

Verifying Chain Replication using Events

Mark Bickford

September 20, 2006

1 Notation and Preliminaries

1.1 Event Structures

To show that a concrete machine satisfies a specification, we will be defining constraints on sets of input and output *events* and proving that the constraints corresponding to the implementation machine imply the constraints that constitute the specification. In defining these constraints we make use of some generic structure that the events have as part of an event structure. An *event structure*, $es = \langle E, loc, <_c, \dots \rangle$ is a type E together with a collection of operations and axioms. For our purposes we need mention only a few of these:

1. Every event $e \in E$ has a *location*, $loc(e) \in Loc$.
2. *Causal order* $<_c$ is a well-founded, transitive order on E .
3. *Local order*, defined by $e <_{loc} e' \equiv e <_c e' \wedge loc(e) = loc(e')$, is a total ordering of events with a given location.

All of our definitions will have an event structure parameter es which we treat as implicit.

1.2 Interfaces

An *interface* X of type T provides two things:

1. A decidable set $E(X)$ of events.
2. A function $X(e)$ of type T defined on $E(X)$.

Thus, interface X partitions the events E in the event structure and for those events e that are in $E(X)$ assigns a value of type T . (Formally, X is just a function of type $E \rightarrow T + Top$.) We define $\bar{X}(L)$ to be $map(X, L)$ for a list L of events in $E(X)$. If X is any interface, it is useful to define the interface $prior_X$ so that the events $e \in E(prior_X)$ are just those events e for which there is a prior $e' <_{loc} e$ for which $e' \in E(X)$ and the value $prior_X(e)$ is the last such e' .

A *retraction* on $E(X)$ is a function $f : E(X) \rightarrow E(X)$ such that

$$\forall e : E(X). f(e) \leq_c e$$

Since $<_c$ is well-founded, for any $e \in E(X)$, iterating f eventually reaches a fixed point $f^\infty(e)$. If e' is one of the f -iterates of e then we write $e' \xleftarrow{f} e$. (The arrow points left because we tend to draw causally prior events to the left.)

There is an induction principle for this relation. To show that

$$\forall a, b : E(X). (b \xleftarrow{f} a) \Rightarrow P(a, b)$$

it is enough to show

$$\forall a : E(X). P(a, a)$$

and

$$\forall a, b : E(X). (P(a, b) \wedge f(b) \neq b) \Rightarrow P(a, f(b))$$

1.3 Basic parameters and notation

Fix type parameters Cmd and Rsp for the *commands* and *responses*, and a boolean function $isupdate : Cmd \rightarrow \mathbb{B}$ that partitions the commands into *updates* (commands c for which $isupdate(c)$) and *queries* (commands c for which $\neg isupdate(c)$). Let $Update$ and $Query$ be the subtypes of Cmd defined by this partition. The “correct” response to an update command c that has been preceded by the list of updates cs' will be $\Delta(cs'.c)$ where Δ is a parameter of type $Update List \rightarrow Rsp$. The correct response to a query q will be $Q(cs', q)$ where parameter Q has type $Update List \rightarrow Query \rightarrow Rsp$.

The top-level specification for the chain-replicated state machine will be a relation (defined in terms of parameters Cmd , Rsp , $isupdate$, Δ , and Q) between an *input interface* In of type Cmd and an *output interface* Out of type Rsp . An update command arrives as an event $e \in E(In)$ with value $c = In(e)$ and via a chain of intermediate events arrives at the tail as an output event $e' \in E(Out)$ with value $Out(e')$. A query event is an input

$e_q \in E(In)$ with (input) value $q = In(e_q)$ which is not an update, so, in terms of our parameters, we can define a query event by

$$query(e) \Leftrightarrow e \in E(In) \wedge \neg isupdate(In(e))$$

In addition to the local and causal orderings on events, we will need some more orderings. Let \sqsubseteq be the prefix relation on lists and \subseteq be the sublist ordering.

2 Specification

Definition We say that interfaces In and Out satisfy the *approximate state machine* relation

$$ApproxSM(Cmd, Rsp, isupdate, \Delta, Q, In, Out)$$

if there is an *explanation* function $expl$ of type

$$E(Out) \rightarrow \{e : E(In) \mid isupdate(In(e))\} List$$

such that for every $e \in E(Out)$

1. $Out(e) = \mathbf{if} \ query(e) \ \mathbf{then} \ Q(\vec{In}(expl(e)), In(e)) \ \mathbf{else} \ \Delta(\vec{In}(expl(e)))$
2. $\forall e' : E(In). e' \in expl(e) \Rightarrow e' \leq_c e$
3. $\neg query(e) \Rightarrow expl(e)$ is non-empty
4. $\forall e, e' : E(Out). (expl(e) \sqsubseteq expl(e') \vee expl(e') \sqsubseteq expl(e))$

3 First Refinement

Our goal is to show that in the actual implementation of chain-replication we can identify input and output interfaces In and Out that satisfy $ApproxSM(In, Out)$. We will accomplish this by a series of logical refinements

$$ApproxSM(In, Out) \Leftarrow \psi_1 \Leftarrow \dots \Leftarrow \psi_{k-1} \Leftarrow \psi_k$$

such that the final refinement ψ_k is “obviously” satisfied by the concrete implementation. Ideally, the concrete implementation can be generated automatically from the final refinement.

