrial-grade, it is a sophisticated problem that pose significant specification and design challenges-- its apparent simplicity is rather... Read complete abstract on page 2.

Formal Specifications and Design of a Message Router

Christian Creveull and Gruia-Catalin Roman

Complete Abstract:

Formal derivation refers to a family of design techniques that entail the development of programs which are guaranteed to be correct by construction. This paper investigates the possible application of one such technique-- UNITY-style specification refinement-- to industrial-grade problems. The formal specification and design of a message router illustrates the derivation process and helps identify those methodological elements that are likely to contribute to successful use of this technique in industrial environment. Although, the message router cannot be characterized as being industrial-grade, it is a sophisticated problem that pose significant specification and design challenges-- its apparent simplicity is rather deceiving, The main body of the paper consists of a complete formal specification of the router and a series of successive refinements that eventually lead to a trivial construction of a correct UNITY program. Each refinement is accompanied by its design rational and is explained both formally (proofs being included in an appendix) and informally, in a manner accessible to a broad audience. We use this example to make the case that program derivation provides a good basis for introducing rigor in the design strategy, regardless of the degrees of formality one is willing to consider.

Formal Specification and Design of a Message Router

Christian Creveuil
Gruia-Catalin Roman

WUCS-92-44

December 1992

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Printed 12/29/92

Abstract

Formal derivation refers to a family of design techniques that entail the development of programs which are guaranteed to be correct by construction. This paper investigates the possible application of one such technique—UNITY-style specification refinement—to industrial-grade problems. The formal specification and design of a message router illustrates the derivation process and helps identify those methodological elements that are likely to contribute to successful use of this technique in an industrial environment. Although, the message router cannot be characterized as being industrial-grade, it is a sophisticated problem that poses significant specification and design challenges—its apparent simplicity is rather deceiving. The main body of the paper consists of a complete formal specification of the router and a series of successive refinements that eventually lead to a trivial construction of a correct UNITY program. Each refinement is accompanied by its design rational and is explained both formally (proofs being included in an appendix) and informally, in a manner accessible to a broad audience. We use this example to make the case that program derivation provides a good basis for introducing rigor in the design strategy, regardless of the degrees of formality one is willing to consider.

Acknowledgments: The first author was supported by the Institut National de la Recherche en Informatique et en Automatique under a postdoctoral grant. The second author was supported in part by the National Science Foundation under the Grant CCR-9015677. The government has certain rights in this material.

The authors thank H. C. Cunningham, K. C. Cox, and J. Y. Plun for their reviews of the manuscript.

Correspondence: All communications regarding this paper should be addressed to

Dr. Gruia-Catalin Roman
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

office: (314) 935-6190
secretary: (314) 935-6160
fax: (314) 935-7302

roman@cs.wustl.edu

1. Introduction

Increasing demands for reliable performance provide a strong impetus for the software engineering community to evaluate and adopt formal methods. Formal notations led to the development of specification languages; formal verification contributed to the application of mechanical theorem provers to program checking; and formal derivation—a class of techniques that ensure correctness by construction—has the potential to reshape the way software will be developed in the future. Program derivation is less costly than *post-factum* verification, is incremental in nature, and can be applied with varying degrees of rigor in conjunction with or completely apart from program verification. More significantly, while verification is tied to analysis and support tools, program derivation deals with the very essence of the design process, the way one thinks about problems and constructs solutions.

In sequential programming, formal derivation enjoys a long standing and prestigious tradition [4, 5, 7, 8, 15, 16]. By contrast, derivation is a relatively new concern in concurrent programming. Although a clean and comprehensive characterization of the field is difficult to make and is beyond the scope of this paper, three general directions seem to have emerged in the concurrency arena. *Constructivist approaches* start with simple components having known properties and combine them into larger ones whose properties may be computed. CSP-related efforts [6, 9, 11, 12] appear to favor this approach in part due to the algebraic mindset that characterizes the work on abstract CSP. *Specification refinement* has been advocated strongly in the work on UNITY [2, 10, 20]. An initial highly-abstract specification is gradually refined up to the point when it contains so much detail that writing a correct program becomes trivial. *Program refinement* uses a correct program as starting point and alters it until a new program satisfying some additional desired properties is produced. In some of the work on action systems [1], for instance, sequential programs are transformed into concurrent or distributed ones. Mixed specification and program refinement [19] has been used in conjunction with the Swarm model and its proof logic [3, 17, 18].

Encouraged by these recent developments, it is reasonable to pose the question whether program derivation is a viable substitute for current, mostly ad-hoc, methods employed by concurrent system designers. With this aim in mind, our research group has embarked on a number of case studies whose immediate objective is to develop an understanding of how program derivation may be applied to industrial-grade problems. The emphasis is not on tool development but on identifying a design style and associated skills that can be taught effectively and applied productively. We want to show that, given the right model and heuristics, program derivation can be made simple to the point that it can be explained even to the non-specialist.

The message router is one of the program derivation exercises we carried out recently. In the simplest terms, the message router is a device which accepts messages on a number of input lines and delivers them to its output lines. Each message consists of a header, one or more body packets, and a tail. The header contains the packet destination. Packet ordering within a message is preserved, packets belonging to different messages are not interleaved, and messages from the same input line going to the same output line are not reordered. We chose this problem due to its deceiving simplicity. Anyone can understand the basic problem but its formal treatment requires careful attention to detail.

The body of this paper consists of a formal specification of the router problem introduced in Section 2 and a UNITY-style specification refinement process developed in Section 3. The UNITY program is derived from the final specification in Section 4, and Section 5 summarizes the lessons we have learned from this exercise. Appendices provide technical details regarding the proof logic, and the proofs of the refinement steps.

2. Specification of the Router Problem

We first give a brief overview of the UNITY logic, then informally describe the router problem, and finally present the formal specification.

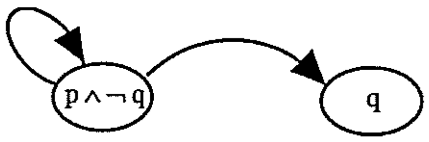
2.1. UNITY Logic

The description we give in this paragraph is intentionally informal and intuitive, in order to allow non-specialist readers to understand the router specification and design. A more formal description may be found in appendix A.

Before presenting the logic, we need to say a few words about UNITY programs, and especially the way they are executed. A typical UNITY program consists of three sections: a *declare* section, which contains Pascal-style declarations of variables; an *initially* section, where all or some of the variables are initialized; an *assign* section, which is a set of multiple assignment statements. The execution of a UNITY program consists simply in repeating forever the following sequence: select at random a statement in the assign section, and execute it. The only constraint on the selection process is that each statement is chosen infinitely often. The semantics of a UNITY program can then be defined as a set of execution sequences. Each sequence begins with the initial state, as defined in the initially section, and each of the following states is obtained from the previous one by executing a statement.

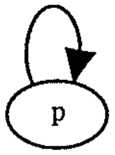
The UNITY logic is used both as a specification language and as a proof logic. It is composed of several unary and binary operators, which allow to define global properties over the execution sequences. Let p and q be two predicates. The list of properties, with their intuitive definition, is given below. Let us note that the program we refer to in the remaining of the paragraph is either the program we want to derive, if the logic is used as a specification language, or the program we want to verify, if the logic is used as a proof logic.

- p **unless** q: this property means that whenever predicate p holds for a program state, p continues to hold in the execution sequence at least until q holds. In other words, from a state verifying $p \wedge \neg q$, the execution can either stay in this state, or move to a state satisfying q.



Note that p ***unless*** q does not require q to hold; in such a case, p must hold forever.

- **stable** p: predicate p is a *stable* predicate if it remains true forever once it becomes true. This property is equivalent to p ***unless*** *false.*



- **const** p: predicate p is *constant* if both p and $\neg p$ are stable predicates. In other words, p remains true forever if it is initially true, and false forever if it is initially false.

- **inv** p: predicate p is *invariant* if it holds in the initial state, and is stable along any execution sequence.

The four properties described above are *safety* properties, in the sense that they prevent the occurrence of certain state transitions. For instance p ***unless*** q disallows the transition from $p \wedge \neg q$ to $\neg p \wedge \neg q$. However, to specify problems, we must also be able to state that some *progress* is made, i.e., that certain predicates hold at some point in the future. For this, the UNITY logic offers the following properties:

- p $\mapsto$ q: predicate p leads to predicate q if, from a state in which p holds, a state in which q holds is eventually reached.

- p **until** q: this property is slightly stronger than $p \mapsto q$, since it guarantees also that p holds at least until q holds. Its definition is: p ***unless*** $q \wedge p \mapsto q$.

- p **ensures** q: this last property is strongly related to the text of the program. It states that, whenever p holds, it must hold at least until q holds (p ***unless*** q), and that there must exist a statement in the program that establishes the truth of q. The ***ensures*** property is thus stronger than ***until***, since the truth of q in the ***until*** case can be established by the execution of more than one statement.

2.2. Description of the Problem

We consider a communication network that connects N senders of messages to M receivers via a message router. Each sender is connected to one of the input ports of the router, and each receiver to one of the output ports (Figure 1).

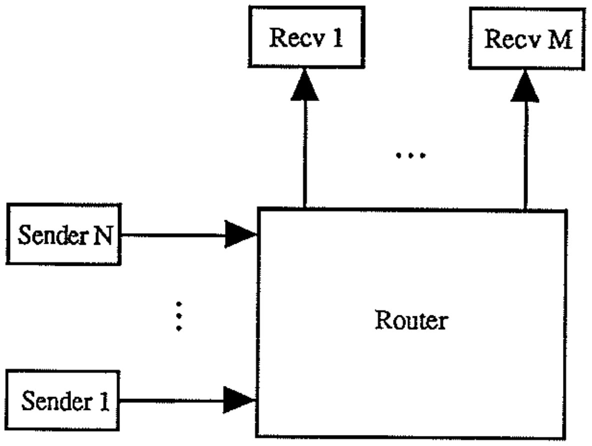


Figure 1. Structure of the system.

Each message is composed of a finite number of packets that can be of three different types: *header*, *body*, and *tail*. The header, which is the first packet of the message, contains the port address of the message destination. Each header is followed by one or more body packets which contain the actual data. Finally, the tail packet marks the end of the message.

The behavior of the router is defined by the following requirements:

- Each packet that is sent must eventually be delivered to the intended receiver.
- The value of the body packets must not be modified, but, for control purposes, the router may modify the value of the header and tail packets.
- Packet ordering within a message must be preserved.
- Messages from the same source going to the same destination must not be reordered.
- Messages from different sources going to the same destination must not be interleaved.

2.3. Formal Specification

The system we want to specify is composed of three interacting entities: an input environment (the N senders), an output environment (the M receivers), and the router. The UNITY formalism offers the possibility to specify those three parts separately, and then to compose the three specifications. However, this implies the use of conditional properties (for instance, assuming that property $P1$ is true in the input environment, property $P2$ is true in the router), which make reasoning and understanding more difficult. We believe it is easier to deal with a unique specification describing the behavior of the system in its entirety, without any interaction with the outside world.

To do so, we abstract the senders and the receivers as infinite input and output queues (Figure 2). Initially the input queues contain all the packets that have to be sent, and the output queues, as well as the router, are empty. Each packet in the input and output environments resides at some distinct location, which is a pair of coordinates *(row, column)*. For instance, packets in the input queues have a row coordinate that ranges from 1 to N, and a column coordinate that ranges from 0 to $-\infty$.

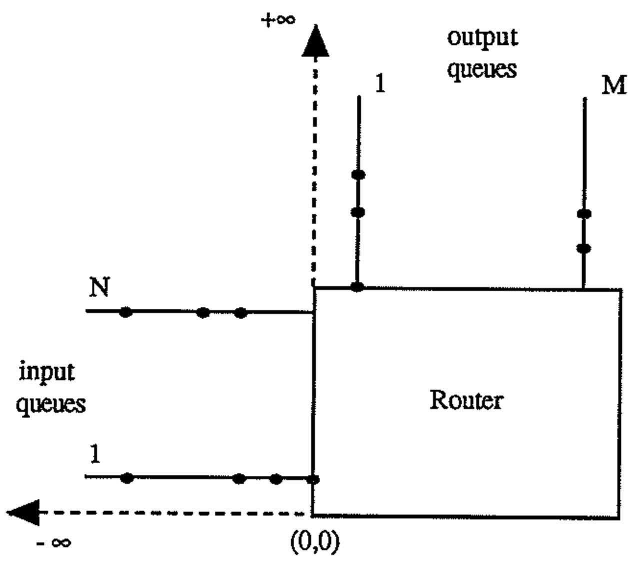


Figure 2. Abstract structure of the system.

Let us define some notation. Packets are designated by notation $\tau[v]$, where τ, the packet type, is equal to h, b, or t—header, body, or tail—and v, the packet value, belongs to some set V. Each packet is augmented with four auxiliary variables: n, m, i, and j. These auxiliary variables cannot be used by the program we want to derive, their only purpose is to make the specification and design process easier. Variables n and m, ranging respectively from 1 to N and from 1 to M, are the addresses of the sender and receiver of the packet (or, in other words, the numbers identifying the source input queue and destination output queue). Variable i is the number of the message the packet belongs to. We define the number of the first message sent by each sender to be equal to 1, the number of the second message to be equal to 2, and so on. Finally, variable j is the packet number, that is the position of the packet within the message. A packet augmented with the auxiliary variables is called a *logical* packet, as opposed to the *physical* packets carried by the network. We use notation $\tau(n,m,i,j)[v]$ to designate a logical packet. Note that the pair (n,i)—(sender number, message number)—uniquely identifies a message in the system, and that the triple (n,i,j)—(sender number, message number, packet number)—uniquely identifies a packet.

In the specification, the notation $\tau(n,m,i,j)[v]$ is never used. We rather designate logical packets by Greek letters (α, β, or δ), and access the packet attributes through several access functions. Function *type*, applied to a packet, returns the type of the packet (τ). Functions *src* and *dest* return the identity of the source input queue (n) and of the destination output queue (m). The message number (i) and packet number (j) are given by functions *mnr* and *pnr*. Finally, we use the functions *mid* and *pid* to access the message identifier (i.e., the pair (n,i)) and the packet identifier (i.e., the triple (n,i,j)). When it is necessary to deal explicitly with the value of a packet, we use the notation $\sigma[v]$, where σ represents $\tau(n,m,i,j)$.

The location of packets in the system is given by the function Π. Let A be the set of logical packets in the router and its environment. Let E be the set of locations in the environment:

$$E=\langle \mathbf{set}\ p,q : (1\le p\le N \wedge q\le 0) \vee (p\ge N+1 \wedge 1\le q\le M) :: (p,q)\rangle.[1]$$

1 This is an example of a *constructor*, a syntactic element which occurs frequently in UNITY notation. The general form of the constructor is:

Let *R* be the set of locations in the router (the value of *R* is unknown at this time). Function *Π* is then defined as follows:

$$\Pi : \Lambda \rightarrow E \cup R.$$

To make the specification easier to read, we use notation *α@l* to mean that packet *α* is at location *l*—i.e., *Π.α=l* (throughout the paper, the operator "." is used to denote function application). We sometimes add a type designator (*h*, *b*, or *t*) to the variables representing packets. For instance, $\alpha^h@l$ means that a header packet *α* is at location *l*: *α@l ∧ type.α=h*. To state the no-reordering and non-interleaving constraints, we define a relation ⊏ on packets. Its definition is:

$$\begin{aligned}\alpha \sqsubset \beta \Leftrightarrow\ & \langle \exists p,q,q' : q<q'\le 0 :: \alpha@(p,q) \wedge \beta@(p,q') \rangle \vee \\ & \langle \exists p,p',q : N+1\le p<p' :: \alpha@(p,q) \wedge \beta@(p',q) \wedge src.\alpha=src.\beta \rangle \vee \\ & \langle \exists p,p',q,q' : q'\le 0 \wedge p'\ge N+1 :: \alpha@(p,q') \wedge \beta@(p',q) \wedge src.\alpha=src.\beta \rangle.\end{aligned}$$

That is, *β* is ahead of *α* according to ⊏ iff *β* is in front of *α* in the same input queue, or *β* is in front of *α* in the same output queue and both packets come from the same source, or *β* is in the output environment, *α* is in the input environment, and both packets have the same source.

The top-level specification S_0 is given below. Explanations follow the specification. Any free variable appearing in the UNITY assertions is assumed to be universally quantified over all the elements of its domain.

Message Representation

$$\textbf{inv}\ \langle \exists l :: \alpha@l \rangle \wedge k=\langle \Sigma\ \beta,l : \beta@l \wedge mid.\beta=mid.\alpha :: 1 \rangle \Rightarrow k\ge 3 \wedge (pnr.\alpha=1 \Leftrightarrow type.\alpha=h) \wedge (1<pnr.\alpha<k \Leftrightarrow type.\alpha=b) \wedge (pnr.\alpha=k \Leftrightarrow type.\alpha=t) \quad \text{(P1)}$$

$$\textbf{const}\ \langle \exists v,l :: \sigma[v]^h@l \rangle \quad \text{(P2)}$$

$$\textbf{const}\ \langle \exists l :: \sigma[v]^b@l \rangle \quad \text{(P3)}$$

$$\textbf{const}\ \langle \exists v,l :: \sigma[v]^t@l \rangle \quad \text{(P4)}$$

Message Location in the Environment

$$\textbf{inv}\ \alpha@(p,q) \wedge (p\ge N+1 \vee q\le 0) \wedge \beta@(p',q') \wedge (p'\ge N+1 \vee q'\le 0) \Rightarrow (pid.\alpha=pid.\beta \Leftrightarrow (p,q)=(p',q')) \quad \text{(P5)}$$

$$\textbf{inv}\ (\alpha@(p,q) \wedge q\le 0 \Rightarrow p=src.\alpha) \wedge (\alpha@(p,q) \wedge p\ge N+1 \Rightarrow q=dest.\alpha) \quad \text{(P6)}$$

Queue Properties

$$\langle \exists\ q,q' : q<q'\le 0 :: \alpha@(p,q) \wedge \beta@(p,q') \rangle\ \textbf{unless}\ \neg\langle \exists\ q' : q'\le 0 :: \beta@(p,q') \rangle \quad \text{(P7)}$$

$$\textbf{stable}\ \langle \exists\ p : p\ge N+1 :: \alpha@(p,q) \wedge \neg\langle \exists p' : p'>p :: \beta@(p',q) \rangle \rangle \quad \text{(P8)}$$

No-Reordering Property

$$\textbf{inv}\ \alpha \sqsubset \beta \Rightarrow pid.\beta<pid.\alpha \quad \text{(P9)}$$

Non-Interleaving Property

$$\textbf{inv}\ \alpha \sqsubset \beta \sqsubset \delta \wedge mid.\alpha=mid.\delta \Rightarrow mid.\beta=mid.\alpha \quad \text{(P10)}$$

Packet Movement

$$\sigma[v]@(p,q) \wedge q\le 0 \mapsto \langle \exists\ v',p',q' : p'\ge N+1 :: \sigma[v']@(p',q') \rangle \quad \text{(P11)}$$

⟨ **op** dummy_variables : range_constraint :: expression ⟩
where **op** is typically a binary, associative, and commutative operator (such as +, *, ∧, ∨, written Σ, Π, ∀, ∃, respectively). Logically, the constructor creates a multi-set of values $\{v_1, v_2, \ldots, v_n\}$ by evaluating the *expression* for every possible instantiation of the *dummy_variables* satisfying the *range_constraint*. The final value of the constructor is obtained by evaluating the expression v_1 **op** v_2 **op** ... **op** v_n. If the range is empty the zero-element for the operator is returned. Other frequently used operators are *min*, *max*, and *set*, having the obvious interpretations.

The first four properties are related to the representation of the messages. Property *(P1)* states that messages are properly structured. If α is a packet at some location in the system, and k is the number of packets of the message α belongs to, then:

- k is greater than or equal to 3, i.e., each message is composed of at least three packets;
- the packet number of α is equal to 1 iff α is a header, ranges strictly from 1 to k iff α is a body, and is equal to k iff α is a tail.

Properties *(P2)* to *(P4)* state that packets are neither created nor destroyed, and that the value of body packets may not be modified. Let us for instance explain property *(P2)*. Going back to the definition of ***const***, *(P2)* means that:

- If the header packet $\sigma[v]$ exists initially in the system, it will exist forever, but not necessarily with the same value (existential quantification on v).
- If, for any value v, $\sigma[v]$ does not initially exist in the system, then it will never exist.

Property *(P4)* is the symmetric of *(P2)* for tail packets. In addition, property *(P3)* prevents the value of the body packets to be changed.

Properties *(P5)* and *(P6)* specify packet locations in the environment. *(P5)* states that each packet has a unique location, and that each location contains at most one packet. Property *(P6)* expresses the requirement that each packet in the input environment resides in its source input queue, and each packet in the output environment resides in its destination output queue.

Properties *(P7)* and *(P8)* specify that the input rows and output columns are queues. *(P7)* states that packets are sent in the order they are queued, i.e., a packet cannot be sent if there are packets in front of it in the queue: considering packet β in its input queue, and packet α behind β in the same queue, α must stay behind β as long as β is in the input queue. Property *(P8)* states that packets received in the output queues can only appear behind the packets already present in the queues. A way to specify it is to say that, considering two packets α and β, if α is in its output queue and β is not ahead of α in the queue, then it will never be.

The no-reordering and non-interleaving constraints are expressed by properties *(P9)* and *(P10)*. Initially, the packets in the input environment are queued in the order they have to be sent—i.e., the packets with the lexicographically smallest identifiers are in the first positions of the queues. This means that if packet β is ahead of α ($\alpha \sqsubset \beta$), β's pid is smaller that α's one. To express the no-reordering constraint, we have to state that this relation is invariant all along the execution, as expressed by property *(P9)*. The non-interleaving requirement is specified by property *(P10)*. Considering three packets α, β, and δ such that δ is ahead of β which is ahead of α, if α and δ belong to the same message, β must also belong to the same message.

The first ten properties are all safety properties. The only progress requirement is stated by property *(P11)*. Each packet $\sigma[v]$ in the input environment must eventually reach the output environment, but not necessarily with the same value.

3. Router Design

The specification S_0 is composed of input/output properties of the router, but never mentions the router itself. The reason is that we were only interested in specifying *what* the router is supposed to do (i.e., deliver messages from its input to its output with some no-reordering and non-interleaving constraints), and not *how* it is actually doing it. Consequently, S_0 can lead to many different solutions for the router design. We describe one of these solutions in this section.

Our design process entails several refinement steps. The objective of these steps is to gradually give a more and more detailed description of the router, up to the point where a UNITY program can be derived trivially from the specification. To be correct, each refined specification must imply the specification of the previous step. We start with a brief overview of the refinement steps.

