Timely Rollback: Specification and Verification

Martín Abadi[1] and Michael Isard[2]

[1] University of California at Santa Cruz
[2] Microsoft Research*

Abstract. This paper presents a formal description and analysis of a technique for distributed rollback recovery. The setting for this work is a model for data-parallel computation with a notion of virtual time. The technique allows the selective undo of work at particular virtual times. A refinement theorem ensures the consistency of rollbacks.

1 Introduction

Rollback recovery plays an important role in ensuring fault-tolerance in many distributed systems [7]. In this paper we initiate the development and study of a rollback technique for distributed data-parallel computing. This technique relies on the timely-dataflow model [11], in which computations are organized as dataflow graphs (e.g., [9]) and events are associated with virtual times [8] in a partial order. The technique guarantees consistency and transparency to applications while allowing the selective undo of work that corresponds to particular virtual times. For example, if a system has processed messages associated with virtual times t_1 and t_2, with $t_2 \not\leq t_1$, the work for time t_1 may be preserved while that for time t_2 may be undone, independently of the order in which the work was originally performed.

More generally, each node p in a dataflow graph may roll back to a set of times $f(p)$. This set is not necessarily the same for all nodes, but consistency constrains its choice. For example, virtual times for which a node p has produced visible external output cannot, in general, be outside $f(p)$: the output represents a commitment. Despite such constraints, the flexibility of not having the same $f(p)$ for all nodes is attractive in practice. In particular, choosing a particular $f(p)$ may imply the availability of corresponding logs or checkpoints, and allowing $f(p)$ to vary means that subsystems may adopt their own policies for logging and checkpointing.

The goal of this paper is to describe the design of our technique abstractly, and to present specifications and proofs that have been essential for this design. These specifications and proofs, which require a non-trivial use of prophecy variables [4], go beyond the analysis of internal dependencies (cf., e.g., [6]) to ensure that rollbacks are observationally correct in the sense that every execution with rollbacks is externally indistinguishable from an execution without rollbacks. We

* Most of this work was done at Microsoft Research. M. Abadi is now at Google.

are implementing the design in the context of the Naiad system [11]; we hope to report on this implementation in the future.

The next section reviews the framework that we use for specifications and the model of computation. Section 3 motivates an assumption on buffering external outputs. Section 4 introduces auxiliary concepts and notations needed for the specification of rollback and the corresponding proofs. That specification is the subject of Section 5. Section 6 outlines the main steps of our proof that the model of Section 5, which includes rollback, is a correct refinement of the high-level model of Section 2.3. Section 7 briefly suggests four further elaborations of the rollback mechanism. Finally, Section 8 concludes, and in particular comments on the broader applicability of our ideas and results. An appendix contains additional definitions and proofs.

2 Model of computation

This section describes our model of computation; it is based on another paper [2], which provides further details.

2.1 Basics of specifications and implementations

In this work, as in much work based on temporal logic (e.g., [4, 10]), specifications describe allowed behaviors, which are sequences of states. Each of the sequences starts in an initial state, and every pair of consecutive states is either identical (a "stutter") or related by a step of computation. Formally, a specification is a *state machine*, that is, a triple (Σ, F, N) where the *state space* Σ is a subset of the product of a fixed set Σ_E of externally visible states with a set Σ_I of internal states; the set F of *initial* states is a subset of Σ; and the *next-state relation* N is a subset of $\Sigma \times \Sigma$. The *complete property generated by* a state machine (Σ, F, N) consists of all infinite sequences $\langle\langle s_0, s_1, \ldots \rangle\rangle$ such that $s_0 \in F$ and, for all $i \geq 0$, either $\langle s_i, s_{i+1} \rangle \in N$ or $s_i = s_{i+1}$. This set is closed under stuttering and is a safety property. (We omit fairness conditions, for simplicity.) The *externally visible property generated by* a state machine is the externally visible property induced by its complete property via projection onto Σ_E and closure under stuttering. It need not be a safety property. A state machine $\mathbf{S}$ *implements* a state machine $\mathbf{S}'$ if and only if the externally visible property generated by $\mathbf{S}$ is a subset of the externally visible property generated by $\mathbf{S}'$.

2.2 Basics of timely dataflow

A system is a directed graph (possibly with cycles), in which nodes do the processing and messages are communicated on edges. We write P for the set of nodes. The set of edges is partitioned into input edges I, internal edges E, and output edges O. Edges have sources and destinations (but not always both): for each $i \in I$, $dst(i) \in P$, and $src(i)$ is undefined; for each $e \in E$, $src(e), dst(e) \in P$, and we require that they are distinct; for each $o \in O$, $src(o) \in P$, and $dst(o)$ is

undefined. We refer to nodes and edges as locations. We write M for the set of messages, and M^* for the set of finite sequences of messages.

We assume a partial order of virtual times $(T, \leq)$, and a function *time* from M to T (independent of the order of processing of messages). Each node can request to be notified when it has received all messages for a given virtual time. We allow T to include multiple time domains, that is, subsets that may use different coordinate systems or express different concerns. For example, inside loops, virtual times may be tuples with coordinates that represent iteration counters. Therefore, it is not always meaningful to compare virtual times at different graph locations. For simplicity, we assume that all inputs, notifications, and notification requests (but not outputs) at each node are in the same time domain; if inputs in different time domains are desired, auxiliary relay nodes can translate across time domains.

The state of a system consists of a mapping from nodes to their local states and outstanding notification requests plus a mapping from edges to their contents. We write $\mathit{LocState}(p)$ for the local state of node p, and Σ_{Loc} for the set of local states; $\mathit{NotRequests}(p)$ for p's outstanding notification requests, which are elements of T; and $Q(e)$ for the finite sequence of messages on edge e.

A local history for a node p is a finite sequence that starts with an initial local state that satisfies a given predicate $\mathit{Initial}(p)$, and a set N of initial notification requests, and is followed by events of the forms t and (e, m); these events indicate the received notifications and the received messages with corresponding edges. We write $\mathit{Histories}(p)$ for the set of local histories of p.

We assume that initially, in every behavior of a system, each node p is in a local state that satisfies $\mathit{Initial}(p)$, and p has some set of notification requests; and for each edge $i \in I$ we let $Q(i)$ contain an arbitrary finite sequence of messages, and for each edge $e \in E \cup O$ we let $Q(e)$ be empty. Thereafter, in the absence of rollback, at each step of computation (atomically, for simplicity), a node consumes a notification or a message, and produces notification requests and places messages on its outgoing edges.

The processing of events is defined by a function $g_1(p)$ for each node p, which is applied to p's local state s and to an event x (either a time t or a pair (e, m)), and which produces a new state s', a set of times N, and finite sequences of messages $\mu_1, \ldots, \mu_k$ on p's outgoing edges $e_1, \ldots, e_k$, respectively. We write:

$$g_1(p)(s, x) = (s', N, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle)$$

where $\langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle$ is the function that maps e_1 to μ_1, ..., e_k to μ_k. Iterating $g_1(p)$, we obtain a function $g(p)$ which takes as input a local history h and produces a new state s' and the resulting cumulative notification requests and sequences of messages $\mu_1, \ldots, \mu_k$:

$$g(p)(h) = (s', N, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle)$$

We let $\mathit{\Pi}_{\mathrm{Loc}}(s', N, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = s'$, $\mathit{\Pi}_{\mathrm{NR}}(s', N, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = N$, and $\mathit{\Pi}_{e_i}(s', N, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = \mu_i$ for $i = i \ldots k$.

When one event at a given virtual time t and location l in a dataflow graph can potentially result in another event at a virtual time t' and location l' in the same graph, we say that (l, t) could-result-in (l', t'), and write $(l, t) \rightsquigarrow (l', t')$. For example, when a node p forwards on an outgoing edge e all the messages that it receives on an incoming edge d, we have that $(d, t) \rightsquigarrow (e, t)$ for all t. The could-result-in relation enables an implementation of timely dataflow to support completion notifications, which tell a node when it will no longer see messages for a given time, and also to reclaim resources that correspond to pairs (l, t) at which no more events are possible. Another paper [2] gives a precise definition of $\rightsquigarrow$ and of the assumptions and properties on which we base our proofs. These include, in particular, that $\rightsquigarrow$ is reflexive and transitive.

A set $S \subseteq ((I \cup E \cup O) \cup P) \times T$ is *upward closed* if and only if, for all (l, t) and (l', t'), $(l, t) \in S$ and $(l, t) \rightsquigarrow (l', t')$ imply $(l', t') \in S$. We write $\mathit{Close}_{\uparrow}(S)$ for the least upward closed set that contains S. Since $\rightsquigarrow$ is reflexive and transitive, $\mathit{Close}_{\uparrow}(S)$ consists of the pairs (l', t') such that $(l, t) \rightsquigarrow (l', t')$ for some $(l, t) \in S$.

2.3 High-level specification

Throughout this paper, each element of Σ_E is an assignment of a value to $Q(e)$ for each $e \in I \cup O$ (that is, to $Q{\restriction}(I \cup O)$, where the symbol $\restriction$ denotes function restriction). In other words, the externally visible state consists of the contents of input and output channels. In the high-level specification, each element of Σ_I is an assignment of a value to $\mathit{LocState}(p)$ and $\mathit{NotRequests}(p)$ for each $p \in P$, and to $Q(e)$ for each $e \in E$ (that is, to $Q{\restriction}E$). In our lower-level specifications, below, each element of Σ_I has additional components.

Loosely adopting the TLA [10] approach, we define a high-level specification *SpecR* in Figure 1. We use the following TLA notations. A primed state function (for example, Q') in an action refers to the value of the state function in the "next" state (the state after the action); $\Box$ is the temporal-logic operator "always"; given an action N and a list of expressions $v_1, \ldots, v_k$, $[N]_{v_1,\ldots,v_k}$ abbreviates $N \vee ((v_1' = v_1) \wedge \ldots \wedge (v_k' = v_k))$. Internal state functions are existentially quantified. We also write v for the list of the state components *LocState*, *NotRequests*, and Q, and use the auxiliary state function *Clock* which indicates pairs of a location and a time for which events may remain:

$$\mathit{Clock} = \mathit{Close}_{\uparrow}\left(\begin{array}{c}\{(e, \mathit{time}(m)) \mid e \in I \cup E \cup O, m \in Q(e)\} \\ \cup \\ \{(p, t) \mid p \in P, t \in \mathit{NotRequests}(p)\}\end{array}\right)$$

The predicate *InitProp* defines the initial states of a state machine, while the action $\mathit{MessR} \vee \mathit{Not} \vee \mathit{Inp} \vee \mathit{Outp}$ defines its next-state relation. The disjuncts *MessR*, *Not*, *Inp*, and *Outp* correspond, respectively, to processing messages, processing notification requests, external changes to input edges, and external changes to output edges. Action *Inp* could be further constrained to ensure that it only shrinks *Clock* or leaves it unchanged. Importantly, *MessR* does not strictly require FIFO behavior. Given a queue $Q(e)$, a node may process any message

$$InitProp = \begin{pmatrix} \forall e \in E \cup O.Q(e) = \emptyset \wedge \forall i \in I.Q(i) \in M^* \\ \wedge \forall p \in P.(LocState(p), NotRequests(p)) \in Initial(p) \end{pmatrix}$$

$$MessR = \exists p \in P.MessR1(p)$$

$$MessR1(p) = \begin{pmatrix} \exists e \in I \cup E.p = dst(e) \wedge \exists m \in M.\exists u, v \in M^*. \\ Q(e) = u{\cdot}m{\cdot}v \wedge Q'(e) = u{\cdot}v \wedge \forall n \in u.time(n) \not\leq time(m) \\ \wedge Mess2(p, e, m) \end{pmatrix}$$

$$Mess2(p, e, m) = \begin{pmatrix} let\ \{e_1, \ldots, e_k\} = \{d \in E \cup O \mid src(d) = p\}, \\ s = LocState(p), \\ (s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k\rangle) = g_1(p)(s, (e, m)) \\ in\ LocState'(p) = s' \\ \wedge\ NotRequests'(p) = NotRequests(p) \cup \{t_1, \ldots, t_n\} \\ \wedge\ Q'(e_1) = Q(e_1){\cdot}\mu_1 \ldots Q'(e_k) = Q(e_k){\cdot}\mu_k \\ \wedge\ \forall q \in P \neq p.LocState'(q) = LocState(q) \\ \wedge\ \forall q \in P \neq p.NotRequests'(q) = NotRequests(q) \\ \wedge\ \forall d \in I \cup E \cup O - \{e, e_1, \ldots, e_k\}.Q'(d) = Q(d) \end{pmatrix}$$

$$Not = \exists p \in P.Not1(p)$$

$$Not1(p) = \exists t \in NotRequests(p).$$
$$\forall e \in I \cup E\ such\ that\ dst(e) = p.(e, t) \notin Clock \wedge Not2(p, t)$$

$$Not2(p, t) = \begin{pmatrix} let\ \{e_1, \ldots, e_k\} = \{d \in E \cup O \mid src(d) = p\}, \\ s = LocState(p), \\ (s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k\rangle) = g_1(p)(s, t) \\ in\ LocState'(p) = s' \\ \wedge\ NotRequests'(p) = NotRequests(p) - \{t\} \cup \{t_1, \ldots, t_n\} \\ \wedge\ Q'(e_1) = Q(e_1){\cdot}\mu_1 \ldots Q'(e_k) = Q(e_k){\cdot}\mu_k \\ \wedge\ \forall q \in P \neq p.LocState'(q) = LocState(q) \\ \wedge\ \forall q \in P \neq p.NotRequests'(q) = NotRequests(q) \\ \wedge\ \forall d \in I \cup E \cup O - \{e_1, \ldots, e_k\}.Q'(d) = Q(d) \end{pmatrix}$$

$$Inp = \begin{pmatrix} \forall p \in P.LocState'(p) = LocState(p) \\ \wedge\ \forall p \in P.NotRequests'(p) = NotRequests(p) \\ \wedge\ \forall i \in I.Q(i)\ is\ a\ subsequence\ of\ Q'(i) \\ \wedge\ \forall d \in E \cup O.Q'(d) = Q(d) \end{pmatrix}$$

$$Outp = \begin{pmatrix} \forall p \in P.LocState'(p) = LocState(p) \\ \wedge\ \forall p \in P.NotRequests'(p) = NotRequests(p) \\ \wedge\ \forall o \in O.Q'(o)\ is\ a\ subsequence\ of\ Q(o) \\ \wedge\ \forall d \in I \cup E.Q'(d) = Q(d) \end{pmatrix}$$

$$SpecR = \exists LocState, NotRequests, Q \upharpoonright E.$$
$$InitProp \wedge \Box[MessR \vee Not \vee Inp \vee Outp]_v$$

Fig. 1. High-level specification

m such that there is no message n ahead of m with $time(n) \leq time(m)$. This relaxation has various benefits, for example in supporting optimizations. For our purposes, it is crucial for obtaining a flexible and correct rollback technique. For example, suppose that a node receives a message for time 2 and then a message for time 1. We would like to be able to undo the work for time 2 while preserving the work for time 1. The system will then behave as though the message for time 1 had overtaken the message for time 2. Therefore, our high-level specification should enable such overtaking.

3 An assumption on external outputs

The model of Section 2.2 allows each node to consume and produce multiple events in one atomic action. While such behavior does not pose problems when it is limited to internal edges, it complicates selective rollback when it becomes visible on output edges, as the following example illustrates.

Example 1. Suppose that q has an outgoing edge $o \in O$. Suppose that t_1 and t_2 are incomparable times, and t_3 is greater than both. As long as q receives messages only for time t_1, it forwards them, outputting them on o, with the same "payload" but at time t_3. As soon as q receives a notification for time t_2, it stops doing any forwarding. Suppose that in a run q has received a notification for time t_2 followed by 50 messages for time t_1, so q has not output anything on o. Suppose that we wish to roll back to a state where q has received the messages for time t_1 but not the notification for time t_2. Consistency requires that there should be 50 messages on o. But a rollback action cannot put them there all atomically, since in a run without the notification for time t_2 they would have appeared one after another, not all at once.

This example suggests that rollback can benefit from buffering external outputs. Buffering may be "a simple matter of programming". Alternatively, we can achieve it "for free" by adding buffer nodes (between q and o in the example). More generally, buffer nodes can support asynchronous behavior (see, e.g., [12]). Formally, we say that $p \in P$ is a *buffer node* if there exists exactly one $e_1 \in I \cup E$ such that $dst(e_1) = p$; there exists exactly one $e_2 \in E \cup O$ such that $src(e_2) = p$; for this e_2, $g_1(p)(s,t) = (s, \emptyset, \langle e_2 \mapsto \emptyset \rangle)$; and for this e_1 and e_2, $g_1(p)(s,(e_1,m)) = (s, \emptyset, \langle e_2 \mapsto \langle\!\langle m \rangle\!\rangle \rangle)$. Such a node p is simply a relay between queues. We assume:

Condition 1 *If $o \in O$ and $src(o) = p$ then p is a buffer node.*

4 Auxiliary concepts

This section reviews a few auxiliary concepts and corresponding notations (introduced in the study of information-flow security properties [3]).

4.1 Sequences, frontiers, filtering, and reordering

We write $\emptyset$ for the empty sequence, $\langle\!\langle a_0, a_1, \ldots \rangle\!\rangle$ for a sequence that contains a_0, a_1, ... (as above), and use $\cdot$ both for adding elements to sequences and for appending sequences. We define *subtraction* for sequences, inductively, by:

$$\begin{array}{ll} u - \emptyset = u & \emptyset - m = \emptyset \\ u - m{\cdot}v = (u - m) - v & (m{\cdot}u - m) = u \\ & (n{\cdot}u - m) = n{\cdot}(u - m) \text{ for } n \neq m \end{array}$$

A subset S of T is *downward closed* if and only if, for all t and t', $t \in S$ and $t' \leq t$ imply $t' \in S$. We call such a subset a *frontier*, and write F for the set of frontiers; we often let f range over frontiers. (In the parlance of mathematics, a frontier might be called an ideal; in that of distributed systems, frontiers resemble consistent cuts.) When $S \subseteq T$, we write $\mathit{Close}_{\downarrow}(S)$ for the downward closure of S (the least frontier that contains S).