In a typical refinement step $\psi_i \Leftarrow \psi_{i+1}$, the more abstract ψ_i will be a constraint between some sets of events given by interfaces, like In and Out ,

and the more concrete ψ_{i+1} will introduce additional intermediate events and constraints between these and the events from the abstract specification. For our first refinement of the top-level specification $ApproxSM(In, Out)$, we will introduce only the intermediate events that correspond to the forwarding of update commands along the chain but including the events that resend update commands after a reconfiguration. The reconfiguration events themselves will be introduced in a later refinement. Since the intermediate events include normal forwarding of an update command along the chain as well as the resend events that can, in effect, forward a list of update commands, it will be most convenient to introduce the intermediate events as an interface Sys of type $Cmd List$, since we can think of a normal forwarding event $e \in E(Sys)$ that forwards update command c as having value $Sys(e) = [c]$, a singleton list.

The events in $E(Sys)$ will include the input and output events in $E(In)$ and $E(Out)$, so all the events of interest at this level of refinement are in $E(Sys)$, and we also introduce an *antecedent function*

$$f : E(Sys) \rightarrow E(Sys)$$

The antecedent function f will be a retraction ($f(e) \leq_c e$) and the idea is that $f(e)$ is the proximate *cause* of event e and traces back where the data in e (the commands coded as $Sys(e)$) came from. There will be three cases:

$f(e) = e$: Event e is an input.

$f(e) \neq e \wedge loc(f(e)) = loc(e)$: Event e is a resend event and $f(e)$ is its local predecessor.

$f(e) \neq e \wedge loc(f(e)) \neq loc(e)$: Event e is a receipt event and $f(e)$ is the event that sent it.

We will define a set of constraints on Sys and f whose conjunction we will call $AbstractChainRepl(Sys, f, In, Out)$ and which will constitute our first refinement. The “combinatorial core” of the chain replication protocol is captured in the proof that

$$AbstractChainRepl(Sys, f, In, Out) \Rightarrow ApproxSM(In, Out)$$

4 Chain-consistency

Definition The first constraint on the antecedent function f is that its fixedpoints are exactly the events in $E(In)$.

$$\forall e : E(Sys). e \in E(In) \Leftrightarrow f(e) = e \tag{1}$$

The antecedent function f is *chain consistent* if there exists a function $chain : E(Sys) \rightarrow Loc List$ such that:

$$\forall e : E(Sys). e \in chain(e) \wedge chain(e) \text{ has no repeats} \quad (2)$$

$$\forall e_1, e_2 : E(Sys). chain(e_1) \subseteq chain(e_2) \vee chain(e_2) \subseteq chain(e_1) \quad (3)$$

$$\forall e_1, e_2 : E(Sys). e_1 <_{loc} e_2 \Rightarrow chain(e_2) \subseteq chain(e_1) \quad (4)$$

$$\forall e : E(Out). loc(e) = last(chain(e)) \quad (5)$$

$$\forall e : E(In). loc(e) = \mathbf{if} \text{ query}(e) \mathbf{ then } last(chain(e)) \mathbf{ else } head(chain(e)) \quad (6)$$

$$\begin{aligned} \forall e : E(Sys). loc(f(e)) \neq loc(e) \Rightarrow \\ loc(f(e)), loc(e) \text{ adjacent in } chain(f(e)) \wedge \\ loc(f(e)), loc(e) \text{ adjacent in } chain(e) \quad (7) \end{aligned}$$

$$\begin{aligned} \forall e : E(Sys). \neg(e \in E(In)) \wedge loc(f(e)) = loc(e) \Rightarrow \\ f(e) = \text{prior}_{Sys}(e) \quad (8) \end{aligned}$$

Definition The antecedent function f is *fifo* if

$$\forall e, e' : E(Sys). loc(e) = loc(e') \wedge f(e) \leq_{loc} f(e') \Rightarrow e \leq_{loc} e' \quad (9)$$

Before we define the other constraints that make up *AbstractChainRepl*, we will derive a number of properties that follow from chain-consistency and fifo alone. For the remainder, suppose that f is chain-consistent and that $chain$ is the witness, and assume that f is fifo.

We start by defining one more ordering $i \prec j$, this time on locations, as follows:

Definition

$$i \prec j \equiv \exists e : E(Sys). i \text{ is before } j \text{ in } chain(e)$$

Lemma 1 \prec is a linear order on the set $\{loc(e) \mid e \in E(Sys)\}$ and

$$e' \xleftarrow{f} e \Rightarrow loc(e') \preceq loc(e)$$

proof: \prec is antireflexive because of (2).

\prec is transitive and total because of (2) and (3). The second assertion follows from the induction principle for $e' \xleftarrow{f} e$ if we show that $loc(f(e)) \preceq loc(e)$. This is trivial if $loc(f(e)) = loc(e)$ and follows from (7) otherwise.

Definition There is a *path* between locations i and j if there exists $e \xleftarrow{f} e'$ with $loc(e) = i$ and $loc(e') = j$. The path *goes through* location k if there exists an e'' with $loc(e'') = k$ and $e \xleftarrow{f} e''$ and $e'' \xleftarrow{f} e'$.