Refinement 1. In the first refinement, we define the general topology of the router as a grid of $N\times M$ switches. The N input lines and M output lines are thus extended inside the router. Each switch can receive packets from its left neighbor on the row or its bottom neighbor on the column, and can route them either to its right neighbor on the row or to its upper neighbor on the column, depending on the destination of the packets. So, to move from its source row to its destination column, each packet first travels along the row (one switch at a time) until it reaches the destination column, and then just moves up the column (also one switch at a time).

Refinement 2. The second refinement provides additional details about the behavior of each single switch, by defining the mechanism that prevents messages from being interleaved along the columns. We will see that this can be done by associating two mutually exclusive signals—*turn* and *up*—with each switch. Signal *turn* prevents messages to move through the switch along the column when a message is currently passing through the switch from the row to the column, and the other way around for signal *up*.

Refinement 3. In the third refinement, we specify further the behavior of the switches by introducing a strong fairness constraint, which prevents more than one message to pass through a switch on the column (from the row to the column) when another message is waiting on the row (on the column).

Refinement 4. At this point in the design, each switch on the rows makes the decision to route messages either to the next switch on the row, or to the next switch on the column, by comparing the message destination to the number of the column it is located at. This implies that each switch has to know its location. The purpose of the fourth refinement is to eliminate this knowledge by using the value of the header packets. We will make the value of each header packet decreased by one each time the packet passes through a switch along the row. Since the value is initially equal to the destination column, this implies that a message will have to take a turn when the value is equal to 1.

Refinement 5. The fifth refinement deals with the execution control we impose on the switches. A possible choice is to have the switches running asynchronously, another choice is to have them working in a synchronous way. We chose the more realistic asynchronous behavior. Since each location can contain at most one packet, this implies that a packet will not be able to move to the next location, unless it is empty.

Refinement 6. Finally, the last refinement step is the transformation of the leads-to properties into ensures properties. As we said earlier in the paper, ensures properties are strongly related to the text of the program, which implies that we can easily derive assignment statements from them.

3.1. Refinement 1: Definition of the Router Topology

The router we are designing consists of a $N\times M$ grid of switches. As an example, a three-sender four-receiver router is depicted in Figure 3.

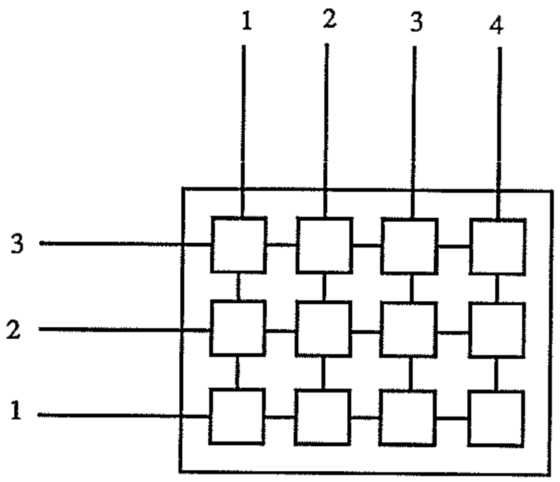


Figure 3. Internal structure of a three-sender four-receiver router

The location of each switch in the grid is given by the pair *(p,q)* where *p* is the row number, and *q* the column number. Each switch *(p,q)* contains two registers—a row and a column register—and an arbitration element (Figure 4). We identify the location of the row register by the triple *(p,q,0)*, and the location of the column register by *(p,q,1)*. The function of the arbitration element is to handle the movement of the packets showing up in the row and column registers, in order to avoid interleaving of messages on the columns. Its design will be the subject of refinement 2.

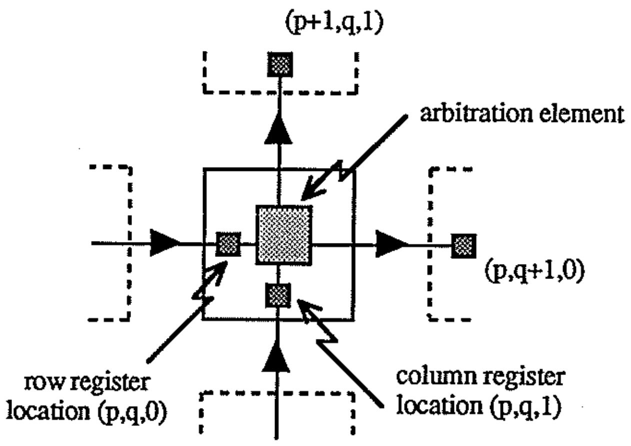

(p+1,q,1)
arbitration element
(p,q+1,0)
row register location (p,q,0)
column register location (p,q,1)

Figure 4. Internal structure of switch (p,q).

The packet movement inside the router is defined as follows: from the row register of switch *(p,q)*—i.e., location *(p,q,0)*—a packet moves to the row register of switch *(p,q+1)*—i.e., location *(p,q+1,0)*—if *q* is not the destination column, or to the column register of switch *(p+1,q)*—i.e., location *(p+1,q,1)*—if *q* is the destination column. Once it has reached its destination column, a packet just goes up, one location at a time.

Let us now give some definitions and notation. To make the location spaces inside and outside the router uniform, and thus make the specification simpler, we redefine environment locations *(p,q)* to be triples *(p,q,0)* in the input queues, and triples *(p,q,1)* in the output queues. The previous set *E* of environment locations is thus refined into the following set *E'*:

$$E' = \langle \mathbf{set}\ p,q,r : (1\le p\le N \wedge q\le 0 \wedge r=0) \vee (p\ge N+1 \wedge 1\le q\le M \wedge r=1) :: (p,q,r)\rangle.$$

The definition of *R*, the set of locations inside the router, is:

$$R \equiv \langle \mathbf{set}\ p,q,r : 1\le p\le N \wedge 1\le q\le M \wedge 0\le r\le 1 :: (p,q,r)\rangle.$$

The location function *Π* is refined into the function *Π'*:

$$\Pi' : \Lambda \rightarrow E' \cup R$$

and the relation between *Π* and *Π'* is given by the following coupling invariant:

$$\begin{aligned}\langle \forall \alpha,p,q,r, ::\ & p\le N \wedge q\ge 1 \wedge 0\le r\le 1 \Rightarrow [\Pi.\alpha=(p,q,r) \Leftrightarrow \Pi'.\alpha=(p,q,r)]\\ \wedge\ & q\le 0 \Rightarrow [\Pi.\alpha=(p,q) \Leftrightarrow \Pi'.\alpha=(p,q,0)]\\ \wedge\ & p\ge N+1 \Rightarrow [\Pi.\alpha=(p,q) \Leftrightarrow \Pi'.\alpha=(p,q,1)]\rangle.\end{aligned}$$

We now use *α@(p,q,r)* to mean *Π'.α=(p,q,r)*, and we introduce the notation *(p,q,r)»(p',q',r')* to mean:

$$(p'=p \wedge q'=q+1 \wedge r=r'=0) \vee (p'=p+1 \wedge q'=q \wedge r'=1).$$

The operator "»" relates valid pairs of consecutive locations. Location *(p,q,0)* is thus in relation with *(p,q+1,0)* and *(p+1,q,1)*, and location *(p,q,1)* is in relation with *(p+1,q,1)*. The relation $\sqsubset$ is refined into the relation $\prec$. Its definition is:

$$\begin{aligned}\alpha\prec\beta \Leftrightarrow\ & \langle \exists p,q,q' : q<q' :: \alpha@(p,q,0) \wedge \beta@(p,q',0)\rangle \vee\\ & \langle \exists p,p',q : p<p' :: \alpha@(p,q,1) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle \vee\\ & \langle \exists p,p',q,q' : q'\le q :: \alpha@(p,q',0) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle.\end{aligned}$$

That is, β is ahead of α according to $\prec$ iff β is in front of α on the same row, or β is in front of α on the same column and both packets come from the same source, or β is on its column, α is on its row at the left of β, and both packets have the same source. The refined specification S_1 is the following:

Message Representation

Properties (P1), (P2), (P3) and (P4).

Message Location

inv $\alpha@(p,q,r) \wedge \beta@(p',q',r') \Rightarrow (pid.\alpha=pid.\beta \Leftrightarrow (p,q,r)=(p',q',r'))$ (P5.1)
inv $\alpha@(p,q,r) \Rightarrow (p=src.\alpha \wedge q\leq dest.\alpha \wedge r=0) \vee (p>src.\alpha \wedge q=dest.\alpha \wedge r=1)$ (P6.1)

No-Reordering Property

inv $\alpha\prec\beta \Rightarrow pid.\beta<pid.\alpha$ (P9.1)

Non-Interleaving Property

inv $\alpha\prec\beta\prec\delta \wedge mid.\alpha=mid.\delta \Rightarrow mid.\beta=mid.\alpha$ (P10.1)

Packet Movement in the Environment

$\sigma[v]@(p,q,r) \wedge (p\geq N+1 \vee q\leq 0)$ **until** $\langle \exists\, p',q',r' : (p,q,r)\gg(p',q',r') :: \sigma[v]@(p',q',r')\rangle$ (P11.1)

Packet Movement inside the Router

$\sigma[v]@(p,q,r) \wedge p\leq N\wedge q\geq 1$ **until** $\langle \exists\, v',p',q',r' : (p,q,r)\gg(p',q',r') :: \sigma[v']@(p',q',r')\rangle$ (P11.2)

Properties *(P5)*, *(P6)*, *(P9)*, *(P10)*, and *(P11)* of specification S_0, which define properties of the packets in the environment, have been extended to the entire system. Property *(P5.1)* specifies that each packet in the environment or the router has a unique location, and that each location contains at most one packet. Property *(P6.1)* defines the set of locations packets are allowed to be at. Each packet can only be located in the source input row or the destination output column, but cannot move beyond the destination column on the row, and below the input row on the column. Property *(P9.1)* states that packets remain ordered according to their identifiers, and *(P10.1)* specifies that messages are not interleaved. Finally, property *(P11)*, which states that every packet in the input eventually moves to the output, has been refined into two progress properties, one for the environment, and the other one for the router. Property *(P11.1)* states that each packet in the environment eventually moves to the next location, as defined by the operator "$\gg$". Since it is an ***until*** property, *(P11.1)* also specifies that packets cannot move anywhere but to the next location. Property *(P11.2)* is quasi symmetrical to *(P11.1)* for packets inside the router. The only difference is that packet values are allowed to change.

Proof Obligations. We need to show that each property of S_0 is implied by S_1. Note that even though properties *(P1)* to *(P4)* are not syntactically modified, the definition of the function @ has changed, which implies that these properties also need to be verified.

Proof Outline. The formal proof of each of the refinement steps can be found in appendix B. In the proof outline sections, we just give an intuitive explanation in order to convince the reader of the validity of the refinements.

The proof of properties *(P1)*-*(P6)* and *(P9)*-*(P10)* is straightforward. Each property is directly implied by the corresponding property in S_1 and the coupling invariant.

The proof that packets are sent in the order they are queued (property *(P7)*) follows from the fact that packets in the input queues move one location at a time (property *(P11.1)*), and that each location can contain at most one packet (property *(P5.1)*). This implies that packets cannot pass each other, and hence that they can only be sent in the order they are queued.

The proof of *(P8)*—packets in the output queues can only appear behind the packets already present in the queues— is a bit more complicated. We need to show that, considering two packets α and β, if α is in its output queue and β is not ahead of α in the queue, then it will never be. Three cases are possible. If β does not exist in the system, then it will never be ahead of α since packets cannot be created (properties *(P2)-(P4)*). If β exists in the system, but its destination column is different from the destination column of α, then it will also never be ahead of α since a packet in the output environment can only reside in its destination column. The final case is β existing in the system with the same destination column as α. In this case, the only way for β to move ahead of α is to overtake it, which is not possible, as stated above.

Finally, the fact that packets in the input eventually move to the output (property *(P11)*) can be proved from *(P11.1)* and *(P11.2)* by a double application of the induction principle. We can first show that a packet in the input environment eventually moves to its destination column, by using the distance between the destination column and the position of the packet in the row as the metric. This distance decreases by one with each move. We can also prove that a packet on the destination column inside the router eventually moves to the output environment. The metric in this case is the distance between row *N+1* and the position of the packet in the column; it also decreases by one with each move. The truth of *(P11)* follows from the transitivity of leads-to.

3.2. Refinement 2: Arbitration Element Refinement

The role of the arbitration element is to route out of the switch the packets showing up in the row and column registers. The main problem is to avoid the interleaving of messages passing through the switch on the column and from the row to the column. To solve this problem, we introduce two mutually exclusive signals—*turn* and *up*—whose purpose is to regulate the movement of packets at the intersection of the row and the column (Figure 5). When the signal *turn* is on—which implies that *up* is off—the paths from location *(p,q,0)* to location *(p,q+1,0)*, and from *(p,q,1)* to *(p+1,q,1)* are broken. Packets can only move from location *(p,q,0)* on the row to location *(p+1,q,1)* on the column. When the signals *turn* and *up* are off, packets can only move through the switch along the row. Finally, when the signal *up* is on—which implies that *turn* is off—packets can simultaneously move through the switch along the row and along the column.

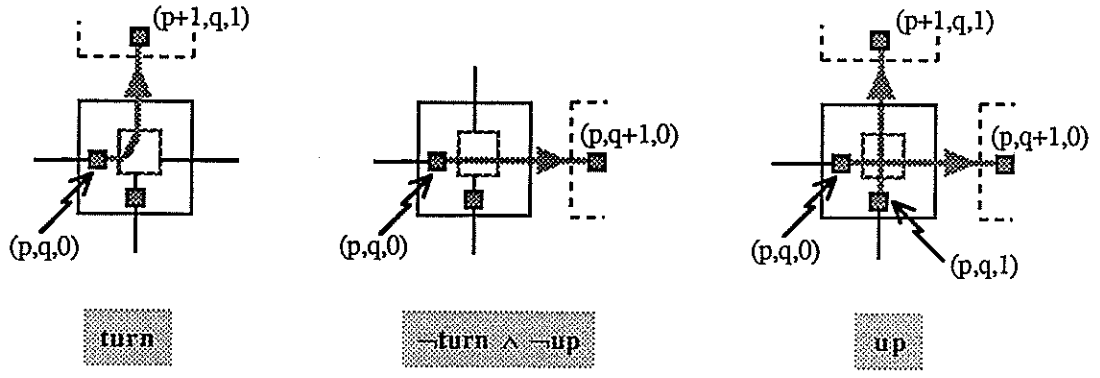


Figure 5. Possible states of the arbitration element.

The *turn* and *up* signals are initially false, and are triggered by the arrival of header packets in the row and column registers. The *turn* signal is eventually set when a header packet whose destination column is equal to *q* shows up at location *(p,q,0)*. The signal remains on as long as the tail of the same message has not showed up, and is turned off as the tail moves through the switch. The policy associated with the *up* signal is symmetrical. Let us mention that we do not impose any priority between the row and column registers. That is, when two headers (going to the same location *(p+1,q,1)*) are in the row and column registers at the same time, the choice of which signal to be turned on is non-deterministic.

We define the predicate *turn(p,q)* to mean that the *turn* signal in switch *(p,q)* is on, and *up(p,q)* to mean that the *up* signal in switch *(p,q)* is on. We also define a two-parameter function *message*. The expression *message.l.l'* is interpreted to mean that there is a message in the system whose head is at location *l*, and tail at location *l'*. Formally:

$$\text{message.l.l'} \equiv \langle \exists \alpha,\beta : \text{mid}.\alpha=\text{mid}.\beta :: \alpha^h@l \wedge \beta^t@l' \rangle.$$

The refined specification S_2 is the following:

Message Representation

Properties (P1), (P2), (P3), and (P4).

Message Location

Properties (P5.1) and (P6.1).

No-Reordering and Non-Interleaving Properties

Properties (P9.1) and (P10.1).

Signal Properties

$\textbf{inv}\ \neg(turn(p,q) \wedge up(p,q))$ (P12)

$\textbf{inv}\ q\geq 1 \wedge \langle \exists p',q',r',q'' : q''\leq q<q' :: message.(p',q',r').(p,q'',0)\rangle \Rightarrow \neg turn(p,q)$ (P13)

$\textbf{inv}\ q\geq 1 \wedge \langle \exists p',q' : p'>p \wedge q'\leq q :: message.(p',q,1).(p,q',0)\rangle \Rightarrow turn(p,q)$ (P14)

$\textbf{inv}\ p\leq N \wedge \langle \exists p',p'',q'',r'': p'>p \wedge (p''<p \vee (p''=p \wedge r''=1)) :: message.(p',q,1).(p'',q'',r'')\rangle \Rightarrow up(p,q)$ (P15)

$\textbf{inv}\ turn(p,q) \Rightarrow \langle \exists p',q',r' : ((r'=0 \wedge p'=p) \vee (r'=1 \wedge p'>p)) \wedge q'\leq q :: message.(p',q,r').(p,q',0)\rangle$ (P16)

$\textbf{inv}\ up(p,q) \Rightarrow \langle \exists p',p'',q'',r'': p'\geq p \wedge (p''<p \vee (p''=p \wedge r''=1)) :: message.(p',q,1).(p'',q'',r'')\rangle$ (P17)

$\alpha^h@(p,q,0) \wedge q=dest.\alpha \wedge turn(p,q)\ \textbf{unless}\ \neg\alpha@(p,q,0)$ (P18)

$\alpha^h@(p,q,1) \wedge p\leq N \wedge up(p,q)\ \textbf{unless}\ \neg\alpha@(p,q,1)$ (P19)

$\alpha^t@(p,q,0) \wedge q\geq 1 \wedge turn(p,q)\ \textbf{unless}\ \neg\alpha@(p,q,0) \wedge \neg turn(p,q)$ (P20)

$\alpha^t@(p,q,1) \wedge p\leq N \wedge up(p,q)\ \textbf{unless}\ \neg\alpha@(p,q,1) \wedge \neg up(p,q)$ (P21)

Packet Movement in the Environment

Property (P11.1).

Packet Movement inside the Router

$\sigma[v]@(p,q,r) \wedge p\leq N \wedge q\geq 1\ \textbf{unless}\ \langle \exists v',p',q',r' : (p,q,r)\gg(p',q',r') :: \sigma[v']@(p',q',r')\rangle$ (P11.2.1)

$\sigma[v]^h@(p,q,0) \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \mapsto \langle \exists v'::\sigma[v']@(p,q+1,0)\rangle$ (P11.2.2)

$\sigma[v]^\tau@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge \neg turn(p,q) \mapsto \langle \exists v'::\sigma[v']@(p,q+1,0)\rangle$ (P11.2.3)

$\sigma[v]^h@(p,q,0) \wedge q=dest.\alpha \mapsto \sigma[v]@(p,q,0) \wedge turn(p,q)$ (P11.2.4)

$\sigma[v]@(p,q,0) \wedge q\geq 1 \wedge turn(p,q) \mapsto \langle \exists v' :: \sigma[v']@(p+1,q,1)\rangle$ (P11.2.5)

$\sigma[v]^h@(p,q,1) \wedge p\leq N \mapsto \sigma[v]@(p,q,1) \wedge up(p,q)$ (P11.2.6)

$\sigma[v]@(p,q,1) \wedge p\leq N \wedge up(p,q) \mapsto \langle \exists v' :: \sigma[v']@(p+1,q,1)\rangle$ (P11.2.7)

The safety properties of the switch signals are described by *(P12)-(P21).* The fact that the two signals are mutually exclusive is expressed by *(P12).* The meaning of invariant *(P13)* is illustrated in Figure 6. If a message is in transit through the switch *(p,q)* along the row—i.e., the header is past the switch, either still on the row or already on the destination column, and the tail has not passed the switch yet—then the turn signal is off.

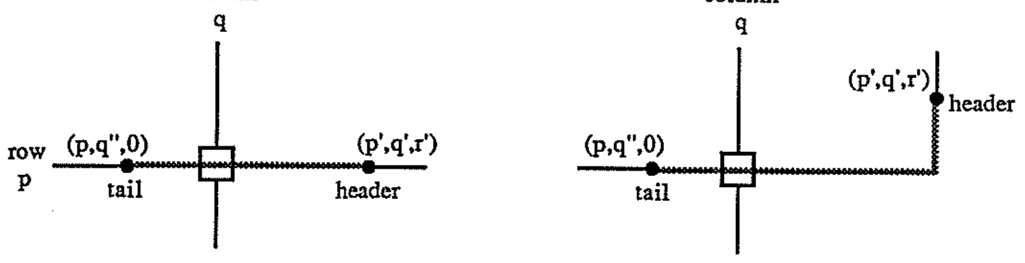


Figure 6. The *turn* signal is off when a message is in transit through the switch along the row.

As depicted in Figure 7, invariant *(P14)* states that, if a message is in transit through the switch *(p,q)* from the row to the column—i.e., the header is past the switch on the column, and the tail is still on the row—then the *turn* signal is on.

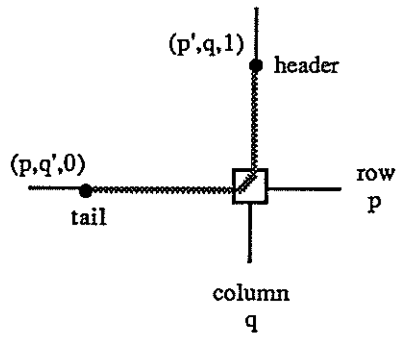


Figure 7. The *turn* signal is on when a message is in transit from the row to the column.

Invariant *(P15)* is symmetric to *(P14)* for the *up* signal. It states that, if a message is in transit through the switch *(p,q)* along the column—i.e., the header is past the switch on the column, and the tail is either on the column and has not moved through the switch yet, or is still on the source row—then the *up* signal is off.

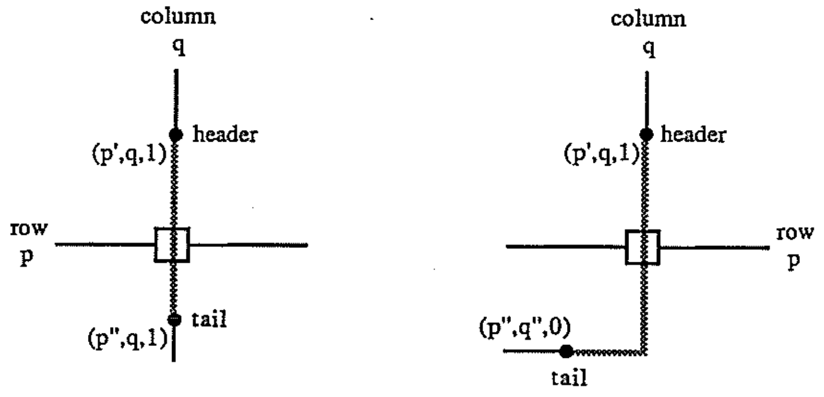


Figure 8. The *up* signal is on when a message is in transit through the switch along the column.