Filtering operations on histories and on sequences of messages keep or remove elements in a given frontier. Given a local history $h = \langle\!\langle (s, N), x_1, \ldots, x_k \rangle\!\rangle$ and a frontier f, where each x_i is of the form t_i or (d_i, m_i), we write $h@f$ for the subsequence of h obtained by removing all $t_i \notin f$ and all (d_i, m_i) such that $\mathit{time}(m_i) \notin f$. When u is a sequence of messages, we write $u@f$ for the subsequence obtained by removing those messages whose times are not in f. Finally, given a sequence of messages u and a frontier f, we write $u@\!\!\setminus f$ for the subsequence of u consisting only of messages whose times are not in f.

The *reordering* relation $\hookrightarrow$ on finite sequences of messages is the least reflexive and transitive relation such that, for $u, v \in M^*$ and $m_1, m_2 \in M$, if $\mathit{time}(m_1) \not\leq \mathit{time}(m_2)$ then $u{\cdot}m_1{\cdot}m_2{\cdot}v \hookrightarrow u{\cdot}m_2{\cdot}m_1{\cdot}v$. This relation models the reordering that happens in message processing according to action *MessR*, so serves in reasoning with this action.

4.2 Expressing dependencies

Rollback can exploit information on whether a history or a part of a history at a node suffices for determining a notification request or message generated by the node. For example, if we know that p's outputs up to time 1 are determined entirely by its history up to time 1, and a rollback does not affect p's history up to time 1, then the outputs up to time 1 and their consequences downstream do not need to be retracted. Here we consider how to capture such information.

We simply assume that every node's notification requests up to time t are determined by its local history up to time t, for all t. For messages, on the other hand, many useful nodes do not satisfy an analogous property (for example, because of the use of different time domains for inputs and outputs) or satisfy stronger properties that we would want to leverage (as is the case for nodes that increment loop counters). Therefore, we make a more flexible hypothesis: for each edge $e \in E \cup O$, we assume a function $\phi(e)$ from frontiers to frontiers (a *frontier transformer*) such that h gives rise to a message on e in $\phi(e)(f)$ if and only if so does $h@f$, and with messages in the same order and multiplicity:

Condition 2 *For all $f \in F$, if $g(p)(h) = (\ldots, N, \langle \ldots e_i \mapsto \mu_i \ldots \rangle)$ and $g(p)(h@f) = (\ldots, N', \langle \ldots e_i \mapsto \mu'_i \ldots \rangle)$ then $N@f = N'@f$ and $\mu_i@\phi(e_i)(f) = \mu'_i@\phi(e_i)(f)$.*

For example, when e is the output edge of a buffer node, we can let $\phi(e)(f) = f$ for all f, that is, let $\phi(e)$ be the identity function. More generally, $\phi(e)$ may be the identity function for many other edges, but this is not required. Neither is it required that $\phi(e)$ be as precise as possible, though a more precise $\phi(e)$ will generally be more useful. In this paper, we do not investigate how to check that Condition 2 holds for a given ϕ: we simply posit that we can find a correct, useful ϕ. Our experience indicates that this assumption is reasonable.

Additionally, we require that ϕ satisfy the following properties:

Condition 3 *For all $e \in E \cup O$:*

1. *$\phi(e)(f_1) \cap \phi(e)(f_2) \subseteq \phi(e)(f_1 \cap f_2)$ for all $f_1, f_2 \in F$,*
2. *$\phi(e)(T) = T$,*
3. *$\phi(e)$ is monotonic.*

Conditions 3(1) and 3(3) imply that $\phi(e)(f_1) \cap \phi(e)(f_2) = \phi(e)(f_1 \cap f_2)$. In combination with Condition 3(2), they say that $\phi(e)$ distributes over all finite intersections, including the empty intersection that yields T. (We can justify a stronger property, namely that $\phi(e)$ distributes over arbitrary intersections [3].)

5 Low-level specification (with rollback)

The low-level specification, which permits rollback, has the same state components as the high-level specification plus an internal variable H that maps each $p \in P$ to a local history in $\mathit{Histories}(p)$. The resulting state space is Σ_{Low}. Figure 2 defines the specification. There, we write v for the list of the state components *LocState*, *NotRequests*, Q, and H.

The main novelties are in the action *RollbackL*. This action creates a global state from local histories filtered down to frontiers $f(p)$ by applying the function $g(p)$, for each node p. Among other things, for each outgoing internal edge e_i this function yields messages μ_i, from which messages in $\phi(e_i)(f(p)) \cap f(\mathit{dst}(e_i))$ are expunged, intuitively because e_i's destination should already have them.

Crucially, the global state is completely determined by the local histories, since $g(p)$ is a function. So we are not concerned with recording non-deterministic choices, other than those encoded in local histories, in order to ensure consistency (for example, in order to ensure that any internal choices revealed by external outputs are made in the same way at each rollback). As discussed in Section 7.2, the creation of the global state may be accelerated by precomputation; the specification is silent on such implementation matters.

The choice of the frontiers $f(p)$ is subject to several constraints. (Section 7.3 briefly considers how to pick frontiers that satisfy these constraints.) A guard in *RollbackL2*(f, p) requires that $f(p)$ cannot contain times for which there are messages in transit towards p on internal edges, basically because, in practice,

$$InitPropL = InitProp \wedge \forall p \in P.H(p) = \langle\!\langle (LocState(p), NotRequests(p)) \rangle\!\rangle$$

$$\begin{aligned} MessL = {} & \exists p \in P. \\ & MessR1(p) \wedge H'(p) = H(p) \cdot (e, m) \wedge \forall q \in P \neq p.H'(q) = H(q) \end{aligned}$$

$$NotL = \exists p \in P.NotL1(p)$$

$$\begin{aligned} NotL1(p) = {} & \exists t \in NotRequests(p). \\ & \begin{pmatrix} \forall e \in I \cup E \text{ such that } dst(e) = p.(e,t) \notin Clock \\ \wedge\ Not2(p,t) \\ \wedge\ H'(p) = H(p) \cdot t \wedge \forall q \in P \neq p.H'(q) = H(q) \end{pmatrix} \end{aligned}$$

$$InpL = Inp \wedge \forall p \in P.H'(p) = H(p)$$

$$OutpL = Outp \wedge \forall p \in P.H'(p) = H(p)$$

$$RollbackL = \exists f \in P \rightarrow F.RollbackL1(f)$$

$$RollbackL1(f) = \begin{pmatrix} \forall p \in P, i \in I \text{ such that } dst(i) = p. \\ \quad \{time(m) \mid (i,m) \in H(p)\} \subseteq f(p) \\ \wedge\ \forall p \in P, o \in O \text{ such that } src(o) = p. \\ \quad \{time(m) \mid \exists e.(e,m) \in H(p)\} \subseteq f(p) \\ \wedge\ \forall p, q \in P, e \in E \text{ such that } src(e) = p \wedge dst(e) = q. \\ \quad \{time(m) \mid (e,m) \in H(q)\} \cap f(q) \subseteq \phi(e)(f(p)) \\ \wedge\ \forall p, q \in P, e_1, e_2 \in E \text{ such that } src(e_1) = p \wedge dst(e_2) = q, \\ \quad t_1 \in T, t_2 \in H(q)@f(q). \\ \quad \text{if } (e_1, t_1) \rightsquigarrow (e_2, t_2) \text{ then } t_1 \in \phi(e_1)(f(p)) \\ \wedge\ \forall e \in I \cup O.Q'(e) = Q(e) \\ \wedge\ \forall p \in P.RollbackL2(f,p) \end{pmatrix}$$

$$RollbackL2(f,p) = \begin{pmatrix} f(p) \cap \{time(m) \mid \exists e \in E, m \in M.dst(e) = p \wedge m \in Q(e)\} = \emptyset \\ \wedge \\ let\ \{e_1, \ldots, e_k\} = \{d \in E \cup O \mid src(d) = p\}, \\ h = H(p)@f(p), \\ (s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = g(p)(h) \\ in \\ \forall i \in 1 \ldots k.\ \text{if } e_i \in E \\ \qquad \text{then } Q'(e_i) = \mu_i @ (\phi(e_i)(f(p)) \cap f(dst(e_i))) \\ \wedge\ LocState'(p) = s' \\ \wedge\ NotRequests'(p) = \{t_1, \ldots, t_n\} \\ \wedge\ H'(p) = h \end{pmatrix}$$

$$\begin{aligned} SpecL = {} & \exists LocState, NotRequests, H, Q \restriction E. \\ & InitPropL \wedge \Box[MessL \vee NotL \vee InpL \vee OutpL \vee RollbackL]_v \end{aligned}$$

Fig. 2. Low-level specification

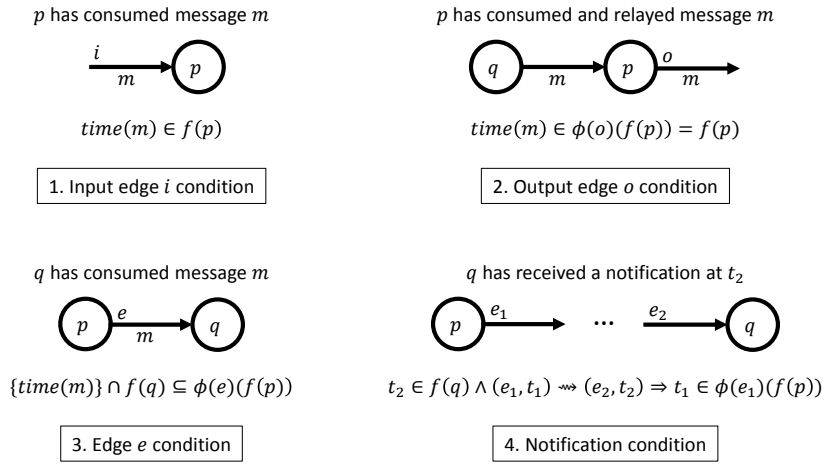

Fig. 3. The guards in *RollbackL1*(f)

any messages in transit on internal edges may be lost during the failures that cause rollbacks. One way to ensure that this guard holds is to pick $f(p)$ so that it contains only times t for which $(p, t) \notin \mathit{Clock}$. In addition, while the frontiers need not be the same at all nodes, guards in *RollbackL1*(f) ensure that they are chosen to be consistent with one another and with external inputs and outputs:

1. For each node p connected to an input edge $i \in I$, $f(p)$ must contain the times of all messages that p has consumed: we do not assume any external mechanism for replaying those messages after a rollback, so p should not forget them, in general. (Optimizations may however allow p to forget many messages in practice.)
2. For each node p connected to an output edge $o \in O$, which must be a buffer node, $f(p)$ must contain the times of all messages that p has consumed and therefore relayed: they cannot be retracted.
3. For each pair of nodes p and q and edge e from p to q, $\phi(e)(f(p))$ must contain the times of all the messages that q has consumed and that are in $f(q)$: since q keeps those messages, p must keep the part of its local history that determines them.
4. Finally, an analogous but more complicated guard refers to notifications. It ensures that if a node q has received a notification for a time $t_2 \in f(q)$, then every node p that might cause events at t_2 at q keeps the part of its local history that would determine those events.
   Below, in Section 7.4, we develop a refinement of *SpecL* that simplifies the treatment of this guard.

Figure 3 summarizes these four guards; when a message m is shown under an edge, it means that m has been transmitted on that edge (not that it is currently

in transit). The following two examples illustrate the role of guards (3) and (4), respectively.

Example 2. Suppose that p has incoming edges d_0 and d_1 and outgoing edges e_0 and e_1 to q_0 and q_1, respectively; and that q_0 and q_1 are buffer nodes, with respective output edges o_0 and o_1. Suppose that p forwards all messages from d_0 on e_0 or from d_1 on e_1, but not both, depending on whether it reads first from d_0 or from d_1. For simplicity, initially, we let $\phi(e)$ be the identity function for every edge e. Assume that, in a particular run, a message with time 0 has travelled from d_0 to o_0 via p, e_0, and q_0. Upon a rollback, let $0 \in f(q_0)$, as suggested by guard (2). Guard (3) then dictates that $0 \in f(p)$. If instead we had $f(p) = \emptyset$, and upon recovery p first reads from d_1, it would cause an output on o_1, which is inconsistent with the previous output on o_0.

As a variant, suppose that p increments all virtual times, so the message from d_0 yields an output on o_0 with time 1. We take $\phi(e_0)(f) = \{0\} \cup \{t+1 \mid t \in f\}$. Upon a rollback, we should have $1 \in f(q_0)$, but we do not need to impose that $1 \in f(p)$: the application of $\phi(e_0)$ implies that $0 \in f(p)$ suffices.

Example 3. Suppose that p_0 and p_1 each send a message to p_2, with times 0 and 1 respectively. Suppose further that, when it receives a message with time 0, p_2 forwards its "payload" with time 1 to p_3 on an edge e, but only if it has not yet processed a message with time 1. Therefore, we have that $(p_2, 0) \rightsquigarrow (p_3, 1)$ but not $(p_2, 1) \rightsquigarrow (p_3, 1)$, and we can let $\phi(e)$ be the identity function. Assume that, in a particular run, p_2 hears from p_1 first, then from p_0, so p_2 never sends anything to p_3; then p_3 receives a completion notification for time 1; and then a rollback takes place with $f(p_2) = \{0\}$. Upon recovery, it appears to p_2 that it heard from p_0 first, so it should send a message at time 1 to p_2. As this message would contradict the completion notification for time 1, we cannot have $1 \in f(p_3)$. Guard (4) prevents it.

As mentioned in Section 4.2, we need not have the most accurate function ϕ. Since the occurrences of ϕ in the guards for *RollbackL* are all in positive positions, using a more informative ϕ makes rollbacks more liberal. On the other hand, using a less informative ϕ (obtained by some under-approximation) does not compromise soundness. Section 7.1 discusses other approximations that may be attractive in practice.

6 Refinement theorem

Our main result is that the low-level specification *SpecL* implements the high-level specification *SpecR*. In other words, the behaviors in the externally visible property of the system with rollback are all in the externally visible property of the system without rollback; so, externally, one cannot tell whether a behavior includes rollback transitions.

For safety properties, we can prove implementation relations by reasoning only about finite prefixes of behaviors: when *Spec* is a safety property, if every

prefix of a behavior in *Spec′* is the prefix of a behavior in *Spec* then *Spec′* implements *Spec*. While such reasoning may not be easy, it can avoid complications related to liveness properties, such as finiteness requirements on prophecy variables [4]. Unfortunately, although *SpecR*'s body ($\mathit{InitProp} \wedge \Box[\mathit{MessR} \vee \mathit{Not} \vee \mathit{Inp} \vee \mathit{Outp}]_v$) is clearly a safety property, *SpecR* itself need not be one, because safety properties are not closed under existential quantification. Therefore, our proof does have to address those complications.

The proof is rather long, so we cannot present it in full detail, but we hope to convey its main elements.

Invariant As in many refinement proofs, the first step is to establish an inductive invariant of the low-level specification. In this case, the invariant, which we call *Inv*, relates the function g to elements of the state, at each node and edge. It is the conjunction of the following formulas:

$$\begin{array}{l}\forall p \in P. \Pi_{\mathrm{Loc}} g(p)(H(p)) = \mathit{LocState}(p) \\ \forall p \in P. \Pi_{\mathrm{NR}} g(p)(H(p)) = \mathit{NotRequests}(p) \\ \forall p, q \in P, e \in E \text{ such that } \mathit{src}(e) = p \wedge \mathit{dst}(e) = q. \\ \quad \Pi_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e, m) \in H(q) \rangle \cdot Q(e))\end{array}$$

Prophecy variable Constructing a refinement mapping is a sound method for proving an implementation relation. Unfortunately, it is not complete on its own [4]. Auxiliary variables are often required to complement refinement mappings. In our case, we cannot find a refinement mapping basically because we cannot predict the effects of future rollbacks, but the addition of a prophecy variable to the low-level specification can provide the required, missing information. (Another paper [1] explains this situation in detail with much smaller examples, not tied to timely dataflow.) Specifically, we add an auxiliary variable D that maps each node p to a frontier. Intuitively, $D(p)$ consists of times not affected by future rollbacks at p. Formally, $D(p)$ is subject to a number of constraints, imposed via the definition of an enriched state space Σ^P_{Low}:

Construction 1 *The enriched state space Σ^P_{Low} consists of pairs of a state from Σ_{Low} (with components LocState, NotRequests, Q, and H) and a function D from P to F such that:*

1. $\forall p \in P, \forall i \in I$ *such that* $\mathit{dst}(i) = p. \{\mathit{time}(m) \mid (i, m) \in H(p)\} \subseteq D(p)$,
2. $\forall p \in P, \forall o \in O$ *such that* $\mathit{src}(o) = p. \{\mathit{time}(m) \mid \exists e. (e, m) \in H(p)\} \subseteq D(p)$,
3. $\forall p, q \in P, \forall e \in E$ *such that* $\mathit{src}(e) = p, \mathit{dst}(e) = q, \forall m \in M$ *such that* $m \in Q(e) \vee (e, m) \in H(q)$. *if* $\mathit{time}(m) \in D(q)$ *then* $\mathit{time}(m) \in \phi(e)(D(p))$,
4. $\forall p, q \in P, \forall e_1, e_2 \in E$ *such that* $\mathit{src}(e_1) = p, \mathit{dst}(e_2) = q, \forall t_1 \in T, t_2 \in H(q) @ D(q)$. *if* $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ *then* $t_1 \in \phi(e_1)(D(p))$.

These conditions are analogous to those in the definition of the action *RollbackL*. However, they apply at all times, not just during rollbacks. This distinction largely accounts for the small differences between them.