Lemma 2 *If $i \preceq j \preceq k$ and there is an update input e' at location j then any path from i to k goes through j . (cite)*

proof: We prove this using the induction principle on paths. The base case is trivial since then $i=j=k$. The induction step reduces to showing that $i = loc(f(e)) \prec j \prec loc(e) = k$ is impossible. By (3), either $chain(e) \subseteq chain(e')$ or $chain(e') \subseteq chain(e)$, but the former is impossible since $i \prec j$ and, by (6), $j = head(chain(e'))$, and the latter is impossible since $i \prec j \prec k$ but, by (7), i and k are adjacent in $chain(e)$.

Definition The paths defined by antecedent function f are *ordered* if for all $a_1, a_2, b_1, b_2 \in E(Sys)$: If $a_1 \xleftarrow{f} b_1$ and $a_2 \xleftarrow{f} b_2$ and $\neg(a_1 \xleftarrow{f} a_2)$, and if $a_1 <_{loc} a_2$ and $loc(b_1) = loc(b_2)$, then $b_1 <_{loc} b_2$.

remarks This says that two paths from i to j that are ordered one way at location i are ordered the same way at location j , but since our paths can have several events at one location (because of the resend events which have $f(e) \neq e$ but $loc(f(e)) = loc(e)$) we have to include the extra condition $\neg(a_1 \xleftarrow{f} a_2)$ to avoid degenerate cases. If for some event e , $f^3(e) <_{loc} f^2(e)$ and $f(e) <_{loc} e$ then the paths $a_1 = f^3(e) \xleftarrow{f} e = b_1$ and $a_2 = f^2(e) \xleftarrow{f} f(e) = b_2$ would contradict the ordering property if the extra condition $\neg(a_1 \xleftarrow{f} a_2)$ were absent.

Lemma 3 *The paths defined by a chain-consistent, fifo, antecedent function f are ordered. (cite)*

proof: We prove this using the induction principle on paths, first on the path $a_1 \xleftarrow{f} b_1$ and then on the path $a_2 \xleftarrow{f} b_2$.

base case for path #1: If the first path begins and ends with the same event a_1 , and the second path $a_2 \xleftarrow{f} b_2$ has $a_1 <_{loc} a_2$ and $loc(b_2) = loc(a_1)$ then, since f is a retraction, $a_2 \leq_c b_2$ and hence, since

$loc(a_2) = loc(b_2)$, $a_2 \leq_{loc} b_2$ and hence $a_1 <_{loc} b_2$.

base case for path #2: If the second path begins and ends with the same event a_2 , and the first path $a_1 \xleftarrow{f} b_1$ has $a_1 <_{loc} a_2$ and $loc(b_1) = loc(a_2)$ then, by Lemma 1, all the events on the first path have the same location and if it were the case that $a_2 \leq_{loc} b_1$ then (because by (8) all the f steps move to the prior *Sys* event) the path between a_1 and b_1 must go through a_2 and this would imply $a_1 \xleftarrow{f} a_2$ contrary to assumption. Thus $b_1 <_{loc} a_2$ must hold as required.

inductive step: We may assume the ordering property for shorter paths and we have paths

$$f(a_1) \xleftarrow{f} a_1 \xleftarrow{f} b_1 \quad (path_1)$$

$$f(a_2) \xleftarrow{f} a_2 \xleftarrow{f} b_2 \quad (path_2)$$

and $f(a_1) <_{loc} f(a_2)$ and $loc(b_1) = loc(b_2)$ and $\neg(f(a_1) \xleftarrow{f} f(a_2))$ and we must show $b_1 <_{loc} b_2$.

Case $loc(f(a_2)) = loc(a_2)$: In this case, we have $f(a_2) <_{loc} a_2$ (because f is a retraction) and hence $f(a_1) <_{loc} a_2$, so the paths $f(a_1) \xleftarrow{f} b_1$ and $a_2 \xleftarrow{f} b_2$ satisfy all the same hypotheses as the paths $path_1$ and $path_2$, but the second one is shorter, so we conclude $b_1 <_{loc} b_2$ from the induction hypothesis.

Case $loc(a_1) = loc(a_2)$: In this case, we have $a_1 <_{loc} a_2$ by the fifo property of f , and again we may use the induction hypothesis on the paths $a_1 \xleftarrow{f} b_1$ and $a_2 \xleftarrow{f} b_2$ to conclude that $b_1 <_{loc} b_2$.

Case $loc(f(a_1)) = loc(a_1)$: In this case, we have $f(a_1) <_{loc} a_1$ and, by (8), $f(a_1)$ is the prior *Sys* event to a_1 . Thus, since $f(a_1) <_{loc} f(a_2)$, we must have $a_1 \leq_{loc} f(a_2)$, and in fact, $a_1 <_{loc} f(a_2)$ (because if $a_1 = f(a_2)$ then $a_1 \xleftarrow{f} a_2$ which implies $f(a_1) \xleftarrow{f} f(a_2)$ contrary to assumption). Thus the paths $a_1 \xleftarrow{f} b_1$ and $f(a_2) \xleftarrow{f} b_2$ satisfy the inductive hypotheses and we again conclude $b_1 <_{loc} b_2$.

