

# The IMAP CmRDT

Tim Jungnickel, Lennart Oldenburg, Matthias Loibl

June 16, 2019

## Abstract

We provide our Isabelle/HOL formalization of a Conflict-free Replicated Data Type for Internet Message Access Protocol commands. To this end, we show that Strong Eventual Consistency (SEC) is guaranteed by proving the commutativity of concurrent operations. We base our formalization on the recently proposed "framework for establishing Strong Eventual Consistency for Conflict-free Replicated Datatypes" (AFP.CRDT) by Gomes et al. Hence, we provide an additional example of how the recently proposed framework can be used to design and prove CRDTs.

## Contents

<b>1</b>	<b>Preface</b>	<b>1</b>
1.1	The IMAP CmRDT . . . . .	2
1.2	Proof Guide . . . . .	2
<b>2</b>	<b>IMAP-CRDT Definitions</b>	<b>3</b>
<b>3</b>	<b>Commutativity of IMAP Commands</b>	<b>5</b>
<b>4</b>	<b>Proof Helpers</b>	<b>7</b>
<b>5</b>	<b>Independence of IMAP Commands</b>	<b>11</b>
<b>6</b>	<b>Convergence of the IMAP-CRDT</b>	<b>23</b>

## 1 Preface

A Conflict-free Replicated Data Type (CRDT) [5] ensures convergence of replicas without requiring a central coordination server or even a distributed coordination system based on consensus or locking. Despite the fact that Shapiro et al. provide a comprehensive collection of definitions for the most useful data types such as registers, sets, and lists [4], we observe that the use of CRDTs in standard IT services is rather uncommon. Therefore, we use the Internet Message Access Protocol (IMAP)—the de-facto standard protocol to retrieve and manipulate mail messages on an email server—as an example to show the feasibility of using CRDTs for replicating state of a standard IT service to achieve planetary scale.

Designing a *correct* CRDT is a challenging task. A CmRDT, the operation-based variant of a CRDT, requires all operations to commute. To this end, Gomes et al. recently published a CmRDT verification framework [1] in Isabelle/HOL.

In our most recent work [3], we presented *pluto*, our research prototype of a planetary-scale IMAP service. To achieve the claimed planet-scale, we designed a CmRDT that provides multi-leader replication of mailboxes without the need of synchronous operations. In order to ensure the correctness of our proposed IMAP CmRDT, we implemented it in the verification framework proposed by Gomes et al.

In this work, we present our Isabelle/HOL proof of the necessary properties and show that our CmRDT indeed guarantees Strong Eventual Consistency (SEC). We contribute not only the certainty that our CmRDT design is correct, but also provide one more example of how the verification framework can be used to prove the correctness of a CRDT.

## 1.1 The IMAP CmRDT

In the rest of this work, we show how we modeled our IMAP CmRDT in Isabelle/HOL. We start by presenting the original IMAP CmRDT, followed by the implementation details of the Isabelle/HOL formalization. The presentation of our CmRDT in Spec. 1 is based on the syntax introduced in [4]. We highly recommend reading the foundational work by Shapiro et al. prior to following our proof documentation.

In essence, the IMAP CmRDT represents the state of a mailbox, containing folders (of type  $\mathcal{N}$ ) and messages (of type  $\mathcal{M}$ ). Moreover, we introduce metadata in form of tags (of type ID). All modeling details and a more detailed description of the CmRDT are provided in the original paper [3].

The only notable difference between the presented specification and our Isabelle/HOL formalization is, that we no longer distinguish between sets ID and  $\mathcal{M}$  and that the generated tags of *create* and *expunge* are handled explicitly. This makes the formalization slightly easier, because less type variables are introduced. The concrete definition can be found in the *IMAP-CRDT Definitions* section of the `IMAP-def.thy` file.

## 1.2 Proof Guide

*Hint:* In our proof, we build on top of the definitions given by Gomes et al. in [2]. We strongly recommend to read their paper first before following our proof. In fact, in our formalization we reuse the *locales* of the proposed framework and therefore this work cannot be compiled without the reference to [1].

Operation-based CRDTs require all concurrent operations to commute in order to ensure convergence. Therefore, we begin our verification by proving the commutativity of every combination of possible concurrent operations. Initially, we used *nitpick* to identify corner cases in our implementation. We prove the commutativity in Section 3 of the `IMAP-proof-commute.thy` file. The *critical conditions* to satisfy in order to commute, can be summarized as follows:

- The tags of a *create* and *expunge* operation or the messages of an *append* and *store* operation are never in the removed-set of a concurrent *delete* operation.
- The message of an *append* operation is never the message that is deleted by a concurrent *store* or *expunge* operation.
- The message inserted by a *store* operation is never the message that is deleted by a concurrent *store* or *expunge* operation.

The identified conditions obviously hold in regular traces of our system, because an item that has been inserted by one operation cannot be deleted by a concurrent operation. It simply cannot be present at the time of the initiation of the concurrent operation.

---

**Specification 1** The IMAP CmRDT

---

```
1: payload map  $u : \mathcal{N} \rightarrow \mathcal{P}(\text{ID}) \times \mathcal{P}(\mathcal{M})$   $\triangleright \{\text{foldername } f \mapsto (\{\text{tag } t\}, \{\text{msg } m\}), \dots\}$ 
2:   initial  $(\lambda x.(\emptyset, \emptyset))$ 
3: update create (foldername  $f$ )
4:   atSource
5:     let  $\alpha = \text{unique}()$ 
6:   downstream  $(f, \alpha)$ 
7:      $u(f) \mapsto (u(f)_1 \cup \{\alpha\}, u(f)_2)$ 
8: update delete (foldername  $f$ )
9:   atSource  $(f)$ 
10:    let  $R_1 = u(f)_1$ 
11:    let  $R_2 = u(f)_2$ 
12:  downstream  $(f, R_1, R_2)$ 
13:     $u(f) \mapsto (u(f)_1 \setminus R_1, u(f)_2 \setminus R_2)$ 
14: update append (foldername  $f$ , message  $m$ )
15:   atSource  $(m)$ 
16:   pre  $m$  is globally unique
17:   downstream  $(f, m)$ 
18:      $u(f) \mapsto (u(f)_1, u(f)_2 \cup \{m\})$ 
19: update expunge (foldername  $f$ , message  $m$ )
20:   atSource  $(f, m)$ 
21:   pre  $m \in u(f)_2$ 
22:   let  $\alpha = \text{unique}()$ 
23:   downstream  $(f, m, \alpha)$ 
24:      $u(f) \mapsto (u(f)_1 \cup \{\alpha\}, u(f)_2 \setminus \{m\})$ 
25: update store (foldername  $f$ , message  $m_{\text{old}}$ , message  $m_{\text{new}}$ )
26:   atSource  $(f, m_{\text{old}}, m_{\text{new}})$ 
27:   pre  $m_{\text{old}} \in u(f)_2$ 
28:   pre  $m_{\text{new}}$  is globally unique
29:   downstream  $(f, m_{\text{old}}, m_{\text{new}})$ 
30:      $u(f) \mapsto (u(f)_1, (u(f)_2 \setminus \{m_{\text{old}}\}) \cup \{m_{\text{new}}\})$ 
```

---

Next, we show that the identified conditions actually hold for all concurrent operations. Because all tags and all inserted messages are globally unique, it can easily be shown that all conditions are satisfied. In Isabelle/HOL, showing this fact takes some effort. Fortunately, we were able to reuse parts of the Isabelle/HOL implementation of the OR-Set proof in [1]. The Isabelle/HOL proofs for the *critical conditions* are encapsulated in the `IMAP-proof-independent.thy` file.

With the introduced lemmas, we prove the final theorem that states that convergence is guaranteed. Due to all operations being commutative in case the *critical conditions* are satisfied and the *critical conditions* indeed are holding for all concurrent updates, all concurrent operations commute. The Isabelle/HOL proof is contained in the `IMAP-proof.thy` file.

## 2 IMAP-CRDT Definitions

We begin by defining the operations on a mailbox state. In addition to the interpretation of the operations, we define valid behaviours for the operations as assumptions for the network. We

use the `network_with_constrained_ops` locale from the framework.

**theory**

*IMAP-def*

**imports**

*CRDT.Network*

**begin**

**datatype** (*'id*, *'a*) *operation* =

*Create 'id 'a* |  
*Delete 'id set 'a* |  
*Append 'id 'a* |  
*Expunge 'a 'id 'id* |  
*Store 'a 'id 'id*

**type-synonym** (*'id*, *'a*) *state* = *'a*  $\Rightarrow$  (*'id set*  $\times$  *'id set*)

**definition** *op-elem* :: (*'id*, *'a*) *operation*  $\Rightarrow$  *'a* **where**

*op-elem oper*  $\equiv$  *case oper of*  
*Create i e*  $\Rightarrow$  *e* |  
*Delete is e*  $\Rightarrow$  *e* |  
*Append i e*  $\Rightarrow$  *e* |  
*Expunge e mo i*  $\Rightarrow$  *e* |  
*Store e mo i*  $\Rightarrow$  *e*

**definition** *interpret-op* :: (*'id*, *'a*) *operation*  $\Rightarrow$  (*'id*, *'a*) *state*  $\rightarrow$  (*'id*, *'a*) *state*  
( $\langle \cdot \rangle$ ) [0] 1000) **where**

*interpret-op oper state*  $\equiv$   
*let metadata* = *fst (state (op-elem oper))*;  
*files* = *snd (state (op-elem oper))*;  
*after* = *case oper of*  
*Create i e*  $\Rightarrow$  (*metadata*  $\cup$   $\{i\}$ , *files*) |  
*Delete is e*  $\Rightarrow$  (*metadata* - *is*, *files* - *is*) |  
*Append i e*  $\Rightarrow$  (*metadata*, *files*  $\cup$   $\{i\}$ ) |  
*Expunge e mo i*  $\Rightarrow$  (*metadata*  $\cup$   $\{i\}$ , *files* -  $\{mo\}$ ) |  
*Store e mo i*  $\Rightarrow$  (*metadata*, *insert i (files - {mo})*)  
*in Some (state ((op-elem oper) := after))*

In the definition of the valid behaviours of the operations, we define additional assumption the state where the operation is executed. In essence, a the tag of a *create*, *append*, *expunge*, and *store* operation is identical to the message number and therefore unique. A *delete* operation deletes all metadata and the content of a folder. The *store* and *expunge* operations must refer to an existing message.