The following specification extends *SpecL* with conjuncts that describe the changes in D. Here, v is the list of all previous state components plus D.

$$
\begin{aligned}
\mathit{InitPropP} &= \mathit{InitPropL} \\
\mathit{MessP} &= \mathit{MessL} \land D = D' \\
\mathit{NotP} &= \mathit{NotL} \land D = D' \\
\mathit{InpP} &= \mathit{InpL} \land D = D' \\
\mathit{OutpP} &= \mathit{OutpL} \land D = D' \\
\mathit{RollbackP} &= \exists f \in P \to F.\mathit{RollbackL1}(f) \land D = (D' \cap f) \\
\mathit{SpecP} &= \exists \mathit{LocState}, \mathit{NotRequests}, H, Q \restriction E, D \textit{ as per Construction 1.} \\
&\qquad \mathit{InitPropP} \land \Box[\mathit{MessP} \lor \mathit{NotP} \lor \mathit{InpP} \lor \mathit{OutpP} \lor \mathit{RollbackP}]_v
\end{aligned}
$$

Most transitions are such that $D = D'$. The exception is in rollbacks, where we have $D = D' \cap f$. In other words, at each node p, the times $D(p)$ that will survive future rollbacks, starting from before a particular rollback to $f(p)$, are those in $f(p)$ that will survive future rollbacks after the transition, that is, those in $D'(p) \cap f(p)$. Characteristically, the current value D of the prophecy variable is defined from its next value D'.

There are standard conditions on what it means to be a prophecy variable. Unfortunately, D, as we have defined it, does not quite satisfy them, because each state in Σ_{Low} may yield infinitely many states in Σ^P_{Low}. We address this difficulty by quotienting by the equivalence relation $\mathcal{Q}$ such that $(s_1, D_1)\mathcal{Q}(s_2, D_2)$ if and only if $s_1 = s_2$ and D_1 and D_2 coincide on the finite set of times $\mathit{ITimes}(H)$, where H is the history component of s_1 and s_2, and $\mathit{ITimes}(H)$ consists of all $t \in T$ such that, for some p, $\mathit{time}(m) = t$ for some $(e, m) \in H(p)$ or $t \in H(p)$. Intuitively, $\mathit{ITimes}(H)$ consists of the "interesting" times given H. For each $s \in \Sigma_{\mathrm{Low}}$, the set of equivalence classes $\{(s, D) \in \Sigma^P_{\mathrm{Low}}\}/\mathcal{Q}$ is finite. The soundness of prophecy variables yields that *SpecL* implements the quotient of *SpecP* by $\mathcal{Q}$, which we call $\mathit{SpecP}_{/\mathcal{Q}}$.

Refinement mapping Using the invariant *Inv* and the auxiliary variable D, we construct a refinement mapping from the low-level specification to the high-level specification. This mapping is a function from Σ^P_{Low} to Σ_{High} that preserves externally visible state components, maps initial states to initial states, and maps steps to steps or to stutters. Basically, it maps a low-level state to a high-level state by pretending that each node p did not process any events with times outside $D(p)$. Formally, it is defined by the following state functions:

$$
\begin{aligned}
\mathit{HLocState}(p) &= \Pi_{\mathrm{Loc}} g(p)(H(p)@D(p)) \\
\mathit{HNotRequests}(p) &= \Pi_{\mathrm{NR}} g(p)(H(p)@D(p)) \\
\mathit{HQ}(e) &= Q(e) \qquad \text{for } e \in I \cup O \\
\mathit{HQ}(e) &= \Pi_e g(p)(H(p)@D(p)) - \langle m \mid m \in M, (e, m) \in H(q)\rangle @D(q) \\
&\qquad \text{where } p = \mathit{src}(e) \text{ and } q = \mathit{dst}(e), \text{ for } e \in E
\end{aligned}
$$

Thus, for each node p, $HLocState(p)$ and $HNotRequests(p)$ are obtained by applying $g(p)$ to p's filtered local history. Similarly, for an internal edge e from a node p to a node q, $HQ(e)$ is obtained by applying $g(p)$ to p's filtered local history, but subtracting messages that q has consumed according to its filtered local history. When e is an input or an output edge, $HQ(e)$ simply equals $Q(e)$, since $Q(e)$ is externally visible.

This refinement mapping respects the equivalence relation $\mathcal{Q}$. In other words, it maps equivalent low-level states to the same high-level state. Therefore, the refinement mapping from Σ^P_{Low} induces a refinement mapping from $\mathcal{Q}$'s equivalence classes, and the soundness of refinement mappings yields that $SpecP_{/\mathcal{Q}}$ implements $SpecR$. By transitivity, we conclude:

Theorem 1. *SpecL implements SpecR.*

7 Further refinements

In this section we consider several further refinements of the low-level specification. Our main goal is to provide evidence that the low-level specification is a useful step towards concrete, correct implementations, and to indicate some possible features of those implementations.

7.1 Approximating the clock and the could-result-in relation

The low-level specification mentions the state function *Clock* and also refers to the relation $\rightsquigarrow$ directly. Both of these may be hard to calculate precisely and efficiently. Fortunately, it is sound to replace *Clock* with any over-approximation. Using a bigger state function will mean that fewer notifications may be delivered at any point, so may result in a smaller set of behaviors. Similarly, it is sound to over-approximate the relation $\rightsquigarrow$ in its other use in the specification (since it is in a negative position).

7.2 Precomputations

While the specification of *RollbackL* suggests that a new state can be computed by applying the function g to filtered local histories, this computation may be expensive, and there is no requirement that an implementation perform it naively and on the fly. In particular, an implementation may compute and store the values of $g(p)(h)$ for certain local histories h, as it runs. For this purpose, an implementation might leverage commutativity properties that could require careful analysis. In any case, much like traditional checkpoints, these values can later facilitate rollback. Formally, the precomputation simply provides an efficient way of satisfying the equations $h = H(p)@f(p)$ and $(s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = g(p)(h)$ in *RollbackL*.

7.3 Choosing frontiers

The specification of *RollbackL* does not describe how to find a function f that satisfies its constraints. A possible refinement consists in specifying that it chooses the largest solution (the function that yields the largest frontiers), which is the one that entails the least rollback. A largest solution exists because the set of functions that satisfy the constraints is closed under arbitrary unions. A further refinement consists in specifying that it chooses the largest solution that has some additional properties. For example, each processor may be willing to roll back only to some subset of "available" frontiers (possibly ones for which it has checkpoints, or ones that do not contain times of events deemed problematic for whatever reason). A largest solution will still exist as long as the set of "available" frontiers is closed under unions. We are currently exploring whether and how finding largest solutions can be practical, at least in special cases.

7.4 Implementing the guard on notifications

One of the guards in *RollbackL* (guard (4)) includes a relatively complex condition that refers to the relation $\rightsquigarrow$ and potentially requires some checking for all pairs of edges. We can replace that condition with one that necessitates only simpler checks. For this purpose, we introduce an additional function f_c. For all p, $f_c(p)$ is a subset of $f(p)$ and, intuitively, represents those times for which events must be preserved in order to respect notifications. The specification *SpecS* is obtained from *SpecL* by replacing that guard with the requirement that, for some $f_c : P \rightarrow F$:

1. $\forall p \in P. f_c(p) \subseteq f(p)$,
2. $\forall p \in P, t \in T.$ *if* $t \in H(p)@f(p)$ *then* $t \in f_c(p)$, and
3. $\forall p, q \in P, e \in E$ *such that* $src(e) = p \wedge dst(e) = q. f_c(q) \subseteq \phi(e)(f_c(p))$.

We can prove that these conditions are sufficient (but not necessary). We obtain:

Theorem 2. *SpecS implies SpecL.*

8 Conclusion

This paper describes and studies, formally, the design of a technique for rollback recovery. This technique is delicate, so its rigorous development has been beneficial. The required proofs have been challenging but, in our opinion, interesting, in particular because of the advanced application of prophecy variables.

The main motivation for our work has been fault-tolerance in the timely-dataflow model of computation. However, some of the machinery that we have developed is more broadly applicable. In particular, rollbacks may arise not only because of failures but also, for example, to undo speculative computations or to revert the effects of attacks. Moreover, the use of functions on frontiers for expressing dependencies and some of the corresponding end-to-end results

may be valuable even for systems without rollback. (Some of the proofs in our work on information-flow security properties [3] resemble those of this paper, but they are considerably simpler, and in particular do not include prophecy variables.) Finally, some of the ideas developed here may help explain, in a common framework, specific schemes for recovery in models less general than timely dataflow (e.g., [5]).

Acknowledgments We are grateful to our coauthors in work on Naiad for discussions that led to this paper.

References

1. Abadi, M.: The prophecy of undo. In: Egyed, A., Schaefer, I. (eds.) Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering. Springer (2015), to appear.
2. Abadi, M., Isard, M.: Timely dataflow: A model (2014), in preparation, draft available at `https://users.soe.ucsc.edu/~abadi/allpapers-chron.html`.
3. Abadi, M., Isard, M.: On the flow of data, information, and time. In: Focardi, R., Myers, A. (eds.) Proceedings of the 4th Conference on Principles of Security and Trust. Springer (2015), to appear.
4. Abadi, M., Lamport, L.: The existence of refinement mappings. Theoretical Computer Science 82(2), 253–284 (1991)
5. Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., Whittle, S.: MillWheel: Fault-tolerant stream processing at Internet scale. Proceedings of the VLDB Endowment 6(11) (Aug 2013)
6. Alvisi, L., Marzullo, K.: Message logging: Pessimistic, optimistic, causal, and optimal. IEEE Transactions on Software Engineering 24(2), 149–159 (1998)
7. Elnozahy, E.N., Alvisi, L., Wang, Y., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)
8. Jefferson, D.R.: Virtual time. ACM Transactions on Programming Languages and Systems 7(3), 404–425 (Jul 1985)
9. Kahn, G.: The semantics of simple language for parallel programming. In: IFIP Congress. pp. 471–475 (1974)
10. Lamport, L.: Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002)
11. Murray, D.G., McSherry, F., Isaacs, R., Isard, M., Barham, P., Abadi, M.: Naiad: a timely dataflow system. In: ACM SIGOPS 24th Symposium on Operating Systems Principles. pp. 439–455 (2013)
12. Selinger, P.: First-order axioms for asynchrony. In: Mazurkiewicz, A.W., Winkowski, J. (eds.) CONCUR '97: Concurrency Theory, 8th International Conference. vol. 1243, pp. 376–390. Springer (1997)

Appendix

This appendix contains additional definitions and proofs. Some of them, in particular basic ones about the model of computation, are from other papers; we include them here for the convenience of readers. The proofs of the main results of this paper are of course new.

Proof techniques

The following results (from [4, 1]) provide basic proof techniques. We state them without justification.

Proposition 1 (Soundness of refinement mappings). *If there exists a refinement mapping from* $\mathbf{S}$ *to* $\mathbf{S}'$*, then* $\mathbf{S}$ *implements* $\mathbf{S}'$*.*

We say that $\mathbf{S^P} = (\Sigma^P, F^P, N^P)$ is *obtained from* $\mathbf{S} = (\Sigma, F, N)$ *by adding a very simple prophecy variable* when the following conditions are satisfied:

P1. $\Sigma^P \subseteq \Sigma \times \Sigma_P$ for some set Σ_P.
P2′. $F^P = \{(s,p) \in \Sigma^P \mid s \in F\}$.
P4′. If $\langle s, s' \rangle \in N$ and $(s', p') \in \Sigma^P$
then there exists $(s,p) \in \Sigma^P$ such that $\langle (s,p), (s',p') \rangle \in N^P$.
P6. For all $s \in \Sigma$, the set $\{(s,p) \in \Sigma^P\}$ is finite and nonempty.

We say that $\mathbf{S^P} = (\Sigma^P, F^P, N^P)$ is *obtained from* $\mathbf{S} = (\Sigma, F, N)$ *almost by adding a very simple prophecy* when the usual conditions P1, P2′, and P4′ hold, and instead of P6 we have only part of it:

P6′. For all $s \in \Sigma$, the set $\{(s,p) \in \Sigma^P\}$ is nonempty.

Proposition 2 (Soundness of very simple prophecy variables). *If* $\mathbf{S^P}$ *is obtained from* $\mathbf{S}$ *by adding a very simple prophecy variable, then* $\mathbf{S}$ *implements* $\mathbf{S^P}$*.*

When $\mathcal{Q}$ is an equivalence relation on Σ, the *quotient* of the state machine $\mathbf{S} = (\Sigma, F, N)$ by $\mathcal{Q}$ is the state machine $\mathbf{S}_{/\mathcal{Q}} = (\Sigma_{/\mathcal{Q}}, F_{/\mathcal{Q}}, N_{/\mathcal{Q}})$ such that:

Q1. $\Sigma_{/\mathcal{Q}} = \Sigma/\mathcal{Q}$ (the set of equivalence classes of states from Σ).
Q2. For $s \in \Sigma_{/\mathcal{Q}}$, $s \in F_{/\mathcal{Q}}$ if and only if there exists $s' \in s$ such that $s' \in F$.
Q3. For $s, t \in \Sigma_{/\mathcal{Q}}$, $\langle s, t \rangle \in N_{/\mathcal{Q}}$ if and only if
there exist $s' \in s$ and $t' \in t$ such that $\langle s', t' \rangle \in N$.

We represent each equivalence class in $\Sigma/\mathcal{Q}$ by an arbitrary member (say, the smallest in a fixed ordering of Σ), so that $\Sigma/\mathcal{Q} \subseteq \Sigma$ and $\Sigma/\mathcal{Q} \subseteq \Sigma_E \times \Sigma_I$ for some set Σ_I. We write $[s]$ for s's equivalence class.

We say that a function f with domain Σ *respects* the equivalence relation $\mathcal{Q}$ on Σ when, for all $s, t \in \Sigma$, if $s \mathcal{Q} t$ then $f(s) = f(t)$.

Proposition 3. *Let* $\mathbf{S} = (\Sigma, F, N)$ *and* $\mathbf{S}' = (\Sigma', F', N')$*, and let* $\mathbf{S}_{/\mathcal{Q}}$ *be the quotient of* $\mathbf{S}$ *by an equivalence relation* $\mathcal{Q}$ *on* Σ*. Assume that:*

- f *is a refinement mapping from* $\mathbf{S}$ *to* $\mathbf{S}'$*,*
- f *respects* $\mathcal{Q}$*.*

Let $f_{/\mathcal{Q}} : \Sigma/\mathcal{Q} \to \Sigma'$ *be such that, for all* $s \in \Sigma/\mathcal{Q}$*,* $f_{/\mathcal{Q}}(s) = f(s)$*. Then* $f_{/\mathcal{Q}}$ *is a refinement mapping from* $\mathbf{S}_{/\mathcal{Q}}$ *to* $\mathbf{S}'$*.*

Proposition 4. *Let* $\mathbf{S} = (\Sigma, F, N)$ *and* $\mathbf{S^P} = (\Sigma^P, F^P, N^P)$*, and let* $\mathbf{S^P_{/\mathcal{Q}}} = (\Sigma^P_{/\mathcal{Q}}, F^P_{/\mathcal{Q}}, N^P_{/\mathcal{Q}})$ *be the quotient of* $\mathbf{S^P}$ *by the equivalence relation* $\mathcal{Q}$ *on* Σ^P*. Assume that:*

1. $\mathbf{S^P}$ *is obtained from* $\mathbf{S}$ *almost by adding a very simple prophecy variable,*
2. *for all* $(s_1, p_1), (s_2, p_2) \in \Sigma^P$*, if* $(s_1, p_1)\mathcal{Q}(s_2, p_2)$ *then* $s_1 = s_2$ *(in other words, projecting to the first component respects* $\mathcal{Q}$*),*
3. *for all* $s \in \Sigma$*, the set* $\{(s, p) \in \Sigma^P\}/\mathcal{Q}$ *is finite.*

Then $\mathbf{S^P_{/\mathcal{Q}}}$ *is obtained from* $\mathbf{S}$ *by adding a very simple prophecy variable.*

We say that *Inv* is an *inductive invariant* of the state machine $\mathbf{S} = (\Sigma, F, N)$ if $Inv \subseteq \Sigma$, $F \subseteq Inv$, and, for all $s, t \in \Sigma$, if $\langle s, t\rangle \in N$ and $s \in Inv$ then $t \in Inv$. We sometimes refer to inductive invariants as invariants, for short, but there is a distinction between the two in the literature.

Given a subset *Inv* of Σ and an equivalence relation $\mathcal{Q}$ on Σ, we write $Inv_{/\mathcal{Q}}$ for the subset of $\Sigma_{/\mathcal{Q}}$ such that $s \in Inv_{/\mathcal{Q}}$ if and only if there exists $s' \in s$ such that $s' \in Inv$. (This notation generalizes the definition of $F_{/\mathcal{Q}}$.)

We say that *Inv respects* the equivalence relation $\mathcal{Q}$ when, for all $s, t \in \Sigma$, if $s\mathcal{Q}t$ and $s \in Inv$ then $t \in Inv$.

Proposition 5. *Assume that:*

- *Inv is an inductive invariant of* $\mathbf{S} = (\Sigma, F, N)$*,*
- *Inv respects the equivalence relation* $\mathcal{Q}$ *on* Σ*.*

Then $Inv_{/\mathcal{Q}}$ *is an inductive invariant of* $\mathbf{S}_{/\mathcal{Q}} = (\Sigma_{/\mathcal{Q}}, F_{/\mathcal{Q}}, N_{/\mathcal{Q}})$*.*

Given a specification $\mathbf{S} = (\Sigma, F, N)$ and a subset *Inv* of Σ, we write $\mathbf{S} + Inv$ for $(\Sigma, F, N \cap (Inv \times \Sigma))$.