Case $i = \text{loc}(f(a_1)) = \text{loc}(f(a_2)) \prec j = \text{loc}(a_2) \prec \text{loc}(a_1) = k$:

$$f(a_1) \xleftarrow{f} a_1 \xleftarrow{f} b_1 \quad (\text{path}_1)$$

$$f(a_2) \xleftarrow{f} a_2 \xleftarrow{f} b_2 \quad (\text{path}_2)$$

This case is impossible because, by (7), i and j are adjacent in $\text{chain}(f(a_2))$ and i and k are adjacent in $\text{chain}(f(a_1))$ but by (4) $\text{chain}(f(a_2)) \subseteq \text{chain}(f(a_1))$.

Case $i = \text{loc}(f(a_1)) = \text{loc}(f(a_2)) \prec k = \text{loc}(a_1) \prec \text{loc}(a_2) = j$:

$$f(a_1) \xleftarrow{f} a_1 \xleftarrow{f} b_1 \quad (\text{path}_1)$$

$$f(a_2) \xleftarrow{f} a_2 \xleftarrow{f} b_2 \quad (\text{path}_2)$$

In this case we will show that the path $a_1 \xleftarrow{f} b_1$ goes through location $j = \text{loc}(a_2)$, so there is an event e with $\text{loc}(e) = \text{loc}(a_2)$ and

$$a_1 \xleftarrow{f} e \xleftarrow{f} b_1$$

We claim that $e \prec_{\text{loc}} a_2$ and hence we can use the induction hypothesis on paths $e \xleftarrow{f} b_1$ and $a_2 \xleftarrow{f} b_2$ to conclude that $b_1 \prec_{\text{loc}} b_2$.

The reason that $e \prec_{\text{loc}} a_2$ is that otherwise, by (4), we would have $\text{chain}(e) \subseteq \text{chain}(a_2)$ and by (7), if $i' = \text{loc}(f(e))$, $i' \prec j$ and i' and j are adjacent in $\text{chain}(e)$. But i and j are adjacent in $\text{chain}(a_2)$ and so $i \prec i' \prec j$ is impossible.

So to finish the proof of Lemma 3 it remains to prove that the path $a_1 \xleftarrow{f} b_1$ goes through location $j = \text{loc}(a_2)$. We have $i \prec k \prec j$ and i and j are adjacent in $\text{chain}(a_2)$, so $\text{chain}(a_2)$ contains no location in the interval $I = (i, j)$ between i and j but does contain j . This means that (by (3))

(*) every configuration (i.e. every $\text{chain}(e)$ for $e \in E(\text{Sys})$) that contains any location in I also contains j .

Since the path $a_1 \xleftarrow{f} b_1$ starts at location $k \in I$ and ends at a location $l = \text{loc}(b_1)$ that is past j (i.e. $j \preceq l$), we prove (by the induction principle) that there must be a link $f(e) \xleftarrow{f} e$ with $\text{loc}(f(e)) \prec j$ and $j \preceq \text{loc}(e)$. If $\text{loc}(e) = j$ then we are done, and otherwise $\text{chain}(e)$ contains $\text{loc}(f(e)) \in I$ but does not contain j contradicting (*).

4.1 Explanation function

We can now construct the explanation function

$$expl : E(Out) \rightarrow \{e : E(In) \mid isupdate(In(e))\} List$$

which is the witness to $ApproxSM(In, Out)$ and show that it satisfies clauses 2–4 of the specification (to show that it also satisfies clause 1 we will need a further constraint). The explanation $expl(e)$ of an output event e will be the update events from the “prior fixedpoints”. The prior fixedpoints $pf(e)$ is a list defined recursively by appending the fixedpoint $f^\infty(e)$ to the prior fixedpoints of the Sys event preceding it (if any).

Definition

$$\begin{aligned} pf(e) &= \text{if } f^\infty(e) \in E(\text{prior}_{Sys}) \text{ then } pf(\text{prior}_{Sys}(f^\infty(e))).f^\infty(e) \\ &\quad \text{else } [f^\infty(e)] \\ expl(e) &= \text{filter}(\lambda e. isupdate(In(e)), pf(e)) \end{aligned}$$

Since, by (1), the fixedpoints of f are input events, the function $expl$ has type

$$expl : E(Sys) \rightarrow \{e : E(In) \mid isupdate(In(e))\} List$$

(we need the fact that f is a retraction in order to prove that the recursion terminates and $expl(e)$ is well-defined). Thus, as long as $E(Out) \subseteq E(Sys)$, $expl$ has the right type for the explanation function we need, and because f is a retraction, it satisfies clause 2 of $ApproxSM(In, Out)$. To see that it satisfies clause 3 we show

Lemma 4

$$\forall e : E(Out). \neg query(e) \Rightarrow f^\infty(e) \in expl(e)$$

proof: Since $f^\infty(e) \in pf(e)$ we only have to show $isupdate(In(f^\infty(e)))$. If not, then, by definition, $query(f^\infty(e))$, so since e is not a query $f^\infty(e) \neq e$, and hence e is not an input. Since $f^\infty(e)$ is a query, $loc(f^\infty(e)) = last(chain(f^\infty(e)))$ and this implies that $loc(e) = loc(f^\infty(e))$. Thus e satisfies $loc(f(e)) = loc(e)$ and $f(e) \neq e$, i.e. e is a resend event. Since we don't want resend events to send redundant responses we have to rule out such events also being output events. So the lemma is proved once we introduce one more constraint:

$$\forall e : E(Sys). f(e) \neq e \wedge loc(f(e)) = loc(e) \Rightarrow \neg e \in E(Out) \quad (10)$$

Lemma 5

$$\forall e, e' : E(Sys). e' \in pfp(e) \Rightarrow pfp(e') \sqsubseteq pfp(e)$$

(cite)

proof: This follows easily from the inductive definition of the list of prior fixedpoints.