Invariants *(P16)* and *(P17)* further specify that when the *turn* signal (*up* signal) is on, there must exist a message in transit through the switch from the row to the column (on the column). Note that, as opposed to *(P14)* and *(P15)*, properties *(P16)* and *(P17)* provide for the case where the header of the message is still in the row register (column register) of the switch. The reason is that, when a header is in the row or column register, the proper signal is first turned on, and only in a subsequent step the packet is allowed to move.

Properties *(P18)* and *(P19)* state that once a signal is turned on, it remains on until the header moves. This prevents signals from being turned on and off repeatedly while the header of the message is still waiting in the row or in the column register. Finally, properties *(P20)* and *(P21)* express the fact that signals are turned off at the same time the tail of the message moves through the switch.

The movement of the packets inside the router is now described by properties *(P11.2.1)-(P11.2.7)*. As in the previous specification, packets can only move to the next location (property *(P11.2.1)*). Properties *(P11.2.2)* and *(P11.2.3)* specify the movement of the packets along the rows. The former states that a header packet at location *(p,q,0)*, with *q* different from the destination column, eventually moves to the next location on the row. The latter states that the body and tail packets showing up at location *(p,q,0)* eventually move to the next location on the row if the signal *turn* is off, i.e., if it has not be turned on by the header of the message. Properties *(P11.2.4)* and *(P11.2.5)* deal with the movement of the packets from the row to the column. The *turn* signal in switch *(p,q)* is eventually turned on after the arrival of a header packet whose destination column is *q* (property *(P11.2.4)*). Once the signal has been turned on, packets are allowed to move to location *(p+1,q,1)* on the column (*property (P11.2.5)*). Properties *(P11.2.6)* and *(P11.2.7)* are symmetric to *(P11.2.4)* and *(P11.2.5)* for the *up* signal.

__Proof Obligations__. The only change brought to specification S_1 is the refinement of *(P11.2)* into *(P11.2.1)-(P11.2.7)*. So we only need to show that S_2 implies *(P11.2)*.

__Proof Outline__. The unless part of *(P11.2)* is equivalent to *(P11.2.1)*. To prove the leads-to part, we need to show that every packet inside the router eventually moves to the next location. The movement on the rows is implied by properties *(P11.2.2)*, *(P11.2.3)*, and invariant *(P13)*. Property *(P11.2.2)* directly states that the header of each message will eventually move to the next location. The movement of the remaining packets of the message is guaranteed by *(P11.2.3)* when the *turn* signal is off, which is implied by invariant *(P13)*. The movement from the row to the column is implied by properties *(P11.2.4)*, *(P11.2.5)*, and invariant *(P14)*. The movement of the header packets follows from the transitivity of leads-to applied to *(P11.2.4)* and *(P11.2.5)*. The movement of the non-header packets is guaranteed by *(P11.2.5)* when the *turn* signal is on, which is implied by *(P14)*. The proof of the movement on the columns follows similarly from *(P11.2.6)*, *(P11.2.7)*, and *(P15)*.

3.3. Refinement 3: Introduction of a Fairness Constraint

At this point in the design, packets are guaranteed to move through the switches without being interleaved on the columns, but no property in the specification constrains the arbitration elements to behave fairly. Consider for instance the case where *n* consecutive messages going to the same destination *q* have been sent by the same sender *p*. Suppose that when the first of these messages arrives at switch *(p,q)*, another message is waiting on the column for moving up through the switch. Then a possible scenario is to have the *n* messages on the row moving through the switch up to the column, before the message on the column is allowed to do so, thus blocking all the messages behind it. A symmetric problem occurs when a message is waiting on the row, while several messages are passing through the switch along the column.

We want to prevent such undesirable behaviors by imposing a strong fairness constraint on the arbitration elements. No more than one message must pass through a switch from the row to the column (along the column), while another message is waiting on the column (on the row). Figures 9 and 10 describe a simple mechanism implementing this requirement. In the first case—a message m_2 is waiting on the column while another message m_1 is moving from the row to the column (Figure 9)—the solution is to turn the *up* signal on as the tail of m_1 is passing through the switch. Since the specification guarantees that the *up* signal cannot not be turned off before the head of m_2 moves (property *(P21)*), no other message will be able to move from the row to the column.

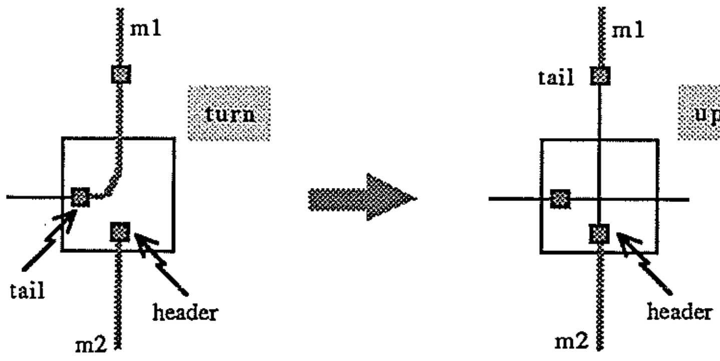


Figure 9. No more than one message can move from the row to the column, while another one is waiting on the column.

In the second case—a message m_2 is waiting on the row while another message m_1 is moving along the column (Figure 10)—the solution is to turn the *turn* signal on as the tail of m_1 is passing through the switch.

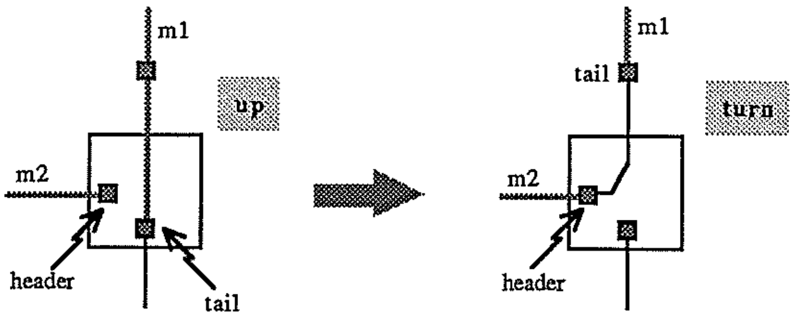


Figure 10. No more than one message can move along the column, while another one is waiting on the row.

Formally, this refinement entails the addition of two new properties describing the preceding behaviors. The refined specification S_3 is:

Message Representation

Properties (P1)-(P4).

Message Location

Properties (P5.1) and (P6.1).

No-Reordering and Non-Interleaving Properties

Properties (P9.1) and (P10.1).

Signal Properties

Properties (P12)-(P21).

$\alpha^t@(p,q,0) \wedge q \geq 1 \wedge \text{turn}(p,q) \wedge \beta^h@(p,q,1)$ **unless** $\neg\alpha^t@(p,q,0) \wedge \text{up}(p,q)$ (P22)

$\alpha^t@(p,q,1) \wedge p \leq N \wedge \text{up}(p,q) \wedge \beta^h@(p,q,0) \wedge q=\text{dest}.\beta$ **unless** $\neg\alpha^t@(p,q,1) \wedge \text{turn}(p,q)$ (P23)

Packet Movement in the Environment

Property (P11.1).

Packet Movement inside the Router

Properties (P11.2.1)-(P11.2.7).

The proof of the refinement is trivial since the specification S_3 includes all the properties of S_2.

3.4. Refinement 4: Refinement of the Header Values

A shortcoming of the design at this point is that switches need to know their location in the grid to route the packets. This knowledge appears in properties *(P11.2.2)* and *(P11.2.4)*, which state that when a header packet shows up in a row register, the switch either routes the packet to the next location on the row if *the number of the column* is different from the packet destination (*(P11.2.2)*), or sets the *turn* signal if *the number of the column* is equal to the packet destination (*(P11.2.4)*).

The purpose of the fourth refinement is to eliminate this knowledge by making the value of each header decreased by one, each time the packet moves through a switch along the row. Since the value is initially equal to the number of the destination column, switches can now make the decision to route the header packets to the next location on the row when the header value is different from 1, and to set the *turn* signal when the header value is equal to 1. The location knowledge is not needed anymore. The refined specification S_4 is:

Message Representation

Properties (P1)-(P4).

Message Location

Properties (P5.1) and (P6.1).

No-Reordering and Non-Interleaving Properties

Properties (P9.1) and (P10.1).

Signal Properties

Properties (P12)-(P23).

Header Value Invariant

$\mathbf{inv}\ \sigma[v]^h@(p,q,r) \wedge q\geq 1 \Rightarrow v=dest.\sigma[v]-q+1$ (P24)

Packet Movement in the Environment

Property (P11.1).

Packet Movement inside the Router

Property (P11.2.1).

$\sigma[v]^h@(p,q,0) \wedge q\geq 1 \wedge v\neq 1 \mapsto \sigma[v-1]@(p,q+1,0)$ (P11.2.2.1)
$\sigma[v]^\tau@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge \neg turn(p,q) \mapsto \sigma[v]@(p,q+1,0)$ (P11.2.3.1)

$\sigma[v]^h@(p,q,0) \wedge v=1 \mapsto \sigma[v]@(p,q,0) \wedge turn(p,q)$ (P11.2.4.1)
$\sigma[v]@(p,q,0) \wedge q\geq 1 \wedge turn(p,q) \mapsto \sigma[v]@(p+1,q,1)$ (P11.2.5.1)

$\sigma[v]^h@(p,q,1) \wedge p\leq N \mapsto \sigma[v]@(p,q,1) \wedge up(p,q)$ (P11.2.6.1)
$\sigma[v]@(p,q,1) \wedge p\leq N \wedge up(p,q) \mapsto \sigma[v]@(p+1,q,1)$ (P11.2.7.1)

The value of the header packets is constrained by the invariant *(P24)* which states that, along the rows inside the router, the value is decreased by one with each move, and that it remains constant once the packet has reached the destination column. The movement of the packets inside the router is trivially refined by properties *(P11.2.2.1)-(P11.2.7.1)*.

<u>Proof Obligations</u>. We need to show that properties *(P11.2.1)-(P11.2.7)* are implied by S_4.

<u>Proof Outline</u>. The proof is straightforward. Properties *(P11.2.3)*, *(P11.2.5)*, *(P11.2.6)*, and *(P11.2.7)* are directly implied by the corresponding refined properties *(P11.2.3.1)*, *(P11.2.5.1)*, *(P11.2.6.1)*, and *(P11.2.7.1)*. To prove that *(P11.2.2)* and *(P11.2.4)* are implied by *(P11.2.2.1)* and *(P11.2.4.1)*, we only need to show:

$\mathbf{inv}\ \sigma[v]^h@(p,q,0) \wedge q\geq 1 \Rightarrow (q=dest.\sigma[v] \Leftrightarrow v=1)$

which is trivially implied by the invariant *(P24)*.

3.5. Refinement 5: Switches Work in an Asynchronous Way

The last design decision concerns the execution control we impose on the switches. Between the two possible choices—either synchronous behavior, or asynchronous behavior—we chose the more realistic asynchronous behavior. Since each location can contain at most one packet, this means that, before routing a packet to the next location, a switch must first check whether this location is empty. Note that, in the case of a synchronous behavior, this constraint would not have been needed, since two consecutive packets have to move synchronously in this case.

Let us define the predicate *empty@(p,q,r)* to mean that location *(p,q,r)* does not contain any packet:

$$\text{empty}@(p,q,r) \equiv \langle \forall\alpha :: \neg\alpha@(p,q,r)\rangle.$$

A way to specify the asynchronous behavior is to state that each packet inside the router must leave an empty location behind it when it moves:

$$\alpha@(p,q,r) \wedge p\leq N \wedge q\geq 1 \textbf{ unless } \text{empty}@(p,q,r).$$

This forces the switches to wait for the next location to be emptied, before routing a packet. The refined specification S_5 contains all the properties of S_4 plus the previous one. The proof of the refinement is therefore straightforward.

Message Representation

Properties (P1)-(P4).

Message Location

Properties (P5.1) and (P6.1).

No-Reordering and Non-Interleaving Properties

Properties (P9.1) and (P10.1).

Signal Properties

Properties (P12)-(P23).

Header Value Invariant

Property (P24).

Packet Movement in the Environment

Property (P11.1).

Packet Movement inside the Router

Property (P11.2.1).
Properties (P11.2.2.1)-(P11.2.7.1).
$\alpha@(p,q,r) \wedge p\leq N \wedge q\geq 1 \textbf{ unless } \text{empty}@(p,q,r).$ (P25)

3.6. Refinement 6: Transformation of the Leads-to Properties into Ensures Properties

This last refinement is not motivated by a design decision. The motivation here is to transform the specification into a form that can be directly translated into program text, i.e., introduce *ensures* properties expressing atomic transformations. The progress properties are property *(P11.1)* specifying the packet movement in the environment, and properties *(P11.2.2.1)-(P11.2.7.1)* expressing the packet movement inside the router. Since we are only interested in the design of the message router, we will not transform the environment property *(P11.1)* into an *ensures* property. We will just make sure that it is satisfied when deriving the program. However, to make it more explicit, we can split it into two properties, one for the input queues and the other one for the output queues:

$\sigma[v]@(p,q,0) \wedge q\leq 0 \textbf{ until } \sigma[v]@(p,q+1,0)$ (P11.1.1)
$\sigma[v]@(p,q,1) \wedge p\geq N+1 \textbf{ until } \sigma[v]@(p+1,q,1).$ (P11.1.2)

Let us now focus on properties *(P11.2.2.1)-(P11.2.7.1)*. The simplest way to transform a *leads-to* property into an *ensures* property is to directly replace "$\mapsto$" by "**ensures**", and to see if the resulted property can be satisfied in an atomic transformation. By applying this method on *(P11.2.2.1)*, we get:

$$\sigma[v]^h@(p,q,0) \wedge q\geq 1 \wedge v\neq 1 \textbf{ ensures } \sigma[v-1]@(p,q+1,0).$$

This property cannot be satisfied in an atomic transformation since the next location may be busy at the time. This suggests the following property:

$\sigma[v]^h@(p,q,0) \wedge q\geq 1 \wedge v\neq 1 \wedge \text{empty}@(p,q+1,0) \textbf{ ensures } \sigma[v-1]@(p,q+1,0)$ (P11.2.2.1.1) which can easily be satisfied in an atomic step. In the same way, we get from *(P11.2.3.1)*:

$$\alpha[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge \neg turn(p,q) \wedge empty@(p,q+1,0)\ \textbf{ensures}\ \sigma[v]@(p,q+1,0). \qquad (P11.2.3.1.1)$$

The transformation of the properties *(P11.2.5.1)* and *(P11.2.7.1)* is slightly more complicated. We need to separate the cases where packets stay inside the router (*p<N*)or move from the router to the output environment (*p=N*). In the former case, we can transform the properties in the same way we did before. We get:

$$\sigma[v]@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge empty@(p+1,q,1)\ \textbf{ensures}\ \sigma[v]@(p+1,q,1) \qquad (P11.2.5.1.1)$$

and:

$$\sigma[v]@(p,q,1) \wedge p<N \wedge up(p,q) \wedge empty@(p+1,q,1)\ \textbf{ensures}\ \sigma[v]@(p+1,q,1). \qquad (P11.2.7.1.1)$$

Since we do not require that locations outside the router be emptied before packets move, properties *(P11.2.5.1)* and *(P11.2.7.1)* with *p* equal to *N*, can simply be transformed into:

$$\sigma[v]@(N,q,0) \wedge q\geq 1 \wedge turn(N,q)\ \textbf{ensures}\ \sigma[v]@(N+1,q,1) \qquad (P11.2.5.1.2)$$

and:

$$\sigma[v]@(N,q,1) \wedge up(N,q)\ \textbf{ensures}\ \sigma[v]@(N+1,q,1). \qquad (P11.2.7.1.2)$$

This implies that packets moving out of the router are allowed to simultaneously push forward all the packets in the output queues.

Let us now concentrate on property *(P11.2.4.1)*. A direct transformation into an *ensures* property:

$$\sigma[v]^{h}@(p,q,0) \wedge v=1\ \textbf{ensures}\ \sigma[v]@(p,q,0) \wedge turn(p,q)$$

is not possible because the *turn* signal cannot be set when a message is currently passing through the switch along the column, i.e., when the *up* signal is on. This suggests the following transformation:

$$\sigma[v]^{h}@(p,q,0) \wedge v=1 \wedge \neg up(p,q)\ \textbf{ensures}\ \sigma[v]@(p,q,0) \wedge turn(p,q).$$

This is however not yet satisfactory. Consider the case where a header packet with value 1 is in the row register, the *up* and *turn* signals are off, and another header packet is in the column register. It is easy to see that the *up* signal cannot be turned on without invalidating the *unless*-part of the previous property. This means that we need to find a mechanism preventing the *up* signal to be turned on in this case. Even though it is possible to do it, we do not want to establish any priority between the two signals, in order to have a non-deterministic behavior. The solution to this problem is to transform *(P11.2.4.1)* in the following way:

$$\sigma[v]^{h}@(p,q,0) \wedge v=1 \wedge \neg up(p,q)\ \textbf{ensures}\ (\sigma[v]@(p,q,0) \wedge turn(p,q)) \vee up(p,q) \qquad (P11.2.4.1.1)$$

which allows the *up* signal to be turned on. A similar transformation on *(P11.2.6.1)* leads to:

$$\sigma[v]^{h}@(p,q,1) \wedge p\leq N \wedge \neg turn(p,q)\ \textbf{ensures}\ (\sigma[v]@(p,q,1) \wedge up(p,q)) \vee turn(p,q). \qquad (P11.2.6.1.1)$$

The final and complete specification S_6 is:

Message Representation

$$\textbf{inv}\ \langle \exists l :: \alpha@l\rangle \wedge k=\langle \Sigma\ \beta,l : \beta@l \wedge mid.\beta=mid.\alpha :: 1\rangle$$
$$\Rightarrow \qquad (P1)$$
$$k\geq 3 \wedge (pnr.\alpha=1 \Leftrightarrow type.\alpha=h) \wedge (1<pnr.\alpha<k \Leftrightarrow type.\alpha=b) \wedge (pnr.\alpha=k \Leftrightarrow type.\alpha=t)$$

$$\textbf{const}\ \langle \exists v,l :: \sigma[v]^{h}@l\rangle \qquad (P2)$$
$$\textbf{const}\ \langle \exists l :: \sigma[v]^{b}@l\rangle \qquad (P3)$$
$$\textbf{const}\ \langle \exists v,l :: \sigma[v]^{t}@l\rangle \qquad (P4)$$

Message Location

$$\textbf{inv}\ \alpha@(p,q,r) \wedge \beta@(p',q',r') \Rightarrow (pid.\alpha=pid.\beta \Leftrightarrow (p,q,r)=(p',q',r')) \qquad (P5.1)$$
$$\textbf{inv}\ \alpha@(p,q,r) \Rightarrow (p=src.\alpha \wedge q\leq dest.\alpha \wedge r=0) \vee (p>src.\alpha \wedge q=dest.\alpha \wedge r=1) \qquad (P6.1)$$

No-Reordering Property

inv $\alpha<\beta \Rightarrow \text{pid}.\beta<\text{pid}.\alpha$ (P9.1)

Non-Interleaving Property

inv $\alpha<\beta<\delta \wedge \text{mid}.\alpha=\text{mid}.\delta \Rightarrow \text{mid}.\beta=\text{mid}.\alpha$ (P10.1)

Signal Properties

inv $\neg(\text{turn}(p,q) \wedge \text{up}(p,q))$ (P12)

inv $q\geq 1 \wedge \langle\exists p',q',r',q'' : q''\leq q<q' :: \text{message}.(p',q',r').(p,q'',0)\rangle \Rightarrow \neg\text{turn}(p,q)$ (P13)
inv $q\geq 1 \wedge \langle\exists p',q' : p'>p \wedge q'\leq q :: \text{message}.(p',q,1).(p,q',0)\rangle \Rightarrow \text{turn}(p,q)$ (P14)
inv $p\leq N \wedge \langle\exists p',p'',q'',r'': p'>p \wedge (p''<p \vee (p''=p \wedge r''=1)) :: \text{message}.(p',q,1).(p'',q'',r'')\rangle \Rightarrow \text{up}(p,q)$ (P15)

inv $\text{turn}(p,q) \Rightarrow \langle\exists p',q',r' : ((r'=0 \wedge p'=p) \vee (r'=1 \wedge p'>p)) \wedge q'\leq q :: \text{message}.(p',q,r').(p,q',0)\rangle$ (P16)
inv $\text{up}(p,q) \Rightarrow \langle\exists p',p'',q'',r'': p'\geq p \wedge (p''<p \vee (p''=p \wedge r''=1)) :: \text{message}.(p',q,1).(p'',q'',r'')\rangle$ (P17)

$\alpha^h@(p,q,0) \wedge q=\text{dest}.\alpha \wedge \text{turn}(p,q)$ **unless** $\neg\alpha@(p,q,0)$ (P18)
$\alpha^h@(p,q,1) \wedge p\leq N \wedge \text{up}(p,q)$ **unless** $\neg\alpha@(p,q,1)$ (P19)

$\alpha^t@(p,q,0) \wedge q\geq 1 \wedge \text{turn}(p,q)$ **unless** $\neg\alpha@(p,q,0) \wedge \neg\text{turn}(p,q)$ (P20)
$\alpha^t@(p,q,1) \wedge p\leq N \wedge \text{up}(p,q)$ **unless** $\neg\alpha@(p,q,1) \wedge \neg\text{up}(p,q)$ (P21)

$\alpha^t@(p,q,0) \wedge q\geq 1 \wedge \text{turn}(p,q) \wedge \beta^h@(p,q,1)$ **unless** $\neg\alpha@(p,q,0) \wedge \text{up}(p,q)$ (P22)
$\alpha^t@(p,q,1) \wedge p\leq N \wedge \text{up}(p,q) \wedge \beta^h@(p,q,0) \wedge q=\text{dest}.\beta$ **unless** $\neg\alpha@(p,q,1) \wedge \text{turn}(p,q)$ (P23)

Header Value Invariant

inv $\sigma[v]^h@(p,q,r) \wedge q\geq 1 \Rightarrow v=\text{dest}.\sigma[v]-q+1$ (P24)

Packet Movement in the Environment

$\sigma[v]@(p,q,0) \wedge q\leq 0$ **until** $\sigma[v]@(p,q+1,0)$ (P11.1.1)
$\sigma[v]@(p,q,1) \wedge p\geq N+1$ **until** $\sigma[v]@(p+1,q,1)$. (P11.1.2)

Packet Movement inside the Router

$\sigma[v]@(p,q,r) \wedge p\leq N \wedge q\geq 1$ **unless** $\langle\exists v',p',q',r' : (p,q,r)\gg(p',q',r') :: \sigma[v']@(p',q',r')\rangle$ (P11.2.1)
$\sigma[v]@(p,q,r) \wedge p\leq N \wedge q\geq 1$ **unless** $\text{empty}@(p,q,r)$ (P25)

$\sigma[v]^h@(p,q,0) \wedge q\geq 1 \wedge v\neq 1 \wedge \text{empty}@(p,q+1,0)$ **ensures** $\sigma[v]@(p,q+1,0)$ (P11.2.2.1.1)
$\alpha[v]^\tau@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge \neg\text{turn}(p,q) \wedge \text{empty}@(p,q+1,0)$ **ensures** $\sigma[v]@(p,q+1,0)$ (P11.2.3.1.1)

$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge \neg\text{up}(p,q)$ **ensures** $(\sigma[v]@(p,q,0) \wedge \text{turn}(p,q)) \vee \text{up}(p,q)$ (P11.2.4.1.1)
$\sigma[v]@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge \text{turn}(p,q) \wedge \text{empty}@(p+1,q,1)$ **ensures** $\sigma[v]@(p+1,q,1)$ (P11.2.5.1.1)
$\sigma[v]@(N,q,0) \wedge q\geq 1 \wedge \text{turn}(N,q)$ **ensures** $\sigma[v]@(N+1,q,1)$ (P11.2.5.1.2)

$\sigma[v]^h@(p,q,1) \wedge p\leq N \wedge \neg\text{turn}(p,q)$ **ensures** $(\sigma[v]@(p,q,1) \wedge \text{up}(p,q)) \vee \text{turn}(p,q)$ (P11.2.6.1.1)
$\sigma[v]@(p,q,1) \wedge p<N \wedge \text{up}(p,q) \wedge \text{empty}@(p+1,q,1)$ **ensures** $\sigma[v]@(p+1,q,1)$ (P11.2.7.1.1)
$\sigma[v]@(N,q,1) \wedge \text{up}(N,q)$ **ensures** $\sigma[v]@(N+1,q,1)$ (P11.2.7.1.2)

Proof Obligations. We need to show that properties *(P11.2.1.1)-(P11.2.7.1)* are implied by S_6.