Proposition 6. *Assume that Inv is an inductive invariant of* $\mathbf{S} = (\Sigma, F, N)$*. Then* $\mathbf{S}$ *and* $\mathbf{S} + Inv$ *generate the same complete property.*

Proposition 7. *Assume that Inv respects* $\mathcal{Q}$*. Then the complete property that* $\mathbf{S}_{/\mathcal{Q}} + Inv_{/\mathcal{Q}}$ *generates is included in that of* $(\mathbf{S} + Inv)_{/\mathcal{Q}}$*.*

Review of the could-result-in relation

We review the definition of the could-result-in relation and some of its properties. Some of this material is from a paper on the computational model [2].

A pointstamp is a pair (x, t) of a location x (node or edge) in a graph and a time t. Thus, the set of pointstamps is $((I \cup E \cup O) \cup P) \times T$. We say that pointstamp (x, t) could-result-in pointstamp (x', t'), and write $(x, t) \rightsquigarrow (x', t')$, if a message or notification at location x and time t may lead to a message or notification at location x' and time t'. We define $\rightsquigarrow$ via an auxiliary relation $\rightsquigarrow^1$ that reflects one step of computation.

Definition 1. *$(p, t) \rightsquigarrow^1 (d, t')$ if and only if $src(d) = p$ and there exists a history h for p and a state s such that*

$$g(p)(h) = (s, \ldots)$$

and an event x such that either $x = t$ or $x = (e, m)$ for some e and m such that $t = time(m)$, and

$$g_1(p)(s, x) = (\ldots, \langle \ldots d \mapsto \mu \ldots \rangle)$$

where some element of μ has time t'.

Definition 2. *$(x, t) \rightsquigarrow (x', t')$ if and only if*

- *$x = x'$ and $t \leq t'$, or*
- *there exist $k > 1$, distinct x_i for $i = 1 \ldots k$, and (not necessarily distinct) t_i for $i = 1 \ldots k$, such that $x = x_1$, $x' = x_k$, $t \leq t_1$, and $t_k \leq t'$, and for all $i = 1 \ldots k - 1$:*
  - *$x_i \in I \cup E$, $x_{i+1} \in P$, $dst(x_i) = x_{i+1}$, and $t_i = t_{i+1}$, or*
  - *$x_i \in P$, $x_{i+1} \in E \cup O$, $src(x_{i+1}) = x_i$, and there exist $t'_i \geq t_i$ and $t''_i \leq t_{i+1}$ such that $(x_i, t'_i) \rightsquigarrow^1 (x_{i+1}, t''_i)$.*

In the first case, we say that the proof of $(x, t) \rightsquigarrow (x', t')$ has length 1; in the second, that it has length k. (These lengths are helpful in inductive arguments. Despite the informal wording, we do not need proofs to be unique.)

Proposition 8.

1. *If $(p, t_1) \rightsquigarrow (e, t_2)$ then there are $e' \in E \cup O$ and $t' \in T$ such that $src(e') = p$, $(p, t_1) \rightsquigarrow (e', t')$ and $(e', t') \rightsquigarrow (e, t_2)$.*
   Moreover, the proof of $(e', t') \rightsquigarrow (e, t_2)$ is strictly shorter than that of $(p, t_1) \rightsquigarrow (e, t_2)$.
2. *If $(e_1, t_1) \rightsquigarrow (e_2, t_2)$, where e_1 and e_2 are different, and $dst(e_1) = p$, then $(p, t_1) \rightsquigarrow (e_2, t_2)$.*
   Moreover, the proof of $(p, t_1) \rightsquigarrow (e_2, t_2)$ is strictly shorter than that of $(e_1, t_1) \rightsquigarrow (e_2, t_2)$.
3. *If $(x, t_1) \rightsquigarrow (x, t_2)$ then $t_1 \leq t_2$.*

4. *If* $(p, t_1) \rightsquigarrow (e, t_2)$ *and* $src(e) = p$ *then there exist a history* h *for* p *and a state* s *such that*

$$g(p)(h) = (s, \ldots)$$

and an event t' *for some* t' *such that* $t_1 \leq t'$ *or* $x = (d, m)$ *for some* d *and* m *such that* $t_1 \leq time(m)$*, and*

$$g_1(p)(s, x) = (\ldots, \langle \ldots e \mapsto \mu \ldots \rangle)$$

where some element of μ *has time* $\leq t_2$*.*
5. *If* $(x, t_1) \rightsquigarrow (i, t_2)$ *for* $i \in I$*, then* $x = i$*.*
6. *If* $(o, t_1) \rightsquigarrow (x, t_2)$ *for* $o \in O$*, then* $x = o$*.*
7. *If* $(x_1, t_1) \rightsquigarrow (x_2, t_2)$*,* $t_1' \leq t_1$*, and* $t_2 \leq t_2'$*, then* $(x_1, t_1') \rightsquigarrow (x_2, t_2')$*.*

Proof:

1. Suppose that $(p, t) \rightsquigarrow (e, t')$. Since p and e are different, by the definition of $\rightsquigarrow$ there exist distinct x_i for $i = 1 \ldots k$, t_i for $i = 1 \ldots k$ such that $p = x_1$, $e = x_k$, $t \leq t_1$ and $t_k \leq t'$, and for all $i = 1 \ldots k - 1$,
   (a) if x_i is an edge $e' \in I \cup E$, then x_{i+1} is a node in P, $dst(e') = x_{i+1}$, and $t_i = t_{i+1}$,
   (b) if x_i is a node $p' \in P$, then x_{i+1} is some edge $e' \in E \cup O$, and $src(e') = p'$, and there exists a history h for p' and a state s such that

$$g(p')(h) = (s, \ldots)$$

   and an event t' for some t' such that $t_i \leq t'$ or $x = (d, m)$ for some d and m such that $t_i \leq time(m)$, and

$$g_1(p')(s, x) = (\ldots, \langle \ldots e' \mapsto \mu \ldots \rangle)$$

   where some element of μ has time $\leq t_{i+1}$,

   In particular, $x_1 = p$, and x_2 is some edge $e' \in E \cup O$, and $src(e') = p$, and there exists a history h for p and a state s such that the above property holds, namely,

$$g(p)(h) = (s, \ldots)$$

   and an event t" for some t" such that $t \leq t$" or $x = (d, m)$ for some d and m such that $t \leq time(m)$, and

$$g_1(p)(s, x) = (\ldots, \langle \ldots e' \mapsto \mu \ldots \rangle)$$

   where some element of μ has time $\leq t_2$.

   We obtain $(p, t) \rightsquigarrow (e', t_2)$. If $e = e'$, then the distinctness requirement implies $k = 2$ and $t_2 \leq t'$, and we obtain $(e, t_2) \rightsquigarrow (e, t')$. If e and e' are different, the cases above yield $(e, t_2) \rightsquigarrow (e, t')$.

   In all cases, the proof of $(e, t_2) \rightsquigarrow (e, t')$ is shorter than that of $(p, t) \rightsquigarrow (e, t')$.
2. Suppose that (e, t) could-result in (e', t'), where e and e' are different. Then there exist x_i, etc., with $x_2 = dst(e)$, $t_2 = t_1$, and $(p, t_1) \rightsquigarrow (e', t')$, as in (1). The proof of $(p, t_1) \rightsquigarrow (e', t')$ is shorter than that of $(e, t) \rightsquigarrow (e', t')$.

3. If $(x,t) \rightsquigarrow (x,t')$, the distinctness requirement in the definition of could-result-in implies that we are in the first case (that of $x = x'$), where $t \leq t'$.
4. If $(p,t) \rightsquigarrow (e,t')$ and $src(e) = p$, the definition of could-result-in (in particular, the distinctness requirement) imply that we are in the case $k = 2$; in that case, there exist $t_1 \geq t$ and $t_2 \leq t'$ and a history h for p and a state s such that
$$g(p)(h) = (s, \ldots)$$
and an event t" for some t" such that $t_1 \leq t$", and therefore $t \leq t'$, or $x = (d,m)$ for some d and m such that $t_1 \leq time(m)$, and therefore $t \leq time(m)$, and
$$g_1(p)(s,x) = (\ldots, \langle \ldots e \mapsto \mu \ldots \rangle)$$
where some element of μ has time $\leq t_2$, and therefore $\leq t'$.
5. Suppose that $(x,t_1) \rightsquigarrow (i,t_2)$ for $i \in I$. We assume that x is not i, in order to obtain a contradiction. Since x and i are different, by the definition of could-result-in, there exists a sequence of distinct nodes and edges from x to i, where the next-to-last element is a node p with $src(i) = p$. But no such p exists.
6. Suppose that $(o,t_1) \rightsquigarrow (x,t_2)$ for $o \in O$. We assume that x is not o, in order to obtain a contradiction. Since x and o are different, by the definition of could-result-in, there exists a sequence of distinct nodes and edges from o to x, where the second element is a node p with $dst(o) = p$. But no such p exists.
7. We argue by cases on the proof of $(x_1,t_1) \rightsquigarrow (x_2,t_2)$. Both cases are trivial.

□

We assume the following condition:

Condition 4 *For all $p \in P$, $e \in E$ with $dst(e) = p$, and $t, t' \in T$, if $(p,t) \rightsquigarrow (e,t')$ then $t \leq t'$.*

It is equivalent to transitivity:

Theorem 3. *The relation $\rightsquigarrow$ is transitive if and only if Condition 4 holds.*

We omit the proof of this theorem since, for the present purposes, we may as well assume transitivity.

Proposition 9. *$Close_{\uparrow}(S) = \{(x',t') \mid \exists (x,t) \in S.(x,t) \rightsquigarrow (x',t')\}$.*

Proof: By the reflexivity and transitivity (Condition 4 and Theorem 3) of the could-result-in relation. □

Proposition 10. *Suppose $src(e) = p$ and $(p,t_1) \rightsquigarrow (e,t_2)$. Then, for all f, if $t_2 \in \phi(e)(f)$ then $t_1 \in f$.*

Proof: Suppose $src(e) = p$, $(p, t_1) \rightsquigarrow (e, t_2)$, and $t_2 \in \phi(e)(f)$. By Proposition 8(4), there is a history $h = h_1 \cdot x$ for p that ends with an event x at some time $t' \geq t_1$ that results in an output on e at a time $t_3 \leq t_2$. Let μ be $\Pi_e g(p)(h)$, and μ_1 be $\Pi_e g(p)(h_1)$. Because the output is at time $t_3 \leq t_2$, and $t_2 \in \phi(e)(f)$, we have $t_3 \in \phi(e)(f)$, so $\mu @ \phi(e)(f) \neq \mu_1 @ \phi(e)(f)$. By Condition 2, $\mu @ \phi(e)(f) = \Pi_e g(p)(h @ f) @ \phi(e)(f)$ and $\mu_1 @ \phi(e)(f) = \Pi_e g(p)(h_1 @ f) @ \phi(e)(f)$. If t_1 were not in f, then t' would not be in f either, and we have $h @ f = h_1 @ f$, so

$$\Pi_e g(p)(h @ f) @ \phi(e)(f) = \Pi_e g(p)(h_1 @ f) @ \phi(e)(f)$$

and by transitivity we would obtain that $\mu @ \phi(e)(f) = \mu_1 @ \phi(e)(f)$, which is a contradiction. □

Proposition 11. *Let G be any function from P to F. Suppose that $p \in P$, $t \in T$, and and, for all $p_0 \in P$, $e_0, e \in E$ such that $src(e_0) = p_0$ and $dst(e) = p$, and all $t_0 \in T$, if $(e_0, t_0) \rightsquigarrow (e, t)$ then $t_0 \in \phi(e_0)(G(p_0))$. Then for all $p_0 \in P$, $e \in E$ such that $dst(e) = p$, and all $t_0 \in T$, if $(p_0, t_0) \rightsquigarrow (e, t)$ then $t_0 \in G(p_0)$.*

Proof: Suppose that $p_0 \in P$, $e \in E$, $dst(e) = p$, $t_0 \in T$, and $(p_0, t_0) \rightsquigarrow (e, t)$. Proposition 8(1) says that there exists (e', t') such that $src(e') = p_0$, $(p_0, t_0) \rightsquigarrow (e', t')$ and $(e', t') \rightsquigarrow (e, t)$. Obviously it cannot be the case that $e' \in I$, since $src(e') = p_0$. In case $e' \in E$, we obtain $t' \in \phi(e')(G(p_0))$, by the hypothesis, and then that $t_0 \in G(p_0)$, by Proposition 10. In case $e' \in O$, Proposition 8(6) yields that $e \in O$, contradicting that $dst(e) = p$. □

Properties of filtering and reordering

We establish a few simple properties of operations such as filtering and reordering. Throughout, f, f_1, f_2, f_3 range over frontiers; u, v, w range over message sequences; h ranges over histories. Some of this basic material is also needed for our work on security, and included in the corresponding paper [3].

Proposition 12. *If $f_1 = f_2 \cap f_3$ then $h @ f_1 = h @ f_2 @ f_3$.*

Proof: By a trivial induction. □

Proposition 13. $u \hookrightarrow u @ f \cdot u @\!\!\!\backslash f$.

Proof: Suppose that m_1 is a message in $u @ f$ and m_2 is a message in $u @\!\!\!\backslash f$. Because f is downward closed, it cannot be the case that $time(m_2) \leq time(m_1)$. So each such m_1 in u (starting with the leftmost one) can be moved left past each such m_2. □

Proposition 14. *If $u \hookrightarrow v$ then $u @ f \hookrightarrow v @ f$.*

Proof: The proof is by induction on the derivation of $u \hookrightarrow v$. The cases of reflexivity and transitivity are trivial. The base case $u \cdot m_1 \cdot m_2 \cdot v \hookrightarrow u \cdot m_2 \cdot m_1 \cdot v$ breaks down into subcases depending on whether m_1 and m_2 are in f, but each is trivial too, using that $(u_1 \cdot u_2) @ f = u_1 @ f \cdot u_2 @ f$. □

Proposition 15. *If $u \hookrightarrow v$ then $(u - w) \hookrightarrow (v - w)$.*

Proof: By induction on the derivation of $u \hookrightarrow v$. If $u = v$ this is obvious. If $u \hookrightarrow v$ follows by transitivity from $u \hookrightarrow u_1 \hookrightarrow v$ then $(u-w) \hookrightarrow (u_1-w) \hookrightarrow (v-w)$, by induction hypothesis. If $u{\cdot}m_1{\cdot}m_2{\cdot}v \hookrightarrow u{\cdot}m_2{\cdot}m_1{\cdot}v$ with $time(m_1) \not\leq time(m_2)$, then $(u{\cdot}m_1{\cdot}m_2{\cdot}v - w) \hookrightarrow (u{\cdot}m_2{\cdot}m_1{\cdot}v - w)$, possibly by reflexivity if subtracting w removes m_1 or m_2. □

Proposition 16. *If $u \hookrightarrow v{\cdot}w$ then $(u - v@f) \hookrightarrow v@\!\!\!\setminus f{\cdot}w$.*

Proof: By Proposition 15, $(u - v@f) \hookrightarrow (v{\cdot}w - v@f) = v@\!\!\!\setminus f{\cdot}w$. □

Proposition 17. *If m is in $(u - v@f)$ and $u \hookrightarrow v{\cdot}w$ and $time(m) \in f$, then m is in w.*

Proof: By Proposition 16, $u \hookrightarrow v{\cdot}w$ implies $(u - v@f) \hookrightarrow v@\!\!\!\setminus f{\cdot}w$. Since m is in $(u - v@f)$, m is in $v@\!\!\!\setminus f{\cdot}w$. Since $time(m) \in f$, we obtain that m is in w. □

Proposition 18. *If $u@f = v@f$, m is in $u - w$, and $time(m) \in f$, then m is in $v - w$.*

Proof: If m is in $u - w$ and $time(m) \in f$, then m is in $u@f - w$. If in addition $u@f = v@f$, we obtain that m is in $v@f - w$. Therefore, m is in $v - w$, because, for all v', if m is in $v' - w$ and v' is a subsequence of v then m is in $v - w$. □

Proposition 19. *If $u \hookrightarrow u'$, $u' = v'{\cdot}m{\cdot}w'$, and $time(n) \not\leq time(m)$ for all n in v', then there exist v and w such that $u = v{\cdot}m{\cdot}w$, and $time(n) \not\leq time(m)$ for all n in v.*

Proof: The proof is by induction on the derivation of $u \hookrightarrow u'$. The cases of reflexivity and transitivity are trivial. The base case

$$\ldots{\cdot}m_1{\cdot}m_2{\cdot}\ldots \hookrightarrow \ldots{\cdot}m_2{\cdot}m_1{\cdot}\ldots$$

is also trivial when (the leftmost occurrence of) m is not m_1 or m_2. Otherwise, if $m = m_1$, the desired conclusion follows from the hypothesis that $time(n) \not\leq time(m)$ for all n in v'; if $m = m_2$, the desired conclusion follows from that hypothesis plus $time(m_1) \not\leq time(m_2)$, which is required for the reordering. □

Proposition 20. *If $u@f = u'@f$, $u' = v'{\cdot}m{\cdot}w'$, $time(m) \in f$, and $time(n) \not\leq time(m)$ for all n in v', then there exist v and w such that $u = v{\cdot}m{\cdot}w$, and $time(n) \not\leq time(m)$ for all n in v.*

Proof: Since $time(m) \in f$, $u@f = u'@f$, and m occurs in u', we have that m occurs in u as well. Let v be the prefix of u to the left of the leftmost occurrence of m. Let v'' be the prefix of u' to the left of the leftmost occurrence of m; it is a prefix of v'.

Since $time(m) \in f$ and $u@f = u'@f$, v and v'' may differ only by elements with times not in f. None of those elements can have a time $\leq time(m)$, since $time(m) \in f$ and f is a frontier. Therefore, since $time(n) \not\leq time(m)$ for all n in v', we obtain $time(n) \not\leq time(m)$ for all n in v. □

Proposition 21. *If $u@f = u'@f$, then $(u - v@f)@f = (u' - v@f)@f$.*

Proof: $u@f = u'@f$ implies $(u@f - v@f) = (u'@f - v@f)$, and $(u - v@f)@f = (u@f - v@f)$ and $(u' - v@f)@f = (u'@f - v@f)$ by the distributivity of filtering over subtraction and the idempotence of filtering. □

Proofs on prophecy variables

Lemma 1. *SpecP is obtained from SpecL almost by adding a very simple prophecy variable.*

Proof: We need to show that D satisfies conditions P1, P2', P4', and P6'.