To show that the explanation function $expl$ satisfies the consistency clause 4 of $ApproxSM(In, Out)$ we have to show:

Lemma 6

$$\forall e_1, e_2 : E(Out). (expl(e_1) \sqsubseteq expl(e_2) \vee expl(e_2) \sqsubseteq expl(e_1))$$

cite

proof: We will prove this by generalizing it to all events e_1 and e_2 in $E(Sys)$ that are *effective*, where e is effective if $e \xleftarrow{f} e'$ for some $e' \in E(Out)$. The effective events are the ones that eventually make it as far as the tail and generate an output. We then proceed by induction on causal order, so we may assume that the result is true for effective events causally before e_1 and e_2 . Since $expl(e) = expl(f^\infty(e))$ and $f^\infty(e) \leq_c e$ we are reduced to the case where both e_1 and e_2 are fixedpoints, and hence in $E(In)$.

If e_1 is a query, then event e_1 itself is removed from $expl(e_1)$ by the filtering operation that keeps only the update inputs, so $expl(e_1)$ is either an empty list or else equal to $expl(\text{prior}_{Sys}(e_1))$. In the first case, clearly $expl(e_1) \sqsubseteq expl(e_2)$ and in the second case we can use the induction hypothesis since $\text{prior}_{Sys}(e_1)$ is causally before e_1 . The case when e_2 is a query is proved in the same way. Thus we may assume that both e_1 and e_2 are update inputs.

Now we will show that either $e_1 \in pfp(e_2)$ or $e_2 \in pfp(e_1)$ and hence by Lemma 5 the explanation of one is a prefix of the other.

Case $loc(e_1) = loc(e_2)$: In this case either $e_1 <_{loc} e_2$, $e_2 <_{loc} e_1$, or $e_1 = e_2$. The third case is trivial, and the other two are covered by the following lemma

Lemma 7 For e_1 an update input and $e_2 \in E(Sys)$

$$e_1 \leq_{loc} e_2 \Rightarrow e_1 \in pfp(e_2)$$

proof of lemma 7: By causal induction on e_2 , if $loc(f(e_2)) = loc(e_2)$ then the result follows from the definition of $pfp(e_2)$ and the induction hypothesis. But it must be the case that $i = loc(f(e_2)) = loc(e_2) = j$ for otherwise i is adjacent to j in $chain(e_2) \subseteq chain(e_1)$ but, by (6), $j = head(chain(e_1))$ because e_1 is an update input .

Case $loc(e_1) \neq loc(e_2)$: In this case, without loss of generality we may assume that

$$loc(e_1) \prec loc(e_2)$$

In this case we will show $e_1 \in pfp(e_2)$. To review the assumptions for this case, we have e_1 and e_2 are both effective update input events and $loc(e_1) \prec loc(e_2)$. Both e_1 and e_2 , trivially, satisfy the following property P_1 :

$$P_1(e) \equiv \exists b : E(Sys). e \leq_{loc} b \wedge isupdate(In(b))$$

By the same reasoning used in the proof of Lemma 7, events e_1 and e_2 both satisfy the following property P_2 :

$$P_2(e) \equiv \forall b : E(Sys). e \leq_{loc} b \Rightarrow loc(f(b)) = loc(b)$$

We will use these properties as inductive hypotheses and we will be finished once we establish the following

Lemma 8 *For any effective update input a and any input $b \in E(In)$*

$$(P_1(b) \wedge P_2(b) \wedge loc(a) \prec loc(b)) \Rightarrow a \in pfp(b)$$

proof: By causal induction on b , we may assume the lemma holds for any b' causally before b , and our first step is to show that b must have a locally prior Sys event. The reason for this is that since a is effective, there is a path $a \xleftarrow{f} e$ to some output event e , and because there is an update input at the location of b (by $P_1(b)$), by Lemma 2 the path from a to e must go through $loc(b)$. (To derive this from Lemma 2 we need to know that $loc(b) \preceq loc(e)$ but this follows easily from chain-consistency and the fact that there is an update input at $loc(b)$ and an output at $loc(e)$.)

Thus there is an event e' with $loc(e') = loc(b)$ such that $a \xleftarrow{f} e' \xleftarrow{f} e$ and since $loc(a) \prec loc(b)$ we may (by choosing the first e' on the path at location $loc(b)$) assume that $loc(f(e')) \neq loc(e')$. But then by property $P_2(b)$, we must have $e' <_{loc} b$ and hence b has a prior Sys event and we may let $b' = prior_{Sys}(b)$. The event e' on the path $a \xleftarrow{f} e' \xleftarrow{f} e$ is then

$e' \leq_{loc} b'$. Let $v = f^\infty(b')$. We will show that $a \in \text{pfp}(v)$ (and hence $a \in \text{pfp}(b)$ since $\text{pfp}(v) \sqsubseteq \text{pfp}(b)$). Note that v is causally before b .

There are three cases depending on the relative order of $loc(a)$ and $loc(v)$ in the chain ordering \prec .