**definition** *valid-behaviours* :: (*'id*, *'a*) *state*  $\Rightarrow$  *'id*  $\times$  (*'id*, *'a*) *operation*  $\Rightarrow$  *bool* **where**

*valid-behaviours state msg*  $\equiv$   
*case msg of*  
(*i*, *Create j e*)  $\Rightarrow$  *i = j* |  
(*i*, *Delete is e*)  $\Rightarrow$  *is = fst (state e)  $\cup$  snd (state e)* |  
(*i*, *Append j e*)  $\Rightarrow$  *i = j* |  
(*i*, *Expunge e mo j*)  $\Rightarrow$  *i = j  $\wedge$  mo  $\in$  snd (state e)* |

$$(i, \text{Store } e \text{ mo } j) \Rightarrow i = j \wedge \text{mo} \in \text{snd} (\text{state } e)$$

**locale** *imap* = *network-with-constrained-ops* - *interpret-op*  $\lambda x. (\{\}, \{\})$  *valid-behaviours*

**end**

### 3 Commutativity of IMAP Commands

In this section we prove the commutativity of operations and identify the edge cases.

**theory**

*IMAP-proof-commute*

**imports**

*IMAP-def*

**begin**

**lemma** (**in** *imap*) *create-create-commute*:

**shows**  $\langle \text{Create } i1 \ e1 \rangle \triangleright \langle \text{Create } i2 \ e2 \rangle = \langle \text{Create } i2 \ e2 \rangle \triangleright \langle \text{Create } i1 \ e1 \rangle$

**by**(*auto simp add: interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (**in** *imap*) *create-delete-commute*:

**assumes**  $i \notin is$

**shows**  $\langle \text{Create } i \ e1 \rangle \triangleright \langle \text{Delete } is \ e2 \rangle = \langle \text{Delete } is \ e2 \rangle \triangleright \langle \text{Create } i \ e1 \rangle$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def, fastforce*)

**lemma** (**in** *imap*) *create-append-commute*:

**shows**  $\langle \text{Create } i1 \ e1 \rangle \triangleright \langle \text{Append } i2 \ e2 \rangle = \langle \text{Append } i2 \ e2 \rangle \triangleright \langle \text{Create } i1 \ e1 \rangle$

**by**(*auto simp add: interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (**in** *imap*) *create-expunge-commute*:

**shows**  $\langle \text{Create } i1 \ e1 \rangle \triangleright \langle \text{Expunge } e2 \ \text{mo } i2 \rangle = \langle \text{Expunge } e2 \ \text{mo } i2 \rangle \triangleright \langle \text{Create } i1 \ e1 \rangle$

**by**(*auto simp add: interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (**in** *imap*) *create-store-commute*:

**shows**  $\langle \text{Create } i1 \ e1 \rangle \triangleright \langle \text{Store } e2 \ \text{mo } i2 \rangle = \langle \text{Store } e2 \ \text{mo } i2 \rangle \triangleright \langle \text{Create } i1 \ e1 \rangle$

**by**(*auto simp add: interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (**in** *imap*) *delete-delete-commute*:

**shows**  $\langle \text{Delete } i1 \ e1 \rangle \triangleright \langle \text{Delete } i2 \ e2 \rangle = \langle \text{Delete } i2 \ e2 \rangle \triangleright \langle \text{Delete } i1 \ e1 \rangle$

**by**(*unfold interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (**in** *imap*) *delete-append-commute*:

**assumes**  $i \notin is$

**shows**  $\langle \text{Delete } is \ e1 \rangle \triangleright \langle \text{Append } i \ e2 \rangle = \langle \text{Append } i \ e2 \rangle \triangleright \langle \text{Delete } is \ e1 \rangle$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def, fastforce*)

**lemma** (**in** *imap*) *delete-expunge-commute*:

**assumes**  $i \notin is$

**shows**  $\langle \text{Delete } is \ e1 \rangle \triangleright \langle \text{Expunge } e2 \ \text{mo } i \rangle = \langle \text{Expunge } e2 \ \text{mo } i \rangle \triangleright \langle \text{Delete } is \ e1 \rangle$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def, fastforce*)

**lemma** (*in imap*) *delete-store-commute*:

**assumes**  $i \notin is$

**shows**  $\langle Delete\ is\ e1 \rangle \triangleright \langle Store\ e2\ mo\ i \rangle = \langle Store\ e2\ mo\ i \rangle \triangleright \langle Delete\ is\ e1 \rangle$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def, fastforce*)

**lemma** (*in imap*) *append-append-commute*:

**shows**  $\langle Append\ i1\ e1 \rangle \triangleright \langle Append\ i2\ e2 \rangle = \langle Append\ i2\ e2 \rangle \triangleright \langle Append\ i1\ e1 \rangle$

**by**(*auto simp add: interpret-op-def op-elem-def kleisli-def, fastforce*)

**lemma** (*in imap*) *append-expunge-commute*:

**assumes**  $i1 \neq mo$

**shows**  $(\langle Append\ i1\ e1 \rangle \triangleright \langle Expunge\ e2\ mo\ i2 \rangle) = (\langle Expunge\ e2\ mo\ i2 \rangle \triangleright \langle Append\ i1\ e1 \rangle)$

**proof**

**fix**  $x$

**show**  $(\langle Append\ i1\ e1 \rangle \triangleright \langle Expunge\ e2\ mo\ i2 \rangle) x = (\langle Expunge\ e2\ mo\ i2 \rangle \triangleright \langle Append\ i1\ e1 \rangle) x$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def*)

**qed**

**lemma** (*in imap*) *append-store-commute*:

**assumes**  $i1 \neq mo$

**shows**  $(\langle Append\ i1\ e1 \rangle \triangleright \langle Store\ e2\ mo\ i2 \rangle) = (\langle Store\ e2\ mo\ i2 \rangle \triangleright \langle Append\ i1\ e1 \rangle)$

**proof**

**fix**  $x$

**show**  $(\langle Append\ i1\ e1 \rangle \triangleright \langle Store\ e2\ mo\ i2 \rangle) x = (\langle Store\ e2\ mo\ i2 \rangle \triangleright \langle Append\ i1\ e1 \rangle) x$

**using** *assms* **by**(*auto simp add: interpret-op-def kleisli-def op-elem-def*)

**qed**

**lemma** (*in imap*) *expunge-expunge-commute*:

**shows**  $(\langle Expunge\ e1\ mo1\ i1 \rangle \triangleright \langle Expunge\ e2\ mo2\ i2 \rangle) = (\langle Expunge\ e2\ mo2\ i2 \rangle \triangleright \langle Expunge\ e1\ mo1\ i1 \rangle)$

**proof**

**fix**  $x$

**show**  $(\langle Expunge\ e1\ mo1\ i1 \rangle \triangleright \langle Expunge\ e2\ mo2\ i2 \rangle) x$

$= (\langle Expunge\ e2\ mo2\ i2 \rangle \triangleright \langle Expunge\ e1\ mo1\ i1 \rangle) x$

**by**(*auto simp add: interpret-op-def kleisli-def op-elem-def*) **qed**

**lemma** (*in imap*) *expunge-store-commute*:

**assumes**  $i1 \neq mo2$  **and**  $i2 \neq mo1$

**shows**  $(\langle Expunge\ e1\ mo1\ i1 \rangle \triangleright \langle Store\ e2\ mo2\ i2 \rangle) = (\langle Store\ e2\ mo2\ i2 \rangle \triangleright \langle Expunge\ e1\ mo1\ i1 \rangle)$

**proof**

**fix**  $x$

**show**  $(\langle Expunge\ e1\ mo1\ i1 \rangle \triangleright \langle Store\ e2\ mo2\ i2 \rangle) x = (\langle Store\ e2\ mo2\ i2 \rangle \triangleright \langle Expunge\ e1\ mo1\ i1 \rangle) x$

**unfolding** *interpret-op-def kleisli-def op-elem-def* **using** *assms(2)* **by** (*simp, fastforce*)

**qed**

```

lemma (in imap) store-store-commute:
  assumes i1 ≠ mo2 and i2 ≠ mo1
  shows (⟨Store e1 mo1 i1⟩ ▷ ⟨Store e2 mo2 i2⟩) = (⟨Store e2 mo2 i2⟩ ▷ ⟨Store e1 mo1 i1⟩)
proof
  fix x
  show (⟨Store e1 mo1 i1⟩ ▷ ⟨Store e2 mo2 i2⟩) x = (⟨Store e2 mo2 i2⟩ ▷ ⟨Store e1 mo1 i1⟩)
x
  unfolding interpret-op-def kleisli-def op-elem-def using assms by (simp, fastforce)
qed

end