P1 holds because the enriched state space is included in the original state space times the domain of prophecy variables.

P2' is $InitPropP = InitPropL$.

P4′ concerns backward steps. We wish to show that if $s \to s'$ is a state transition of *SpecL* and (s', D') is an enriched state then there exists an enriched state (s, D) such that $(s, D) \to (s', D')$ in *SpecP*.

- For transitions where *LocState*, *NotRequests*, H, and Q are unchanged, we let $D = D'$.
- For *MessL*, *NotL*, *InpL*, and *OutpL* transitions, we let $D = D'$, as suggested by the definitions of *MessP*, *NotP*, *InpP*, and *OutpP*.
- For *RollbackL*, we let $(D = D' \cap f)$, as suggested by the definition of *RollbackP*.

As defined, $D(p)$ is a frontier for all p. In the case of *RollbackL*, this property follows from the fact that downward closure is closed by intersection.

The rest of the proof of P4′ is by cases on the type of the transition $s \to s'$.

- When the state transition leaves *LocState*, *NotRequests*, H, and Q unchanged, if (s', D') is an enriched state, then so is (s, D), trivially.
- When the state transition $s \to s'$ satisfies *MessL* or *NotL*, we argue that if (s', D') is an enriched state then so is (s, D) as follows:
  - $\forall p \in P, \forall i \in I$ *such that* $dst(i) = p.\{time(m) \mid (i, m) \in H(p)\} \subseteq D(p)$:
    For $p \in P$ and $i \in I$ with $dst(i) = p$, *MessL* and *NotL* imply that $H(p) = H'(p)$ or $H'(p)$ extends $H(p)$, so $\{time(m) \mid (i, m) \in H(p)\}$ is a subset of $\{time(m) \mid (i, m) \in H'(p)\}$. We have assumed that $\{time(m) \mid (i, m) \in H'(p)\}$ is included in $D'(p)$. Since $D = D'$ in these cases, we obtain that $\{time(m) \mid (i, m) \in H(p)\}$ is included in $D(p)$.
  - $\forall p \in P, \forall o \in O$ *such that* $src(o) = p.\{time(m) \mid \exists e.(e, m) \in H(p)\} \subseteq D(p)$:
    Similarly, for $p \in P$ and $o \in O$ with $src(o) = p$, *MessL* and *NotL* imply that $H(p) = H'(p)$ or $H'(p)$ extends $H(p)$, so $\{time(m) \mid \exists e.(e, m) \in H(p)\}$ is a subset of $\{time(m) \mid \exists e.(e, m) \in H'(p)\}$. We have assumed that $\{time(m) \mid \exists e.(e, m) \in H'(p)\}$ is included in $D'(p)$. Since $D = D'$ in these cases, we obtain that $\{time(m) \mid \exists e.(e, m) \in H'(p)\}$ is included in $D(p)$.

- $\forall p, q \in P, \forall e \in E$ *such that* $src(e) = p, dst(e) = q, \forall m \in M$ *such that* $m \in Q(e) \vee (e, m) \in H(q)$.*if* $time(m) \in D(q)$ *then* $time(m) \in \phi(e)(D(p))$:
  For any $p, q \in P$, and $e \in E$ such that $src(e) = p$, and $dst(e) = q$, *MessL* and *NotL* only leave unchanged or extend $\{m \mid m \in Q(e) \vee (e, m) \in H(q)\}$. So any m such that $m \in Q(e)$ or $(e, m) \in H(q)$ is also such that $m \in Q'(e)$ or $(e, m) \in H'(q)$, and if $time(m) \in D'(q)$ then $time(m) \in \phi(e)(D'(p))$ implies if $time(m) \in D(q)$ then $time(m) \in \phi(e)(D(p))$.
- $\forall p, q \in P, \forall e_1, e_2 \in E$ *such that* $src(e_1) = p, dst(e_2) = q, \forall t_1 \in T, t_2 \in H(q)@D(q)$. *if* $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ *then* $t_1 \in \phi(e_1)(D(p))$:
  For each q, *MessL* and *NotL* only leave unchanged or extend $\{t \mid t \in H(q)\}$, while $D' = D$. So any t_2 such that $t_2 \in H(q)@D(q)$ is also such that $t_2 \in H'(q)@D'(q)$, and if $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(D'(p))$ implies if $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(D(p))$.

– When the state transition $s \rightarrow s'$ satisfies *InpL* or *OutpL*, if (s', D') is an enriched state, then so is (s, D), because being an enriched state space does not depend on $Q(d)$ for $d \in I \cup O$, and because *InpL* and *OutpL* both imply that all other state components are unchanged.
– When the state transition $s \rightarrow s'$ satisfies *RollbackL*, we argue that if (s', D') is an enriched state then so is (s, D) as follows:
  - $\forall p \in P, \forall i \in I$ *such that* $dst(i) = p.\{time(m) \mid (i, m) \in H(p)\} \subseteq D(p)$:
    For $p \in P$ and $i \in I$ with $dst(i) = p$, the definition of *RollbackL* implies for all $(i, m) \in H(p)$, $time(m) \in f(p)$. We have assumed that for all $(i, m) \in H'(p)$, $time(m) \in D'(p)$. Since *RollbackL* implies that $H'(p) = H(p)@f(p)$, we obtain that for all $(i, m) \in H(p)$, $time(m) \in D'(p)$. The definition of D as the intersection of f and D' then yields that for all $(i, m) \in H(p)$, $time(m) \in D(p)$.
  - $\forall p \in P, \forall o \in O$ *such that* $src(o) = p.\{time(m) \mid \exists e.(e, m) \in H(p)\} \subseteq D(p)$:
    Similarly, for $p \in P$ and $o \in O$ with $src(o) = p$, the definition of *RollbackL* implies that for all $(e, m) \in H(p)$, $time(m) \in f(p)$. We have assumed that for all $(e, m) \in H'(p)$, $time(m) \in D'(p)$. Since *RollbackL* implies that $H'(p) = H(p)@f(p)$, we obtain that for all $(e, m) \in H(p)$, $time(m) \in D'(p)$. The definition of D as the intersection of f and D' then yields that for all $(e, m) \in H(p)$, $time(m) \in D(p)$.
  - $\forall p, q \in P, \forall e \in E$ *such that* $src(e) = p, dst(e) = q, \forall m \in M$ *such that* $m \in Q(e) \vee (e, m) \in H(q)$.*if* $time(m) \in D(q)$ *then* $time(m) \in \phi(e)(D(p))$:
    Consider any $p, q \in P$, and $e \in E$ such that $src(e) = p$, and $dst(e) = q$, and a message m, with $m \in Q(e)$ or $(e, m) \in H(q)$. Suppose that $time(m) \in D(q)$. Therefore, $time(m) \in f(q)$ and $D'(q)$.
    In case $m \in Q(e)$, we have obtained a contradiction with the precondition of *RollbackL* for $e \in E$ that implies that $f(q)$ does not intersect $\{t \mid \exists m \in M.m \in Q(e) \wedge time(m) = t\}$.
    So we proceed only with the case of $(e, m) \in H(q)$. Since $time(m) \in f(q)$, and $H'(q) = H(q)@f(q)$, we obtain $(e, m) \in H'(q)$. From $time(m) \in D'(q)$ we then obtain $time(m) \in \phi(e)(D'(p))$. To conclude that $time(m) \in \phi(e)(D(p))$, we use that $D(p)$ is the intersection of $D'(p)$ with $f(p)$,

that by Condition 3(1) $\phi(e)(D(p))$ includes $\phi(e)(D'(p)) \cap \phi(e)(f(p))$, and that *RollbackL* implies that if $(e, m) \in H(q)$ and $time(m) \in f(q)$ then $time(m) \in \phi(e)(f(p))$.

- $\forall p, q \in P, \forall e_1, e_2 \in E$ *such that* $src(e_1) = p, dst(e_2) = q, \forall t_1 \in T, t_2 \in H(q)@D(q)$. *if* $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ *then* $t_1 \in \phi(e_1)(D(p))$:
  In the case of *RollbackL*, we have assumed that, for all $p, q \in P$, for all $e_1, e_2 \in E$ such that $src(e_1) = p$ and $dst(e_2) = q$, $t_1 \in T$, $t_2 \in H'(q)@D'(q)$, we have that if $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then t_1 in $\phi(e_1)(D'(p))$. The definitions of D and H' in *RollbackP* and Proposition 12 imply $H'(q)@D'(q) = H(q)@f(q)@D'(q) = H(q)@D(q)$, so if $t_2 \in H(q)@D(q)$ and $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(D'(p))$.
  RollbackL also implies that if $t_2 \in H(q)@f(q)$ and $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(f(p))$. Since $D \subseteq f$, it follows that if $t_2 \in H(q)@D(q)$ and $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(f(p))$.
  Putting these facts together, we have that if $t_2 \in H(q)@D(q)$ and $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ then $t_1 \in \phi(e_1)(D'(p)) \cap \phi(e_1)(f(p))$, and hence, by Condition 3(1), $t_1 \in \phi(e_1)(D'(p) \cap f(p))$, that is, $t_1 \in \phi(e_1)(D(p))$, as desired.

P6' says that for every $(LocState, NotRequests, H, Q)$ there exists D such that $(LocState, NotRequests, H, Q, D)$ in the enriched state space. We can simply take $D(p) = T$ for all p. The correctness of this choice requires Condition 3(2). □

Proposition 22. *The set* $\{(LocState, NotRequests, H, Q, D) \in \Sigma^P_{\text{Low}}\}/\mathcal{Q}$ *is finite for each* $(LocState, NotRequests, H, Q)$.

Proof: For each $(LocState, NotRequests, H, Q)$, there is at most one equivalence class of $(LocState, NotRequests, H, Q, D)$ for each function in

$$P \rightarrow (ITimes(H) \rightarrow 2)$$

Since $H(p)$ is finite for each p, and P is finite as well, the set $ITimes(H)$ is finite. □

Lemma 2. *SpecP*$_{/\mathcal{Q}}$ *is obtained from SpecL by adding a very simple prophecy variable.*

Proof: *SpecP* is obtained from *SpecL* almost by adding a very simple prophecy variable, by Lemma 1. In addition,

$$\begin{array}{c}(LocState_1, NotRequests_1, H_1, Q_1, D_1)\\ \mathcal{Q}\\ (LocState_2, NotRequests_2, H_2, Q_2, D_2)\end{array}$$

trivially implies

$$(LocState_1, NotRequests_1, H_1, Q_1) = (LocState_2, NotRequests_2, H_2, Q_2)$$

Finally, by Proposition 22, for all $(\mathit{LocState}_1, \mathit{NotRequests}_1, H_1, Q_1)$, the set

$$\{(\mathit{LocState}_1, \mathit{NotRequests}_1, H_1, Q_1, D) \in \Sigma^P_{\mathrm{Low}}\}/\mathcal{Q}$$

is finite. So we can apply Proposition 4, and obtain that $\mathit{SpecP}_{/\mathcal{Q}}$ is obtained by adding a very simple prophecy variable. □

Proposition 23. *SpecL implements* $\mathit{SpecP}_{/\mathcal{Q}}$.

Proof: This follows immediately from Lemma 2 and the soundness of very simple prophecy variables (Proposition 2). □

Proofs of main invariants and refinement

This section contains proofs of our main invariants and of the refinement theorem. In some cases, analogous but not identical arguments are needed for work on security [3]. Importantly, however, that work does not require a prophecy variable, nor does it cover notifications, the clock, input and output edges, and related concepts.

We introduce names for *Inv*'s conjuncts:

- Let Inv_1 be
$$\forall p. \mathit{\Pi}_{\mathrm{Loc}} g(p)(H(p)) = \mathit{LocState}(p)$$
- Let Inv_2 be
$$\forall p. \mathit{\Pi}_{\mathrm{NR}} g(p)(H(p)) = \mathit{NotRequests}(p)$$
- Let Inv_3 be:
$$\forall p, q \in P, e \in E \text{ such that } \mathit{src}(e) = p \wedge \mathit{dst}(e) = q.$$
$$\mathit{\Pi}_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e, m) \in H(q)\rangle \cdot Q(e))$$

Lemma 3. *Inv is an inductive invariant of SpecL and SpecP.*

Proof: We do the proof for *SpecL*, but the same argument applies to *SpecP*.

InitPropL implies that initially $H(p)$ is a sequence of the form

$$\langle\!\langle (\mathit{LocState}(p), \mathit{NotRequests}(p)) \rangle\!\rangle$$

for all $p \in P$, and that $Q(e)$ is empty for all $e \in E$. The definition of g then yields the desired properties.

For showing that *Inv* is preserved by steps, we treat the three conjuncts separately, using the first conjunct in the arguments for the subsequent ones.

1. For showing that Inv_1 is preserved by steps, suppose that
$$\mathit{\Pi}_{\mathrm{Loc}} g(p)(H(p)) = \mathit{LocState}(p)$$
for all p, in order to show that
$$\mathit{\Pi}_{\mathrm{Loc}} g(p)(H'(p)) = \mathit{LocState}'(p)$$
for a particular p. We consider *MessL*, *NotL*, *InpL*, *OutpL*, and *RollbackL* transitions.

- *MessL* at p implies $H'(p) = H(p)\cdot(e,m)$; and both $\Pi_{\mathrm{Loc}}g(p)(H'(p))$ and $\mathit{LocState}'(p)$ are obtained from $g(p)(H(p))$ by applying $g_1(p)$ to $\mathit{LocState}(p)$ and (e,m), then taking the state component of the result.
  MessL at other nodes q leaves $H(p)$ and $\mathit{LocState}(p)$ unchanged.
- Similarly, *NotL* at p implies $H'(p) = H(p)\cdot t$; and both $\Pi_{\mathrm{Loc}}g(p)(H'(p))$ and $\mathit{LocState}'(p)$ are obtained from $g(p)(H(p))$ by applying $g_1(p)$ to $\mathit{LocState}(p)$ and t, then taking the state component of the result.
  NotL at other nodes q leaves $H(p)$ and $\mathit{LocState}(p)$ unchanged.
- *InpL* and *OutpL* both imply $\mathit{LocState}'(p) = \mathit{LocState}(p)$ and $H'(p) = H(p)$ for all p, so these cases are trivial.
- In *RollbackL*, $\mathit{LocState}'(p)$ is obtained by applying $g(p)$ to $H(p)@f(p)$ which equals $H'(p)$, for all p.

2. For showing that Inv_2 is preserved by steps, suppose that

$$\Pi_{\mathrm{NR}}g(p)(H(p)) = \mathit{NotRequests}(p)$$

for all p, in order to show that

$$\Pi_{\mathrm{NR}}g(p)(H'(p)) = \mathit{NotRequests}'(p)$$

for a particular p. We consider *MessL*, *NotL*, *InpL*, *OutpL*, and *RollbackL* transitions.

- *MessL* at p implies $H'(p) = H(p)\cdot(e,m)$. We have:

$$\mathit{NotRequests}'(p) = \mathit{NotRequests}(p) \cup (\Pi_{\mathrm{NR}}g_1(p)(\mathit{LocState}(p),(e,m)))$$

  We also have:

$$\begin{array}{l}(\Pi_{\mathrm{NR}}g(p)(H'(p)))\\ = (\Pi_{\mathrm{NR}}g(p)(H(p))) \cup (\Pi_{\mathrm{NR}}g_1(p)((\Pi_{\mathrm{Loc}}g(p)(H(p))),(e,m)))\\ = \mathit{NotRequests}(p) \cup (\Pi_{\mathrm{NR}}g_1(p)(\mathit{LocState}(p),(e,m))\end{array}$$

  by induction hypothesis and Inv_1. So they are equal.
  MessL at other nodes q leaves $H(p)$ and $\mathit{NotRequests}(p)$ unchanged.
- *NotL* at p implies $H'(p) = H(p)\cdot t$. We have:

$$\mathit{NotRequests}'(p) = \mathit{NotRequests}(p) - \{t\} \cup (\Pi_{\mathrm{NR}}g_1(p)(\mathit{LocState}(p),(e,m)))$$

  We also have:

$$\begin{array}{l}(\Pi_{\mathrm{NR}}g(p)(H'(p)))\\ = (\Pi_{\mathrm{NR}}g(p)(H(p))) - \{t\} \cup (\Pi_{\mathrm{NR}}g_1(p)((\Pi_{\mathrm{Loc}}g(p)(H(p))),(e,m)))\\ = \mathit{NotRequests}(p) - \{t\} \cup (\Pi_{\mathrm{NR}}g_1(p)(\mathit{LocState}(p),(e,m)))\end{array}$$

  by induction hypothesis and Inv_1. So they are equal.
  NotL at other nodes q leaves $H(p)$ and $\mathit{NotRequests}(p)$ unchanged.
- *InpL* and *OutpL* both imply $\mathit{NotRequests}'(p) = \mathit{NotRequests}(p)$ and $H'(p) = H(p)$ for all p, so these cases are trivial.

– Finally, in *RollbackL*,

$$\mathit{NotRequests}'(p) = \Pi_{\mathrm{NR}} g(p)(H(p)@f(p))$$

which is the same as $\Pi_{\mathrm{NR}} g(p)(H'(p))$ since $H(p)@f(p) = H'(p)$, for all p.

3. For showing that Inv_3 is preserved by steps, suppose that, for all p, q, and e with $\mathit{src}(e) = p$ and $\mathit{dst}(e) = q$, we have

$$\Pi_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e,m) \in H(q)\rangle \cdot Q(e))$$

in order to show that

$$\Pi_e g(p)(H'(p)) \hookrightarrow (\langle m \mid (e,m) \in H'(q)\rangle \cdot Q'(e))$$

for particular p, q, and e. We consider *MessL*, *NotL*, *InpL*, *OutpL*, and *RollbackL* transitions.