Case $loc(a) \prec loc(v)$:

Event v is a fixedpoint, so it is an input event. If it is an update input, then v satisfies the induction hypothesis and we conclude that $a \in \text{pfp}(v)$. If it is a query, then $loc(v) = loc(b')$ and hence $v \leq_{loc} b'$ and $v \prec_{loc} b$. In this case v also satisfies properties $P_1(v)$ and $P_2(v)$, so again it satisfies the induction hypothesis.

Case $loc(v) = loc(a)$:

In this case we claim that $a \leq_{loc} v$ and hence, by Lemma 7, $a \in \text{pfp}(v)$. The claim follows from the ordering property proved in Lemma 3 because if $v \prec_{loc} a$ then $path_1 = v \xleftarrow{f} b'$ and $path_2 = a \xleftarrow{f} e'$ contradict the ordering conditions because $e' \leq_{loc} b'$.

Case $loc(v) \prec loc(a)$:

In this case we again use Lemma 2 to deduce that the path from v to b' goes through $loc(a)$, so there is an event e'' with $loc(e'') = loc(a)$ and $v \xleftarrow{f} e'' \xleftarrow{f} b'$, and we may assume that $loc(f(e'')) \neq loc(e'')$. If $e'' \prec_{loc} a$ then the paths $path_1 = e'' \xleftarrow{f} b'$ and $path_2 = a \xleftarrow{f} e'$ contradict the path ordering property. Thus $a \leq_{loc} e''$. But since a is an update input, there cannot be an e'' after a with $loc(f(e'')) \neq loc(e'')$, so this case is impossible, and we are finished with the proof of Lemma 8 and also Lemma 6

5 Constraints on Output

We will next introduce the abstract constraints needed to derive the first clause of $ApproxSM(In, Out)$, which is that for every $e \in E(Out)$:

$$Out(e) = \mathbf{if} \text{ query}(e) \mathbf{ then } Q(\vec{In}(expl(e)), In(e)) \mathbf{ else } \Delta(\vec{In}(expl(e)))$$

The machines at each location will only be able to generate responses that are a function of their *local state*, but as long as we constrain the response to be a function of the *local history* of events, we can relate the local history to a local state variable in a later refinement step. So let us now define the abstract local history in terms of the interface Sys of type $Cmd List$.

Definition The *local history*, $hist(e)$, is the concatenation of the list of

lists

$$[Sys(e') \mid (e' \leq_{loc} e) \wedge e' \in E(Sys)]$$

The *local update history*, $uhist(e)$, is

$$filter(isupdate, hist(e))$$

The *prior update history*, $priorhist(e)$, is

$$\mathbf{if} \ e \in E(\text{prior}_{Sys}) \ \mathbf{then} \ uhist(\text{prior}_{Sys}(e)) \ \mathbf{else} \ \text{nil}$$

The output constraint is: $\forall e : E(Out)$.

$$Out(e) = \mathbf{if} \ query(e) \ \mathbf{then} \ Q(uhist(e), In(e)) \ \mathbf{else} \ \Delta(uhist(e)) \quad (11)$$

To prove the final clause of $ApproxSM(In, Out)$ it will clearly be sufficient to prove the following invariant that relates the local update history to the explanation function:

Lemma 9

$$\forall e : E(Sys). \ uhist(e) = \vec{In}(expl(e))$$

To prove this invariant, we need constraints on how values are forwarded along the chain.

6 Forwarding Constraints

Definition Event e is the (receipt of the) *first message* (from its current predecessor) if $\text{firstMsg}(e) \equiv$

$$\forall e' : E(Sys). \ e' <_{loc} e \Rightarrow loc(f(e')) \neq loc(f(e))$$

Definition Event e *did forward* if it is the cause of an event at another location:

$$\text{did-forward}(e) \equiv \exists a : E(Sys). \ e = f(a) \wedge loc(a) \neq loc(e)$$

And e *should forward* if it is an update input or a receipt of a forwarded command.

$$\text{should-forward}(e) \equiv (e \in E(In) \wedge isupdate(In(e))) \vee loc(f(e)) \neq loc(e)$$

Definition The antecedent function f is *command forwarding* if, for all $e \in E(Sys)$:

$$e \in E(In) \Rightarrow Sys(e) = [In(e)] \quad (12)$$

$$(\neg(e \in E(In)) \wedge loc(f(e)) = loc(e)) \Rightarrow Sys(e) = nil \quad (13)$$

$$(loc(f(e)) \neq loc(e) \wedge firstMsg(e)) \Rightarrow \\ Sys(e) = nthTail(length(priorhist(e)), uhist(f(e))) \quad (14)$$

$$(loc(f(e)) \neq loc(e) \wedge \neg firstMsg(e)) \Rightarrow \\ Sys(e) = Sys(f(e)) \quad (15)$$

$$did-forward(e) \Rightarrow \\ \forall a : E(Sys). (a <_{loc} e \wedge should-forward(a)) \Rightarrow \\ did-forward(a) \quad (16)$$

Remarks: Clause (12) expresses our earlier remark that we think of an input as coding a singleton list of commands. Clause (13) says that resend events will not add anything to the local history. Clause (14) says that the value of the first message sent between two locations will attempt to make the local history of the sender and receiver the same by sending a tail of the sender's history such that the receiver ends up with a history of the same length (we will prove that this is, in fact, enough to make the invariant in Lemma 9 true). Clause (15) says that later messages between two locations merely forward the current commands. Finally, clause (16) is a weak abstract *fail stop* condition that says that if one event at a location successfully forwarded its command, then all prior events that should have forwarded did also forward. This means that once a node fails to forward a command, it will not forward some later command.