```

## 4 Proof Helpers

In this section we define and prove lemmas that help to show that all identified critical conditions hold for concurrent operations. Many of the following parts are derivations from the definitions and lemmas of Gomes et al.

```

theory
  IMAP-proof-helpers
imports
  IMAP-def

begin

lemma (in imap) apply-operations-never-fails:
  assumes xs prefix of i
  shows apply-operations xs ≠ None
  using assms proof(induction xs rule: rev-induct, clarsimp)
  case (snoc x xs) thus ?case
proof (cases x)
  case (Broadcast e) thus ?thesis
    using snoc by force
next
  case (Deliver e) thus ?thesis
    using snoc apply clarsimp unfolding interp-msg-def apply-operations-def
    by (metis (no-types, lifting) bind.bind-lunit interpret-op-def prefix-of-appendD)
qed
qed

```

```

lemma (in imap) create-id-valid:
  assumes xs prefix of j
  and Deliver (i1, Create i2 e) ∈ set xs
  shows i1 = i2
proof –
  have ∃ s. valid-behaviours s (i1, Create i2 e)
    using assms deliver-in-prefix-is-valid by blast
  thus ?thesis
    by(simp add: valid-behaviours-def)

```

qed

**lemma** (in *imap*) *append-id-valid*:

assumes *xs* prefix of *j*  
and *Deliver* (*i1*, *Append i2 e*) ∈ *set xs*  
shows *i1* = *i2*

**proof** —

have ∃ *s*. *valid-behaviours s* (*i1*, *Append i2 e*)  
using *assms deliver-in-prefix-is-valid* **by** *blast*  
**thus** *?thesis*  
**by**(*simp add: valid-behaviours-def*)

qed

**lemma** (in *imap*) *expunge-id-valid*:

assumes *xs* prefix of *j*  
and *Deliver* (*i1*, *Expunge e mo i2*) ∈ *set xs*  
shows *i1* = *i2*

**proof** —

have ∃ *s*. *valid-behaviours s* (*i1*, *Expunge e mo i2*)  
using *assms deliver-in-prefix-is-valid* **by** *blast*  
**thus** *?thesis*  
**by**(*simp add: valid-behaviours-def*)

qed

**lemma** (in *imap*) *store-id-valid*:

assumes *xs* prefix of *j*  
and *Deliver* (*i1*, *Store e mo i2*) ∈ *set xs*  
shows *i1* = *i2*

**proof** —

have ∃ *s*. *valid-behaviours s* (*i1*, *Store e mo i2*)  
using *assms deliver-in-prefix-is-valid* **by** *blast*  
**thus** *?thesis*  
**by**(*simp add: valid-behaviours-def*)

qed

**definition** (in *imap*) *added-ids* :: ('*id* × ('*id*, '*b*) operation) event list ⇒ '*b* ⇒ '*id* list **where**

*added-ids es p* ≡ *List.map-filter* (λ*x*. case *x* of  
  *Deliver* (*i*, *Create j e*) ⇒ if *e* = *p* then *Some j* else *None* |  
  *Deliver* (*i*, *Expunge e mo j*) ⇒ if *e* = *p* then *Some j* else *None* |  
  - ⇒ *None*) *es*

**definition** (in *imap*) *added-files* :: ('*id* × ('*id*, '*b*) operation) event list ⇒ '*b* ⇒ '*id* list **where**

*added-files es p* ≡ *List.map-filter* (λ*x*. case *x* of  
  *Deliver* (*i*, *Append j e*) ⇒ if *e* = *p* then *Some j* else *None* |  
  *Deliver* (*i*, *Store e mo j*) ⇒ if *e* = *p* then *Some j* else *None* |  
  - ⇒ *None*) *es*

— added files simplifier

**lemma** (in *imap*) [*simp*]:  
**shows** *added-files* [] *e* = []  
**by** (auto *simp*: *added-files-def map-filter-def*)

**lemma** (in *imap*) [*simp*]:  
**shows** *added-files* (*xs* @ *ys*) *e* = *added-files xs e* @ *added-files ys e*  
**by** (auto *simp*: *added-files-def map-filter-append*)

**lemma** (in *imap*) *added-files-Broadcast-collapse* [*simp*]:  
**shows** *added-files* ([*Broadcast e*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Delete-collapse* [*simp*]:  
**shows** *added-files* ([*Deliver (i, Delete is e)*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Create-collapse* [*simp*]:  
**shows** *added-files* ([*Deliver (i, Create j e)*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Expunge-collapse* [*simp*]:  
**shows** *added-files* ([*Deliver (i, Expunge e mo j)*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Append-diff-collapse* [*simp*]:  
**shows**  $e \neq e' \implies$  *added-files* ([*Deliver (i, Append j e)*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Append-same-collapse* [*simp*]:  
**shows** *added-files* ([*Deliver (i, Append j e)*]) *e* = [*j*]  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Store-diff-collapse* [*simp*]:  
**shows**  $e \neq e' \implies$  *added-files* ([*Deliver (i, Store e mo j)*]) *e'* = []  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

**lemma** (in *imap*) *added-files-Deliver-Store-same-collapse* [*simp*]:  
**shows** *added-files* ([*Deliver (i, Store e mo j)*]) *e* = [*j*]  
**by** (auto *simp*: *added-files-def map-filter-append map-filter-def*)

— added ids simplifier

**lemma** (in *imap*) [*simp*]:  
**shows** *added-ids* [] *e* = []  
**by** (auto *simp*: *added-ids-def map-filter-def*)

**lemma** (in *imap*) *split-ids* [*simp*]:  
**shows** *added-ids* (*xs* @ *ys*) *e* = *added-ids xs e* @ *added-ids ys e*

by (auto simp: added-ids-def map-filter-append)

**lemma** (in *imap*) *added-ids-Broadcast-collapse* [*simp*]:  
 shows *added-ids* ([*Broadcast e*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Delete-collapse* [*simp*]:  
 shows *added-ids* ([*Deliver (i, Delete is e)*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Append-collapse* [*simp*]:  
 shows *added-ids* ([*Deliver (i, Append j e)*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Store-collapse* [*simp*]:  
 shows *added-ids* ([*Deliver (i, Store e mo j)*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Create-diff-collapse* [*simp*]:  
 shows  $e \neq e' \implies$  *added-ids* ([*Deliver (i, Create j e)*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Expunge-diff-collapse* [*simp*]:  
 shows  $e \neq e' \implies$  *added-ids* ([*Deliver (i, Expunge e mo j)*])  $e' = []$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Create-same-collapse* [*simp*]:  
 shows *added-ids* ([*Deliver (i, Create j e)*])  $e = [j]$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *added-ids-Deliver-Expunge-same-collapse* [*simp*]:  
 shows *added-ids* ([*Deliver (i, Expunge e mo j)*])  $e = [j]$   
 by (auto simp: added-ids-def map-filter-append map-filter-def)

**lemma** (in *imap*) *expunge-id-not-in-set*:  
 assumes  $i1 \notin \text{set } (\text{added-ids } [\text{Deliver } (i, \text{Expunge } e \text{ mo } i2)]) e$   
 shows  $i1 \neq i2$   
 using *assms* by *simp*

**lemma** (in *imap*) *apply-operations-added-ids*:  
 assumes *es* prefix of *j*  
 and *apply-operations*  $es = \text{Some } f$   
 shows  $\text{fst } (f x) \subseteq \text{set } (\text{added-ids } es x)$   
 using *assms* **proof** (*induct es arbitrary: f rule: rev-induct, force*)  
 case (*snoc x xs*) **thus** ?*case*  
**proof** (*cases x, force*)  
 case (*Deliver e*)  
 moreover **obtain** *a b* **where**  $e = (a, b)$  **by** *force*  
 ultimately **show** ?*thesis*

```

using snoc by(case-tac b; clarsimp simp: interp-msg-def split: bind-splits,
  force split: if-split-asm simp add: op-elem-def interpret-op-def)
qed
qed

lemma (in imap) apply-operations-added-files:
  assumes es prefix of j
  and apply-operations es = Some f
  shows snd (f x) ⊆ set (added-files es x)
  using assms proof (induct es arbitrary: f rule: rev-induct, force)
  case (snoc x xs) thus ?case
  proof (cases x, force)
    case (Deliver e)
      moreover obtain a b where e = (a, b) by force
      ultimately show ?thesis
        using snoc by(case-tac b; clarsimp simp: interp-msg-def split: bind-splits,
          force split: if-split-asm simp add: op-elem-def interpret-op-def)
      qed
    qed

lemma (in imap) Deliver-added-files:
  assumes xs prefix of j
  and i ∈ set (added-files xs e)
  shows Deliver (i, Append i e) ∈ set xs ∨ (∃ mo . Deliver (i, Store e mo i) ∈ set xs)
  using assms proof (induct xs rule: rev-induct, clarsimp)
  case (snoc x xs) thus ?case
  proof (cases x, force)
    case X: (Deliver e')
      moreover obtain a b where E: e' = (a, b) by force
      ultimately show ?thesis using snoc
        apply (case-tac b; clarify) apply (simp,metis prefix-of-appendD,force)
        using append-id-valid apply simp
        using E apply (metis
          added-files-Deliver-Append-diff-collapse added-files-Deliver-Append-same-collapse
          empty-iff in-set-conv-decomp list.set(1) prefix-of-appendD set-ConsD, simp)
        using E apply-operations-added-files apply (blast,simp)
        using E apply-operations-added-files
        by (metis Un-iff
          added-files-Deliver-Store-diff-collapse added-files-Deliver-Store-same-collapse empty-iff
          empty-set list.set-intros(1) prefix-of-appendD set-ConsD set-append store-id-valid)
      qed
    qed

end

```

## 5 Independence of IMAP Commands

In this section we show that two concurrent operations that reference to the same tag must be identical.