– *MessL* at p implies $H'(p) = H(p) \cdot (e_0, m_0)$. Let

$$\mu = \Pi_e g_1(p)(\mathit{LocState}(p), (e_0, m_0))$$

Then $Q'(e) = Q(e) \cdot \mu$. Moreover $H'(q) = H(q)$ (by the assumption that edge source and destination are always different), so

$$\langle m \mid (e,m) \in H'(q)\rangle = \langle m \mid (e,m) \in H(q)\rangle$$

By Inv_1, $\mathit{LocState}(p) = \Pi_{\mathrm{Loc}} g(p)(H(p))$, so

$$\Pi_e g(p)(H'(p)) = (\Pi_e g(p)(H(p))) \cdot \mu$$

Moreover,

$$\Pi_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e,m) \in H(q)\rangle \cdot Q(e))$$

by induction hypothesis, so

$$(\Pi_e g(p)(H(p))) \cdot \mu \hookrightarrow (\langle m \mid (e,m) \in H(q)\rangle \cdot Q(e)) \cdot \mu$$

$$\Pi_e g(p)(H'(p)) \hookrightarrow (\langle m \mid (e,m) \in H(q)\rangle \cdot Q(e) \cdot \mu)$$

and finally

$$\Pi_e g(p)(H'(p)) \hookrightarrow (\langle m \mid (e,m) \in H'(q)\rangle \cdot Q'(e))$$

MessL at q implies $H'(q) = H(q) \cdot (e_1, m_1)$ and $H'(p) = H(p)$. If $e_1 \neq e$, then $Q'(e) = Q(e)$ and $\langle m \mid (e,m) \in H'(q)\rangle = \langle m \mid (e,m) \in H(q)\rangle$, so we are done by induction hypothesis. If $e_1 = e$, then there exist $u, v \in M^*$ such that $Q(e) = u \cdot m_1 \cdot v$, $Q'(e) = u \cdot v$, $\mathit{time}(n) \not\leq \mathit{time}(m_1)$

for all $n \in u$, and $H'(q) = H(q) \cdot (e, m_1)$. Since $H'(p) = H(p)$, we have $g(p)(H'(p)) = g(p)(H(p))$. So,

$$\begin{array}{l} \Pi_e g(p)(H'(p)) = \Pi_e g(p)(H(p)) \\ \hookrightarrow (\langle m \mid (e,m) \in H(q) \rangle \cdot Q(e)) \text{ by induction hypothesis} \\ = (\langle m \mid (e,m) \in H(q) \rangle \cdot u \cdot m_1 \cdot v) \\ \hookrightarrow (\langle m \mid (e,m) \in H(q) \rangle \cdot m_1 \cdot u \cdot v) \\ = (\langle m \mid (e,m) \in H(q) \rangle \cdot m_1 \cdot Q'(e)) \\ = (\langle m \mid (e,m) \in H'(q) \rangle \cdot Q'(e)) \end{array}$$

MessL elsewhere (not at p or q) does not affect $H(p)$, $H(q)$, or $Q(e)$, so the induction hypothesis immediately implies the desired conclusion.

- *NotL* at p does not affect $\langle m \mid (e,m) \in H(q) \rangle$, and extends $Q(e)$ and $\Pi_e g(p)(H(p))$ in the same way, since Inv_1 says that $\Pi_{\mathrm{Loc}} g(p)(H(p)) = \mathit{LocState}(p)$.
  NotL at q does not affect $H(p)$ or $Q(e)$, nor $\langle m \mid (e,m) \in H(q) \rangle$ (since it adds only a notification to $H(q)$), so the induction hypothesis immediately implies the desired conclusion.
  NotL elsewhere (not at p or q) does not affect $H(p)$, $H(q)$, or $Q(e)$, so the induction hypothesis immediately implies the desired conclusion.
- *InpL* and *OutpL* both imply $Q'(e) = Q(e)$ for all $e \in E$ and $H'(p) = H(p)$ for all p, so these cases are trivial.
- Finally, in *RollbackL*, with a frontier f, $H'(p) = H(p)@f(p)$ and $H'(q) = H(q)@f(q)$, so

$$\Pi_e g(p)(H'(p)) = \Pi_e g(p)(H(p)@f(p))$$

and

$$\langle m \mid (e,m) \in H'(q) \rangle = \langle m \mid (e,m) \in H(q)@f(q) \rangle$$

In addition,

$$Q'(e) = \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))$$

The induction hypothesis says that

$$\Pi_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e,m) \in H(q) \rangle \cdot Q(e))$$

It follows that

$$\begin{array}{l} \Pi_e g(p)(H(p))@(\phi(e)(f(p)) \cap f(q)) \\ \hookrightarrow (\langle m \mid (e,m) \in H(q) \rangle \cdot Q(e))@(\phi(e)(f(p)) \cap f(q)) \end{array}$$

by Proposition 14. Since, for $e \in E$, the definition of *RollbackL* implies that if $m \in Q(e)$ and $\mathit{time}(m) = t$ then $t \notin f(q)$, we obtain

$$\begin{array}{l} \Pi_e g(p)(H(p))@(\phi(e)(f(p)) \cap f(q)) \\ \hookrightarrow \langle m \mid (e,m) \in H(q) \rangle @(\phi(e)(f(p)) \cap f(q)) \end{array}$$

Since

$$\Pi_e g(p)(H(p))@\phi(e)(f(p)) = \Pi_e g(p)(H(p)@f(p))@\phi(e)(f(p))$$

by Condition 2, we obtain

$$\begin{array}{l}\Pi_e g(p)(H(p))@(\phi(e)(f(p)) \cap f(q)) \\ = \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))\end{array}$$

It follows that

$$\begin{array}{l}\Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q)) \\ \hookrightarrow \langle m \mid (e,m) \in H(q)\rangle@(\phi(e)(f(p)) \cap f(q))\end{array}$$

and therefore

$$\begin{array}{l}\begin{pmatrix}\Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q)) \\ \quad . \\ \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))\end{pmatrix} \\ \quad \hookrightarrow \\ \begin{pmatrix}\langle m \mid (e,m) \in H(q)\rangle@(\phi(e)(f(p)) \cap f(q)) \\ \quad . \\ \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))\end{pmatrix}\end{array}$$

and thus, by Proposition 13,

$$\begin{array}{l}\Pi_e g(p)(H(p)@f(p)) \\ \quad \hookrightarrow \\ \begin{pmatrix}\langle m \mid (e,m) \in H(q)\rangle@(\phi(e)(f(p)) \cap f(q)) \\ \quad . \\ \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))\end{pmatrix}\end{array}$$

Since by the definition of *RollbackL*,

$$\{time(m) \mid (e,m) \in H(q)\} \cap f(q) \subseteq \phi(e)(f(p))$$

we have

$$\langle m \mid (e,m) \in H(q)\rangle@f(q) = \langle m \mid (e,m) \in H(q)\rangle@(\phi(e)(f(p)) \cap f(q))$$

so

$$\begin{array}{l}\Pi_e g(p)(H(p)@f(p)) \\ \hookrightarrow \langle m \mid (e,m) \in H(q)\rangle@f(q) \cdot \Pi_e g(p)(H(p)@f(p))@(\phi(e)(f(p)) \cap f(q))\end{array}$$

that is,

$$\Pi_e g(p)(H'(p)) \hookrightarrow \langle m \mid (e,m) \in H'(q)\rangle \cdot Q'(e)$$

as desired.

□

Proposition 24. *Let* $\mu = \Pi_e g(p)(H(p)@D(p))$ *and* $\nu = \langle m \mid m \in M, (e,m) \in H(q)\rangle$. *Then Inv implies that* $\mu \cdot u - \nu@D(q) = (\mu - \nu@D(q))\cdot u$, *for all* u.

Proof: *Inv* implies that

$$\Pi_e g(p)(H(p)) \hookrightarrow (\langle m \mid (e,m) \in H(q)\rangle \cdot Q(e))$$

that is,

$$\Pi_e g(p)(H(p)) \hookrightarrow \nu \cdot Q(e)$$

So Condition 2 and Proposition 14 imply that

$$\Pi_e g(p)(H(p)@D(p))@\phi(e)(D(p)) \hookrightarrow \nu@\phi(e)(D(p))\cdot Q(e)@\phi(e)(D(p))$$

that is,

$$\mu@\phi(e)(D(p)) \hookrightarrow \nu@\phi(e)(D(p))\cdot Q(e)@\phi(e)(D(p))$$

So $\mu@\phi(e)(D(p))$ and a fortiori μ include every element of $\nu@\phi(e)(D(p))$, and with at least the same multiplicity. By the construction of the enriched state space (which implies that for $(e,m) \in H(q)$, if $time(m) \in D(q)$ then $time(m) \in \phi(e)(D(p))$, in Construction 1(3)), we have that $\nu@D(q)$ is a subsequence of $\nu@\phi(e)(D(p))$. So μ includes every element of $\nu@D(q)$, and with at least the same multiplicity. It follows that $\mu\cdot u - \nu@D(q) = (\mu - \nu@D(q))\cdot u$, for all u. □

The definitions of *HQ* and *HNotRequests* induce a definition for a state function *HClock*:

$$HClock = Close_{\uparrow}\left(\begin{array}{c}\{(e, time(m)) \mid e \in I \cup E \cup O, m \in HQ(e)\} \\ \cup \\ \{(p,t) \mid p \in P, t \in HNotRequests(p)\}\end{array}\right)$$

Propositions 25 and 26 relate *HClock* to *Clock*.

Proposition 25. *For all* $t \in T$, $i \in I$, *if* $(i,t) \in HClock$ *then* $(i,t) \in Clock$.

Proof: Suppose that $i \in I$ and $(i,t) \in HClock$. By Proposition 9 and the definition of *HClock*, that means that there exists (x_0, t_0) such that $(x_0,t_0) \rightsquigarrow (i,t)$ and either

1. x_0 is some $d \in I \cup E \cup O$, and for some $m \in M$, $m \in HQ(d)$ and $time(m) = t_0$, or
2. x_0 is some $q \in P$, and $t_0 \in HNotRequests(q)$.

We have $x_0 = i$ by Proposition 8(5), which yields that for some $m \in M$, $m \in HQ(i)$ and $time(m) = t_0$. The definition of $HQ(i)$ as $Q(i)$ then yields $m \in Q(i)$. The definition of *Clock* finally yields $(i,t) \in Clock$. □

Let *ClockCorrespondence* be:

$$\begin{array}{l}\forall p \in P, t \in D(p). \\ \textit{if } \left(\begin{array}{l}\forall p_0 \in P, t_0 \in T, e_0, e \in E \textit{ such that } src(e_0) = p_0 \wedge dst(e) = p. \\ \textit{if } (e_0,t_0) \rightsquigarrow (e,t) \textit{ then } t_0 \in \phi(e_0)(D(p_0))\end{array}\right) \\ \textit{then } \forall e \in E \textit{ such that } dst(e) = p. \textit{ if } (e,t) \in HClock \textit{ then } (e,t) \in Clock\end{array}$$

Proposition 26. *Inv_2 and Inv_3 imply ClockCorrespondence.*

Proof: Suppose that p and t are such that $t \in D(p)$ and, for all $p_0 \in P$, $t_0 \in T$, $e_0, e \in E$ such that $src(e_0) = p_0$ and $dst(e) = p$, if $(e_0, t_0) \rightsquigarrow (e, t)$ then $t_0 \in \phi(e_0)(D(p_0))$.

Suppose that $dst(e) = p$ and $(e, t) \in HClock$. By Proposition 9 and the definition of $HClock$, that means that there exists (x_0, t_0) such that $(x_0, t_0) \rightsquigarrow (e, t)$ and either

1. x_0 is some $d \in I \cup E \cup O$, and for some $m \in M$, $m \in HQ(d)$ and $time(m) = t_0$, or
2. x_0 is some $q \in P$, and $t_0 \in HNotRequests(q)$.

In these two cases, for establishing that $(e, t) \in Clock$, it suffices to show, respectively, that:

1. $m \in Q(d)$, or
2. $t_0 \in NotRequests(q)$.

We prove these two memberships as follows:

1. We have that $m \in HQ(d)$ with $time(m) = t_0$.
   In case $d \in I \cup O$, the definition of HQ says that $HQ(d) = Q(d)$, so $m \in Q(d)$.
   Otherwise ($d \in E$), the definition of HQ says that m is in

   $$\mu - \nu @ D(q_2)$$

   where $q_1 = src(d)$, $q_2 = dst(d)$, $\mu = \Pi_d g(q_1)(H(q_1) @ D(q_1))$, and $\nu = \langle m \mid m \in M, (d, m) \in H(q_2) \rangle$.
   Since $(d, t_0) \rightsquigarrow (e, t)$ and $src(d) = q_1$, the hypothesis yields $t_0 \in \phi(d)(D(q_1))$.
   By Condition 2, $\mu @ \phi(d)(D(q_1)) = \Pi_d g(q_1)(H(q_1)) @ \phi(d)(D(q_1))$. So m is in

   $$\Pi_d g(q_1)(H(q_1)) - \nu @ D(q_2)$$

   by Proposition 18.
   We argue that $t_0 \in D(q_2)$ as well, by cases.
   - If $d = e$, then $q_2 = p$, the fact that $(d, t_0) \rightsquigarrow (e, t)$ implies that $t_0 \leq t$ by Proposition 8(3), so $t \in D(p)$ implies $t_0 \in D(p)$, that is, $t_0 \in D(q_2)$.
   - Otherwise ($d \neq e$), $(q_2, t_0) \rightsquigarrow (e, t)$ by Proposition 8(2), so the hypothesis and Proposition 11 yield that $t_0 \in D(q_2)$.

   By Inv_3, $\Pi_d g(q_1)(H(q_1)) \hookrightarrow \nu \cdot Q(d)$. By Proposition 17, we obtain $m \in Q(d)$, as desired. Recall that Proposition 17 says that if m is in $u - v @ f$ and $u \hookrightarrow v \cdot w$ and $time(m) \in f$, then m is in w. We apply it with

   $$\begin{aligned} u &= \Pi_d g(q_1)(H(q_1)) \\ v &= \nu \\ w &= Q(d) \\ f &= D(q_2) \end{aligned}$$

2. We have that $t_0 \in \mathit{HNotRequests}(q)$, which is $\Pi_{\mathrm{NR}} g(p)(H(q)@D(q))$, by definition of *HNotRequests*. Since $(q, t_0) \rightsquigarrow (e, t)$, the hypothesis and Proposition 11 yield that $t_0 \in D(q)$. Condition 2 then yields $t_0 \in \Pi_{\mathrm{NR}} g(p)(H(q))$, that is (by Inv_2 in Lemma 3), $t_0 \in \mathit{NotRequests}(q)$, as desired.

□

Lemma 4. *$(\mathit{HLocState}, \mathit{HNotRequests}, HQ)$ constitutes a refinement mapping from SpecP + Inv to SpecR.*

Proof: For any expression Exp, we write $\overline{\mathit{Exp}}$ for the result of applying the substitution

$$[\mathit{HLocState}/\mathit{LocState}, \mathit{HNotRequests}/\mathit{NotRequests}, HQ/Q]$$

to Exp.

We check conditions on initial predicates and on the next-state relation, as follows:

R1. $HQ(e) = Q(e)$ for $e \in I \cup O$, by definition.

R2. *InitPropP* implies $\overline{\mathit{InitProp}}$.
We need: for all $e \in E \cup O$, $HQ(e) = \emptyset$, for all $i \in I$, $HQ(i) \in M^*$, and for all $p \in P$, $(\mathit{HLocState}(p), \mathit{HNotRequests}(p)) \in \mathit{Initial}(p)$.

- For the third conjunct:
  By definition $\mathit{HLocState}(p)$ and $\mathit{HNotRequests}(p)$ are the first two components of $g(p)(H(p)@D(p))$, which equals

  $$g(p)(\langle\!\langle(\mathit{LocState}(p), \mathit{NotRequests}(p))\rangle\!\rangle)$$

  since *InitPropP* implies $H(p) = \langle\!\langle(\mathit{LocState}(p), \mathit{NotRequests}(p))\rangle\!\rangle$. Furthermore,

  $$\begin{aligned} &g(p)(\langle\!\langle(\mathit{LocState}(p), \mathit{NotRequests}(p))\rangle\!\rangle) \\ &= (\mathit{LocState}(p), \mathit{NotRequests}(p), \ldots) \end{aligned}$$

  so $\mathit{HLocState}(p) = \mathit{LocState}(p)$ and $\mathit{HNotRequests}(p) = \mathit{NotRequests}(p)$. Moreover, *InitPropP* implies $(\mathit{LocState}(p), \mathit{NotRequests}(p)) \in \mathit{Initial}(p)$.
- For the first conjunct:
  For $e \in E$, $HQ(e)$ contains at most the messages in $\Pi_e g(p)(H(p)@D(p))$ where $p = \mathit{src}(e)$. Since

  $$g(p)(H(p)@D(p)) = g(p)(\langle\!\langle(\mathit{LocState}(p), \mathit{NotRequests}(p))\rangle\!\rangle)$$

  we have that $\Pi_e g(p)(H(p)@D(p))$ is empty, and a fortiori $HQ(e)$ is empty as well.
  For $o \in O$, $HQ(o) = Q(o)$, so $Q(o) = \emptyset$ implies $HQ(o) = \emptyset$.
- For second conjunct:
  For $i \in I$, $HQ(i) = Q(i)$, so $Q(i) \in M^*$ implies $HQ(i) \in M^*$.