7 Proof of the Invariant

Assuming now that the events in $E(Sys)$ and the antecedent function f satisfy the command forwarding constraints (12) through (16), we can prove Lemma 9, the invariant:

$$\forall e : E(Sys). uhist(e) = \vec{In}(expl(e))$$

Proof: By causal induction on e , we assume the invariant holds for all events in $E(Sys)$ causally before e .

Case $f(e) = e$: In this case e is an input event, and by induction $uhist(\text{prior}_{Sys}(e)) = \vec{In}(\text{expl}(\text{prior}_{Sys}(e)))$. Unfolding the definitions and using (12) to get $Sys(e) = [In(e)]$, we see that $uhist(e) = \vec{In}(\text{expl}(e))$ because each side of the equation merely appends $In(e)$ or nil to the inductive case depending on whether e is a query or an update.

Case $f(e) \neq e$: In this case $f(e) <_c e$ (because f is a retraction) so $uhist(f(e)) = \vec{In}(\text{expl}(f(e)))$ by the induction hypothesis. But $\vec{In}(\text{expl}(e)) = \vec{In}(\text{expl}(f(e)))$ so it will be enough to show that

$$uhist(e) = uhist(f(e)) \quad (\text{goal})$$

This will certainly be true if $loc(f(e)) = loc(e)$ because then e is a resend event and, by (13), $Sys(e) = \text{nil}$ and, by (8), $f(e) = \text{prior}_{Sys}(e)$. So we may assume that e is a receipt, i.e. that

$$loc(f(e)) \neq loc(e)$$

Case $\text{firstMsg}(e)$: By (14) we have

$$Sys(e) = nthTail(\text{length}(\text{priorhist}(e)), uhist(f(e)))$$

If there is no Sys event prior to e , then the goal is true because then $\text{length}(\text{priorhist}(e)) = 0$ and so $Sys(e) = uhist(f(e))$ and

$$uhist(e) = filter(isupdate, Sys(e)) = filter(isupdate, uhist(f(e))) = uhist(f(e))$$

So let $p = \text{prior}_{Sys}(e)$, then (using $+$ to mean append), and using the induction hypothesis on both e' and $f(e)$,

$$\begin{aligned} uhist(e) &= uhist(p) + filter(isupdate, Sys(e)) \\ &= \vec{In}(\text{expl}(p)) + nthTail(\text{length}(uhist(p)), uhist(f(e))) \\ &= \vec{In}(\text{expl}(p)) + nthTail(\text{length}(\vec{In}(\text{expl}(p))), uhist(f(e))) \\ &= \vec{In}(\text{expl}(p)) + nthTail(\text{length}(\vec{In}(\text{expl}(p))), \vec{In}(\text{expl}(f(e)))) \\ (\text{claim}) &= \vec{In}(\text{expl}(f(e))) \\ &= uhist(f(e)) \end{aligned}$$

To prove the claim, note that if $\text{expl}(p) = \text{nil}$ the claim is trivial, and otherwise there must be an event e' before e with $loc(f(e')) \neq loc(e')$ because,

since e is a receipt, there can be no update inputs prior to e . For the last such e' we have

$$\text{expl}(p) = \text{expl}(e')$$

and to prove the claim it is enough to show that $\vec{In}(\text{expl}(e')) \subseteq \vec{In}(\text{expl}(f(e)))$

To show that, we will be done if we can show that there is an event a with $a \leq_{loc} f(e)$ and $a \xleftarrow{f} e'$ because then

$$\vec{In}(\text{expl}(e')) = \vec{In}(\text{expl}(a)) = \text{uhist}(a) \subseteq \text{uhist}(f(e)) = \vec{In}(\text{expl}(f(e)))$$

Because of the ordering property, Lemma 3, it is enough to show that there is an a with $a \xleftarrow{f} e'$ and $\text{loc}(a) = \text{loc}(f(e))$, i.e. that the path $f^\infty(e') \xleftarrow{f} e'$ goes through location $j = \text{loc}(f(e))$.

Let $v = f^\infty(e')$ and $i = \text{loc}(v)$ and consider the relative order of i and j . If $i=j$ we are done, since we can take $a=v$. If $j \prec i$ then we have

$$j \prec i \prec \text{loc}(e')$$

and v must be an update input at location j . Then by Lemma 2, the path $f(e) \xleftarrow{f} e$ would have to go through location j , but this is impossible since $f(e)$ and e have to be at adjacent locations.