**theory**

*IMAP-proof-independent*

**imports**

*IMAP-def*

*IMAP-proof-helpers*

**begin**

**lemma** (in *imap*) *Broadcast-Expunge-Deliver-prefix-closed:*

**assumes**  $xs @ [Broadcast\ (i,\ Expunge\ e\ mo\ i)]$  *prefix of j*

**shows**  $Deliver\ (mo,\ Append\ mo\ e) \in set\ xs \vee$

$(\exists\ mo2.\ Deliver\ (mo,\ Store\ e\ mo2\ mo) \in set\ xs)$

**proof** –

**obtain**  $y$  **where** *apply-operations xs = Some y*

**using** *assms broadcast-only-valid-msgs* **by** *blast*

**moreover** **hence**  $mo \in snd\ (y\ e)$

**using** *broadcast-only-valid-msgs[of xs (i, Expunge e mo i) j]*

*valid-behaviours-def[of y (i, Expunge e mo i)] assms* **by** *auto*

**ultimately** **show** *?thesis*

**using** *assms Deliver-added-files apply-operations-added-files* **by** *blast*

**qed**

**lemma** (in *imap*) *Broadcast-Store-Deliver-prefix-closed:*

**assumes**  $xs @ [Broadcast\ (i,\ Store\ e\ mo\ i)]$  *prefix of j*

**shows**  $Deliver\ (mo,\ Append\ mo\ e) \in set\ xs \vee$

$(\exists\ mo2.\ Deliver\ (mo,\ Store\ e\ mo2\ mo) \in set\ xs)$

**proof** –

**obtain**  $y$  **where** *apply-operations xs = Some y*

**using** *assms broadcast-only-valid-msgs* **by** *blast*

**moreover** **hence**  $mo \in snd\ (y\ e)$

**using** *broadcast-only-valid-msgs[of xs (i, Store e mo i) j]*

*valid-behaviours-def[of y (i, Store e mo i)] assms* **by** *auto*

**ultimately** **show** *?thesis*

**using** *assms Deliver-added-files apply-operations-added-files* **by** *blast*

**qed**

**lemma** (in *imap*) *Deliver-added-ids:*

**assumes**  $xs$  *prefix of j*

**and**  $i \in set\ (added-ids\ xs\ e)$

**shows**  $Deliver\ (i,\ Create\ i\ e) \in set\ xs \vee$

$(\exists\ mo.\ Deliver\ (i,\ Expunge\ e\ mo\ i) \in set\ xs)$

**using** *assms* **proof** (*induct xs rule: rev-induct, clarsimp*)

**case** (*snoc x xs*) **thus** *?case*

**proof** (*cases x, force*)

**case**  $X:$  (*Deliver e'*)

**moreover** **obtain**  $a\ b$  **where**  $e' = (a,\ b)$  **by** *force*

**ultimately** **show** *?thesis*

**using** *snoc* **apply** (*case-tac b; clarify*)

**apply** (*simp,metis added-ids-Deliver-Create-diff-collapse*

*added-ids-Deliver-Create-same-collapse empty-iff list.set(1) set-ConsD create-id-valid*

*in-set-conv-decomp prefix-of-appendD, force)*  
**using** *append-id-valid* **apply** (*simp, metis (no-types, lifting) prefix-of-appendD, simp,*  
*metis*  
*Un-iff added-ids-Deliver-Expunge-diff-collapse added-ids-Deliver-Expunge-same-collapse*  
*empty-iff expunge-id-valid list.set(1) list.set-intros(1) prefix-of-appendD set-ConsD*  
*set-append)*  
**by** (*simp, blast*)  
**qed**  
**qed**

**lemma** (*in imap*) *Broadcast-Deliver-prefix-closed:*

**assumes** *xs @ [Broadcast (r, Delete ix e)] prefix of j*  
**and** *i ∈ ix*  
**shows** *Deliver (i, Create i e) ∈ set xs ∨*  
*Deliver (i, Append i e) ∈ set xs ∨*  
*(∃ mo . Deliver (i, Expunge e mo i) ∈ set xs) ∨*  
*(∃ mo . Deliver (i, Store e mo i) ∈ set xs)*

**proof** –

**obtain** *y* **where** *apply-operations xs = Some y*  
**using** *assms broadcast-only-valid-msgs* **by** *blast*  
**moreover** **hence** *ix = fst (y e) ∪ snd (y e)*  
**by** (*metis (mono-tags, lifting) assms(1) broadcast-only-valid-msgs operation.case(2)*  
*option.simps(1) valid-behaviours-def case-prodD)*  
**ultimately show** *?thesis*  
**using** *assms Deliver-added-ids apply-operations-added-ids*  
**by** (*metis Deliver-added-files Un-iff apply-operations-added-files le-iff-sup prefix-of-appendD)*  
**qed**

**lemma** (*in imap*) *concurrent-create-delete-independent-technical:*

**assumes** *i ∈ is*  
**and** *xs prefix of j*  
**and** *(i, Create i e) ∈ set (node-deliver-messages xs)*  
**and** *(ir, Delete is e) ∈ set (node-deliver-messages xs)*  
**shows** *hb (i, Create i e) (ir, Delete is e)*

**proof** –

**have** *f1: Deliver (i, Create i e) ∈ set (history j)*  
**using** *assms prefix-msg-in-history* **by** *blast*  
**obtain** *pre k* **where** *P: pre@[Broadcast (ir, Delete is e)] prefix of k*  
**using** *assms delivery-has-a-cause events-before-exist prefix-msg-in-history* **by** *blast*  
**hence** *f2: Deliver (i, Create i e) ∈ set pre ∨*  
*Deliver (i, Append i e) ∈ set pre ∨*  
*(∃ mo . Deliver (i, Expunge e mo i) ∈ set pre) ∨*  
*(∃ mo . Deliver (i, Store e mo i) ∈ set pre)*  
**using** *Broadcast-Deliver-prefix-closed assms* **by** *auto*  
**have** *f3: Deliver (i, Append i e) ∉ set pre* **using** *f1 P*  
**by** (*metis (full-types) Pair-inject fst-conv network.delivery-has-a-cause network.msg-id-unique*  
  
*network-axioms operation.simps(9) prefix-elem-to-carriers prefix-of-appendD)*  
**have** *f4: ∀ mo . Deliver (i, Expunge e mo i) ∉ set pre* **using** *f1 P*

by (metis delivery-has-a-cause fst-conv msg-id-unique old.prod.inject operation.simps(11)  
 prefix-elem-to-carriers prefix-of-appendD)  
 have  $\forall mo . \text{Deliver } (i, \text{Store } e \text{ mo } i) \notin \text{set pre}$  using f1 P  
 by (metis delivery-has-a-cause fst-conv msg-id-unique old.prod.inject operation.simps(13)  
 prefix-elem-to-carriers prefix-of-appendD)  
 thus ?thesis using f2 f3 f4 P events-in-local-order hb-deliver by blast  
 qed

lemma (in imap) concurrent-store-expunge-independent-technical:

assumes  $xs$  prefix of  $j$   
 and  $(i, \text{Store } e \text{ mo } i) \in \text{set } (\text{node-deliver-messages } xs)$   
 and  $(r, \text{Expunge } e \text{ i } r) \in \text{set } (\text{node-deliver-messages } xs)$   
 shows  $hb (i, \text{Store } e \text{ mo } i) (r, \text{Expunge } e \text{ i } r)$   
 proof –  
 obtain pre  $k$  where  $P: \text{pre}@[Broadcast (r, \text{Expunge } e \text{ i } r)]$  prefix of  $k$   
 using assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast  
 moreover hence f1:  $\text{Deliver } (i, \text{Append } i \text{ e}) \in \text{set pre} \vee$   
 $(\exists mo2 . \text{Deliver } (i, \text{Store } e \text{ mo2 } i) \in \text{set pre})$   
 using Broadcast-Expunge-Deliver-prefix-closed assms(1) by auto  
 hence f2:  $\text{Deliver } (i, \text{Append } i \text{ e}) \notin \text{set } (\text{history } k)$   
 by (metis Pair-inject assms(1) assms(2) fst-conv msg-id-unique network.delivery-has-a-cause  
 network-axioms operation.distinct(17) prefix-msg-in-history)  
 from f1 obtain mo2 :: 'a where  
 $\text{Deliver } (i, \text{Store } e \text{ mo2 } i) \in \text{set } (\text{history } k)$  using f2  
 using P prefix-elem-to-carriers by blast  
 hence  $\text{Deliver } (i, \text{Store } e \text{ mo } i) \in \text{set } (\text{history } k)$  using assms f1 f2 P  
 by (metis fst-conv msg-id-unique network.delivery-has-a-cause network-axioms  
 prefix-msg-in-history)  
 then show ?thesis  
 using hb.intros(2) events-in-local-order f1 f2 P  
 by (metis delivery-has-a-cause fst-conv msg-id-unique node-histories.prefix-of-appendD  
 node-histories-axioms prefix-elem-to-carriers)  
 qed