Proof Outline. The proof that packets inside the router eventually move to the next location—properties *(P11.2.2.1)*, *(P11.2.3.1)*, *(P11.2.5.1)*, and *(P11.2.7.1)*—can be decomposed into three cases: packet movement along the columns, from the rows to the columns, and finally along the rows. The movement along the columns can be proved by induction on the row number. The base case is established by the property *(P11.2.7.1.2)* which directly implies the packet movement from the upper locations *(N,q,1)* to the output queues. Since a packet leaves an empty location behind it when it moves, the base case and the property *(P11.2.5.1.1)* allow us to prove that packets at locations *(N-1,q,1)* will move to locations *(N,q,1)*, and so on, down to row 1.

Packet movement from the row *N* to the output environment is directly implied by *(P11.2.5.1.2)*. For *p* between 1 and *N-1*, we have just proved that packets along the columns are guaranteed to move to the next location. This implies that locations along the columns will eventually be emptied, and thus, according to *(P11.2.5.1.1)*, that packets at their turning switches will eventually move to their destination columns.

The packet movement along the rows can be proved by induction on the column number. We know that packets at the right end of the rows (locations *(p,M,0)*) are necessarily at their turning switches. They will thus move to the columns and leave empty locations behind them. This will allow packets at locations *(p,M-1,0)* to move to locations *(p,M,0)* (properties *(P11.2.2.1.1)* and *(P11.2.3.1.1)*), and so on, down to column 1.

Finally, we need to prove that the proper signals are eventually turned on when header packets show up in the registers (properties *(P11.2.4.1)* and *(P11.2.6.1)*). From *(P11.2.4.1.1)* we know that the *turn* signal will eventually be turned on, unless the *up* signal is turned on. In this case, the fairness property *(P23)* assures that the *turn* signal will also be turned on, as the tail of the message moving along the column will pass through the switch. The proof of *(P11.2.6.1)* is symmetrical.

4. Derivation of a Program from the Specification

The final step of the design consists of writing the program text. We first define the types and data structures used in the program, and then derive the program statements from the progress properties of the final specification.

4.1. Types and Data Structures of the Program

We define the type *packet* to be the Cartesian product between *{h,b,t}* and the set *V* of all the possible packet values. If *p* is a variable of type *packet*, we use the notation *p.1* to access the type, and *p.2* to access the value. The grid of *N×M* switches is implemented by a 3-dimensional array:

switch : array[1..N,1..M,0..1] of packet ∪ {⊥}

such that *switch[p,q,0]* represents the row register of switch *(p,q)* and *switch[p,q,1]* the column register. When a register does not contain any packet, we assume that its value is equal to ⊥. We define functions *header*, *body*, *tail*, *empty*, and *val* in the following way:

```
header.(p,q,r)  ≡ (switch[p,q,r].1=h),
body.(p,q,r)    ≡ (switch[p,q,r].1=b),
tail.(p,q,r)    ≡ (switch[p,q,r].1=t),
empty.(p,q,r)   ≡ (switch[p,q,r]=⊥),
val.(p,q,r)     ≡ switch[p,q,r].2 .
```

We also define the function *dec_val* which accepts a header packet *(h,v)* as argument and returns the packet *(h,v-1)*. The switch signals *turn* and *up* are implemented by two Boolean arrays:

turn,up : array[1..N,1..M] of Boolean

and the *N* input queues and *M* output queues are represented by two arrays of sequences of packets:

```
input   : array[1..N] of sequence of packet,
output  : array[1..M] of sequence of packet.
```

We suppose that the output queues are initially empty, and that the input queues contain all the packets that have to be sent. Finally, we use the following operations on sequences:

```
hd.s   ≡ head element of the sequence s,
tl.s   ≡ tail sequence of the sequence s,
(s;x)  ≡ sequence obtained by appending the element x at the end of the sequence s,
nil.s  ≡ s is an empty sequence.
```

4.2. Program Text

Let us first briefly describe the UNITY program notation. As we stated at the beginning of the paper, a typical UNITY program consists of a *declare* section, which contains Pascal-style declarations of variables; an *initially* section, where all or some of the variables are initialized; and an *assign* section, which is a set of multiple assignment statements separated by the operator "[]". The statements may be of the form:

var_list := exp_list

or may be conditional:

var_list_1 := exp_list_1 **if** bexp_1 ~ exp_list_2 **if** bexp_2 ~ ...

Several statements may be composed with the parallel bar to form a bigger statement:

statement_1 || statement_2 || ...

Finally, it is possible to generate a list of statements by using the following constructor:

⟨[] dummy_variables : range_constraint :: statement⟩.

The program derived from the specification is as follows (we omitted the *declare* section):

```
Program Message Router
initially
    ⟨[] p,q : 1≤p≤N ∧ 1≤q≤M :: switch[p,q,0],switch[p,q,1],turn[p,q],up[p,q] := ⊥,⊥,false,false⟩
assign
    {Packet movement from the input to the router: property (P11.1.1)}
    ⟨[] p : 1≤p≤N :: switch[p,1,0], input[p] := hd.input[p], tl.input[p]  if  empty.(p,1,0) ∧ ¬nil.input[p]⟩
[]  {Packet movement along the rows: properties (P11.2.2.1.1) and (P11.2.3.1.1)}
    ⟨[] p,q : 1≤p≤N ∧ 1≤q<M ::
        switch[p,q,0], switch[p,q+1,0] :=
            ⊥, dec_val.switch[p,q,0]  if  header.(p,q,0) ∧ val.(p,q,0)≠1 ∧ empty.(p,q+1,0)  ~
            ⊥, switch[p,q,0]          if  (body.(p,q,0) ∨ tail.(p,q,0)) ∧ ¬turn[p,q] ∧ empty.(p,q+1,0)⟩
[]  {Packet movement from the rows (<N) to the columns: properties (P11.2.5.1.1), (P20), and (P22)}
    ⟨[] p,q : 1≤p<N ∧ 1≤q≤M ::
        switch[p,q,0], switch[p+1,q,1]  := ⊥, switch[p,q,0]  if  turn[p,q] ∧ ¬empty.(p,q,0) ∧ empty.(p+1,q,1)
      ||                    turn[p,q]  := false  if  turn[p,q] ∧ tail.(p,q,0) ∧ empty.(p+1,q,1)
      ||                      up[p,q]  := true  if  turn[p,q] ∧ tail.(p,q,0) ∧ empty.(p+1,q,1) ∧ header.(p,q,1)⟩
[]  {Packet movement from row N to the output: properties (P11.2.5.1.2), (P20), and (P22)}
    ⟨[] q : 1≤q≤M ::
        switch[N,q,0], output[q]  := ⊥, (output[q];switch[N,q,0])  if  turn[N,q] ∧ ¬empty.(N,q,0)
      ||              turn[N,q]  := false  if  turn[N,q] ∧ tail.(N,q,0)
      ||                up[N,q]  := true  if  turn[N,q] ∧ tail.(N,q,0) ∧ header.(N,q,1)⟩
[]  {Packet movement along the columns inside the router: properties (P11.2.7.1.1), (P21), and (P23)}
    ⟨[] p,q : 1≤p<N ∧ 1≤q≤M ::
        switch[p,q,1], switch[p+1,q,1]  := ⊥, switch[p,q,1]  if  up[p,q] ∧ ¬empty.(p,q,1) ∧ empty.(p+1,q,1)
      ||                      up[p,q]  := false  if  up[p,q] ∧ tail.(p,q,1) ∧ empty.(p+1,q,1)
      ||                    turn[p,q]  := true  if  up[p,q] ∧ tail.(p,q,1) ∧ empty.(p+1,q,1) ∧ header.(p,q,0)
                                                  ∧ val.(p,q,0)=1⟩
[]  {Packet movement from the columns to the output: properties (P11.2.7.1.2), (P21), and (P23)}
    ⟨[] q : 1≤q≤M ::
        switch[N,q,1], output[q]  := ⊥, (output[q];switch[N,q,1])  if  up[N,q] ∧ ¬empty.(N,q,1)
      ||                up[N,q]  := false  if  up[N,q] ∧ tail.(N,q,1)
      ||              turn[N,q]  := true  if  up[N,q] ∧ tail.(N,q,1) ∧ header.(N,q,0) ∧ val.(N,q,0)=1⟩
[]  {Signal changes: Properties (P11.2.4.1.1) and (P11.2.6.1.1)}
    ⟨[] p,q : 1≤p≤N ∧ 1≤q≤M :: turn[p,q] := true  if ¬turn[p,q] ∧ ¬up[p,q] ∧ header.(p,q,0) ∧ val.(p,q,0)=1⟩
[]  ⟨[] p,q : 1≤p≤N ∧ 1≤q≤M :: up[p,q] := true  if ¬up[p,q] ∧ ¬turn[p,q] ∧ header.(p,q,1)⟩
end
```

Initially, all the row and column registers are empty, and the *turn* and *up* signals are off. The packet movement in the input queues (property *(P11.1.1)*) is implemented by the first statement of the *assign* section. As long as there exists an input queue that is not empty, its head element is removed and assigned to the first register on the row, if it is empty. This implies that all the packets in the queue simultaneously move forward from one position.

The second statement takes care of the packet movement along the rows. It was trivially derived from the properties *(P11.2.2.1.1)* and *(P11.2.3.1.1)*.

The movement from the lower rows (below row *N*) to the columns is realized by the third statement, which consists of three components. The first component was trivially derived from the progress property *(P11.2.5.1.1)*. The second component makes sure that the *turn* signal is turned off when the tail of the message is passing through the switch. It was suggested by the safety property *(P20)*, or more precisely by the conjunction of *(P20)* and *(P11.2.5.1.1)*. The third component, which was derived from the conjunction of *(P22)* and *(P11.2.5.1.1)*, makes sure that the *up* signal is turned on when the tail of the message is passing through the switch and a header packet is waiting in the column register. The packet movement along the columns inside the router (fifth statement) was derived from the properties *(P11.2.7.1.1)*, *(P21)*, and *(P23)* in a similar way.

The movement of the packets from the upper locations inside the router—either on the rows (fourth statement) or on the columns (sixth statement)— is achieved by just appending the outgoing packet at the end of the output queue. Note that this implies that packets simultaneously move forward from one position in the output queue, and thus satisfies the property *(P11.1.2)*.

Finally, the properties *(P11.2.4.1.1)* and *(P11.2.6.1.1)*, specifying that the *turn* or *up* signals are turned off when a header packet shows up in the row or column registers, suggested the last two statements. To be complete, we also need to show that each statement preserves the safety and invariant properties of the specification. We omit the proofs here since they can be verified easily using the program text.

5. Conclusion

This paper presents the formal derivation of a message router. The refinement method is that of UNITY. Although UNITY-style formal derivations have appeared in print before, the questions addressed by this case study are different and the lessons learnt bear careful scrutiny and further investigation. Others have been concerned with the issue of whether formal derivation can be applied successfully to sophisticated problems, such as the router. We are intrigued by the possibility that careful management of the refinement process may render formal derivation capable of supporting industrial-grade applications. Most of our effort went not into the router derivation per se but, in fine tuning the derivation process. In earlier sections we described the ultimate outcome of this study, a series of refinements and their motivation. Now we turn our attention to those elements of the derivation process that have been instrumental in shaping the specification style and the derivation strategy.

Early on we observed that it is better to specify and reason about a closed system—i.e., a system and its environment—rather than an open one. In the latter case, conditional properties make specifications more complex and proofs more difficult. In the router example, we have been able to specify a closed system by representing the input and output environments as infinite input and output queues, with the input queues initially containing all the packets that had to be sent. The unified formal treatment of the system and its environment reduced complexity without compromising the desire to separate concerns, the two types of properties were simply identified as addressing distinct issues. Naturally, only system properties were refined while the environmental properties were left unchanged.

Another critical element is the formulation of the top-level specification. On one hand, it should be as general as possible so as not to restrict prematurely the range of possible designs; on the other hand, it ought to make the refinements simple. The former is achievable by focusing the top-level specification on input/output properties alone while the latter is facilitated by selecting the "right" notation. For instance, by identifying packet locations in the environment as pairs of coordinates (row,column) the refinement of the router as a grid of switches became very natural. One does not stumble upon appropriate notation. Experience, exploration, and some looking ahead can provide the required insight. Derivation papers are often criticized for fortuitous early decisions which seem to indicate that the authors already have seen the final solution. Such objections are valid only if one claims that the nature of the specification dictates (in some mechanistic way) the next refinement step. True design, however, never makes such pretense. Looking ahead and backtracking are part of the method. Insights gained from considering various alternatives ultimately lead—relatively early in the process—to the notation we adopted and to the choice of auxiliary variables: *n* (source row), *m* (destination column), *i* (message number), and *j* (packet number). The selection of auxiliary variables was one of the key decisions we had to make; it enabled simple formulation of many central properties such as those involving message location, no-reordering, and non-interleaving. Note, for instance, that using the destination column variable in the early stages of the design allowed us to deal with the value of the header packets only in the fourth refinement.

The scope of the individual refinements is another issue affecting the ease with which the derivation can be explained to others and verified formally. As expected, small refinements proved helpful in both respects. They also seem to help the designer avoid premature commitments. The refinement process can be viewed as a tree, where internal nodes represent specifications, and leafs represent programs. The more internal nodes, the more leafs at the bottom of the tree. This conservative strategy leads to a broader exploration of alternatives and a decrease in the likelihood that not all implications of each design decision are well understood and considered.

Finally, we discovered that it was relatively easy to separate the formal treatment of the proofs from the refinement process itself. We spent a lot of investigation time on choosing the right top-level specification, the right notation and auxiliary variables, and the right sequence of refinements. During all this time, we came up with different solutions, but we hardly ever felt the need to formally verify the correctness of the refinements we generated—only at the end of the design, when all the design decisions were set, we generated the required formal proofs for each refinement step. This made us conclude that the refinement process is sufficiently intuitive to allow a rigorous design methodology to proceed without the burden of unnecessary and cumbersome proofs. This also means that design and verification can actually be carried out by different people. People with synthetic skills could focus their energy on the design while people with strong analytical skills could deal mostly with the proofs, often without even requiring an understanding of the design details. These observations strengthen our belief that formal methods may soon play an important role in the development of industrial-grade software systems.

References

[1] Back, R. J. R., and Sere, K., "Stepwise Refinement of Parallel Algorithms," *Science of Computer Programming*, vol. 13, no. 2-3, pp. 133-180, 1990.

[2] Chandy, K. M., and Misra, J., *Parallel Program Design: A Foundation*, Addison-Wesley, New York, NY, 1988.

[3] Cunningham, H. C., and Roman, G.-C., "A UNITY-Style Programming Logic for a Shared Dataspace Language," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 365-376, 1990.

[4] Dijkstra, E. D., *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[5] Dijkstra, E. W., and Fiejen, W. H. J., *A Method of Programming*, Addison-Wesley, Wokingham, England, 1988.

[6] Ebergen, J. C., and Hoogerwoord, R. R., "A Derivation of a Serial-Parallel Multiplier," *Science of Computer Programming*, vol. 15, pp. 201-215, 1990.

[7] Gries, D., *The Science of Programming*, Springer-Verlag, New York, NY, 1981.

[8] Gries, D., and Prins, J., "A New Notion of Encapsulation," *ACM SIGPLAN NOTICES*, vol. 20, no. 6, pp. 131-139, 1985.

[9] Hoogerwoord, R. R., "A Calculational Derivation of the CASOP Algorithm," *Information Processing Letters*, vol. 36, pp. 297-299, 1990.

[10] Knapp, E., "An Exercise in the Formal Derivation of Parallel Programs: Maximum Flows in Graphs," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 2, pp. 203-223, 1990.

[11] Lengauer, C., "A Methodology for Programming with Concurrency: the Formalism," in *Science of Computer Programming*, North-Holland, vol. 2, pp. 19-52, 1982.

[12] Lengauer, C., and Hehner, E. C. R., "A Methodology for Programming with Concurrency: an Informal Presentation," in *Science of Computer Programming*, North-Holland, vol. 2, pp. 1-18, 1982.

[13] Misra, J., "General Conjunction and Disjunction Rules for *unless*," Dept of CS, The University of Texas at Austin, Austin, TX 78712, Notes on UNITY 01-88, 1988.

[14] Misra, J., "Stable Conjunction," Dept of CS, The University of Texas at Austin, Austin, TX 78712, Notes on UNITY 21-90, 1990.

[15] Morgan, C. C., "The specification statement," *ACM TOPLAS*, vol. 10, pp. 403-419, 1988.

[16] Morris, J. M., "Laws of data refinement," *Acta Informatica*, vol. 26, pp. 287-308, 1989.

[17] Roman, G.-C., and Cunningham, H. C., "Mixed Programming Metaphors in a Shared Dataspace Model of Concurrency," *IEEE Transactions on Software Engineering*, vol. 16, no. 12, pp. 1361-1373, 1990.

[18] Roman, G.-C., and Cunningham, H. C., "Reasoning about Synchronic Groups," in *Research Directions in High-Level Parallel Programming Languages*, J. P. Banâtre, D. L. Métayer, Eds., Springer-Verlag, New York, NY, vol. 574, pp. 21-38, 1992.

[19] Roman, G.-C., Wilcox, C. D., and Plun, J. Y., "On Deriving Distributed Programs from Formal Specifications of Functional Requirements and Architectural Constraints," *12th International Conference on Distributed Computing Systems.*, pp. 494-501, 1991.

[20] Staskauskas, M., "A Formal Specification and Design of a Distributed Electronic Funds-Transfer Network," *IEEE Transactions on Computers*, vol. 37, no. 12, pp. 1515-1528, 1988.

A. FORMAL DEFINITION OF THE UNITY LOGIC OPERATORS

The definition of the UNITY operators is based on the Hoare triple:

$\{p\}\, s\, \{q\}$

which means that, starting in a state satisfying the precondition *p*, the execution of the statement *s* ends in a state satisfying the postcondition *q*. Let *F* be a UNITY program, *F.assign* be the assign the section of *F*, and *INIT* a predicate denoting the initial state of *F*. The UNITY operators are formally defined in the following way:

- p **unless** $q \equiv \langle \forall s : s \in F.assign :: \{p \wedge \neg q\}\ s\ \{p \vee q\} \rangle$.

  Whenever the predicate *p* holds for a program state, it continues to hold at least until *q* holds.

- **stable** $p \equiv \langle \forall s : s \in F.assign :: \{p\}\ s\ \{p\} \rangle \equiv p$ **unless** false.

  The predicate *p* is a *stable* predicate if it remains true forever once it becomes true.

- **const** $p \equiv$ **stable** $p \wedge$ **stable** $\neg p$.

  The predicate *p* is *constant* if it remains true forever if it is initially true, and false forever if it is initially false.

- **inv** $p \equiv (INIT \Rightarrow p) \wedge$ **stable** p.

  The predicate *p* is *invariant* if it holds in the initial state, and is stable all along the execution.

- p **ensures** $q \equiv p$ **unless** $q \wedge \langle \exists s : s \in F.assign :: \{p \wedge \neg q\}\ s\ \{q\} \rangle$.

  Whenever *p* holds, it must hold at least until *q* holds (*p* ***unless*** *q*), and there must exist a statement in the program that establishes the truth of *q*.

- $p \mapsto q$.

  The assertion $p \mapsto q$ (*p* leads to *q*) is true iff it can be derived by a finite number of applications of the following inference rules:

$$\frac{p \textbf{ ensures } q}{p \mapsto q}$$

$$\frac{p \mapsto q\ ,\ q \mapsto r}{p \mapsto r} \qquad \text{(transitivity)}$$

$$\frac{\langle \forall m : m \in W :: p(m) \mapsto q \rangle}{\langle \exists m : m \in W :: p(m) \rangle \mapsto q} \qquad \text{for any set W (disjunction)}$$

B. FORMAL PROOFS OF THE REFINEMENTS

We give here the proofs of the first, second, and sixth refinement. The proofs of the other refinements, which are trivial, have been done in the body of the paper. We sometimes give numbers to properties, using the following conventions: numbers *(DP1)*, *(DP2)*, ... are used for properties that are derived from the specifications; numbers *(Q1)*, *(Q2)*, ... are used for intermediate properties that have to be proved; numbers *(R1)*, *(R2)*, ... are finally used for results that have been proved. The numbering starts from 1 at each new refinement proof. Finally, let us mention that the UNITY theorems that we use below are listed in appendix C.

B.1. Proof of Refinement 1

We need to show that specification S_1 implies specification S_0, i.e., properties *(P1)* through *(P11)*.