R3.(*MessP*) *MessP* and *Inv* imply

$$\overline{\mathit{MessR} \vee \langle \mathit{LocState}, \mathit{NotRequests}, Q\rangle' = \langle \mathit{LocState}, \mathit{NotRequests}, Q\rangle}$$

Consider a *MessP* step. So for some $p \in P$, some $e \in I \cup E$, some $m \in M$, $u_0, v_0 \in M^*$, we have $p = \mathit{dst}(e)$, $Q(e) = u_0 \cdot m \cdot v_0$, $Q'(e) = u_0 \cdot v_0$, $\mathit{time}(n) \not\leq \mathit{time}(m)$ for all $n \in u_0$, $H'(p) = H(p) \cdot (e, m)$, and $\mathit{LocState}'(p)$, $\mathit{NotRequests}'(p)$, and $Q'(e_i)$ (for e_i such that $\mathit{src}(e_i) = p$) are updated by calculating $g_1(p)(\mathit{LocState}(p), (e, m))$.
Let $\{e_1, \ldots, e_k\} = \{d \in E \cup O \mid \mathit{src}(d) = p\}$, $s = \mathit{LocState}(p)$, and

$$(s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k\rangle) = g_1(p)(s, (e, m))$$

Then $\mathit{LocState}'(p) = s'$, $\mathit{NotRequests}'(p) = \mathit{NotRequests}(p) \cup \{t_1, \ldots, t_n\}$, and $Q'(e_1) = Q(e_1) \cdot \mu_1$, …, $Q'(e_k) = Q(e_k) \cdot \mu_k$.
Other low-level state components are unchanged.
The proof is by cases on whether $\mathit{time}(m) \in D(p)$.
Suppose first that $\mathit{time}(m) \in D(p)$.
We wish to show that there exist $u, v \in M^*$ such that $HQ(e) = u \cdot m \cdot v$, $HQ'(e) = u \cdot v$, and for all n in u, $\mathit{time}(n) \not\leq \mathit{time}(m)$.
We distinguish the cases of $e \in I$ and $e \in E$.

- If $e \in I$, $HQ(e) = Q(e)$, and $HQ'(e) = Q'(e)$, so we can take $u = u_0$ and $v = v_0$.
- If $e \in E$, we have:

  $$HQ(e) = \mu - \nu @ D(p)$$

  where $p_0 = \mathit{src}(e)$, $\mu = \mathit{\Pi}_e g(p_0)(H(p_0) @ D(p_0))$, and $\nu = \langle n \mid n \in M, (e, n) \in H(p)\rangle$, and

  $$HQ'(e) = \mu - \nu' @ D(p)$$

  where μ is as above (because sources and destinations are distinct, so $H'(p_0) = H(p_0)$), and $\nu' = \langle n \mid n \in M, (e, n) \in H(p) \cdot (e, m)\rangle$.
  By the construction of the enriched state space (Construction 1(3)), $\mathit{time}(m) \in \phi(e)(D(p_0))$ follows from $m \in Q(e)$ and $\mathit{time}(m) \in D(p)$, and we also have that

  $$\begin{aligned} &\langle n \mid n \in M, (e, n) \in H(p)\rangle @ D(p) \\ &\quad = \\ &\langle n \mid n \in M, (e, n) \in H(p)\rangle @ (\phi(e)(D(p_0)) \cap D(p)) \end{aligned}$$

  so

  $$HQ(e) = \mu - \langle n \mid n \in M, (e, n) \in H(p)\rangle @ (\phi(e)(D(p_0)) \cap D(p))$$

  Condition 2 implies that

  $$\mu @ \phi(e)(D(p_0)) = \mathit{\Pi}_e g(p_0)(H(p_0)) @ \phi(e)(D(p_0))$$

and hence, by Proposition 12,

$$\mu@(\phi(e)(D(p_0)) \cap D(p)) = \Pi_e g(p_0)(H(p_0))@(\phi(e)(D(p_0)) \cap D(p))$$

So, by Proposition 21,

$$\begin{array}{l} HQ(e)@(\phi(e)(D(p_0)) \cap D(p)) \\ = \begin{pmatrix} \Pi_e g(p_0)(H(p_0)) \\ -\langle n \mid n \in M, (e,n) \in H(p)\rangle@(\phi(e)(D(p_0)) \cap D(p)) \end{pmatrix} \\ \quad @(\phi(e)(D(p_0)) \cap D(p)) \end{array}$$

For proving that $time(n) \not\leq time(m)$ for all n to the left of (the leftmost occurrence) of m in $HQ(e)$, Proposition 20 implies that it suffices to establish this property for

$$\Pi_e g(p_0)(H(p_0)) - \langle n \mid n \in M, (e,n) \in H(p)\rangle@(\phi(e)(D(p_0)) \cap D(p))$$

instead of $HQ(e)$.
By Inv_3,

$$\begin{array}{l} \Pi_e g(p_0)(H(p_0)) \hookrightarrow \langle n \mid n \in M, (e,n) \in H(p)\rangle \cdot Q(e) \\ = \langle n \mid n \in M, (e,n) \in H(p)\rangle \cdot u_0 \cdot m \cdot v_0 \end{array}$$

So, by Proposition 16,

$$\begin{array}{l} \Pi_e g(p_0)(H(p_0)) - \langle n \mid n \in M, (e,n) \in H(p)\rangle@(\phi(e)(D(p_0)) \cap D(p)) \\ \hookrightarrow \langle n \mid n \in M, (e,n) \in H(p)\rangle \not@(\phi(e)(D(p_0)) \cap D(p)) \cdot u_0 \cdot m \cdot v_0 \end{array}$$

Since $time(m) \in \phi(e)(D(p_0)) \cap D(p)$, and $time(n) \not\leq time(m)$ for all $n \in u_0$, we obtain that $time(n) \not\leq time(m)$ for all n to the left of (the leftmost occurrence) of m in

$$\langle n \mid n \in M, (e,n) \in H(p)\rangle \not@(\phi(e)(D(p_0)) \cap D(p)) \cdot u_0 \cdot m \cdot v_0$$

and hence in

$$\Pi_e g(p_0)(H(p_0)) - \langle n \mid n \in M, (e,n) \in H(p)\rangle@(\phi(e)(D(p_0)) \cap D(p))$$

by Proposition 19, and hence also in $HQ(e)$ by Proposition 20 as indicated above.
We let the prefix of $HQ(e)$, to the left of the leftmost occurrence of m, be u; the suffix (to the right) be v.
Furthermore, we have:

$$\begin{array}{l} HQ'(e) = \mu - \nu'@D(p) \\ = \mu - \langle n \mid n \in M, (e,n) \in H(p) \cdot (e,m)\rangle@D(p) \\ = \mu - \nu@D(p) \cdot m \\ = (\mu - \nu@D(p)) - m \\ = HQ(e) - m \\ = u \cdot v \end{array}$$

In this case ($\mathit{time}(m) \in D(p)$), we also need to show that if

$$(s_1', N_1, \langle e_1 \mapsto \nu_1, \ldots, e_k \mapsto \nu_k \rangle) = g_1(p)(\mathit{HLocState}(p), (e, m))$$

then:

- $\mathit{HLocState}'(p) = s_1'$:
  We have that $\mathit{HLocState}'(p) = \Pi_{\mathrm{Loc}} g(p)(H'(p)@D'(p))$, which is

  $$\Pi_{\mathrm{Loc}} g(p)((H(p) \cdot (e, m))@D(p))$$

  Since $\mathit{time}(m) \in D(p)$, we have that

  $$(H(p) \cdot (e, m))@D(p) = H(p)@D(p) \cdot (e, m)$$

  so $\mathit{HLocState}'(p)$ is obtained by applying $g_1(p)$ to $\Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$, in other words to $\mathit{HLocState}(p)$. So $\mathit{HLocState}'(p) = s_1'$.
- $\mathit{HNotRequests}'(p) = \mathit{HNotRequests}(p) \cup N_1$:
  Let $N = \Pi_{\mathrm{NR}} g(p)(H'(p)@D'(p))$, which is $\mathit{HNotRequests}'(p)$ by definition. Since $\mathit{time}(m) \in D(p)$ and $D = D'$, we obtain

  $$N = \Pi_{\mathrm{NR}} g(p)(H(p)@D(p) \cdot (e, m))$$

  So

  $$\begin{aligned} N &= (\Pi_{\mathrm{NR}} g(p)(H(p)@D(p))) \\ &\quad \cup (\Pi_{\mathrm{NR}} g_1(p)((\Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))), (e, m))) \\ &= \mathit{HNotRequests}(p) \cup (\Pi_{\mathrm{NR}} g_1(p)(\mathit{HLocState}(p), (e, m))) \\ &= \mathit{HNotRequests}(p) \cup N_1 \end{aligned}$$

  since $\mathit{HNotRequests}(p) = (\Pi_{\mathrm{NR}} g(p)(H(p)@D(p)))$ and $\mathit{HLocState}(p) = \Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$.
- $HQ'(e_1) = HQ(e_1) \cdot \nu_1$, …, $HQ'(e_k) = HQ(e_k) \cdot \nu_k$:
  - For $e_i = o \in O$:
    We need to show that $Q'(o) = Q(o) \cdot \nu_i$, by the definition of HQ on O.
    We have that $Q'(o) = Q(o) \cdot \mu_i$, by MessP.
    So we need to show that $\mu_i = \Pi_o g_1(p)(s, (e, m))$ and

    $$\nu_i = \Pi_o g_1(p)(\mathit{HLocState}(p), (e, m))$$

    imply that $\mu_i = \nu_i$.
    By Condition 1, since p must be a buffer in this case, we simply have that $\mu_i = \nu_i = m$.
  - For $e_i \in E$:
    Suppose that $\mathit{dst}(e_i) = q$. We have that

    $$\begin{aligned} HQ'(e_i) = \Pi_{e_i} g(p)(H'(p)@D'(p)) \\ - \langle n \mid n \in M, (e_i, n) \in H'(q) \rangle @D'(q) \end{aligned}$$

Simplifying (using in particular that $time(m) \in D(p)$), we obtain

$$HQ'(e_i) = \Pi_{e_i} g(p)((H(p)@D(p))\cdot(e,m)) \\ - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q)$$

We also have that

$$HQ(e_i) = \Pi_{e_i} g(p)(H(p)@D(p)) \\ - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q)$$

Since $HLocState(p) = \Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$,

$$\Pi_{e_i} g(p)((H(p)@D(p))\cdot(e,m))$$

is obtained from $\Pi_{e_i} g(p)(H(p)@D(p))$ by concatenating

$$\Pi_{e_i} g_1(p)(HLocState(p),(e,m))$$

called ν_i above. Therefore, we have:

$$\begin{aligned} &HQ'(e_i) \\ &= (\Pi_{e_i} g(p)(H(p)@D(p))\cdot\nu_i) - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q) \\ &= (\Pi_{e_i} g(p)(H(p)@D(p))) - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q))\cdot\nu_i \\ &= HQ(e_i)\cdot\nu_i \end{aligned}$$

as desired. The second equality requires Proposition 24.

- All other high-level state components are unchanged. This holds for $HLocState(q)$ for all $q \neq p$ because $H(q)$ does not change in this transition. It holds for $HNotRequests(q)$ for the same reason. It also holds for $HQ(d)$ for all $d \in I \cup E \cup O - \{e, e_1, \ldots, e_k\}$:
  - For $d \in I \cup O$, $HQ'(d) = Q'(d)$, and $HQ(d) = Q(d)$, so $MessP$, which gives us $Q'(d) = Q(d)$, implies $HQ'(d) = HQ(d)$.
  - For $d \in E$, we have:
    - $H'(src(d)) = H(src(d))$ (since d cannot be among $e_1, \ldots, e_k$, so $src(d) \neq p$);
    - if $dst(d) \neq p$, then $H'(dst(d)) = H(dst(d))$;
    - if $dst(d) = p$, then $H'(dst(d)) = H(dst(d))\cdot(e,m)$, so $\langle n \mid n \in M, (d,n) \in H'(p)\rangle = \langle n \mid n \in M, (d,n) \in H(p)\rangle$, so $H'(dst(d))$ and $H(dst(d))\cdot(e,m)$ induce the same $HQ'(d)$.

Now suppose $time(m) \notin D(p)$. We argue that $HLocState$, $HNotRequests$, and HQ are unchanged.

Since $H(q)@D(q) = H'(q)@D'(q)$ in this case, for all q including p, we have $HLocState'(q) = HLocState(q)$ and $HNotRequests'(q) = HNotRequests(q)$.

The argument about HQ is different for each type of edge:

- For $i \in I$, we need $HQ'(i) = HQ(i)$, that is, $Q'(i) = Q(i)$. By the construction of the enriched state space (Construction 1(1)), we have that $time(m_0) \in D'(dst(i))$ for each m_0 such that $(i, m_0) \in H'(dst(i))$. By $MessP$, $D' = D$, so $time(m_0) \in D(dst(i))$ for such m_0. Since $(e,m) \in H'(p)$ and $time(m) \notin D'(p)$, we have that $p \neq dst(i)$, so $e \neq i$. Therefore, $MessP$ implies $Q'(i) = Q(i)$, as desired.

- For $d \in E$, $HQ'(d) = HQ(d)$ follows $H'(p)@D'(p) = H(p)@D(p)$ and $H'(q) = H(q)$ for all $q \neq p$.
- For $o \in O$, we need $HQ'(o) = HQ(o)$, that is, $Q'(o) = Q(o)$. By the construction of the enriched state space (Construction 1(2)), we have that $time(m_0) \in D'(src(o))$ for each m_0 such that, for some e_0, $(e_0, m_0) \in H'(src(o))$. By *MessP*, $D' = D$, so $time(m_0) \in D(src(o))$ for such m_0. Since $(e, m) \in H'(p)$ and $time(m) \notin D(p)$, we obtain that $p \neq src(o)$. Therefore, *MessP* implies $Q'(o) = Q(o)$, as desired.

R3.(*NotP*) *NotP* and *Inv* imply

$$\overline{Not \vee \langle LocState, NotRequests, Q\rangle' = \langle LocState, NotRequests, Q\rangle}$$

Consider a *NotP* step. So for some $p \in P$, some $t \in NotRequests(p)$, we have $(e, t) \notin Clock$ for all $e \in I \cup E$ such that $dst(e) = p$, $H'(p) = H(p)\cdot t$, and $LocState'(p)$, $NotRequests'(p)$, and $Q'(e_i)$ (for e_i such that $src(e_i) = p$) are updated by calculating $g_1(p)(LocState(p), t)$.
Let $\{e_1, \ldots, e_k\} = \{d \in E \cup O \mid src(d) = p\}$, $s = LocState(p)$, and

$$(s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k\rangle) = g_1(p)(s, t)$$

Then $LocState'(p) = s'$, $NotRequests'(p) = NotRequests(p) - \{t\} \cup \{t_1, \ldots, t_n\}$, and $Q'(e_1) = Q(e_1)\cdot\mu_1$, $\ldots$, $Q'(e_k) = Q(e_k)\cdot\mu_k$.
Other low-level state components are unchanged.
The proof is by cases on whether $t \in D(p)$.
Suppose first that $t \in D(p)$.
In this case, we need to show:

- $t \in HNotRequests(p)$:
  We have that $t \in NotRequests(p)$, so $t \in \Pi_{\mathrm{NR}} g(p)(H(p))$ by Inv_2, so $t \in \Pi_{\mathrm{NR}} g(p)(H(p)@D(p))$ by Condition 2, that is, $t \in HNotRequests(p)$.
- For all $e \in I \cup E$ such that $dst(e) = p$, $(e, t) \notin HClock$:
  We have that, for such e, $(e, t) \notin Clock$. For $e \in I$, Proposition 25 immediately yields the desired result. We proceed with the case where $e \in E$. We have $t \in H'(p)$, and also $t \in D'(p)$ since *NotP* implies $D(p) = D'(p)$. So by the construction of the enriched state space (Construction 1(4)), for all $p_0 \in P$, $e_0 \in E$ such that $src(e_0) = p_0$, $e \in E$ such that $dst(e) = p$, $t_0 \in T$, if $(e_0, t_0) \rightsquigarrow (e, t)$ then $t_0 \in \phi(e_0)(D'(p_0))$, and hence in $\phi(e_0)(D(p_0))$. Then, by *ClockCorrespondence* (obtained in Proposition 26), for all $e \in E$ such that $dst(e) = p$, if $(e, t) \in HClock$ then $(e, t) \in Clock$. Since, for all e such that $dst(e) = p$, we have assumed that $(e, t) \notin Clock$, we obtain that $(e, t) \notin HClock$.