lemma (in imap) concurrent-store-expunge-independent-technical2:

assumes  $xs$  prefix of  $j$   
 and  $(i, \text{Store } e1 \text{ mo2 } i) \in \text{set } (\text{node-deliver-messages } xs)$   
 and  $(r, \text{Expunge } e \text{ mo } r) \in \text{set } (\text{node-deliver-messages } xs)$   
 shows  $mo2 \neq r$   
 proof –  
 obtain oid :: 'a  $\times$  ('a, 'b) operation  $\Rightarrow$  nat where  
 $oid: \forall p n. \text{Deliver } p \notin \text{set } (\text{history } n) \vee Broadcast p \in \text{set } (\text{history } (oid \text{ p}))$   
 by (metis (no-types) delivery-has-a-cause)  
 hence f1:  $Broadcast (r, \text{Expunge } e \text{ mo } r) \in \text{set } (\text{history } (oid (r, \text{Expunge } e \text{ mo } r)))$   
 using assms(1) assms(3) prefix-msg-in-history by blast  
 obtain  $k :: 'a \Rightarrow 'b \Rightarrow ('a \times ('a, 'b) \text{ operation}) \text{ event list} \Rightarrow 'a$  where  $k:$   
 $\forall i e \text{ pre}. (\exists mo. \text{Deliver } (i, \text{Store } e \text{ mo } i) \in \text{set pre}) =$   
 $(\text{Deliver } (i, \text{Store } e (k \text{ i } e \text{ pre}) \text{ i}) \in \text{set pre})$

**by** *moura*  
**obtain**  $pre :: nat \Rightarrow ('a \times ('a, 'b) \text{ operation}) \text{ event} \Rightarrow ('a \times ('a, 'b) \text{ operation}) \text{ event list}$   
**where**  $pre: \forall k \text{ op1}. (\exists \text{ op2}. \text{op2} @ [\text{op1}] \text{ prefix of } k) = (pre \ k \ \text{op1} @ [\text{op1}] \text{ prefix of } k)$   
**by** *moura*  
**hence**  $f2: \forall e \ n. e \notin \text{set}(\text{history } n) \vee pre \ n \ e @ [e] \text{ prefix of } n$   
**using** *events-before-exist* **by** *simp*  
**hence**  $f3: pre \ (\text{oid } (i, \text{Store } e1 \ \text{mo2 } i)) \ (\text{Broadcast } (i, \text{Store } e1 \ \text{mo2 } i))$   
*prefix of oid (i, Store e1 mo2 i)*  
**using** *oid assms(1) assms(2) prefix-msg-in-history* **by** *blast*  
**have**  $f4: \text{Deliver } (r, \text{Append } r \ e1) \notin \text{set}(\text{history } (\text{oid } (i, \text{Store } e1 \ \text{mo2 } i)))$   
**by** (*metis (no-types) oid f1 fst-conv msg-id-unique old.prod.inject operation.distinct(15)*)  
**have**  $\text{Deliver } (r, \text{Store } e1 \ (k \ r \ e1 \ (pre \ (\text{oid } (i, \text{Store } e1 \ \text{mo2 } i)) \ (\text{Broadcast } (i, \text{Store } e1 \ \text{mo2 } i)))) \ r) \notin \text{set}(\text{history } (\text{oid } (i, \text{Store } e1 \ \text{mo2 } i)))$   
**by** (*metis (no-types) oid f1 fst-conv msg-id-unique old.prod.inject operation.distinct(19)*)  
**thus** *?thesis* **using** *oid k f2 f3 f4 assms*  
**by** (*metis (no-types, lifting) Broadcast-Store-Deliver-prefix-closed network.prefix-msg-in-history network-axioms prefix-elem-to-carriers*)  
**qed**

**lemma** (*in imap*) *concurrent-store-delete-independent-technical*:

**assumes**  $i \in is$   
**and**  $xs \text{ prefix of } j$   
**and**  $(i, \text{Store } e \ \text{mo } i) \in \text{set}(\text{node-deliver-messages } xs)$   
**and**  $(ir, \text{Delete } is \ e) \in \text{set}(\text{node-deliver-messages } xs)$   
**shows**  $hb \ (i, \text{Store } e \ \text{mo } i) \ (ir, \text{Delete } is \ e)$

**proof** –

**have**  $f1: \text{Deliver } (i, \text{Store } e \ \text{mo } i) \in \text{set}(\text{history } j)$  **using** *assms prefix-msg-in-history* **by** *auto*  
**obtain**  $pre \ k$  **where**  $P: pre @ [\text{Broadcast } (ir, \text{Delete } is \ e)] \text{ prefix of } k$   
**using** *assms delivery-has-a-cause events-before-exist prefix-msg-in-history* **by** *blast*  
**hence**  $f2: \text{Deliver } (i, \text{Create } i \ e) \in \text{set } pre \vee$   
 $\text{Deliver } (i, \text{Append } i \ e) \in \text{set } pre \vee$   
 $(\exists \ \text{mo} . \text{Deliver } (i, \text{Expunge } e \ \text{mo } i) \in \text{set } pre) \vee$   
 $(\exists \ \text{mo} . \text{Deliver } (i, \text{Store } e \ \text{mo } i) \in \text{set } pre)$   
**using** *Broadcast-Deliver-prefix-closed assms(1)* **by** *auto*  
**have**  $f3: \text{Deliver } (i, \text{Create } i \ e) \notin \text{set } pre$  **using**  $f1 \ P$   
**by** (*metis Pair-inject delivery-has-a-cause fst-conv msg-id-unique operation.distinct(7) prefix-elem-to-carriers prefix-of-appendD*)  
**have**  $f4: \text{Deliver } (i, \text{Append } i \ e) \notin \text{set } pre$  **using**  $f1 \ P$   
**by** (*metis delivery-has-a-cause fst-conv msg-id-unique operation.distinct(17) prefix-elem-to-carriers prefix-of-appendD prod.inject*)  
**have**  $\forall \ \text{mo} . \text{Deliver } (i, \text{Expunge } e \ \text{mo } i) \notin \text{set } pre$  **using**  $f1 \ P$   
**by** (*metis Pair-inject delivery-has-a-cause fst-conv msg-id-unique operation.simps(25) prefix-elem-to-carriers prefix-of-appendD*)  
**hence**  $\text{Deliver } (i, \text{Store } e \ \text{mo } i) \in \text{set } pre$  **using**  $f1 \ f2 \ f3 \ f4 \ P$   
**by** (*metis delivery-has-a-cause fst-conv msg-id-unique node-histories.prefix-of-appendD node-histories-axioms prefix-elem-to-carriers*)  
**thus** *?thesis* **using**  $P \ \text{events-in-local-order } hb\text{-deliver}$  **by** *blast*  
**qed**

**lemma** (in *imap*) *concurrent-append-delete-independent-technical*:

**assumes**  $i \in is$   
**and**  $xs$  prefix of  $j$   
**and**  $(i, \text{Append } i \ e) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(ir, \text{Delete } is \ e) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $hb \ (i, \text{Append } i \ e) \ (ir, \text{Delete } is \ e)$

**proof** –

**obtain**  $pre \ k$  **where**  $P: pre@[Broadcast \ (ir, \text{Delete } is \ e)]$  prefix of  $k$   
**using** *assms delivery-has-a-cause events-before-exist prefix-msg-in-history* **by** *blast*  
**hence**  $f1: \text{Deliver } (i, \text{Create } i \ e) \in \text{set } pre \vee$   
 $\text{Deliver } (i, \text{Append } i \ e) \in \text{set } pre \vee$   
 $(\exists \ mo . \text{Deliver } (i, \text{Expunge } e \ mo \ i) \in \text{set } pre) \vee$   
 $(\exists \ mo . \text{Deliver } (i, \text{Store } e \ mo \ i) \in \text{set } pre)$   
**using** *Broadcast-Deliver-prefix-closed assms(1)* **by** *auto*  
**hence**  $\text{Deliver } (i, \text{Append } i \ e) \in \text{set } pre$  **using**  $P \ f1$   
**by** (*metis (no-types, hide-lams) delivery-has-a-cause events-in-local-order fst-conv*  
*hb-broadcast-exists1 hb-deliver msg-id-unique prefix-msg-in-history*)  
**thus** *?thesis* **using**  $P$  *events-in-local-order hb-deliver* **by** *blast*  
**qed**

**lemma** (in *imap*) *concurrent-append-expunge-independent-technical*:

**assumes**  $i = mo$   
**and**  $xs$  prefix of  $j$   
**and**  $(i, \text{Append } i \ e) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(r, \text{Expunge } e \ mo \ r) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $hb \ (i, \text{Append } i \ e) \ (r, \text{Expunge } e \ mo \ r)$

**proof** –

**obtain**  $pre \ k$  **where**  $P: pre@[Broadcast \ (r, \text{Expunge } e \ mo \ r)]$  prefix of  $k$   
**using** *assms delivery-has-a-cause events-before-exist prefix-msg-in-history* **by** *blast*  
**hence**  $f1: \text{Deliver } (mo, \text{Append } mo \ e) \in \text{set } pre \vee$   
 $(\exists \ mo2 . \text{Deliver } (mo, \text{Store } e \ mo2 \ mo) \in \text{set } pre)$   
**using** *Broadcast-Expunge-Deliver-prefix-closed assms(1)* **by** *auto*  
**hence**  $(\forall \ mo2 . \text{Deliver } (mo, \text{Store } e \ mo2 \ mo) \notin \text{set } pre)$  **using**  $P$  *assms*