▼ Proof of Property (P1)

Although properties *(P1)* through *(P4)* are not syntactically changed in the refinement, the definition of the function @ has changed, which implies that they need to be verified. To avoid confusion, we call them *(P'1)*, *(P'2)*, *(P'3)*, and *(P'4)* in S_1, and we explicitly use functions Π and Π' in the formulas. The property *(P1)* is:

$$\mathbf{inv}\ \langle \exists l :: \Pi.\alpha{=}l\rangle \wedge k{=}\langle \Sigma\ \beta,l : \Pi.\beta{=}l \wedge mid.\beta{=}mid.\alpha :: 1\rangle \\ \Rightarrow \\ k{\geq}3 \wedge (pnr.\alpha{=}1 \Leftrightarrow type.\alpha{=}h) \wedge (1{<}pnr.\alpha{<}k \Leftrightarrow type.\alpha{=}b) \wedge (pnr.\alpha{=}k \Leftrightarrow type.\alpha{=}t). \qquad (P1)$$

Let's show that *(P'1)* is equivalent to *(P1)*. For this, we need to show:

$$\langle \exists l :: \Pi'.\alpha{=}l\rangle \Leftrightarrow \langle \exists l :: \Pi.\alpha{=}l\rangle \qquad (Q1)$$

and

$$\langle \Sigma\ \beta,l : \Pi'.\beta{=}l \wedge mid.\beta{=}mid.\alpha :: 1\rangle = \langle \Sigma\ \beta,l : \Pi.\beta{=}l \wedge mid.\beta{=}mid.\alpha :: 1\rangle. \qquad (Q2)$$

Proof of Property (Q1)

$\langle \exists l :: \Pi'.\alpha{=}l\rangle$

$\Leftrightarrow$ {definition of E' and R}

$\quad \langle \exists p,q,r : 1{\leq}p{\leq}N \wedge 1{\leq}q{\leq}M \wedge 0{\leq}r{\leq}1 :: \Pi'.\alpha{=}(p,q,r)\rangle$
$\vee\ \langle \exists p,q : 1{\leq}p{\leq}N \wedge q{\leq}0 :: \Pi'.\alpha{=}(p,q,0)\rangle$
$\vee\ \langle \exists p,q : p{\geq}N{+}1 \wedge 1{\leq}q{\leq}M :: \Pi'.\alpha{=}(p,q,1)\rangle$

$\Leftrightarrow$ {coupling invariant}

$\quad \langle \exists p,q,r : 1{\leq}p{\leq}N \wedge 1{\leq}q{\leq}M \wedge 0{\leq}r{\leq}1 :: \Pi.\alpha{=}(p,q,r)\rangle$
$\vee\ \langle \exists p,q : 1{\leq}p{\leq}N \wedge q{\leq}0 :: \Pi.\alpha{=}(p,q)\rangle$
$\vee\ \langle \exists p,q : p{\geq}N{+}1 \wedge 1{\leq}q{\leq}M :: \Pi.\alpha{=}(p,q)$

$\Leftrightarrow$ {definition of E and R}

$\langle \exists l :: \Pi.\alpha{=}l\rangle$. ❑

Proof of Property (Q2)

$\langle \Sigma\ \beta,l : \Pi'.\beta{=}l \wedge mid.\beta{=}mid.\alpha :: 1\rangle$

$=$ {definition of E' and R}

$\quad \langle \Sigma\ \beta,p,q,r, : 1{\leq}p{\leq}N \wedge 1{\leq}q{\leq}M \wedge 0{\leq}r{\leq}1 \wedge \Pi'.\beta{=}(p,q,r) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$
$+\ \langle \Sigma\ \beta,p,q : 1{\leq}p{\leq}N \wedge q{\leq}0 \wedge \Pi'.\beta{=}(p,q,0) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$
$+\ \langle \Sigma\ \beta,p,q : p{\geq}N{+}1 \wedge 1{\leq}q{\leq}M \wedge \Pi'.\beta{=}(p,q,1) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$

$=$ {coupling invariant}

$\quad \langle \Sigma\ \beta,p,q,r, : 1{\leq}p{\leq}N \wedge 1{\leq}q{\leq}M \wedge 0{\leq}r{\leq}1 \wedge \Pi.\beta{=}(p,q,r) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$
$+\ \langle \Sigma\ \beta,p,q : 1{\leq}p{\leq}N \wedge q{\leq}0 \wedge \Pi.\beta{=}(p,q,0) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$
$+\ \langle \Sigma\ \beta,p,q : p{\geq}N{+}1 \wedge 1{\leq}q{\leq}M \wedge \Pi.\beta{=}(p,q,1) \wedge mid.\beta{=}mid.\alpha :: 1\rangle$

$=$ {definition of E and R}

$\langle \Sigma\ \beta,l : \Pi.\beta{=}l \wedge mid.\beta{=}mid.\alpha :: 1\rangle$. ❑

The proofs of properties *(P2)*, *(P3)*, and *(P4)* are similar.

▼ Proof of Property (P5)

In the environment, each packet has a unique location, and each location contains at most one packet:

$$\mathbf{inv}\ \alpha@(p,q) \wedge (q{\leq}0 \vee p{\geq}N{+}1) \wedge \beta@(p',q') \wedge (q'{\leq}0 \vee p'{\geq}N{+}1) \Rightarrow (pid.\alpha{=}pid.\beta \Leftrightarrow (p,q){=}(p',q')). \qquad (P5)$$

Let us start from property *(P5.1)*:

$\textbf{inv}\ \alpha@(p,q,r) \wedge \beta@(p',q',r') \Rightarrow (pid.\alpha{=}pid.\beta \Leftrightarrow (p,q,r){=}(p',q',r'))$

$\Rightarrow$ {predicate calculus}

$\textbf{inv}\ \alpha@(p,q,r) \wedge [(r{=}0 \wedge q{\le}0) \vee (r{=}1 \wedge p{\ge}N{+}1)] \wedge \beta@(p',q',r') \wedge [(r'{=}0 \wedge q'{\le}0) \vee (r'{=}1 \wedge p'{\ge}N{+}1)]$

$\Rightarrow$

$(pid.\alpha{=}pid.\beta \Leftrightarrow (p,q,r){=}(p',q',r'))$

$\Rightarrow$ {coupling invariant}

$\textbf{inv}\ \alpha@(p,q) \wedge (q{\le}0 \vee p{\ge}N{+}1) \wedge \beta@(p',q') \wedge (q'{\le}0 \vee p'{\ge}N{+}1) \Rightarrow (pid.\alpha{=}pid.\beta \Leftrightarrow (p,q){=}(p',q'))$.

The truth of *(P6)* can easily be derived from *(P6.1)* in the same way.

▼ **Proof of Property (P7)**

Packets are sent in the order they are queued:

$\langle \exists\, q,q' : q{<}q'{\le}0 :: \alpha@(p,q) \wedge \beta@(p,q')\rangle\ \textbf{unless}\ \neg\langle \exists\, q' : q'{\le}0 :: \beta@(p,q')\rangle.$ (P7)

From *(P11.1)*, *(P6.1)*, and the definition of "»", we easily get:

$\alpha@(p,q,0) \wedge q{\le}0\ \textbf{unless}\ \alpha@(p,q{+}1,0) \wedge q{\le}0$

and

$\beta@(p,q',0) \wedge q'{\le}0\ \textbf{unless}\ \beta@(p,q'{+}1,0) \wedge q'{\le}0.$

By applying the conjunction rule, and restricting *q* to be smaller than *q'*, we get:

$\alpha@(p,q,0) \wedge \beta@(p,q',0) \wedge q{<}q'{\le}0$

$\textbf{unless}$

$[(\alpha@(p,q,0) \wedge \beta@(p,q'{+}1,0)) \vee (\alpha@(p,q{+}1,0) \wedge \beta@(p,q',0)) \vee (\alpha@(p,q{+}1,0) \wedge \beta@(p,q'{+}1,0))] \wedge q{<}q'{\le}0$

$\Rightarrow$ {consequence weakening}

$\alpha@(p,q,0) \wedge \beta@(p,q',0) \wedge q{<}q'{\le}0$

$\textbf{unless}$

$\langle \exists\pi,\pi' : \pi{\le}\pi'{\le}1 \wedge \neg(\pi{=}q \wedge \pi'{=}q') :: \alpha@(p,\pi,0) \wedge \beta@(p,\pi',0)\rangle$

$\Leftrightarrow$ {each location contains at most one packet (property *(P5.1)*)}

$\alpha@(p,q,0) \wedge \beta@(p,q',0) \wedge q{<}q'{\le}0$

$\textbf{unless}$

$\langle \exists\pi,\pi' : \pi{<}\pi'{\le}1 \wedge \neg(\pi{=}q \wedge \pi'{=}q') :: \alpha@(p,\pi,0) \wedge \beta@(p,\pi',0)\rangle$

$\Leftrightarrow$ {each packet has a unique location (property *(P5.1)*)}

$\alpha@(p,q,0) \wedge \beta@(p,q',0) \wedge q{<}q'{\le}0$

$\textbf{unless}$

$\neg(\alpha@(p,q,0) \wedge \beta@(p,q',0)) \wedge \langle \exists\pi,\pi' : \pi{<}\pi'{\le}1 :: \alpha@(p,\pi,0) \wedge \beta@(p,\pi',0)\rangle$

$\Leftrightarrow$ {introducing quantification on *q* and *q'*, and naming the predicates}

$\langle \forall q,q' : q{<}q'{\le}0 :: P(q,q')\ \textbf{unless}\ \neg P(q,q') \wedge \langle \exists\pi,\pi' : \pi{<}\pi'{\le}1 :: P(\pi,\pi')\rangle\rangle$

$\Rightarrow$ {corollary of the general disjunction rule}

$\langle \exists q,q' : q{<}q'{\le}0 :: P(q,q')\rangle\ \textbf{unless}\ \langle \forall q,q' : q{<}q'{\le}0 :: \neg P(q,q')\rangle \wedge \langle \exists q,q' : q{<}q'{\le}1 :: P(q,q')\rangle$

$\Rightarrow$ {predicate calculus + consequence weakening}

$\langle \exists q,q' : q{<}q'{\le}0 :: P(q,q')\rangle\ \textbf{unless}\ \langle \exists q : q{<}1 :: P(q,1)\rangle$

$\Rightarrow$ {unfolding *P* + consequence weakening}

$\langle \exists q,q' : q{<}q'{\le}0 :: \alpha@(p,q,0) \wedge \beta@(p,q',0)\rangle\ \textbf{unless}\ \beta@(p,1,0)$

$\Rightarrow$ {consequence weakening: each packet has a unique location}

$\langle \exists q,q' : q{<}q'{\le}0 :: \alpha@(p,q,0) \wedge \beta@(p,q',0)\rangle\ \textbf{unless}\ \neg\langle \exists q' : q'{\le}0 :: \beta@(p,q',0)\rangle$

$\Leftrightarrow$ {coupling invariant}

(P7).

▼ Proof of Property (P8)

Packets received in the output queues can only appear behind the packets already present in the queues:

stable $\langle\exists p : p\geq N+1 :: \alpha@(p,q) \land \neg\langle\exists p' : p'>p :: \beta@(p',q)\rangle\rangle$. (P8)

We first prove some properties that will be useful along the demonstration.

Derived Property 1

Packets on the rows can only move to their destination column:

$\langle\exists v',p',q' :: \sigma'[v']@(p',q',0) \land q=dest.\sigma'[v']\rangle$ **unless** $\langle\exists v',p' : p'\leq N+1 :: \sigma'[v']@(p',q,1)\rangle$. (DP1)

Proof of Property (DP1)

From the unless part of *(P11.1)* and *(P11.2)*, and the definition of "$\gg$", we get:

$\sigma'[v']@(p',q',0) \land q=dest.\sigma'[v'] \land q'\neq q$ **unless** $\langle\exists v'' :: \sigma'[v'']@(p',q'+1,0)\rangle$

$\Rightarrow$ {consequence weakening: each packet has a unique location}

$\sigma'[v']@(p',q',0) \land q=dest.\sigma'[v'] \land q'\neq q$ **unless** $\neg\sigma'[v']@(p',q',0) \land \langle\exists v'',q'' :: \sigma'[v'']@(p',q'',0)\rangle$. (R1)

From *(P11.2)* and the definition of "$\gg$", we also get:

$\sigma'[v']@(p',q,0) \land q=dest.\sigma'[v']$ **unless** $\langle\exists v'' :: \sigma'[v'']@(p'+1,q,1) \land p'+1\leq N+1\rangle$

$\Rightarrow$ {consequence weakening: each packet has a unique location}

$\sigma'[v']@(p',q,0) \land q=dest.\sigma'[v']$ **unless** $\neg\sigma'[v']@(p',q,0) \land \langle\exists v'',p'' : p''\leq N+1 :: \sigma'[v'']@(p'',q,1)\rangle$

$\Rightarrow$ {simple disjunction with *(R1)*}

$\sigma'[v']@(p',q',0) \land q=dest.\sigma'[v']$
unless
$\neg\sigma'[v']@(p',q',0) \land (\langle\exists v'',q'' :: \sigma'[v'']@(p',q'',0)\rangle \lor \langle\exists v'',p'' : p''\leq N+1 :: \sigma'[v'']@(p'',q,1)\rangle)$

$\Rightarrow$ {corollary of the general disjunction rule on *v',p',q*}

$\langle\exists v',p',q' :: \sigma'[v']@(p',q',0) \land q=dest.\sigma'[v']\rangle$
unless
$\langle\forall v',p',q' :: \neg\sigma'[v']@(p',q',0)\rangle \land (\langle\exists v',p',q' :: \sigma'[v']@(p',q',0)\rangle \lor \langle\exists v',p' : p'\leq N+1 :: \sigma'[v']@(p',q,1)\rangle)$

$\Rightarrow$ {predicate calculus + consequence weakening}

$\langle\exists v',p',q' :: \sigma'[v']@(p',q',0) \land q=dest.\sigma'[v']\rangle$ **unless** $\langle\exists v',p' : p'\leq N+1 :: \sigma'[v']@(p',q,1)\rangle$. ❑

Derived Property 2

Packets stay in the output environment forever, once they reach it:

stable $\langle\exists p : p\geq N+1 :: \sigma[v]@(p,q,1)\rangle$. (DP2)

Proof of Property (DP2)

From *(P11.1)* and the definition of "$\gg$", we get:

$\sigma[v]@(p,q,1) \land p\geq N+1$ **unless** $\sigma[v]@(p+1,q,1) \land p\geq N+1$

$\Rightarrow$ {consequence weakening: each packet has a unique location}

$\sigma[v]@(p,q,1) \land p\geq N+1$ **unless** $\neg\sigma[v]@(p,q,1) \land p\geq N+1 \land \langle\exists p' : p'\geq N+2 :: \sigma[v]@(p',q,1)\rangle$

$\Rightarrow$ {corollary of the general disjunction rule on *p*}

$\langle\exists p : p\geq N+1 :: \sigma[v]@(p,q,1)\rangle$ **unless** $\langle\forall p : p\geq N+1 :: \neg\sigma[v]@(p,q,1)\rangle \land \langle\exists p : p\geq N+2 :: \sigma[v]@(p,q,1)\rangle$

$\Leftrightarrow$ {predicatecalculus}

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1)\rangle$ **unless** false. ❑

To prove (P11), we are going to show :

stable $\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \neg\langle \exists p' : p'>p :: \sigma'[v']@(p',q,1)\rangle\rangle$ (Q3)

which, according to the coupling invariant, is equivalent to *(P11)*. Since each packet has a unique value, the second conjunct is equivalent to:

$\neg\langle \exists v',p' : p'>p :: \sigma'[v']@(p',q,1)\rangle$

$\Leftrightarrow$ {predicatecalculus}

$\neg\langle \exists v',p' : p'>p :: \sigma'[v']@(p',q,1)\rangle \wedge (\neg\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle \vee \langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle)$

$\Leftrightarrow$ {predicatecalculus}

$\neg\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle \vee (\neg\langle \exists v',p' : p'>p :: \sigma'[v']@(p',q,1)\rangle \wedge \langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle)$.

The second disjunct is equivalent to:

$\neg\langle \exists v',p',q',r' : p'>p \wedge q'=q \wedge r'=1 :: \sigma'[v']@(p',q',r')\rangle \wedge \langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle$

$\Leftrightarrow$ {predicatecalculus}

$\langle \exists v',p',q',r' : \neg(p'>p \wedge q'=q \wedge r'=1) :: \sigma'[v']@(p',q',r')\rangle$

$\Leftrightarrow$ {predicatecalculus}

$\langle \exists v',p',q',r' : \neg(p'>p \wedge q'=q \wedge r'=1) :: \sigma'[v']@(p',q',r') \wedge (dest.\sigma'[v'] \neq q \vee dest.\sigma'[v']=q)\rangle$

$\Leftrightarrow$ {$dest.\sigma'[v'] \neq q \Rightarrow \neg(q'=q \wedge r'=1)$ + predicate calculus}

$\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r') \wedge dest.\sigma'[v'] \neq q\rangle$

$\vee \langle \exists v',p',q' :: (\sigma'[v']@(p',q',0) \wedge dest.\sigma'[v']=q) \vee (\sigma'[v']@(p',q,1) \wedge p' \leq p)\rangle$

So, the second conjunct of property *(Q3)* is equivalent to:

$\neg\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle$ (R2)
$\vee \langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r') \wedge dest.\sigma'[v'] \neq q\rangle$ (R3)
$\vee \langle \exists v',p',q' :: (\sigma'[v']@(p',q',0) \wedge dest.\sigma'[v']=q) \vee (\sigma'[v']@(p',q,1) \wedge p' \leq p)\rangle$. (R4)

To prove *(Q3)*, it is sufficient to show:

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge (R2)\rangle$ **unless** false (Q4)
$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge (R3)\rangle$ **unless** false (Q5)
$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge (R4)\rangle$ **unless** false. (Q6)

Proof of Property (Q4)

From *(P2)*, *(P3)*, *(P4)*, and the definition of ***const***, we get:

$\neg\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle$ **unless** false

$\Rightarrow$ {simple disjunction with *(DP2)*}

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1)\rangle \wedge \neg\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle$ **unless** false

$\Leftrightarrow$ {predicatecalculus}

(Q4). ❑

Proof of Property (Q5)

From *(P2)*, *(P3)*, *(P4)*, and the definition of ***const***, we get:

$\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r')\rangle$ **unless** false

$\Rightarrow$ {stable conjunction}

$\langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r') \wedge q \neq dest.\sigma'[v'] \rangle$ **unless** false

$\Rightarrow$ {simple disjunction with *(DP2)*}

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \rangle \wedge \langle \exists v',p',q',r' :: \sigma'[v']@(p',q',r') \wedge q \neq dest.\sigma'[v'] \rangle$ **unless** false

$\Leftrightarrow$ {predicatecalculus}

(Q5). ❑

Proof of Property (Q6)

By applying the conjunction rule on *(DP2)* and *(DP1)*, we get:

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \langle \exists v',p',q' :: \sigma'[v']@(p',q',0) \wedge q=dest.\sigma'[v'] \rangle \rangle$
unless
$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \langle \exists v',p' : p' \leq N+1 :: \sigma'[v']@(p',q,1) \rangle \rangle$

$\Rightarrow$ {consequence weakening: each location contains at most one packet}

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \langle \exists v',p',q' :: \sigma'[v']@(p',q',0) \wedge q=dest.\sigma'[v'] \rangle \rangle$
unless (R5)
$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \langle \exists v',p' : p'<p :: \sigma'[v']@(p',q,1) \rangle \rangle$.

From *(P11.1)*, *(P11.2)*, and the definition of "»", we get:

$\sigma[v]@(p,q,1) \wedge p \geq N+1$ **unless** $\sigma[v]@(p+1,q,1) \wedge p \geq N+1$

and

$\sigma'[v']@(p',q,1) \wedge p'<p$ **unless** $\langle \exists w' :: \sigma'[w']@(p'+1,q,1) \rangle \wedge p'<p$.

An application of the conjunction rule leads to :

$\sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1) \wedge p \geq N+1 \wedge p'<p$
unless
$[(\sigma[v]@(p,q,1) \wedge \langle \exists w' :: \sigma'[w']@(p'+1,q,1) \rangle)$
$\vee (\sigma[v]@(p+1,q,1) \wedge \sigma'[v']@(p',q,1))$
$\vee (\sigma[v]@(p+1,q,1) \wedge \langle \exists w' :: \sigma'[w']@(p'+1,q,1) \rangle)] \wedge p \geq N+1 \wedge p'<p$

$\Rightarrow$ {consequenceweakening}

$\sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1) \wedge p \geq N+1 \wedge p'<p$
unless
$\langle \exists w',\pi,\pi' : \pi \geq N+1 \wedge \pi' \leq \pi \wedge \neg(\pi=p \wedge \pi'=p') :: \sigma[v]@(\pi,q,1) \wedge \sigma'[w']@(\pi',q,1) \rangle$

$\Leftrightarrow$ {each location contains at most one packet}

$\sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1) \wedge p \geq N+1 \wedge p'<p$
unless
$\langle \exists w',\pi,\pi' : \pi \geq N+1 \wedge \pi'<\pi \wedge \neg(\pi=p \wedge \pi'=p') :: \sigma[v]@(\pi,q,1) \wedge \sigma'[w']@(\pi',q,1) \rangle$

$\Leftrightarrow$ {each packet has a unique location}

$\sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1) \wedge p \geq N+1 \wedge p'<p$
unless
$\neg(\sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1)) \wedge \langle \exists w',\pi,\pi' : \pi \geq N+1 \wedge \pi'<\pi :: \sigma[v]@(\pi,q,1) \wedge \sigma'[w']@(\pi',q,1) \rangle$

$\Rightarrow$ {universal quantification on *v',p,p'* + naming the predicates}

$\langle \forall v',p,p' : p \geq N+1 \wedge p'<p :: P(v',p,p')$ **unless** $\neg P(v',p,p') \wedge \langle \exists w',\pi,\pi' : \pi \geq N+1 \wedge \pi'<\pi :: \neg P(w',\pi,\pi') \rangle \rangle$

$\Rightarrow$ {corollary of the general disjunction rule}

$\langle \exists v',p,p' : p \geq N+1 \wedge p'<p :: P(v',p,p') \rangle$
unless
$\langle \forall v',p,p' : p \geq N+1 \wedge p'<p :: \neg P(v',p,p') \rangle \wedge \langle \exists v',p,p' : p \geq N+1 \wedge p'<p :: P(v',p,p') \rangle$

$\Leftrightarrow$ {predicate calculus + unfolding *P*}

$\langle \exists v',p,p' : p \geq N+1 \wedge p'<p :: \sigma[v]@(p,q,1) \wedge \sigma'[v']@(p',q,1)\rangle$ **unless** false

$\Leftrightarrow$ {predicatecalculus}

$\langle \exists p : p \geq N+1 :: \sigma[v]@(p,q,1) \wedge \langle \exists v',p' : p'<p :: \sigma'[v']@(p',q,1)\rangle\rangle$ **unless** false

$\Rightarrow$ {cancellation with *(R5)*}

(Q6). ❑

▼ **Proof of Property (P9)**

From the row to the column, packets remain ordered according to their identifiers:

inv $\alpha \sqsubset \beta \Rightarrow pid.\beta < pid.\alpha$. (P9)

From the definition of $\prec$, we get:

$$\begin{aligned}
\textbf{inv}\quad & \langle \exists p,q,q' : q<q' \leq 0 :: \alpha@(p,q,0) \wedge \beta@(p,q',0)\rangle \\
\vee\ & \langle \exists p,p',q : N+1 \leq p<p' :: \alpha@(p,q,1) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle \\
\vee\ & \langle \exists p,p',q,q' : q' \leq 0 \wedge p' \geq N+1 :: \alpha@(p,q',0) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle \\
\Rightarrow\ & \\
& \alpha \prec \beta
\end{aligned}$$

$\Rightarrow$ {property *(P9.1)*}

$$\begin{aligned}
\textbf{inv}\quad & \langle \exists p,q,q' : q<q' \leq 0 :: \alpha@(p,q,0) \wedge \beta@(p,q',0)\rangle \\
\vee\ & \langle \exists p,p',q : N+1 \leq p<p' :: \alpha@(p,q,1) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle \\
\vee\ & \langle \exists p,p',q,q' : q' \leq 0 \wedge p' \geq N+1 :: \alpha@(p,q',0) \wedge \beta@(p',q,1) \wedge src.\alpha=src.\beta\rangle \\
\Rightarrow\ & \\
& pid.\beta < pid.\alpha
\end{aligned}$$

$\Leftrightarrow$ {coupling invariant}

inv $\alpha \sqsubset \beta \Rightarrow pid.\beta < pid.\alpha$.