Most of the rest of the proof for *NotP* is directly analogous to the corresponding parts of the proof for *MessP*, with changes only in easy details.
In this case ($t \in D(p)$), we also need to show that if

$$(s'_1, N_1, \langle e_1 \mapsto \nu_1, \ldots, e_k \mapsto \nu_k\rangle) = g_1(p)(HLocState(p), t)$$

then:

- $HLocState'(p) = s_1'$:
  We have that

  $$HLocState'(p) = \Pi_{\mathrm{Loc}} g(p)(H'(p)@D'(p))$$

  which is $\Pi_{\mathrm{Loc}} g(p)((H(p)\cdot t)@D(p))$. Since $t \in D(p)$, we have that

  $$(H(p)\cdot t)@D(p) = H(p)@D(p)\cdot t$$

  so $HLocState'(p)$ is obtained by applying $g_1(p)$ to $\Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$, in other words to $HLocState(p)$. So $HLocState'(p) = s_1'$.
- $HNotRequests'(p) = HNotRequests(p) - \{t\} \cup N_1$:
  Let $N = \Pi_{\mathrm{NR}} g(p)(H'(p)@D'(p))$, which is $HNotRequests'(p)$ by definition. Since $time(m) \in D(p)$ and $D = D'$, we obtain

  $$N = \Pi_{\mathrm{NR}} g(p)(H(p)@D(p)\cdot(e,m))$$

  So

  $$\begin{aligned} N &= (\Pi_{\mathrm{NR}} g(p)(H(p)@D(p))) - \{t\} \\ &\quad \cup (\Pi_{\mathrm{NR}} g_1(p)((\Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))), (e,m))) \\ &= HNotRequests(p) - \{t\} \cup (\Pi_{\mathrm{NR}} g_1(p)(HLocState(p), (e,m))) \\ &= HNotRequests(p) - \{t\} \cup N_1 \end{aligned}$$

  since $HNotRequests(p) = (\Pi_{\mathrm{NR}} g(p)(H(p)@D(p)))$ and $HLocState(p) = \Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$.
- $HQ'(e_1) = HQ(e_1)\cdot\nu_1, \ldots, HQ'(e_k) = HQ(e_k)\cdot\nu_k$:
  - For $e_i = o \in O$:
    We need to show that $Q'(o) = Q(o)\cdot\nu_i$, by the definition of HQ on O.
    We have that $Q'(o) = Q(o)\cdot\mu_i$, by $NotP$.
    So we need to show that $\mu_i = \Pi_o g_1(p)(s,t)$ and

    $$\nu_i = \Pi_o g_1(p)(HLocState(p), t)$$

    imply that $\mu_i = \nu_i$.
    By Condition 1, since p must be a buffer in this case, we simply have that $\mu_i = \nu_i = \emptyset$.
  - For $e_i \in E$:
    Suppose that $dst(e_i) = q$. We have that

    $$\begin{aligned} HQ'(e_i) = \Pi_{e_i} g(p)(H'(p)@D'(p)) \\ - \langle n \mid n \in M, (e_i, n) \in H'(q)\rangle @D'(q) \end{aligned}$$

    Simplifying (using in particular that $t \in D(p)$), we obtain

    $$\begin{aligned} HQ'(e_i) = \Pi_{e_i} g(p)((H(p)@D(p))\cdot t) \\ - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q) \end{aligned}$$

We also have that

$$HQ(e_i) = \Pi_{e_i} g(p)(H(p)@D(p)) \\ - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q)$$

Since $HLocState(p) = \Pi_{\mathrm{Loc}} g(p)(H(p)@D(p))$,

$$\Pi_{e_i} g(p)((H(p)@D(p))\cdot t)$$

is obtained from $\Pi_{e_i} g(p)(H(p)@D(p))$ by concatenating

$$\Pi_{e_i} g_1(p)(HLocState(p), t)$$

called ν_i above. Therefore, we have:

$$\begin{aligned} &HQ'(e_i) \\ &= (\Pi_{e_i} g(p)(H(p)@D(p))\cdot\nu_i) - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q) \\ &= (\Pi_{e_i} g(p)(H(p)@D(p))) - \langle n \mid n \in M, (e_i, n) \in H(q)\rangle @D(q))\cdot\nu_i \\ &= HQ(e_i)\cdot\nu_i \end{aligned}$$

as desired. The second equality requires Proposition 24.

- All other high-level state components are unchanged. This holds for $HLocState(q)$ for all $q \neq p$ because $H(q)$ does not change in this transition. It holds for $HNotRequests(q)$ for the same reason. It also holds for $HQ(d)$ for all $d \in I \cup E \cup O - \{e_1, \ldots, e_k\}$:
  - For $d \in I \cup O$, $HQ'(d) = Q'(d)$, and $HQ(d) = Q(d)$, so $NotP$, which gives us $Q'(d) = Q(d)$, implies $HQ'(d) = HQ(d)$.
  - For $d \in E$, we have:
    - $H'(src(d)) = H(src(d))$ (since d cannot be among $e_1, \ldots, e_k$, so $src(d) \neq p$),
    - if $dst(d) \neq p$, then $H'(dst(d)) = H(dst(d))$,
    - if $dst(d) = p$, then $H'(dst(d)) = H(dst(d))\cdot t$, so $\langle n \mid n \in M, (d, n) \in H'(p)\rangle = \langle n \mid n \in M, (d, n) \in H(p)\rangle$, so $H'(dst(d))$ and $H(dst(d))\cdot t$ induce the same $HQ'(d)$.

Now suppose $t \notin D(p)$. We argue that $HLocState$, $HNotRequests$, and HQ are unchanged.

Since $H(q)@D(q) = H'(q)@D'(q)$ in this case, for all q including p, we have $HLocState'(q) = HLocState(q)$ and $HNotRequests'(q) = HNotRequests(q)$.

The argument about HQ is different for each type of edge:

- For $i \in I$, we need $HQ'(i) = HQ(i)$, that is, $Q'(i) = Q(i)$, which is immediate for $NotP$.
- For $d \in E$, $HQ'(d) = HQ(d)$ follows $H'(p)@D'(p) = H(p)@D(p)$ and $H'(q) = H(q)$ for all $q \neq p$.
- For $o \in O$, we need $HQ'(o) = HQ(o)$, that is, $Q'(o) = Q(o)$. If $p \neq src(o)$, then $NotP$ immediately implies $Q'(o) = Q(o)$, as desired. If $p = src(o)$, then Condition 1 implies that p must be a buffer, and $NotP$ implies $Q'(o) = Q(o)$ when p is a buffer.

R3.($InpP$) $InpP$ and Inv imply

$$\overline{Inp \vee \langle LocState, NotRequests, Q\rangle' = \langle LocState, NotRequests, Q\rangle}$$

Since $InpP$ implies $H'(p) = H(p)$ and $D'(p) = D(p)$ for all p, and $Q'(o) = Q(o)$ for all $o \in O$, the definitions of the refinement mapping yield

$$\overline{\langle LocState, NotRequests, Q\restriction(E \cup O)\rangle' = \langle LocState, NotRequests, Q\restriction(E \cup O)\rangle)}$$

Since, for every $i \in I$, $InpP$ also implies that $Q(i)$ is a subsequence of $Q'(i)$, those definitions yield that $HQ(i)$ is a subsequence of $HQ'(i)$. We conclude that $InpP$ implies $\overline{Inp}$.

R3.($OutpP$) $OutpP$ and Inv imply

$$\overline{Outp \vee \langle LocState, NotRequests, Q\rangle' = \langle LocState, NotRequests, Q\rangle}$$

Since $OutpP$ implies $H'(p) = H(p)$ and $D'(p) = D(p)$ for all p, and $Q'(i) = Q(i)$ for all $i \in I$, the definitions of the refinement mapping yield

$$\overline{\langle LocState, NotRequests, Q\restriction(I \cup E)\rangle' = \langle LocState, NotRequests, Q\restriction(I \cup E)\rangle)}$$

Since, for every $o \in O$, $OutpP$ also implies that $Q'(o)$ is a subsequence of $Q(o)$, those definitions yield that $HQ'(o)$ is a subsequence of $HQ(o)$. We conclude that $OutpP$ implies $\overline{Outp}$.

R3.($RollbackP$) $RollbackP$ and Inv imply

$$\overline{\langle LocState, NotRequests, Q\rangle' = \langle LocState, NotRequests, Q\rangle}$$

Consider a $RollbackP$ step. So we have some f such that

$$\begin{array}{l}
\forall p \in P, i \in I \text{ such that } dst(i) = p.\\
\{time(m) \mid (i, m) \in H(p)\} \subseteq f(p)\\
\wedge\\
\forall p \in P, o \in O \text{ such that } src(o) = p.\\
\{time(m) \mid \exists e.(e, m) \in H(p)\} \subseteq f(p)\\
\wedge\\
\forall p, q \in P, e \in E \text{ such that } src(e) = p \wedge dst(e) = q.\\
\{time(m) \mid (e, m) \in H(q)\} \cap f(q) \subseteq \phi(e)(f(p))\\
\wedge\\
\forall p, q \in P, e_1, e_2 \in E \text{ such that } src(e_1) = p \wedge dst(e_2) = q,\\
t_1 \in T, t_2 \in H(q)@f(q).\\
\quad \text{if } (e_1, t_1) \rightsquigarrow (e_2, t_2) \text{ then } t_1 \in \phi(e_1)(f(p))\\
\wedge\\
\forall e \in I \cup O.Q'(e) = Q(e)
\end{array}$$

and, for all p,

$$
\begin{array}{l}
f(p) \cap \{t \mid \exists e \in E, m \in M. dst(e) = p \wedge m \in Q(e) \wedge time(m) = t\} = \emptyset \\
\wedge \\
let\ \{e_1, \ldots, e_k\} = \{d \in E \cup O \mid src(d) = p\}, \\
h = H(p) @ f(p), \\
(s', \{t_1, \ldots, t_n\}, \langle e_1 \mapsto \mu_1, \ldots, e_k \mapsto \mu_k \rangle) = g(p)(h) \\
in\ \forall i \in 1 \ldots k.\ if\ \ e_i \in E\ then\ Q'(e_i) = \mu_i @ (\phi(e_i)(f(p)) \cap f(dst(e_i))) \\
\wedge \\
LocState'(p) = s' \\
\wedge \\
NotRequests'(p) = \{t_1, \ldots, t_n\} \\
\wedge \\
H'(p) = h
\end{array}
$$

Consider a node p. We have $HLocState(p) = \Pi_{\mathrm{Loc}} g(p)(H(p) @ D(p))$, and

$$
\begin{aligned}
HLocState'(p) &= \Pi_{\mathrm{Loc}} g(p)(H'(p) @ D'(p)) \\
&= \Pi_{\mathrm{Loc}} g(p)(H(p) @ f(p) @ D'(p)) \\
&= \Pi_{\mathrm{Loc}} g(p)(H(p) @ D(p))
\end{aligned}
$$

by Proposition 12, so they are equal. Similarly, we have $HNotRequests(p) = \Pi_{\mathrm{NR}} g(p)(H(p) @ D(p))$, and

$$
\begin{aligned}
HNotRequests'(p) &= \Pi_{\mathrm{NR}} g(p)(H'(p) @ D'(p)) \\
&= \Pi_{\mathrm{NR}} g(p)(H(p) @ f(p) @ D'(p)) \\
&= \Pi_{\mathrm{NR}} g(p)(H(p) @ D(p))
\end{aligned}
$$

by Proposition 12, so they are equal.
For $e \in I \cup O$, $HQ(e) = Q(e)$, $HQ'(e) = Q'(e)$, so they are equal since *RollbackP* implies $Q'(e) = Q(e)$ for such e.
For $e \in E$, with $dst(e) = q$:

$$HQ(e) = \mu_0 - \nu_0 @ D(q)$$

where

$$
\begin{aligned}
\mu_0 &= \Pi_e g(p)(H(p) @ D(p)) \\
\nu_0 &= \langle m \mid m \in M, (e, m) \in H(q) \rangle
\end{aligned}
$$

Also,

$$HQ'(e) = \mu_1 - \nu_1 @ D'(q)$$

where

$$
\begin{aligned}
\mu_1 &= \Pi_e g(p)(H'(p) @ D'(p)) \\
&= \Pi_e g(p)(H(p) @ D(p)) \quad \text{as above by Proposition 12} \\
&= \mu_0
\end{aligned}
$$

and

$$\begin{aligned}\nu_1 &= \langle m \mid m \in M, (e,m) \in H'(q) \rangle \\ &= \langle m \mid m \in M, (e,m) \in H(q)@f(q) \rangle \\ &= \nu_0@f(q)\end{aligned}$$

Since $D = f \cap D'$, we obtain $\nu_0@D(q) = \nu_1@D'(q)$ by Proposition 12. Therefore, $HQ'(e) = HQ(e)$.

□

Lemma 5. *$(HLocState, HNotRequests, HQ)_{/\mathcal{Q}}$ constitutes a refinement mapping from $(SpecP + Inv)_{/\mathcal{Q}}$ to SpecR.*

Proof: By Proposition 3, it suffices to invoke Lemma 4 and to check that if

$$(LocState_1, NotRequests_1, H_1, Q_1, D_1)\mathcal{Q}(LocState_2, NotRequests_2, H_2, Q_2, D_2)$$

then the refinement mapping maps $(LocState_1, NotRequests_1, H_1, Q_1, D_1)$ and $(LocState_2, NotRequests_2, H_2, Q_2, D_2)$ to the same state.

So, let us assume that

$$(LocState_1, NotRequests_1, H_1, Q_1, D_1)\mathcal{Q}(LocState_2, NotRequests_2, H_2, Q_2, D_2)$$

It follows that $(LocState_1, NotRequests_1, H_1, Q_1) = (LocState_2, NotRequests_2, H_2, Q_2)$ and that D_1 and D_2 coincide on $ITimes(H_1)$. Since the refinement mapping depends on D_1 (or D_2) only via $H_1(p)@D_1(p)$ (or $H_2(p)@D_2(p)$) for $p \in P$, and since $H_1(p)@D_1(p) = H_1(p)@D_2(p)$ when D_1 and D_2 coincide on $ITimes(H_1)$, the refinement mapping maps $(LocState_1, NotRequests_1, H_1, Q_1, D_1)$ and $(LocState_2, NotRequests_2, H_2, Q_2, D_2)$ to the same state. □

Proposition 27. *$SpecP_{/\mathcal{Q}}$ implements SpecR.*

Proof: *Inv* is an inductive invariant of *SpecP* by Lemma 3. Since *Inv* does not mention D, it respects $\mathcal{Q}$. Therefore, $Inv_{/\mathcal{Q}}$ is an inductive invariant of $SpecP_{/\mathcal{Q}}$ by Proposition 5, and $SpecP_{/\mathcal{Q}}$ implements $SpecP_{/\mathcal{Q}} + Inv_{/\mathcal{Q}}$ by Proposition 6. In turn, $SpecP_{/\mathcal{Q}} + Inv_{/\mathcal{Q}}$ implements $(SpecP + Inv)_{/\mathcal{Q}}$ by Proposition 7. Finally, $(SpecP + Inv)_{/\mathcal{Q}}$ implements *SpecR* by Lemma 5 and the soundness of refinement mappings (Proposition 1). By transitivity, $SpecP_{/\mathcal{Q}}$ implements *SpecR*.

(This proof strategy is a little more general than strictly needed: *Inv* respects $\mathcal{Q}$ follows from the fact that *Inv* does not mention D at all. It may be possible to leverage this fact to obtain a simple proof that *SpecL* implements $(SpecP + Inv)_{/\mathcal{Q}}$, which would be a helpful intermediate step.) □

Proof of Theorem 1: This follows immediately from Propositions 27 and 23, by transitivity. □

Further refinement proof

Proof of Theorem 2: We prove that *SpecS* implies *SpecL* by showing that *RollbackS* implies *RollbackL*. This, in turn, we do by showing that conditions (1), (2), and (3) imply the "$\rightsquigarrow$-based condition". In fact, we prove something stronger: for $p, q \in P$, $e_1, e_2 \in E$ such that $src(e_1) = p$ and $dst(e_2) = q$, $t_1, t_2 \in T$, we show that if $(e_1, t_1) \rightsquigarrow (e_2, t_2)$ and $t_2 \in f_c(q)$ then $t_1 \in \phi(e_1)(f_c(p))$.

The "$\rightsquigarrow$-based condition" follows because $t_2 \in H(q)@f(q)$ implies $t_2 \in f_c(q)$ by (2), and $\phi(e_1)(f_c(p)) \subseteq \phi(e_1)(f(p))$ by (1) and Condition 3(3) (monotonicity).

We argue by complete induction on the proof of $(e_1, t_1) \rightsquigarrow (e_2, t_2)$. We analyze this proof.

- If $e_1 = e_2$ then $t_1 \leq t_2$ by Proposition 8(3). So it suffices to show that $f_c(q) \subseteq \phi(e_1)(f_c(p))$. This is what (3) says.
- If $e_1 \neq e_2$, we look at $dst(e_1)$. Call it r. By Proposition 8(2), $(r, t_1) \rightsquigarrow (e_2, t_2)$. By Proposition 8(1), there exist e' and t' such that $src(e') = r$ and $(r, t_1) \rightsquigarrow (e', t')$ and $(e', t') \rightsquigarrow (e_2, t_2)$, with a strictly smaller proof than that of $(e_1, t_1) \rightsquigarrow (e_2, t_2)$. By induction hypothesis, $t' \in \phi(e')(f_c(r))$. By Proposition 10, since $(r, t_1) \rightsquigarrow (e', t')$, $t_1 \in f_c(r)$. By (3), $t_1 \in \phi(e_1)(f_c(p))$.

□

The proof of Theorem 2 above includes an argument that conditions (1), (2), and (3) imply the "$\rightsquigarrow$-based condition". The converse relation between conditions does not hold, at least not for any f, as the following example illustrates.

Example 4. Suppose that we have three nodes r, p, and q, with edge d from r to p and e from p to q. Suppose that $\phi(e)$ and $\phi(d)$ are the identity, and that p is a node such that $(d, 1) \rightsquigarrow (e, 2)$ but not $(d, 2) \rightsquigarrow (e, 2)$. (For example, p could produce an output at time 2 every time it gets an input at time 1, but the output may be affected by previous inputs at time 2.) Let $f(p) = f(q) = \{1, 2\}$ and $f(r) = \{1\}$. Note that *RollbackL* requires that $(\{time(m) \mid (d, m) \in H(p)\}) \cap f(p)) \subseteq f(r)$, but this can hold if r has not sent any messages at time 2 to p. Suppose $H(q)@f(q) = \{1, 2\}$. $H(r)$ and $H(p)$ do not contain notifications. The "$\rightsquigarrow$-based condition" holds; it requires that $1, 2 \in f(p)$ and $1 \in f(r)$. On the other hand, conditions (1), (2), and (3) imply constraints on the choice of f_c:

- $f_c(r) \subseteq f(r)$
- $f_c(p) \subseteq f(p)$
- $f_c(q) \subseteq f(q)$
- $f_c(q) = \{1, 2\}$
- $f_c(p) \subseteq f_c(r)$
- $f_c(q) \subseteq f_c(p)$

Putting these together, we obtain that $f_c(q) \subseteq f(r) = \{1\}$, but also that $f_c(q) = \{1, 2\}$, in contradiction. So we cannot find f_c to satisfy conditions (1), (2), and (3).

Corollary 1. *SpecS implies SpecR.*

Proof: This follows immediately from Theorem 2 and Theorem 1, by transitivity. □