**proof** –

**have**  $\text{Deliver } (mo, \text{Append } mo \ e) \in \text{set } (\text{history } j)$   
**using** *assms(1) assms(2) assms(3) prefix-msg-in-history* **by** *blast*  
**thus** *?thesis*  
**by** (*metis (no-types) P Pair-inject delivery-has-a-cause fst-conv msg-id-unique*  
*operation.simps(23) prefix-elem-to-carriers prefix-of-appendD*)

**qed**

**thus** *?thesis*

**using** *hb.intros(2) events-in-local-order assms(1) P f1* **by** *blast*

**qed**

**lemma** (in *imap*) *concurrent-append-store-independent-technical*:

**assumes**  $i = mo$   
**and**  $xs$  prefix of  $j$   
**and**  $(i, \text{Append } i \ e) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(r, \text{Store } e \ mo \ r) \in \text{set } (\text{node-deliver-messages } xs)$

shows  $hb (i, Append\ i\ e) (r, Store\ e\ mo\ r)$   
**proof** –  
**obtain**  $pre\ k$  **where**  $pre: pre@[Broadcast\ (r, Store\ e\ mo\ r)]\ prefix\ of\ k$   
**using**  $assms\ delivery\ has\ a\ cause\ events\ before\ exist\ prefix\ msg\ in\ history$  **by**  $blast$   
**moreover** **hence**  $f1: Deliver\ (mo, Append\ mo\ e) \in set\ pre \vee$   
 $(\exists\ mo2 . Deliver\ (mo, Store\ e\ mo2\ mo) \in set\ pre)$   
**using**  $Broadcast\ Store\ Deliver\ prefix\ closed\ assms(1)$  **by**  $auto$   
**have**  $f2: Deliver\ (i, Append\ i\ e) \in set\ (history\ j)$   
**by**  $(meson\ assms\ network.\ prefix\ msg\ in\ history\ network\ axioms)$   
**then** **show**  $?thesis$  **using**  $f1$   
**by**  $(metis\ pre\ delivery\ has\ a\ cause\ events\ in\ local\ order\ fst\ conv\ hb\ deliver\ msg\ id\ unique\ node\ histories.\ prefix\ of\ appendD\ node\ histories\ axioms\ prefix\ elem\ to\ carriers)$   
**qed**

**lemma** (**in**  $imap$ ) *concurrent-expunge-delete-independent-technical*:

**assumes**  $i \in is$   
**and**  $xs$  *prefix of j*  
**and**  $(i, Expunge\ e\ mo\ i) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $(ir, Delete\ is\ e) \in set\ (node\ deliver\ messages\ xs)$   
**shows**  $hb (i, Expunge\ e\ mo\ i) (ir, Delete\ is\ e)$   
**proof** –  
**obtain**  $pre\ k$  **where**  $pre: pre@[Broadcast\ (ir, Delete\ is\ e)]\ prefix\ of\ k$   
**using**  $assms\ delivery\ has\ a\ cause\ events\ before\ exist\ prefix\ msg\ in\ history$  **by**  $blast$   
**moreover** **hence**  $A: Deliver\ (i, Create\ i\ e) \in set\ pre \vee$   
 $Deliver\ (i, Append\ i\ e) \in set\ pre \vee$   
 $(\exists\ mo . Deliver\ (i, Expunge\ e\ mo\ i) \in set\ pre) \vee$   
 $(\exists\ mo . Deliver\ (i, Store\ e\ mo\ i) \in set\ pre)$   
**using**  $Broadcast\ Deliver\ prefix\ closed\ assms(1)$  **by**  $auto$   
**hence**  $Deliver\ (i, Expunge\ e\ mo\ i) \in set\ pre$  **using**  $assms$   
**proof** –  
**have**  $f1: \bigwedge e. e \notin set\ pre \vee e \in set\ (history\ k)$   
**using**  $pre\ prefix\ elem\ to\ carriers$  **by**  $blast$   
**have**  $f2: Deliver\ (i, Expunge\ e\ mo\ i) \in set\ (history\ j)$   
**by**  $(meson\ assms\ network.\ prefix\ msg\ in\ history\ network\ axioms)$   
**then** **show**  $?thesis$  **using**  $f1\ A$   
**by**  $(metis\ no\ types,\ lifting)\ fst\ conv\ msg\ id\ unique\ network.\ delivery\ has\ a\ cause\ network\ axioms)$   
**qed**  
**ultimately** **show**  $?thesis$   
**using**  $hb.intros(2)\ events\ in\ local\ order$  **by**  $blast$   
**qed**

**lemma** (**in**  $imap$ ) *concurrent-store-store-independent-technical*:

**assumes**  $xs$  *prefix of j*  
**and**  $(i, Store\ e\ mo\ i) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $(r, Store\ e\ i\ r) \in set\ (node\ deliver\ messages\ xs)$   
**shows**  $hb (i, Store\ e\ mo\ i) (r, Store\ e\ i\ r)$   
**proof** –

**obtain**  $pre\ k$  **where**  $P: pre@[Broadcast\ (r,\ Store\ e\ i\ r)]\ prefix\ of\ k$   
**using**  $assms\ delivery\ has\ a\ cause\ events\ before\ exist\ prefix\ msg\ in\ history$  **by**  $blast$   
**hence**  $f1: \forall e. e \notin set\ pre \vee e \in set\ (history\ k)$   
**using**  $prefix\ elem\ to\ carriers$  **by**  $blast$   
**have**  $f2: Deliver\ (i,\ Append\ i\ e) \in set\ pre \vee (\exists\ mo2.\ Deliver\ (i,\ Store\ e\ mo2\ i) \in set\ pre)$   
**using**  $Broadcast\ Store\ Deliver\ prefix\ closed\ assms(1)\ P$  **by**  $auto$   
**hence**  $Deliver\ (i,\ Store\ e\ mo\ i) \in set\ pre$  **using**  $assms\ f1$   
**by**  $(metis\ delivery\ has\ a\ cause\ fst\ conv\ msg\ id\ unique\ prefix\ msg\ in\ history)$   
**then show**  $?thesis$   
**using**  $hb.intros(2)\ events\ in\ local\ order\ P$  **by**  $blast$   
**qed**

**lemma** (in  $imap$ )  $expunge\ delete\ tag\ causality$ :

**assumes**  $i \in is$   
**and**  $xs\ prefix\ of\ j$   
**and**  $(i,\ Expunge\ e1\ mo\ i) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $(ir,\ Delete\ is\ e2) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $pre@[Broadcast\ (ir,\ Delete\ is\ e2)]\ prefix\ of\ k$   
**shows**  $Deliver\ (i,\ Expunge\ e2\ mo\ i) \in set\ (history\ k)$   
**proof**–  
**have**  $f1: Deliver\ (i,\ Append\ i\ e2) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.deliver\ has\ a\ cause\ network\ axioms\ old.prod.inject$   
 $operation.distinct(15)\ prefix\ msg\ in\ history)$   
**have**  $f2: Deliver\ (i,\ Create\ i\ e2) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.deliver\ has\ a\ cause\ network\ axioms\ old.prod.inject$   
 $operation.distinct(5)\ prefix\ msg\ in\ history)$   
**have**  $f3: \forall\ mo. Deliver\ (i,\ Store\ e2\ mo\ i) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ Pair\ inject\ fst\ conv\ msg\ id\ unique\ network.deliver\ has\ a\ cause\ network\ axioms$   
 $operation.simps(25)\ prefix\ msg\ in\ history)$   
**hence**  $\exists\ mo1. Deliver\ (i,\ Expunge\ e2\ mo1\ i) \in set\ (history\ k)$  **using**  $assms\ f1\ f2$   
**by**  $(meson\ imap.Broadcast\ Deliver\ prefix\ closed\ imap\ axioms\ node\ histories.prefix\ of\ appendD$   
 $node\ histories\ axioms\ prefix\ elem\ to\ carriers)$   
**then obtain**  $mo1 :: 'a$  **where**  
 $Deliver\ (i,\ Expunge\ e2\ mo1\ i) \in set\ (history\ k)$  **by**  $blast$   
**then show**  $?thesis$  **using**  $assms\ f1\ f2\ f3$   
**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.deliver\ has\ a\ cause\ network\ axioms\ old.prod.inject$   
 $operation.inject(4)\ prefix\ msg\ in\ history)$   
**qed**

**lemma** (in  $imap$ )  $expunge\ delete\ ids\ imply\ messages\ same$ :

**assumes**  $i \in is$   
**and**  $xs\ prefix\ of\ j$   
**and**  $(i,\ Expunge\ e1\ mo\ i) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $(ir,\ Delete\ is\ e2) \in set\ (node\ deliver\ messages\ xs)$   
**shows**  $e1 = e2$