Property *(P10)* can be derived from *(P10.1)* in a similar way.

▼ **Proof of Property (P11)**

Every packet in the input eventually moves to the output.

$\sigma[v]@(p,q) \wedge q \leq 0 \mapsto \langle \exists\, v',p',q' : p' \geq N+1 :: \sigma[v']@(p',q')\rangle$. (P11)

To prove *(P11)*, we will use *(P11.1)*, *(P11.2)*, and the induction principle to show that :

- Packets in the input eventually move to the destination column:

  $\sigma[v]@(p,q,0) \wedge q \leq 0 \mapsto \langle \exists v',q' :: \sigma[v']@(p+1,q',1)\rangle$. (Q7)

- Packets on the destination column eventually move to the output:

  $\langle \exists v,p,q : p \leq N :: \sigma[v]@(p,q,1)\rangle \mapsto \langle \exists v,p,q : p \geq N+1 :: \sigma[v]@(p,q,1)\rangle$. (Q8)

Proof of Property (Q7)

From *(P11.1)*, *(P11.2)*, and the definition of "$\gg$" , we easily get:

$\sigma[v]@(p,q,0) \mapsto \langle \exists v' :: \sigma[v']@(p,q+1,0)\rangle \vee \langle \exists v' :: \sigma[v']@(p+1,q,1)\rangle$

$\Rightarrow$ {stable conjunction + *$\langle \forall v,v' :: dest.\sigma[v]=dest.\sigma[v']\rangle$*}

$\sigma[v]@(p,q,0) \wedge dest.\sigma[v]-q=d \mapsto \langle \exists v' :: \sigma[v']@(p,q+1,0) \wedge dest.\sigma[v']-(q+1)=d-1\rangle \vee \langle \exists v' :: \sigma[v']@(p+1,q,1)\rangle$

$\Rightarrow$ {general disjunction on *v* and *q*}

$\langle \exists v,q :: \sigma[v]@(p,q,0) \wedge dest.\sigma[v]-q=d \rangle$
$\mapsto$
$\langle \exists v,q :: \sigma[v]@(p,q,0) \wedge dest.\sigma[v]-q=d-1 \rangle \vee \langle \exists v,q :: \sigma[v]@(p+1,q,1) \rangle$

$\Rightarrow$ {induction principle}

$\langle \exists v,q :: \sigma[v]@(p,q,0) \rangle \mapsto \langle \exists v,q :: \sigma[v]@(p+1,q,1) \rangle$

$\Rightarrow$ {implication rule + transitivity of leads-to}

(Q7). ❑

Proof of Property (Q8)

From *(P11.2)*, we get:

$\sigma[v]@(p,q,1) \wedge p \le N \mapsto \langle \exists v' :: \sigma[v']@(p+1,q,1) \wedge p+1 \le N \rangle \vee \langle \exists v' :: \sigma[v']@(N+1,q,1) \rangle$

$\Rightarrow$ {stable conjunction}

$\sigma[v]@(p,q,1) \wedge p \le N \wedge N-p=d$
$\mapsto$
$\langle \exists v' :: \sigma[v']@(p+1,q,1) \wedge p+1 \le N \wedge N-(p+1)=d-1 \rangle \vee \langle \exists v' :: \sigma[v']@(N+1,q,1) \rangle$

$\Rightarrow$ {general disjunction on *v*, *p*, and *q*}

$\langle \exists v,p,q : p \le N :: \sigma[v]@(p,q,1) \wedge N-p=d \rangle$
$\mapsto$
$\langle \exists v,p,q : p \le N :: \sigma[v]@(p,q,1) \wedge N-p=d-1 \rangle \vee \langle \exists v,q :: \sigma[v]@(N+1,q,1) \rangle$

$\Rightarrow$ {induction principle}

$\langle \exists v,p,q : p \le N :: \sigma[v]@(p,q,1) \rangle \mapsto \langle \exists v,q :: \sigma[v]@(N+1,q,1) \rangle$

$\Rightarrow$ {implication rule + transitivity of leads-to}

(Q8). ❑

The truth of *(P11)* for *p* equal to *N* follows directly from *(Q7)*:

$\sigma[v]@(N,q,0) \wedge q \le 0 \mapsto \langle \exists v',q' :: \sigma[v']@(N+1,q',1) \rangle$

$\Rightarrow$ {coupling invariant}

$\sigma[v]@(N,q) \wedge q \le 0 \mapsto \langle \exists\ v',p',q' : p' \ge N+1 :: \sigma[v']@(p',q') \rangle$.

For *p* smaller than *N*, we have (property *(Q7)*):

$\sigma[v]@(p,q,0) \wedge p<N \wedge q \le 0 \mapsto \langle \exists v',p',q' : p' \le N :: \sigma[v']@(p',q',1) \rangle$

$\Rightarrow$ {transitivity with *(Q8)*}

$\sigma[v]@(p,q,0) \wedge p<N \wedge q \le 0 \mapsto \langle \exists v',p',q' : p' \ge N+1 :: \sigma[v']@(p',q',1) \rangle$

$\Leftrightarrow$ {coupling invariant}

$\sigma[v]@(p,q) \wedge p<N \wedge q \le 0 \mapsto \langle \exists\ v',p',q' : p' \ge N+1 :: \sigma[v']@(p',q') \rangle$.

B.2. Proof of Refinement 2

We first need to show several derived properties.

Derived Property 1

If a non header packet is in the row register of a switch *(p,q)* and *q* is not the destination column, a message is in transit through the switch along the row :

inv $\sigma[v]^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q \ge 1 \wedge q \neq dest.\sigma[v] \Rightarrow \langle \exists p',q',r',q'' : q'' \le q < q' :: message.(p',q',r').(p,q'',0) \rangle$. (DP1)

Proof of Property (DP1)

Let us first prove:

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \Rightarrow \langle \exists\alpha,p',q',r' : mid.\alpha=mid.\sigma[v] \wedge q<q' :: \alpha^{h}@(p',q',r')\rangle$. (Q1)

Since messages are properly structured (property *(P1)*), we have:

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \Rightarrow \langle \exists\alpha,p',q',r' : mid.\alpha=mid.\sigma[v] :: \alpha^{h}@(p',q',r')\rangle$

$\Rightarrow$ {predicatecalculus}

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \Rightarrow \langle \exists\alpha,p',q',r' : mid.\alpha=mid.\sigma[v] :: \alpha^{h}@(p',q',r')\rangle$

$\Leftrightarrow$ {packets reside either on the source row or on the destination column}

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v]$

$\Rightarrow$

$\langle \exists\alpha,q' : mid.\alpha=mid.\sigma[v] :: \alpha^{h}@(p,q',0)\rangle \vee \langle \exists\alpha,p',q' : mid.\alpha=mid.\sigma[v] \wedge q'=dest.\alpha :: \alpha^{h}@(p',q',1)\rangle$

$\Rightarrow$ {packets are ordered according to their identifiers, and the header packet has the smallest identifier}

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v]$

$\Rightarrow$

$\langle \exists\alpha,q' : mid.\alpha=mid.\sigma[v] \wedge q<q' :: \alpha^{h}@(p,q',0)\rangle \vee \langle \exists\alpha,p',q' : mid.\alpha=mid.\sigma[v] \wedge q'=dest.\alpha :: \alpha^{h}@(p',q',1)\rangle$

$\Rightarrow$ {packets from the same message have the same destination}

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v]$

$\Rightarrow$

$\langle \exists\alpha,q' : mid.\alpha=mid.\sigma[v] \wedge q<q' :: \alpha^{h}@(p,q',0)\rangle \vee \langle \exists\alpha,p',q' : mid.\alpha=mid.\sigma[v] \wedge q<q' :: \alpha^{h}@(p',q',1)\rangle$

$\Rightarrow$ {predicatecalculus}

(Q1).

Let us now prove:

$\mathbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \Rightarrow \langle \exists\beta,q'' : mid.\beta=mid.\sigma[v] \wedge q''\leq q :: \beta^{t}@(p,q'',0)\rangle$. (Q2)

If *σ[v]* is a tail packet, *(Q2)* is trivially true. Let us prove *(Q2)* for body packets. Since messages are properly structured, we have:

$\mathbf{inv}\ \sigma[v]^{b}@(p,q,0) \Rightarrow \langle \exists\beta,p'',q'',r'' : mid.\beta=mid.\sigma[v] :: \beta^{t}@(p'',q'',r'')\rangle$

$\Rightarrow$ {predicatecalculus}

$\mathbf{inv}\ \sigma[v]^{b}@(p,q,0) \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \Rightarrow \langle \exists\beta,p'',q'',r'' : mid.\beta=mid.\sigma[v] :: \beta^{t}@(p'',q'',r'')\rangle$

$\Leftrightarrow$ {packets reside either on the source row or on the destination column}

$\mathbf{inv}\ \sigma[v]^{b}@(p,q,0) \wedge q\geq 1 \wedge q\neq dest.\sigma[v]$

$\Rightarrow$

$\langle \exists\beta,q'' : mid.\beta=mid.\sigma[v] :: \beta^{t}@(p,q'',0)\rangle \vee \langle \exists\beta,p'',q'' : mid.\beta=mid.\sigma[v] \wedge q''=dest.\beta :: \beta^{t}@(p'',q'',1)\rangle$

$\Leftrightarrow$ {the identifier of the tail packet is greater than the identifiers of the other packets of the message}

$\mathbf{inv}\ \sigma[v]^{b}@(p,q,0) \wedge q\geq 1 \wedge q\neq dest.\sigma[v]$

$\Rightarrow$

$\langle \exists\beta,q'' : mid.\beta=mid.\sigma[v] \wedge pid.\beta>pid.\sigma[v] :: \beta^{t}@(p,q'',0)\rangle$

$\vee \langle \exists\beta,p'',q'' : mid.\beta=mid.\sigma[v] \wedge q''=dest.\beta \wedge pid.\beta>pid.\sigma[v] :: \beta^{t}@(p'',q'',1)\rangle$

$\Leftrightarrow$ {from the rows to the columns, packets are ordered according to their identifiers (property *(P9.1)*)}

$\mathbf{inv}\ \sigma[v]^{b}@(p,q,0) \wedge q\geq 1 \wedge q\neq dest.\sigma[v] \Rightarrow \langle \exists\beta,q'' : mid.\beta=mid.\sigma[v] \wedge q''<q :: \beta^{t}@(p,q'',0)\rangle$.

The truth of *(DP1)* finally follows from the conjunction of *(Q1)* and *(Q2)*. ❑

In a very similar way, we can deduce the two following properties from S_2:

Derived Property 2

If a non header packet is in the row register of its turning switch, a message is in transit through the switch from the row to the column:

$$\textbf{inv}\ \sigma[v]^{\tau}@(p,q,0) \wedge \tau{\neq}h \wedge q{=}dest.\sigma[v] \Rightarrow \langle \exists p',q' : p'{>}p \wedge q'{\leq}q :: message.(p',q,1).(p,q',0)\rangle. \qquad \text{(DP2)}$$

Derived Property 3

If a non header packet is in the column register of a switch *(p,q)*, a message is in transit through the switch along the column:

$$\begin{array}{l}\textbf{inv}\ \sigma[v]^{\tau}@(p,q,1) \wedge \tau{\neq}h \wedge p{\leq}N \\ \quad\Rightarrow \\ \quad\langle \exists p',p'',q'',r'' : p'{>}p \wedge (p''{<}p \vee (p''{=}p \wedge r''{=}1)) :: message.(p',q,1).(p'',q'',r'')\rangle.\end{array} \qquad \text{(DP3)}$$

Let us now focus on the proof of the refinement, i.e., the proof that S_2 implies property *(P11.2)*:

$$\sigma[v]@(p,q,r) \wedge p{\leq}N \wedge q{\geq}1\ \textbf{until}\ \langle \exists\ v',p',q',r' : (p,q,r){\gg}(p',q',r') :: \sigma[v']@(p',q',r')\rangle. \qquad \text{(P11.2)}$$

The unless part of *(P11.2)* is equivalent to *(P11.2.1)*. The proof of the leads-to part can be decomposed in three steps:

- Packet movement along the rows:

  $$\sigma[v]@(p,q,0) \wedge q{\geq}1 \wedge q{\neq}dest.\sigma[v] \mapsto \langle \exists v' :: \sigma[v']@(p,q{+}1,0)\rangle. \qquad \text{(Q3)}$$

- Packet movement from the rows to the columns:

  $$\sigma[v]@(p,q,0) \wedge q{=}dest.\sigma[v] \mapsto \langle \exists v' :: \sigma[v']@(p{+}1,q,1)\rangle. \qquad \text{(Q4)}$$

- Packet movement along the columns:

  $$\sigma[v]@(p,q,1) \wedge p{\leq}N \mapsto \langle \exists v' :: \sigma[v']@(p{+}1,q,1)\rangle. \qquad \text{(Q5)}$$

The leads-to part of *(P11.2)* can then be derived from *(Q3)*, *(Q4)*, and *(Q5)* by applying the simple disjunction rule.

Proof of Property (Q3)

For header packets, *(Q3)* is equivalent to *(P11.2.2)*. For non header packets, we have:

$$\begin{array}{ll} & \sigma[v]^{\tau}@(p,q,0) \wedge \tau{\neq}h \wedge q{\geq}1 \wedge q{\neq}dest.\sigma[v] \\ \Rightarrow & \{\text{property } \textit{(DP1)}\} \\ & \sigma[v]^{\tau}@(p,q,0) \wedge \tau{\neq}h \wedge q{\geq}1 \wedge \langle \exists p',q',r',q'' : q''{\leq}q{<}q' :: message.(p',q',r').(p,q'',0)\rangle \\ \Rightarrow & \{\text{invariant } \textit{(P13)}\} \\ & \sigma[v]^{\tau}@(p,q,0) \wedge \tau{\neq}h \wedge q{\geq}1 \wedge \neg turn(p,q) \\ \mapsto & \{\text{property } \textit{(P11.2.3)}\} \\ & \langle \exists v' :: \sigma[v']@(p,q{+}1,0)\rangle. \end{array}$$

□

Proof of Property (Q5)

For header packets, *(Q5)* follows from the transitivity of leads-to applied to properties *(P11.2.4)* and *(P11.2.5)*. For non header packets, we have:

$\sigma[v]^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q=dest.\sigma[v]$

$\Rightarrow$ {property *(DP2)* + $dest.\sigma[v] \geq 1$}

$\sigma[v]^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q \geq 1 \wedge \langle \exists p',q' : p'>p \wedge q' \leq q :: message.(p',q,1).@(p,q',0) \rangle$

$\Rightarrow$ {invariant *(P14)*}

$\sigma[v]^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q \geq 1 \wedge turn(p,q)$

$\mapsto$ {property *(P11.2.5)*}

$\langle \exists v' :: \sigma[v']@(p+1,q,1) \rangle$. ❑

The proof of *(Q6)*, based on properties *(P11.2.6)*, *(P11.2.7)*, *(P15)*, and *(DP3)*, is symmetrical.

B.3. Proof of Refinement 6

We must show that S_6 implies properties *(P11.2.2.1)-(P11.2.7.1)* specifying the packet movement inside the router. We are first going to show the packet movement along the columns, and then use that result to show the packet movement at the turning switches and along the rows.

▼ Proof of Properties (P11.2.6.1) and (P11.2.7.1)

We are going to prove the packet movement along the columns:

$$\alpha@(p,q,1) \wedge p \leq N \wedge up(p,q) \mapsto \alpha@(p+1,q,1) \qquad \text{(P11.2.7.1)}$$

and the fact that headers on the columns eventually turn the *up* signal on:

$$\alpha^{h}@(p,q,1) \wedge p \leq N \mapsto \alpha@(p,q,1) \wedge up(p,q) \qquad \text{(P11.2.6.1)}$$

by induction on the row number *p*. However, to prove that each property is true for the row *p*, we need to know that both are true for the row *p+1*. This implies that *(P11.2.7.1)* and *(P11.2.6.1)* must be proved simultaneously in the same inductive proof.

Let us first say a few words about *(P11.2.6.1)*. The refined property *(P11.2.6.1.1)* states that, when a header is waiting on the column and the signal *turn* is off, either the *up* or the *turn* signal will be turned on:

$\alpha^{h}@(p,q,1) \wedge p \leq N \wedge \neg turn(p,q)$ **ensures** $(\alpha^{h}@(p,q,1) \wedge up(p,q)) \vee turn(p,q)$

$\Rightarrow$ {conjunction with $\alpha^{h}@(p,q,1)$ ***unless*** $\langle \exists \alpha' : pid.\alpha'=pid.\alpha :: \alpha'^{h}@(p+1,q,1) \rangle$}

$\alpha^{h}@(p,q,1) \wedge p \leq N \wedge \neg turn(p,q)$
ensures
$(\alpha^{h}@(p,q,1) \wedge up(p,q)) \vee (\alpha^{h}@(p,q,1) \wedge turn(p,q)) \vee (\langle \exists \alpha' : pid.\alpha'=pid.\alpha :: \alpha'^{h}@(p+1,q,1) \rangle \wedge turn(p,q))$

$\Rightarrow$ {property *(P6.1)* $\Rightarrow src.\alpha<p$ + $src.\alpha'=src.\alpha$}

$\alpha^{h}@(p,q,1) \wedge p \leq N \wedge \neg turn(p,q)$
ensures
$(\alpha^{h}@(p,q,1) \wedge up(p,q)) \vee (\alpha^{h}@(p,q,1) \wedge turn(p,q)) \vee (\langle \exists \alpha' : src.\alpha'<p :: \alpha'^{h}@(p+1,q,1) \rangle \wedge turn(p,q))$

$\Rightarrow$ {messages are properly structured}

$\alpha^{h}@(p,q,1) \wedge p \leq N \wedge \neg turn(p,q)$
ensures
$(\alpha^{h}@(p,q,1) \wedge up(p,q)) \vee (\alpha^{h}@(p,q,1) \wedge turn(p,q))$
$\vee (\langle \exists \alpha',\beta',p',q',r' : mid.\beta'=mid.\alpha' \wedge p'<p :: \alpha'^{h}@(p+1,q,1) \wedge \beta'^{t}@(p',q',r') \rangle \wedge turn(p,q))$

$\Leftrightarrow$ {invariants *(P15)* and *(P12)*}

$\alpha^{h}@(p,q,1) \wedge p \leq N \wedge \neg turn(p,q)$ **ensures** $(\alpha^{h}@(p,q,1) \wedge up(p,q)) \vee (\alpha^{h}@(p,q,1) \wedge turn(p,q))$.

To prove *(P11.2.6.1)*, we thus only need to show that when a header is waiting on the column and the signal *turn* is on, the *up* signal will eventually be turned on:

$$\alpha^h@(p,q,1) \wedge p\leq N \wedge turn(p,q) \mapsto \alpha^h@(p,q,1) \wedge up(p,q). \qquad (Q1)$$

Informally, the truth of *(Q1)* is based on the fact that, if the signal *turn* is on, a message is currently taking its turn through the switch *(p,q)* and, according to the fairness property *(P22)*, when the tail of the message will pass through the switch, the *up* signal will be turned on. To prove it formally, we need to show some intermediate results.