Thus, we must have $i \prec j$, and the claim follows from the following lemma:

Lemma 10 *For all $x, y, z \in E(\text{Sys})$, if $x \xleftarrow{f} y$ and $y \leq_{loc} z$ and $\text{loc}(x) \preceq \text{loc}(f(z))$, then the path $x \xleftarrow{f} y$ goes through location $\text{loc}(f(z))$.*

proof of lemma 10: By induction on the path $x \xleftarrow{f} y$. The base case is trivial. So suppose $f(x) \xleftarrow{f} x \xleftarrow{f} y$, $y \leq_{loc} z$, and $\text{loc}(f(x)) \prec \text{loc}(f(z))$. If $\text{loc}(x) \preceq \text{loc}(y)$ then by induction, then path $x \xleftarrow{f} y$ goes through $\text{loc}(f(z))$ and hence so does the path $f(x) \xleftarrow{f} y$. Thus assume that $\text{loc}(f(x)) \prec \text{loc}(y) \prec \text{loc}(x)$. If $\text{loc}(x) = \text{loc}(z)$ then we get a contradiction from the adjacency of $\text{loc}(f(x))$ and $\text{loc}(z)$ in $\text{chain}(x)$ and the adjacency of $\text{loc}(f(z))$ and $\text{loc}(z)$ in $\text{chain}(z) \subseteq \text{chain}(x)$. If $\text{loc}(f(z)) \prec \text{loc}(x) \prec \text{loc}(z)$ then any configuration that contains $\text{loc}(x)$ must also contain $\text{loc}(f(z))$ so $\text{loc}(f(z)) \in \text{chain}(x)$. But $\text{loc}(f(x))$ and $\text{loc}(x)$ are adjacent in $\text{chain}(x)$ so this is impossible.

Back to the proof of Lemma 9, the last case is:

Case \neg firstMsg(e) : In this case there is an event before e whose sender has the same location as the sender of e , so let e' be the last such event. So, $e' <_{loc} e$ and $loc(f(e')) = loc(f(e))$ and this is not the case for any events between e' and e . Then, in fact, all the events a between e' and e must have $loc(f(a)) = loc(a)$ (this follows easily from chain-consistency-by induction again). Thus all the events between e' and e are either resend events that have $Sys(a) = \text{nil}$ or are input events. But if they are input events, then they must be queries, since after an update input there can be no more receipt events like e . Thus,

$$\begin{aligned}
uhist(e) &= uhist(e') + Sys(e) \\
&= uhist(e') + Sys(f(e)) && \text{by (15)} \\
&= \vec{In}(expl(e')) + Sys(f(e)) && \text{by induction} \\
&= \vec{In}(expl(f(e'))) + Sys(f(e)) && \text{by def of expl} \\
&= uhist(f(e')) + Sys(f(e)) && \text{by induction}
\end{aligned}$$

Recall that our goal is to show that $uhist(e) = uhist(f(e))$, so we have only to show that

$$uhist(f(e')) + Sys(f(e)) = uhist(f(e))$$

Since $f(e') <_{loc} f(e)$ follows from the fifo property of f , this final equation follows once we show that for all events a strictly between $f(e')$ and $f(e)$, $filter(isupdate, Sys(a)) = \text{nil}$. Note that we have not yet used the fail-stop clause (16)—that will be needed for the proof of this claim. Suppose $f(e') <_{loc} a <_{loc} f(e)$ and that $filter(isupdate, Sys(a)) \neq \text{nil}$, then a is either an update input or a receipt, so we have should-forward(a). Since, by definition, did-forward($f(e)$), by clause (16), we have did-forward(a). By definition, this implies that there is an event b such that $a = f(b)$. Since both $f(e')$ and $f(e)$ are sending to the same location and a is between them, it follows from chain-consistency, that a is also sending to that location (the location of e and e'). Then from the fifo property we deduce that $e' <_{loc} b <_{loc} e$ but this contradicts our choice of e' as the last event prior to e for which $loc(f(e')) = loc(f(e))$.

This finishes the proof of Lemma 9 and establishes the invariant we needed to prove the first clause of the *ApproxSM(In, Out)* specification.

8 Abstract Chain Replication

We gather up all the constraints needed to prove the preceding lemmas so that we can state the theorem that has been proved.

Definition With parameters Cmd , Rsp , $isupdate$, Δ , and Q , the antecedent function f and interfaces Sys , In , and Out satisfy

$$AbstractChainRepl(Sys, f, In, Out)$$

if

1. $E(In) \subseteq E(Sys) \wedge E(Out) \subseteq E(Sys)$
2. $\forall e : E(Sys). e \in E(In) \Leftrightarrow f(e) = e$ (clause (1)).
3. f is chain-consistent, (clauses (2)-(8)).
4. f is fifo, (clause (9)).
5. $\forall e : E(Sys). f(e) \neq e \wedge loc(f(e)) = loc(e) \Rightarrow \neg e \in E(Out)$
6. $Out(e) = \mathbf{if} \text{ query}(e) \mathbf{ then } Q(\text{uhist}(e), In(e)) \mathbf{ else } \Delta(\text{uhist}(e))$
7. f is command-forwarding, (clauses (12)-(16)).
8. $\forall e : E(In). \neg isupdate(In(e)) \Rightarrow e \in E(Out)$

Remarks: All of these clauses have been introduced earlier and used in the proofs of the various lemmas except for the first clause $E(In) \subseteq E(Sys) \wedge E(Out) \subseteq E(Sys)$ which we have used implicitly, and the last clause, which may not be needed, but which is part of the definition used in the NuPrl proof of the following theorem:

Theorem 11 $AbstractChainRepl(Sys, f, In, Out) \Rightarrow ApproxSM(In, Out)$

proof: The proof has been carried out in the NuPrl theorem proving environment by essentially the same steps used in the above lemmas.