**proof** –

**obtain**  $pre\ k$  **where**  $P: pre@[Broadcast\ (ir,\ Delete\ is\ e2)]\ prefix\ of\ k$   
**using**  $assms\ delivery\ has\ a\ cause\ events\ before\ exist\ prefix\ msg\ in\ history$  **by**  $blast$   
**hence**  $Deliver\ (i,\ Expunge\ e2\ mo\ i) \in set\ (history\ k)$  **using**  $assms\ expunge\ delete\ tag\ causality$   
**by**  $blast$   
**then show**  $?thesis$  **using**  $assms$   
**by**  $(metis\ delivery\ has\ a\ cause\ fst\ conv\ network.\ msg\ id\ unique\ network\ axioms\ operation.\ inject(4)\ prefix\ msg\ in\ history\ prod.\ inject)$

**qed**

**lemma** (in  $imap$ )  $store\ delete\ ids\ imply\ messages\ same:$

**assumes**  $i \in is$   
**and**  $xs\ prefix\ of\ j$   
**and**  $(i,\ Store\ e1\ mo\ i) \in set\ (node\ deliver\ messages\ xs)$   
**and**  $(ir,\ Delete\ is\ e2) \in set\ (node\ deliver\ messages\ xs)$   
**shows**  $e1 = e2$

**proof** –

**obtain**  $pre\ k$  **where**  $P: pre@[Broadcast\ (ir,\ Delete\ is\ e2)]\ prefix\ of\ k$   
**using**  $assms\ delivery\ has\ a\ cause\ events\ before\ exist\ prefix\ msg\ in\ history$  **by**  $blast$   
**have**  $f1: Deliver\ (i,\ Append\ i\ e2) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.\ delivery\ has\ a\ cause\ network\ axioms\ old.\ prod.\ inject$

$operation.\ distinct(17)\ prefix\ msg\ in\ history)$

**have**  $f2: \forall\ mo.\ Deliver\ (i,\ Expunge\ e2\ mo\ i) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ Pair\ inject\ fst\ conv\ msg\ id\ unique\ network.\ delivery\ has\ a\ cause\ network\ axioms\ operation.\ distinct(19)\ prefix\ msg\ in\ history)$

**have**  $f3: Deliver\ (i,\ Create\ i\ e2) \notin set\ (history\ k)$  **using**  $assms$   
**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.\ delivery\ has\ a\ cause\ network\ axioms\ old.\ prod.\ inject$

$operation.\ distinct(8)\ prefix\ msg\ in\ history)$

**hence**  $(\exists\ mo1.\ Deliver\ (i,\ Store\ e2\ mo1\ i) \in set\ pre)$  **using**  $assms\ P\ f1\ f2\ imap\ axioms$   
**by**  $(meson\ imap.\ Broadcast\ Deliver\ prefix\ closed\ prefix\ elem\ to\ carriers\ prefix\ of\ appendD)$

**then obtain**  $mo1 :: 'a$  **where**

$f3: Deliver\ (i,\ Store\ e2\ mo1\ i) \in set\ pre$  **by**  $blast$

**then have**  $f4: Deliver\ (i,\ Store\ e2\ mo1\ i) \in set\ (history\ k)$   
**using**  $P\ prefix\ elem\ to\ carriers$  **by**  $blast$

**hence**  $Deliver\ (i,\ Store\ e2\ mo\ i) \in set\ pre$  **using**  $f2\ f3\ assms$

**by**  $(metis\ fst\ conv\ msg\ id\ unique\ network.\ delivery\ has\ a\ cause\ network\ axioms\ old.\ prod.\ inject$

$operation.\ inject(5)\ prefix\ msg\ in\ history)$

**moreover have**  $Deliver(i,\ Store\ e1\ mo\ i) \in set\ (history\ j)$   
**using**  $assms(2)\ assms(3)\ prefix\ msg\ in\ history$  **by**  $blast$

**ultimately show**  $?thesis$  **using**  $f4$

**by**  $(metis\ delivery\ has\ a\ cause\ fst\ conv\ msg\ id\ unique\ old.\ prod.\ inject\ operation.\ inject(5))$

**qed**

**lemma** (in  $imap$ )  $create\ delete\ ids\ imply\ messages\ same:$

**assumes**  $i \in is$   
**and**  $xs\ prefix\ of\ j$

**and**  $(i, \text{Create } i \ e1) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(ir, \text{Delete } ir \ e2) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $e1 = e2$   
**proof** –  
**obtain**  $pre \ k$  **where**  $P: \text{pre}@[\text{Broadcast } (ir, \text{Delete } ir \ e2)] \text{ prefix of } k$   
**using**  $\text{assms } \text{delivery-has-a-cause } \text{events-before-exist } \text{prefix-msg-in-history}$  **by**  $\text{blast}$   
**have**  $f1: \text{Deliver } (i, \text{Append } i \ e2) \notin \text{set } (\text{history } k)$   
**by**  $(\text{metis } \text{assms}(2) \ \text{assms}(3) \ \text{delivery-has-a-cause } \text{fst-conv } \text{network.msg-id-unique}$   
 $\text{network.prefix-msg-in-history } \text{network-axioms } \text{operation.distinct}(3) \ \text{prod.inject})$   
**have**  $f2: \forall \ mo. \text{Deliver } (i, \text{Expunge } e2 \ mo \ i) \notin \text{set } (\text{history } k)$   
**by**  $(\text{metis } \text{assms}(2) \ \text{assms}(3) \ \text{fst-conv } \text{msg-id-unique } \text{network.delivery-has-a-cause } \text{network-axioms}$   
 $\text{old.prod.inject } \text{operation.distinct}(5) \ \text{prefix-msg-in-history})$   
**have**  $f3: \forall \ mo. \text{Deliver } (i, \text{Store } e2 \ mo \ i) \notin \text{set } (\text{history } k)$   
**by**  $(\text{metis } \text{Pair-inject } \text{assms}(2) \ \text{assms}(3) \ \text{delivery-has-a-cause } \text{fst-conv } \text{msg-id-unique}$   
 $\text{operation.distinct}(7) \ \text{prefix-msg-in-history})$   
**hence**  $\text{Deliver } (i, \text{Create } i \ e2) \in \text{set } pre$  **using**  $\text{assms } P \ f2 \ f1 \ \text{imap-axioms}$   
**by**  $(\text{meson } \text{imap.Broadcast-Deliver-prefix-closed } \text{prefix-elem-to-carriers } \text{prefix-of-appendD})$   
**then show**  $?thesis$  **using**  $f1 \ f2 \ f3$   
**by**  $(\text{metis } (\text{no-types, lifting}) \ P \ \text{assms}(2) \ \text{assms}(3) \ \text{delivery-has-a-cause } \text{fst-conv } \text{msg-id-unique}$   
 $\text{node-histories.prefix-of-appendD } \text{node-histories-axioms } \text{old.prod.inject } \text{operation.inject}(1)$   
 $\text{prefix-elem-to-carriers } \text{prefix-msg-in-history})$   
**qed**

**lemma** **(in**  $\text{imap}$ **)**  $\text{append-delete-ids-imply-messages-same:}$   
**assumes**  $i \in is$   
**and**  $xs$   $\text{prefix of } j$   
**and**  $(i, \text{Append } i \ e1) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(ir, \text{Delete } ir \ e2) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $e1 = e2$   
**proof** –  
**obtain**  $pre \ k$  **where**  $P: \text{pre}@[\text{Broadcast } (ir, \text{Delete } ir \ e2)] \text{ prefix of } k$   
**using**  $\text{assms } \text{delivery-has-a-cause } \text{events-before-exist } \text{prefix-msg-in-history}$  **by**  $\text{blast}$   
**hence**  $f1: \bigwedge e. e \in \text{set } pre \implies e \in \text{set } (\text{history } k)$  **using**  $\text{prefix-elem-to-carriers}$  **by**  $\text{blast}$   
**have**  $f2: \text{Deliver } (i, \text{Create } i \ e2) \notin \text{set } pre$  **using**  $P \ f1$   
**by**  $(\text{metis } \text{assms}(2) \ \text{assms}(3) \ \text{fst-conv } \text{msg-id-unique } \text{network.delivery-has-a-cause } \text{network-axioms}$   
 $\text{old.prod.inject } \text{operation.distinct}(3) \ \text{prefix-msg-in-history})$   
**moreover have**  $D1: \forall \ mo. \text{Deliver } (i, \text{Expunge } e2 \ mo \ i) \notin \text{set } pre$  **using**  $P \ f1$   
**by**  $(\text{metis } \text{Pair-inject } \text{assms}(2) \ \text{assms}(3) \ \text{fst-conv } \text{msg-id-unique } \text{network.delivery-has-a-cause}$   
 $\text{network-axioms } \text{operation.distinct}(15) \ \text{prefix-msg-in-history})$   
**moreover have**  $D2: \forall \ mo. \text{Deliver } (i, \text{Store } e2 \ mo \ i) \notin \text{set } pre$  **using**  $P \ f1$   
**by**  $(\text{metis } \text{Pair-inject } \text{assms}(2) \ \text{assms}(3) \ \text{fst-conv } \text{msg-id-unique } \text{network.delivery-has-a-cause}$   
 $\text{network-axioms } \text{operation.simps}(23) \ \text{prefix-msg-in-history})$   
**moreover hence**  $\text{Deliver } (i, \text{Append } i \ e2) \in \text{set } pre$

using  $P$   $D1$   $D2$   $f2$   $assms(1)$  *Broadcast-Deliver-prefix-closed* **by** *blast*  
**moreover have**  $Deliver(i, Append\ i\ e1) \in set(history\ j)$   
 using  $assms(2)$   $assms(3)$  *prefix-msg-in-history* **by** *blast*  
**ultimately show** *?thesis* **using**  $assms$   
**by** ( $metis\ f1\ msg-id-unique\ network.delivery-has-a-cause\ network-axioms\ old.prod.inject$   
 $operation.inject(3)\ prod.sel(1)$ )  
**qed**