Definition

We define predicate *tail$_{p,q}$@(p,q',0)* to be true iff a message is taking its turn (or waiting for taking its turn) through the switch *(p,q)*, and the tail of the message is at location *(p,q',0)*.

$$tail_{p,q}@(p,q',0)$$
$$\equiv$$
$$\langle \exists \alpha,\beta,p' : dest.\alpha{=}q \wedge mid.\alpha{=}mid.\beta :: \beta^t@(p,q',0) \wedge (\alpha^h@(p,q,0) \vee \alpha^h@(p',q,1))\rangle.$$

Derived Property 1

The tail of the message turning through the switch *(p,q)* can only move one location at a time along the row:

$$tail_{p,q}@(p,q',0) \wedge q'<q \ \textbf{unless}\ tail_{p,q}@(p,q'+1,0). \qquad (DP1)$$

Proof of Property (DP1)

Property *(DP1)* follows from the three properties below, which can be derived from *(P11.2.1)*:

$$\langle \exists v :: \sigma[v]^h@(p,q,0) \wedge q{=}dest.\sigma[v]\rangle\ \textbf{unless}\ \langle \exists v,p' :: \sigma[v]^h@(p',q,1)\rangle \qquad (R1)$$
$$\langle \exists v,p' :: \sigma[v]^h@(p',q,1)\rangle\ \textbf{unless}\ false \qquad (R2)$$
$$\langle \exists w :: \sigma'[w]^t@(p,q',0) \wedge q'<dest.\sigma'[w]\rangle\ \textbf{unless}\ \langle \exists w :: \sigma'[w]^t@(p,q'+1,0)\rangle. \qquad (R3)$$

By applying the cancellation rule on *(R1)* and *(R2)*, we get:

$$\langle \exists v,p' :: (\sigma[v]^h@(p,q,0) \vee \sigma[v]^h@(p',q,1)) \wedge q{=}dest.\sigma[v]\rangle\ \textbf{unless}\ false$$

$\Rightarrow$ {conjunction with *(R3)* and restricting *σ'[w]* to belong to the same message as *σ[v]*}

$$\langle \exists v,w,p' :: \sigma'[w]^t@(p,q',0) \wedge (\sigma[v]^h@(p,q,0) \vee \sigma[v]^h@(p',q,1)) \wedge q'<q \wedge q{=}dest.\sigma[v] \wedge mid.\sigma'[w]{=}mid.\sigma[v]\rangle$$
unless
$$\langle \exists v,w,p' :: \sigma'[w]^t@(p,q'+1,0) \wedge (\sigma[v]^h@(p,q,0) \vee \sigma[v]^h@(p',q,1)) \wedge q{=}dest.\sigma[v] \wedge mid.\sigma'[w]{=}mid.\sigma[v]\rangle$$

$\Rightarrow$ {general disjunction on *σ* and *σ'* + consequence weakening}

$$\langle \exists \alpha,\beta,p' : q{=}dest.\alpha \wedge mid.\beta{=}mid.\alpha :: \beta^t@(p,q',0) \wedge (\alpha^h@(p,q,0) \vee \alpha^h@(p',q,1))\rangle \wedge q'<q$$
unless
$$\langle \exists \alpha,\beta,p' : q{=}dest.\alpha \wedge mid.\beta{=}mid.\alpha :: \beta^t@(p,q'+1,0) \wedge (\alpha^h@(p,q,0) \vee \alpha^h@(p',q,1))\rangle$$

$\Leftrightarrow$ {definition of *tail$_{p,q}$*}

(DP1). ❑

Derived Property 2

The signal *turn(p,q)* stays on until the tail of the message turning through the switch *(p,q)* shows up:

$$turn(p,q)\ \textbf{unless}\ turn(p,q) \wedge tail_{p,q}@(p,q,0). \qquad (DP2)$$

Proof of Property (DP2)

From *(P11.2.1)*, we can easily derive:

$\langle \exists w,q' :: \sigma'[w]^t@(p,q',0) \wedge \text{dest}.\sigma'[w]=q\rangle$ **unless** $\langle \exists w :: \sigma'[w]^t@(p,q,0)\rangle$ (R4)

$\Rightarrow$ {conjunction with *(R2)* and restricting *σ'[w]* to belong to the same message as *σ[v]*}

$\langle \exists v,w,p',q' :: \sigma'[w]^t@(p,q',0) \wedge \sigma[v]^h@(p',q,1) \wedge \text{mid}.\sigma'[w]=\text{mid}.\sigma[v]\rangle$
unless
$\langle \exists v,w,p' :: \sigma'[w]^t@(p,q,0) \wedge \sigma[v]^h@(p',q,1) \wedge \text{mid}.\sigma'[w]=\text{mid}.\sigma[v]\rangle$

$\Rightarrow$ {general disjunction on *σ* and *σ'* + consequence weakening}

$\langle \exists \alpha,\beta,p',q' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p',q,1)\rangle$
unless
$\langle \exists \alpha,\beta,p' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q,0) \wedge \alpha^h@(p',q,1)\rangle$

$\Leftrightarrow$ {invariant *(P14)*}

$\langle \exists \alpha,\beta,p',q' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p',q,1)\rangle \wedge \text{turn}(p,q)$
unless (R5)
$\langle \exists \alpha,\beta,p' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q,0) \wedge \alpha^h@(p',q,1)\rangle \wedge \text{turn}(p,q)$.

Furthermore, from *(P18)*, we get:

$\langle \exists v :: \sigma[v]^h@(p,q,0) \wedge q=\text{dest}.\sigma[v]\rangle \wedge \text{turn}(p,q)$ **unless** $\neg\langle \exists v :: \sigma[v]^h@(p,q,0)\rangle$

$\Rightarrow$ {conjunction with *(R1)* + each packet has a unique location}

$\langle \exists v :: \sigma[v]^h@(p,q,0) \wedge q=\text{dest}.\sigma[v]\rangle \wedge \text{turn}(p,q)$ **unless** $\langle \exists v,p' :: \sigma[v]^h@(p',q,1)\rangle$

$\Rightarrow$ {conjunction with *(R4)* and restricting *σ'[w]* to belong to the same message as *σ[v]*}

$\langle \exists v,w,q' :: \sigma'[w]^t@(p,q',0) \wedge \sigma[v]^h@(p,q,0) \wedge q=\text{dest}.\sigma[v] \wedge \text{mid}.\sigma'[w]=\text{mid}.\sigma[v]\rangle \wedge \text{turn}(p,q)$
unless
$\langle \exists v,w,p',q' :: \sigma'[w]^t@(p,q',0) \wedge \sigma[v]^h@(p',q,1) \wedge \text{mid}.\sigma'[w]=\text{mid}.\sigma[v]\rangle$

$\Rightarrow$ {general disjunction on *σ* and *σ'* + consequence weakening}

$\langle \exists \alpha,\beta,q' : q=\text{dest}.\alpha \wedge \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p,q,0)\rangle \wedge \text{turn}(p,q)$
unless
$\langle \exists \alpha,\beta,p',q' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p',q,1)\rangle$

$\Leftrightarrow$ {invariant *(P14)*}

$\langle \exists \alpha,\beta,q' : q=\text{dest}.\alpha \wedge \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p,q,0)\rangle \wedge \text{turn}(p,q)$
unless
$\langle \exists \alpha,\beta,p',q' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge \alpha^h@(p',q,1)\rangle \wedge \text{turn}(p,q)$

$\Rightarrow$ {cancellation with *(R5)*}

$\langle \exists \alpha,\beta,p',q' : q=\text{dest}.\alpha \wedge \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q',0) \wedge (\alpha^h@(p,q,0) \vee \alpha^h@(p',q,1))\rangle \wedge \text{turn}(p,q)$
unless
$\langle \exists \alpha,\beta,p' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q,0) \wedge \alpha^h@(p',q,1)\rangle \wedge \text{turn}(p,q)$

$\Leftrightarrow$ {invariant *(P16)*}

$\text{turn}(p,q)$ **unless** $\langle \exists \alpha,\beta,p' : \text{mid}.\beta=\text{mid}.\alpha :: \beta^t@(p,q,0) \wedge \alpha^h@(p',q,1)\rangle \wedge \text{turn}(p,q)$

$\Leftrightarrow$ {definition of $tail_{p,q}$}

(DP2). ❑

<u>Derived Property 3</u>

If a non header packet is in the row register of a switch *(p,q)* and *q* is not the destination column, the signal *turn(p,q)* is off:

inv $\alpha^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q \geq 1 \wedge q \neq dest.\sigma[v] \Rightarrow \neg turn(p,q)$. (DP3)

Proof of Property (DP3)

In the proof of the second refinement, we have showed that when a non header packet α is in the row register of a switch *(p,q)* and *q* is not the destination column, there exists a message in transit through the switch along the row:

inv $\alpha^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q \geq 1 \wedge q \neq dest.\sigma[v] \Rightarrow \langle \exists p',q',r',q'' : q'' \leq q < q' :: message.(p',q',r').(p,q'',0) \rangle$. (R6)

This property is still valid since none of the properties used to prove it has been refined. The validity of *(DP3)* follows from *(R6)* and invariant *(P13)*. ❑

In the same way, we can easily prove the properties *(DP4)* and *(DP5)* below:

<u>Derived Property 4</u>

If a non header packet is in the row register of its turning switch, the signal *turn(p,q)* is on:

inv $\alpha^{\tau}@(p,q,0) \wedge \tau \neq h \wedge q = dest.\alpha \Rightarrow turn(p,q)$. (DP4)

<u>Derived Property 5</u>

If a non header packet is in the column register of a switch *(p,q)*, the signal *up(p,q)* is on:

inv $\alpha^{\tau}@(p,q,1) \wedge \tau \neq h \wedge p \leq N \Rightarrow up(p,q)$. (DP5)

<u>Derived Property 6</u>

Assuming that packets are guaranteed to move from the row to the column when the signal *turn* is on :

$\alpha@(p,q,0) \wedge q \geq 1 \wedge turn(p,q) \mapsto empty@(p,q,0)$

all the packets residing inside the router between the switch *(p,q)* and the tail of the turning message will eventually move:

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q' < q \wedge \beta@(p,q'',0) \wedge q'' \geq 1 \wedge q' \leq q'' \leq q \mapsto empty@(p,q'',0)$. (DP6)

Proof of Property (DP6)

We are going to show *(DP6)* by induction on *q''*.

- **Base case: q''=q**

The hypothesis of *(DP6)* implies:

$\beta@(p,q,0) \wedge q \geq 1 \wedge turn(p,q) \mapsto empty@(p,q,0)$

$\Rightarrow$ {implication rule + transitivity of leads-to}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q' < q \wedge \beta@(p,q,0) \wedge q \geq 1 \mapsto empty@(p,q,0)$.

- **Induction Hypothesis**

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \beta@(p,q''+1,0) \wedge q''+1>1 \wedge q'<q''+1\leq q \mapsto empty@(p,q''+1,0)$.

- **Induction Step**

Let us show:

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \mapsto empty@(p,q'',0)$.

From the property *(P11.2.1)*, we get:

$\delta@(p,q'',0) \wedge q''<dest.\delta$ **unless** $\langle\exists\delta' : pid.\delta'=pid.\delta :: \delta'@(p,q''+1,0)\rangle$

$\Rightarrow$ {conjunction with $\beta@(p,q''+1,0) \wedge q''+1<1$ ***unless*** $empty@(p,q''+1,0)$ (property *(P25)*)}

$\delta@(p,q'',0) \wedge 1\leq q''<dest.\delta \wedge \beta@(p,q''+1,0)$ **unless** $\delta@(p,q'',0) \wedge q''<dest.\delta \wedge empty@(p,q''+1,0)$

$\Rightarrow$ {PSP with induction hypothesis}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge 1\leq q''<dest.\delta \wedge q'\leq q''<q \wedge \beta@(p,q''+1,0)$

$\mapsto$

$\delta@(p,q'',0) \wedge q''<dest.\delta \wedge empty@(p,q''+1,0)$

$\Rightarrow$ {general disjunction on β}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge 1\leq q''<dest.\delta \wedge q'\leq q''<q \wedge \neg empty@(p,q''+1,0)$

$\mapsto$

$\delta@(p,q'',0) \wedge q''<dest.\delta \wedge empty@(p,q''+1,0)$.

From the fact that messages are not interleaved, we can deduce that δ belongs to the message that is taking its turn, which implies that $dest.\delta$ is equal to q. Since we have $q''<q$, the expression $q''<dest.\delta$ can be dropped from the left-hand side. Furthermore, from the fact that messages are properly structured, we can deduce that δ cannot be a header packet. So the last property is equivalent to:

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \wedge \neg empty@(p,q''+1,0)$

$\mapsto$

$\delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge q''<dest.\delta \wedge empty@(p,q''+1,0)$

$\Rightarrow$ {Property *(DP3)*}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \wedge \neg empty@(p,q''+1,0)$

$\mapsto$

$\delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge \neg turn(p,q'') \wedge empty@(p,q''+1,0)$

$\Rightarrow$ {transitivity with *(P11.2.3.1.1)*}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \wedge \neg empty@(p,q''+1,0)$

$\mapsto$

$\delta@(p,q''+1,0)$

$\Rightarrow$ {PSP with $\delta@(p,q'',0) \wedge q''\geq 1$ ***unless*** $empty@(p,q'',0)$ (property *(P25)*)}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \wedge \neg empty@(p,q''+1,0)$

$\mapsto$ (R7)

$empty@(p,q'',0)$.

To complete the proof, we now need to show that δ will eventually move when the next location is empty. From *(P11.2.3.1.1)*, we get:

$\delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge q''\geq 1 \wedge \neg turn(p,q'') \wedge empty@(p,q''+1,0) \mapsto \delta@(p,q''+1,0)$

$\Rightarrow$ {PSP with $\delta@(p,q'',0) \wedge q''\geq 1$ ***unless*** $empty@(p,q'',0)$}

$\delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge q''\geq 1 \wedge \neg turn(p,q'') \wedge empty@(p,q''+1,0) \mapsto empty@(p,q'',0)$

$\Leftrightarrow$ {implication rule + transitivity of leads-to}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge 1\leq q''<dest.\delta \wedge q'\leq q''<q \wedge \neg turn(p,q'')$
$\wedge\ empty@(p,q''+1,0)$
$\mapsto$
$empty@(p,q'',0)$

$\Leftrightarrow$ {property *(DP3)*}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta^{\tau}@(p,q'',0) \wedge \tau\neq h \wedge 1\leq q''<dest.\delta \wedge q'\leq q''<q \wedge empty@(p,q''+1,0)$
$\mapsto$
$empty@(p,q'',0)$

$\Leftrightarrow$ {messages are not interleaved and properly structured $\Rightarrow$ $\tau\neq h \wedge dest.\delta=q$}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \wedge empty@(p,q''+1,0)$
$\mapsto$
$empty@(p,q'',0)$

$\Rightarrow$ {disjunction with *(R7)*}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \wedge \delta@(p,q'',0) \wedge q''\geq 1 \wedge q'\leq q''<q \mapsto empty@(p,q'',0)$. ❑

<u>**Derived Property 7**</u>

Assuming that packets are guaranteed to move from the row to the column when the signal *turn* is on :

$\alpha@(p,q,0) \wedge q\geq 1 \wedge turn(p,q) \mapsto empty@(p,q,0)$

the tail of the turning message will eventually move to the next location :

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q \mapsto turn(p,q) \wedge tail_{p,q}@(p,q'+1,0)$. (DP7)

Proof of Property (DP7)

The proof can be decomposed into two cases: $q'\leq 0$ and $q\geq 1$. In the first case, we get from *(P11.1)*:

$\langle\exists\beta :: \beta^{t}@(p,q',0)\rangle \wedge q'\leq 0 \mapsto \langle\exists\beta :: \beta^{t}@(p,q'+1,0)\rangle$

$\Leftrightarrow$ {implication rule + transitivity of leads-to}

$turn(p,q) \wedge \langle\exists\beta :: \beta^{t}@(p,q',0)\rangle \wedge q'\leq 0 \mapsto \langle\exists\beta :: \beta^{t}@(p,q'+1,0)\rangle$. (R8)

By applying the conjunction rule on *(DP1)* and *(DP2)*, we get:

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q$
unless
$turn(p,q) \wedge tail_{p,q}@(p,q'+1,0)$
$\vee\ (turn(p,q) \wedge tail_{p,q}@(p,q,0) \wedge tail_{p,q}@(p,q',0))$
$\vee\ (turn(p,q) \wedge tail_{p,q}@(p,q,0) \wedge tail_{p,q}@(p,q'+1,0))$

$\Leftrightarrow$ {messages are not interleaved}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'<q$ **unless** $turn(p,q) \wedge tail_{p,q}@(p,q'+1,0)$ (R9)

$\Rightarrow$ {PSP with *(R8)*}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge \langle\exists\beta :: \beta^{t}@(p,q',0)\rangle \wedge q'\leq 0 \wedge q'<q$
$\mapsto$
$(turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge \langle\exists\beta :: \beta^{t}@(p,q'+1,0)\rangle) \vee (turn(p,q) \wedge tail_{p,q}@(p,q'+1,0))$

$\Leftrightarrow$ {definition of $tail_{p,q}$ + messages are not interleaved and properly structured}

$turn(p,q) \wedge tail_{p,q}@(p,q',0) \wedge q'\leq 0 \wedge q'<q \mapsto turn(p,q) \wedge tail_{p,q}@(p,q'+1,0)$.

In the second case ($q\geq 1$), we can deduce from *(DP6)*:

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge q'<q \wedge \beta^t@(p,q',0) \wedge q'\geq 1 \mapsto \mathrm{empty}@(p,q',0)$

$\Rightarrow$ {general disjunction on β + definition of $tail_{p,q}$}

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge 1\leq q'<q \mapsto \mathrm{empty}@(p,q',0)$

$\Rightarrow$ {PSP with *(R9)*}

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge 1\leq q'<q \mapsto \mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q'+1,0).$ ❑

<u>**Derived Property 8**</u>

Assuming that packets are guaranteed to move from the row to the column when the signal *turn* is on :

$\alpha@(p,q,0) \wedge q\geq 1 \wedge \mathrm{turn}(p,q) \mapsto \mathrm{empty}@(p,q,0)$

a tail is guaranteed to show up in the row register of the turning switch:

$\mathrm{turn}(p,q) \mapsto \mathrm{turn}(p,q) \wedge \langle \exists\beta :: \beta^t@(p,q,0)\rangle.$ (DP8)

Proof of Property (DP8)

Let us start from (DP7) :

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge q'<q \mapsto \mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q'+1,0)$

$\Leftrightarrow$ {predicate calculus}

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge q'<q$

$\mapsto$

$(\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q'+1,0) \wedge q'+1<q) \vee (\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q,0))$

$\Rightarrow$ {stable conjunction}

$\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q',0) \wedge q'<q \wedge q-q'=d$

$\mapsto$

$(\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q'+1,0) \wedge q'+1<q \wedge q-(q'+1)=d-1) \vee (\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q,0))$

$\Rightarrow$ {general disjunction on q}

$\mathrm{turn}(p,q) \wedge \langle \exists q' : q'<q :: \mathrm{tail}_{p,q}@(p,q',0) \wedge q-q'=d\rangle$

$\mapsto$

$(\mathrm{turn}(p,q) \wedge \langle \exists q' : q'<q :: \mathrm{tail}_{p,q}@(p,q',0) \wedge q-q'=d-1\rangle) \vee (\mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q,0))$

$\Rightarrow$ {induction principle}

$\mathrm{turn}(p,q) \wedge \langle \exists q' : q'<q :: \mathrm{tail}_{p,q}@(p,q',0)\rangle \mapsto \mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q,0)$

$\Leftrightarrow$ {reflexivity of leads-to}

$\mathrm{turn}(p,q) \wedge \langle \exists q' : q'\leq q :: \mathrm{tail}_{p,q}@(p,q',0)\rangle \mapsto \mathrm{turn}(p,q) \wedge \mathrm{tail}_{p,q}@(p,q,0)$

$\Rightarrow$ {$turn(p,q) \Rightarrow \langle \exists q' : q'\leq q :: tail_{p,q}@(p,q',0)\rangle$ (invariant *(P16)*) + definition of $tail_{p,q}@(p,q,0)$}

$\mathrm{turn}(p,q) \mapsto \mathrm{turn}(p,q) \wedge \langle \exists\beta :: \beta^t@(p,q,0)\rangle.$ ❑

We are now able to prove properties *(P11.2.7.1)* and *(Q1)*, by induction on *p*. Let us rename them $T_1(p)$ and $T_2(p)$:

$T_1(p) \equiv \alpha@(p,q,1) \wedge p\leq N \wedge \mathrm{up}(p,q) \mapsto \alpha@(p+1,q,1)$
$T_2(p) \equiv \alpha^h@(p,q,1) \wedge p\leq N \wedge \mathrm{turn}(p,q) \mapsto \alpha^h@(p,q,1) \wedge \mathrm{up}(p,q).$

Base case: p=N

The validity of $T_1(N)$:

$\alpha @(N,q,1) \wedge up(N,q) \mapsto \alpha @(N+1,q,1)$

is directly implied by property *(P11.2.7.1.2)*. To prove $T_2(N)$:

$\alpha^h @(N,q,1) \wedge turn(N,q) \mapsto \alpha^h @(N,q,1) \wedge up(N,q)$

we need the derived property *(DP8)*. So, to be able to use it, we must prove:

$\alpha @(N,q,0) \wedge q \geq 1 \wedge turn(N,q) \mapsto empty@(N,q,0)$.

From *(P11.2.5.1.2)*, we get :

$\alpha @(N,q,0) \wedge q \geq 1 \wedge turn(N,q) \mapsto \alpha @(N+1,q,1)$

$\Rightarrow$ {PSP with $\alpha @(N,q,0) \wedge q \geq 1$ ***unless*** $empty@(N,q,0)$}

$\alpha @(N,q,0) \wedge q \geq 1 \wedge turn(N,q) \mapsto empty@(N,q,0)$.

We can now prove $T_2(N)$. From *(P11.2.1)*, we get:

$\alpha^h @(N,q,1)$ **unless** $\langle \exists \alpha' : pid.\alpha' = pid.\alpha :: \alpha'^h @(N+1,q,1) \rangle$

$\Leftrightarrow$ {invariant *(P15)* + messages are properly structured}

$\alpha^h @(N,q,1)$ **unless** $\langle \exists \alpha' : pid.\alpha' = pid.\alpha :: \alpha'^h @(N+1,q,1) \rangle \wedge up(N,q)$ (R10)

$\Rightarrow$ {conjunction with *(DP2)* + *turn* and *up* are mutually exclusive}

$\alpha^h @(N,q,1) \wedge turn(N,q)$ **unless** $\alpha^h @(N,q,1) \wedge turn(N,q) \wedge tail_{N,q} @(N,q,0)$

$\Rightarrow$ {PSP with *(DP8)* + definition of $tail_{N,q}$}

$\alpha^h @(N,q,1) \wedge turn(N,q) \mapsto \alpha^h @(N,q,1) \wedge turn(N,q) \wedge \langle \exists \beta :: \beta^t @(N,q,0) \rangle$. (R11)

Furthermore, from *(P11.2.1)* and *(P20)*, we get:

$\beta^t @(N,q,0) \wedge q \geq 1 \wedge turn(N,q)$ **unless** $\langle \exists \beta' : pid.\beta' = pid.\beta :: \beta'^t @(N+1,q,1) \rangle \wedge \neg turn(N,q)$

$\Rightarrow$ {conjunction with *(R10)* + consequence weakening}

$\beta^t @(N,q,0) \wedge turn(N,q) \wedge \alpha^h @(N,q,1)$ **unless** $\neg turn(N,q) \wedge \alpha^h @(N,q,1)$

$\Rightarrow$ {conjunction with $\beta^t @(N,q,0) \wedge turn(N,q) \wedge \alpha^h @(N,q,1)$ ***unless*** $up(N,q)$ (property *(P22)*)}

$\beta^t @(N,q,0) \wedge turn(N,q) \wedge \alpha^h @(N,q,1)$ **unless** $up(N,q) \wedge \alpha^h @(N,q,1)$ (R12)

$\Rightarrow$ {PSP with $\beta^t @(N,q,0) \wedge q \geq 1 \wedge turn(N,q) \mapsto \beta^t @(N+1,q,1)$ (property *(P11.2.5.1.2)*)}

$\beta^t @(N,q,0) \wedge turn(N,q) \wedge \alpha^h @(N,q,1) \mapsto up(N,q) \wedge \alpha^h @(N,q,1)$

$\Rightarrow$ {general disjunction on β + transitivity with *(R11)*}

$\alpha^h @(N,q,1) \wedge turn(N,q) \mapsto \alpha^h @(N,q,1) \wedge up(N,q)$.

Induction Hypothesis

Let us suppose $T_1(p+1)$:

$\alpha @(p+1,q,1) \wedge p+1 \leq N \wedge up(p+1,q) \mapsto \alpha @(p+2,q,1)$

and $T_2(p+1)$:

$\alpha^h @(p+1,q,1) \wedge p+1 \leq N \wedge turn(p+1,q) \mapsto \alpha^h @(p+1,q,1) \wedge up(p+1,q)$.

To prove $T_1(p)$ and $T_2(p)$, we will need the following property:

$\neg empty@(p+1,q,1) \wedge p+1 \leq N \mapsto empty@(p+1,q,1)$.

Let us call it $T_3(p+1)$, and let us prove that it is implied by $T_1(p+1)$ and $T_2(p+1)$. The property $T_2(p+1)$, along with the following property we derived from *(P11.2.6.1.1)*:

$\alpha^h@(p+1,q,1) \wedge p+1\leq N \wedge \neg turn(p+1,q)$
ensures
$(\alpha^h@(p+1,q,1) \wedge up(p+1,q)) \vee (\alpha^h@(p+1,q,1) \wedge turn(p+1,q))$.

implies:

$\alpha^h@(p+1,q,1) \wedge p+1\leq N \mapsto \alpha^h@(p+1,q,1) \wedge up(p+1,q)$
$\Rightarrow$ {transitivity with *$T_1(p+1)$*}
$\alpha^h@(p+1,q,1) \wedge p+1\leq N \mapsto \alpha@(p+2,q,1)$.

Furthermore, from *$T_1(p+1)$* we also get :

$\alpha^\tau@(p+1,q,1) \wedge \tau\neq h \wedge p+1\leq N \wedge up(p+1,q) \mapsto \alpha@(p+2,q,1)$
$\Leftrightarrow$ {property *(DP5)*}
$\alpha^\tau@(p+1,q,1) \wedge \tau\neq h \wedge p+1\leq N \mapsto \alpha@(p+2,q,1)$.

So, we have proved :

$\alpha@(p+1,q,1) \wedge p+1\leq N \mapsto \alpha@(p+2,q,1)$
$\Rightarrow$ {PSP with *$\alpha@(p+1,q,1) \wedge p+1\leq N$ **unless** $empty@(p+1,q,1)$*}
$\alpha@(p+1,q,1) \wedge p+1\leq N \mapsto empty@(p+1,q,1)$
$\Rightarrow$ {general disjunction on α}
$T_3(p+1)$.

Induction Step

Let us prove *$T_1(p)$* (for *$p<N$*). From the fact that a packet can move to the next location only if this location is empty, we can easily prove:

$\alpha@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \neg empty@(p+1,q,1)$
unless
$\alpha@(p,q,1) \wedge p<N \wedge up(p,q) \wedge empty@(p+1,q,1)$
$\Rightarrow$ {PSP with *$T_3(p+1)$*}
$\alpha@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \neg empty@(p+1,q,1) \mapsto \alpha@(p,q,1) \wedge p<N \wedge up(p,q) \wedge empty@(p+1,q,1)$

We also know, from *(P11.2.7.1.1)*:

$\alpha@(p,q,1) \wedge p<N \wedge up(p,q) \wedge empty@(p+1,q,1) \mapsto \alpha@(p+1,q,1)$
$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}
$\alpha@(p,q,1) \wedge p<N \wedge up(p,q) \mapsto \alpha@(p+1,q,1)$

As in the base case, to prove *$T_2(p)$* for *$p<N$*, we first need to prove the following property:

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \mapsto empty@(p,q,0)$.

From the fact that a packet can move to the next location only if this location is empty, and by using the PSP theorem with *$T_3(p+1)$* we get:

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge \neg empty@(p+1,q,1)$
$\mapsto$
$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge empty@(p+1,q,1)$
$\Rightarrow$ {transitivity of leads-to + disjunction with *(P11.2.5.1.1)*}
$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \mapsto \alpha@(p+1,q,1)$
$\Rightarrow$ {PSP with *$\alpha@(p,q,0) \wedge q\geq 1$ **unless** $empty@(p,q,0)$*}
$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \mapsto empty@(p+1,q,0)$.

We can now prove *T2(p)*. In the same way we proved *(R11)* in the base case, we have:

$$\alpha^h@(p,q,1) \wedge p<N \wedge turn(p,q) \mapsto \alpha^h@(p,q,1) \wedge p<N \wedge turn(p,q) \wedge \langle \exists\beta :: \beta^t@(p,q,0)\rangle. \qquad (R13)$$

In the same way we proved *(R12)*, we get:

$\beta^t@(p,q,0) \wedge p<N \wedge turn(p,q) \wedge \alpha^h@(p,q,1)$ **unless** $\alpha^h@(p,q,1) \wedge up(p,q)$

$\Rightarrow$ {PSP with $\beta^t@(p,q,0) \wedge p<N \wedge q\geq 1 \wedge turn(p,q) \wedge empty@(p+1,q,1) \mapsto \beta^t@(p+1,q,1)$ *(P11.2.5.1.1)*}

$\beta^t@(p,q,0) \wedge p<N \wedge turn(p,q) \wedge \alpha^h@(p,q,1) \wedge empty@(p+1,q,1) \mapsto \alpha^h@(p,q,1) \wedge up(p,q)$.

We can also easily prove:

$\beta^t@(p,q,0) \wedge p<N \wedge turn(p,q) \wedge \alpha^h@(p,q,1) \wedge \neg empty@(p+1,q,1)$
$\mapsto$
$\beta^t@(p,q,0) \wedge p<N \wedge turn(p,q) \wedge \alpha^h@(p,q,1) \wedge empty@(p+1,q,1)$

$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}

$\beta^t@(p,q,0) \wedge p<N \wedge turn(p,q) \wedge \alpha^h@(p,q,1) \mapsto \alpha^h@(p,q,1) \wedge up(p,q)$

$\Rightarrow$ {general disjunction on β + transitivity with *(R13)*}

$\alpha^h@(p,q,1) \wedge p<N \wedge turn(p,q) \mapsto \alpha^h@(p,q,1) \wedge up(p,q)$.

▼ Proof of Property (P11.2.5.1)

Packets eventually move from the row to the column when the signal *turn* is on:

$$\alpha@(p,q,0) \wedge q\geq 1 \wedge turn(p,q) \mapsto \alpha@(p+1,q,1). \qquad (P11.2.5.1)$$

The truth of *(P11.2.5.1)* for $p=N$ is directly implied by *(P11.2.5.1.2)*. Let us prove it for $p<N$. From the fact that a packet can move to the next location only if it is empty, we get:

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge \neg empty@(p+1,q,1)$
unless
$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge empty@(p+1,q,1)$

$\Rightarrow$ {PSP with $\neg\, empty@(p+1,q,1) \wedge p+1\leq N \mapsto empty@(p+1,q,1)$ (property *T3(p+1)*)}

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge \neg empty@(p+1,q,1)$
$\mapsto$
$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge empty@(p+1,q,1)$.

We also know, from *(P11.2.5.1.1)*:

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \wedge empty@(p+1,q,1) \mapsto \alpha@(p+1,q,1)$

$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}

$\alpha@(p,q,0) \wedge q\geq 1 \wedge p<N \wedge turn(p,q) \mapsto \alpha@(p+1,q,1)$.

▼ Proof of Property (P11.2.4.1)

Headers with value 1 along the row eventually turn the *turn* signal on:

$$\sigma[v]^h@(p,q,0) \wedge v=1 \mapsto \sigma[v]^h@(p,q,0) \wedge turn(p,q). \qquad (P11.2.4.1)$$

The refined property *(P11.2.4.1.1)* states that, when a header is waiting at its turning switch and the signal *up* is off, either the *turn* or the *up* signal will eventually be turned on:

$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge \neg up(p,q)$ **ensures** $(\sigma[v]^h@(p,q,0) \wedge turn(p,q)) \vee up(p,q)$.

In the same way we did for the property *(P11.2.6.1.1)*, we can prove that this implies:

$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge \neg up(p,q)$ **ensures** $(\sigma[v]^h@(p,q,0) \wedge turn(p,q)) \vee (\sigma[v]^h@(p,q,0) \wedge up(p,q))$.

So, to prove *(P11.2.4.1)*, we only need to show that when a header is waiting at its turning switch and signal *up* is on, the *turn* signal will eventually be turned on:

$$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge up(p,q) \mapsto \sigma[v]^h@(p,q,0) \wedge turn(p,q). \qquad (Q2)$$

As in the proof of *(P11.2.6.1)*, we need several intermediate results.

Definition

We define predicate *$tail'_{p,q}@(p',q',r')$* to be true iff a message is passing (or waiting for passing) through the switch *(p,q)* along the column, and the tail of the message is at location *(p',q',r')*.

$$tail'_{p,q}@(p',q',r')$$
$$\equiv$$
$$\langle \exists \alpha,\beta,p'' : p'' \geq p \wedge (p'<p \vee (p'=p \wedge r'=1)) \wedge mid.\alpha=mid.\beta :: \beta^t@(p',q',r') \wedge \alpha^h@(p'',q,1)\rangle.$$

Derived Property 9

The signal *up(p,q)* stays on until the tail of the message passing through the switch *(p,q)* along the column shows up:

$$up(p,q)\ \textbf{unless}\ up(p,q) \wedge tail'_{p,q}@(p,q,1). \qquad (DP9)$$

Derived Property 10

When the signal *up(p,q)* is on, a tail is guaranteed to show up in the column register of the switch *(p,q)*:

$$up(p,q) \mapsto up(p,q) \wedge \langle \exists \alpha :: \alpha^t@(p,q,1)\rangle. \qquad (DP10)$$

The proof of *(DP9)* is symmetrical to the proof of *(DP2)*. The proof of *(DP10)* is based on the same steps as the proof of *(DP8)*. However, it is slightly more complicated, since the tail of the message can either still be on the source row, or already on the column. So, to move to the switch *(p,q)*, the tail may have to first travel along the row up to its turning switch, and then move up the column. Let us also remark that, unlike *(DP8)*, the property *(DP10)* is not a conditional property. We do not need to assume that packets at location *(p,q,1)* will eventually move to the next location when signal *up* is on, since it is implied by the property *(P11.2.7.1)* we have proved early on.

Let us now focus on the proof of property *(Q2)*. From *(P11.2.1)*, *(P6.1)*, and invariant *(P24)*, we can deduce:

$$\sigma[v]^h@(p,q,0) \wedge v=1\ \textbf{unless}\ \sigma[v]^h@(p+1,q,1) \wedge src.\sigma[v]=p$$
$$\Rightarrow \{\text{invariant } (P14) + \text{messages are properly structured}\}$$
$$\sigma[v]^h@(p,q,0) \wedge v=1\ \textbf{unless}\ \sigma[v]^h@(p+1,q,1) \wedge turn(p,q) \qquad (R14)$$
$$\Rightarrow \{\text{conjunction with } (DP9) + turn \text{ and } up \text{ are mutually exclusive}\}$$
$$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge up(p,q)\ \textbf{unless}\ \sigma[v]^h@(p,q,0) \wedge v=1 \wedge up(p,q) \wedge tail'_{p,q}@(p,q,1)$$
$$\Rightarrow \{\text{PSP with } (DP10) + \text{definition of } tail'_{p,q}\}$$
$$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge up(p,q) \mapsto \sigma[v]^h@(p,q,0) \wedge v=1 \wedge up(p,q) \wedge \langle \exists \alpha :: \alpha^t@(p,q,1)\rangle. \qquad (R15)$$

Furthermore, from *(P11.2.1)* and *(P21)*, we get:

$$\alpha^t@(p,q,1) \wedge p \leq N \wedge up(p,q)\ \textbf{unless}\ \langle \exists \alpha' : pid.\alpha'=pid.\alpha :: \alpha'^t@(p+1,q,1)\rangle \wedge \neg up(p,q)$$
$$\Rightarrow \{\text{conjunction with } (R14) + \text{consequence weakening}\}$$
$$\alpha^t@(p,q,1) \wedge p \leq N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1\ \textbf{unless}\ \sigma[v]^h@(p,q,0) \wedge \neg up(p,q)$$

$\Rightarrow$ {conjunction with $\alpha^t@(p,q,1) \wedge p \leq N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0)$ **unless** $turn(p,q)$ (property *(P23)*)}

$\alpha^t@(p,q,1) \wedge p \leq N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1$ **unless** $\sigma[v]^h@(p,q,0) \wedge turn(p,q)$. (R16)

When p is equal to N, *(R16)* is equivalent to:

$\alpha^t@(N,q,1) \wedge up(N,q) \wedge \sigma[v]^h@(N,q,0) \wedge v=1$ **unless** $\sigma[v]^h@(N,q,0) \wedge turn(N,q)$

$\Rightarrow$ {PSP with $\alpha^t@(N,q,1) \wedge up(N,q) \mapsto \alpha^t@(N+1,q,1)$ (property *(P11.2.7.1.2)*)}

$\alpha^t@(N,q,1) \wedge up(N,q) \wedge \sigma[v]^h@(N,q,0) \wedge v=1 \mapsto turn(N,q) \wedge \sigma[v]^h@(N,q,0)$

$\Rightarrow$ {general disjunction on α + transitivity with *(R15)*}

$\sigma[v]^h@(N,q,0) \wedge v=1 \wedge up(N,q) \mapsto \sigma[v]^h@(N,q,1) \wedge turn(N,q)$.

When p is smaller than N, *(R16)* is equivalent to:

$\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1$ **unless** $\sigma[v]^h@(p,q,0) \wedge turn(p,q)$

$\Rightarrow$ {PSP with $\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge empty@(p+1,q,1) \mapsto \alpha^t@(p+1,q,1)$ (property *(P11.2.7.1.1)*)}

$\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1 \wedge empty@(p+1,q,1) \mapsto \sigma[v]^h@(p,q,0) \wedge turn(p,q)$.

We can also easily prove:

$\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1 \wedge \neg empty@(p+1,q,1)$
$\mapsto$
$\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1 \wedge empty@(p+1,q,1)$

$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}

$\alpha^t@(p,q,1) \wedge p<N \wedge up(p,q) \wedge \sigma[v]^h@(p,q,0) \wedge v=1 \mapsto \sigma[v]^h@(p,q,0) \wedge turn(p,q)$

$\Rightarrow$ {general disjunction on α + transitivity with *(R15)*}

$\sigma[v]^h@(p,q,0) \wedge v=1 \wedge p<N \wedge up(p,q) \mapsto \sigma[v]^h@(p,q,0) \wedge turn(p,q)$.

▼ Proof of Properties (P11.2.2.1) and (P11.2.3.1)

We will show that header packets whose value is different from 1 eventually move to the next location on the row:

$\sigma[v]^h@(p,q,0) \wedge v \neq 1 \wedge q \geq 1 \mapsto \sigma[v-1]^h@(p,q+1,0)$ (P11.2.2.1)

and that non header packets eventually move to the next location on the row when signal *turn* is off:

$\alpha^\tau@(p,q,0) \wedge \tau \neq h \wedge q \geq 1 \wedge \neg turn(p,q) \mapsto \alpha^\tau@(p,q+1,0)$ (P11.2.3.1)

by induction on the column number q. Since, from location *(p,M,0)*, packets can only move to the column, the base case is:

Base Case: q=M-1

Let us first show *(P11.2.2.1)*. From the fact that a packet can move to the next location only if it is empty, we can easily prove:

$\sigma[v]^h@(p,M-1,0) \wedge v \neq 1 \wedge \neg empty@(p,M,0)$ **unless** $\sigma[v]^h@(p,M-1,0) \wedge v \neq 1 \wedge empty@(p,M,0)$. (R17)

From the two packet movement properties *(P11.2.4.1)* and *(P11.2.5.1)* we have just proved, we can easily derived that packets at their turning switch eventually move to the next location:

$\alpha@(p,q,0) \wedge q=dest.\alpha \mapsto empty@(p,q,0)$.

Since packets at the right end of the rows are necessarily at their turning switch, we have:

$\alpha@(p,M,0) \mapsto empty@(p,M,0)$

$\Rightarrow$ {general disjunction on α}

$\neg empty@(p,M,0) \mapsto empty@(p,M,0)$ (R18)

$\Rightarrow$ {PSP with *(R17)*}

$\sigma[v]^h@(p,M\text{-}1,0) \wedge v\neq 1 \wedge \neg empty@(p,M,0) \mapsto \sigma[v]^h@(p,M\text{-}1,0) \wedge v\neq 1 \wedge empty@(p,M,0)$.

We also know, from *(P11.2.2.1.1)*:

$\sigma[v]^h@(p,M\text{-}1,0) \wedge v\neq 1 \wedge empty@(p,M,0) \mapsto \sigma[v\text{-}1]^h@(p,M,0)$

$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}

$\sigma[v]^h@(p,M\text{-}1,0) \wedge v\neq 1 \mapsto \sigma[v\text{-}1]^h@(p,M,0)$.

Let us now show *(P11.2.3.1)*. We can easily prove:

$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \wedge \neg empty@(p,M,0)$
unless
$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \wedge empty@(p,M,0)$

$\Rightarrow$ {PSP with *(R18)*}

$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \wedge \neg empty@(p,M,0)$
$\mapsto$
$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \wedge empty@(p,M,0)$.

We also know, from *(P11.2.3.1.1)*:

$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \wedge empty@(p,M,0) \mapsto \alpha^\tau@(p,M,0)$

$\Rightarrow$ {transitivity of leads-to + disjunction with the previous property}

$\alpha^\tau@(p,M\text{-}1,0) \wedge \tau\neq h \wedge \neg turn(p,M\text{-}1) \mapsto \alpha^\tau@(p,M,0)$.

Induction Hypothesis

Let us suppose:

$\sigma[v]^h@(p,q+1,0) \wedge v\neq 1 \wedge q+1>1 \mapsto \sigma[v\text{-}1]^h@(p,q+2,0)$
$\alpha^\tau@(p,q+1,0) \wedge \tau\neq h \wedge q+1>1 \wedge \neg turn(p,q+1) \mapsto \alpha^\tau@(p,q+2,0)$.

From these two properties we can easily prove:

$\alpha@(p,q+1,0) \wedge 1<q+1<dest.\alpha \mapsto empty@(p,q+1,0)$.

We also know:

$\alpha@(p,q+1,0) \wedge q+1=dest.\alpha \mapsto empty@(p,q+1,0)$.

So, we have:

$\alpha@(p,q+1,0) \wedge q+1>1 \mapsto empty@(p,q+1,0)$

$\Rightarrow$ {general disjunction on α}

$\neg empty@(p,q+1,0) \wedge q+1>1 \mapsto empty@(p,q+1,0)$. (R19)

Induction Step

The proofs of:

$\sigma[v]^h@(p,q,0) \wedge v\neq 1 \wedge q\geq 1 \mapsto \sigma[v\text{-}1]^h@(p,q+1,0)$
$\alpha^\tau@(p,q,0) \wedge \tau\neq h \wedge q\geq 1 \wedge \neg turn(p,q) \mapsto \alpha^\tau@(p,q+1,0)$

are identical to the proofs of the base case. We just have to replace *M-1* by *q*, and use *(R19)* instead of *(R18)*.

C. THEOREMS ABOUT UNLESS, ENSURES, AND LEADS TO

The theorems are listed without any proofs. They can be found in [2] and in the notes 01-88 and 21-90 on UNITY [13, 14].

C.1. Theorems about Unless

Consequence Weakening

$$\frac{p \textbf{ unless } q,\ q \Rightarrow r}{p \textbf{ unless } r}$$

Conjunction and Simple Conjunction

$$\frac{p \textbf{ unless } q, p' \textbf{ unless } q'}{(p \wedge p') \textbf{ unless } (p \wedge q') \vee (p' \wedge q) \vee (q \wedge q')}$$

$$\frac{p \textbf{ unless } q, p' \textbf{ unless } q'}{p \wedge p' \textbf{ unless } q \vee q'}$$

Stable Conjunction

$$\frac{p \textbf{ unless } q,\ \textbf{stable } b}{p \wedge b \textbf{ unless } q \wedge b}$$

Disjunction and Simple Disjunction

$$\frac{p \textbf{ unless } q, p' \textbf{ unless } q'}{(p \vee p') \textbf{ unless } (\neg p \wedge q') \vee (\neg p' \wedge q) \vee (q \wedge q')}$$

$$\frac{p \textbf{ unless } q, p' \textbf{ unless } q'}{p \vee p' \textbf{ unless } q \vee q'}$$

General Disjunction and Corollary

$$\frac{\langle \forall i :: p.i \textbf{ unless } q.i \rangle}{\langle \exists i :: p.i \rangle \textbf{ unless } \langle \forall i :: \neg p.i \vee q.i \rangle \wedge \langle \exists i :: q.i \rangle}$$

$$\frac{\langle \forall i :: p.i \textbf{ unless } \neg p.i \wedge q.i \rangle}{\langle \exists i :: p.i \rangle \textbf{ unless } \langle \forall i :: \neg p.i \rangle \wedge \langle \exists i :: q.i \rangle}$$

Cancellation

$$\frac{p \textbf{ unless } q, q \textbf{ unless } r}{p \vee q \textbf{ unless } r}$$

C.2. Theorems about Ensures

Conjunction

$$\frac{p \textbf{ unless } q,\ p' \textbf{ ensures } q'}{(p \wedge p') \textbf{ ensures } (p \wedge q') \vee (p' \wedge q) \vee (q \wedge q')}$$

C.3. Theorems about Leads to

Implication

$$\frac{p \Rightarrow q}{p \mapsto q}$$

Reflexivity

$$p \mapsto p$$

Stable Conjunction

$$\frac{p \mapsto q,\ \textbf{stable } b}{p \wedge b \mapsto q \wedge b}$$

General Disjunction

$$\frac{\langle \forall m : m \in W :: p(m) \mapsto q(m) \rangle}{\langle \exists m : m \in W :: p(m) \rangle \mapsto \langle \exists m : m \in W :: q(m) \rangle} \qquad \text{for any set W}$$

Progress-Safety-Progress (PSP)

$$\frac{p \mapsto q,\ r \textbf{ unless } b}{p \wedge r \mapsto (q \wedge r) \vee b}$$

Induction Principle

Let *(W,≺)* be a well-founded set, and *M* a function from the program state to *W*. We have:

$$\frac{\langle \forall m : m \in W :: p \wedge M{=}m \mapsto (p \wedge M{\prec}m) \vee q \rangle}{p \mapsto q}$$