**lemma** (in *imap*) *append-expunge-ids-imply-messages-same*:

**assumes**  $i = mo$   
**and**  $xs$  *prefix of*  $j$   
**and**  $(i, Append\ i\ e1) \in set(node-deliver-messages\ xs)$   
**and**  $(r, Expunge\ e2\ mo\ r) \in set(node-deliver-messages\ xs)$   
**shows**  $e1 = e2$

**proof** –

**obtain**  $pre\ k$  **where**  $pre$ :  $pre@[Broadcast\ (r, Expunge\ e2\ mo\ r)]$  *prefix of*  $k$   
**using**  $assms\ delivery-has-a-cause\ events-before-exist\ prefix-msg-in-history$  **by** *blast*  
**moreover hence**  $Deliver(mo, Append\ mo\ e2) \in set\ pre \vee$   
 $(\exists\ mo2 . Deliver(mo, Store\ e2\ mo2\ mo) \in set\ pre)$   
**using** *Broadcast-Expunge-Deliver-prefix-closed*  $assms(1)$   
**by** ( $meson\ imap.Broadcast-Deliver-prefix-closed\ imap-axioms$ )  
**hence**  $Deliver(i, Append\ i\ e2) \in set\ pre$  **using**  $assms$   
**by** ( $metis\ (no-types, lifting)\ pre\ delivery-has-a-cause\ fst-conv\ hb-broadcast-exists1$   
 $msg-id-unique\ network.hb-deliver\ network.prefix-msg-in-history\ network-axioms$   
 $node-histories.events-in-local-order\ node-histories-axioms\ operation.distinct(17)$   
 $prod.inject$ )  
**moreover have**  $Deliver(i, Append\ i\ e1) \in set(history\ j)$   
**using**  $assms(2)$   $assms(3)$  *prefix-msg-in-history* **by** *blast*  
**ultimately show** *?thesis*  
**by** ( $metis\ (no-types, lifting)\ fst-conv\ network.delivery-has-a-cause\ network.msg-id-unique$   
 $network-axioms\ operation.inject(3)\ prefix-elem-to-carriers\ prefix-of-appendD\ prod.inject$ )

**qed**

**lemma** (in *imap*) *append-store-ids-imply-messages-same*:

**assumes**  $i = mo$   
**and**  $xs$  *prefix of*  $j$   
**and**  $(i, Append\ i\ e1) \in set(node-deliver-messages\ xs)$   
**and**  $(r, Store\ e2\ mo\ r) \in set(node-deliver-messages\ xs)$   
**shows**  $e1 = e2$

**proof** –

**obtain**  $pre\ k$  **where**  $P$ :  $pre@[Broadcast\ (r, Store\ e2\ mo\ r)]$  *prefix of*  $k$   
**using**  $assms\ delivery-has-a-cause\ events-before-exist\ prefix-msg-in-history$  **by** *blast*  
**moreover hence**  $A$ :  $Deliver(mo, Append\ mo\ e2) \in set\ pre \vee$   
 $(\exists\ mo2 . Deliver(mo, Store\ e2\ mo2\ mo) \in set\ pre)$   
**using** *Broadcast-Store-Deliver-prefix-closed*  $assms(1)$   
**by** ( $meson\ imap.Broadcast-Deliver-prefix-closed\ imap-axioms$ )  
**have**  $f1$ :  $Deliver(i, Append\ i\ e1) \in set(history\ j)$   
**using**  $assms(2)$   $assms(3)$  *prefix-msg-in-history* **by** *blast*  
**hence**  $Deliver(i, Append\ i\ e2) \in set\ pre$  **using**  $assms\ P\ A$

by (metis Pair-inject assms(1) P delivery-has-a-cause fst-conv msg-id-unique  
 operation.simps(23) prefix-elim-to-carriers prefix-of-appendD)  
**then show** ?thesis **using** f1  
 by (metis P delivery-has-a-cause fst-conv msg-id-unique  
 node-histories.prefix-of-appendD node-histories-axioms operation.inject(3)  
 prefix-elim-to-carriers prod.inject)  
**qed**

**lemma** (in *imap*) *expunge-store-ids-imply-messages-same*:

**assumes** *xs* prefix of *j*  
**and** (*i*, Store *e1* mo *i*) ∈ set (node-deliver-messages *xs*)  
**and** (*r*, Expunge *e2* *i* *r*) ∈ set (node-deliver-messages *xs*)  
**shows** *e1* = *e2*  
**proof** –  
**obtain** *pre* *k* **where** *P*: pre@[Broadcast (*r*, Expunge *e2* *i* *r*)] prefix of *k*  
**using** *assms* delivery-has-a-cause events-before-exist prefix-msg-in-history **by** blast  
**hence** *pprefix*: *pre* prefix of *k*  
**using** *P* **by** blast  
**have** *A*: Deliver (*i*, Append *i* *e2*) ∈ set *pre* ∨  
 (∃ *mo2* . Deliver (*i*, Store *e2* mo2 *i*) ∈ set *pre*)  
**using** Broadcast-Expunge-Deliver-prefix-closed *assms*(1) *P* **by** blast  
**have** Deliver (*i*, Store *e2* mo *i*) ∈ set *pre* **using** *A* *P*  
**proof** –  
**obtain** *op1* :: '*a* × ('*a*, '*b*) operation ⇒ nat **where**  
*f1*: Broadcast (*i*, Store *e1* mo *i*) ∈ set (history (*op1* (*i*, Store *e1* mo *i*)))  
**by** (meson *assms*(1) *assms*(2) delivery-has-a-cause prefix-msg-in-history)  
**then show** ?thesis  
**using** *f1* *A* *pprefix* delivery-has-a-cause network.msg-id-unique network-axioms  
 node-histories.prefix-to-carriers node-histories-axioms  
**by** fastforce  
**qed**  
**moreover have** Deliver (*i*, Store *e1* mo *i*) ∈ set (history *j*)  
**using** *assms*(1) *assms*(2) prefix-msg-in-history **by** auto  
**ultimately show** ?thesis **using** *assms* *P*  
**by** (metis delivery-has-a-cause fst-conv msg-id-unique operation.inject(5)  
 prefix-elim-to-carriers prefix-of-appendD prod.inject)  
**qed**

**lemma** (in *imap*) *store-store-ids-imply-messages-same*:

**assumes** *xs* prefix of *j*  
**and** (*i*, Store *e1* mo *i*) ∈ set (node-deliver-messages *xs*)  
**and** (*r*, Store *e2* *i* *r*) ∈ set (node-deliver-messages *xs*)  
**shows** *e1* = *e2*  
**proof** –  
**obtain** *pre* *k* **where** *P*: pre@[Broadcast (*r*, Store *e2* *i* *r*)] prefix of *k*  
**using** *assms* delivery-has-a-cause events-before-exist prefix-msg-in-history **by** blast  
**moreover hence** *A*: Deliver (*i*, Append *i* *e2*) ∈ set *pre* ∨  
 (∃ *mo2* . Deliver (*i*, Store *e2* mo2 *i*) ∈ set *pre*)  
**using** Broadcast-Store-Deliver-prefix-closed *assms*(1) **by** blast

**have**  $\forall e. e \notin \text{set } pre \vee e \in \text{set } (\text{history } k)$   
**using**  $P$  *prefix-elem-to-carriers* **by** *auto*  
**hence**  $\text{Deliver } (i, \text{Store } e2 \text{ mo } i) \in \text{set } pre$   
**by** (*metis A assms(1) assms(2) delivery-has-a-cause fst-conv msg-id-unique*  
*operation.distinct(17) operation.inject(5) prefix-msg-in-history prod.inject*)  
**moreover have**  $\text{Deliver } (i, \text{Store } e1 \text{ mo } i) \in \text{set } (\text{history } j)$   
**using** *assms(1) assms(2) prefix-msg-in-history* **by** *auto*  
**ultimately show** *?thesis* **using** *assms*  
**by** (*metis Pair-inject delivery-has-a-cause msg-id-unique operation.simps(5)*  
*prefix-elem-to-carriers prefix-of-appendD prod.sel(1)*)  
**qed**  
**end**

## 6 Convergence of the IMAP-CRDT

In this final section show that concurrent updates commute and thus Strong Eventual Convergence is achieved.

**theory**

*IMAP-proof*

**imports**

*IMAP-def*

*IMAP-proof-commute*

*IMAP-proof-helpers*

*IMAP-proof-independent*

**begin**

**corollary** (*in imap*) *concurrent-create-delete-independent:*

**assumes**  $\neg hb (i, \text{Create } i \ e1) (ir, \text{Delete } is \ e2)$

**and**  $\neg hb (ir, \text{Delete } is \ e2) (i, \text{Create } i \ e1)$

**and** *xs prefix of j*

**and**  $(i, \text{Create } i \ e1) \in \text{set } (\text{node-deliver-messages } xs)$

**and**  $(ir, \text{Delete } is \ e2) \in \text{set } (\text{node-deliver-messages } xs)$

**shows**  $i \notin is$

**using** *assms create-delete-ids-imply-messages-same concurrent-create-delete-independent-technical*

**by** *fastforce*

**corollary** (*in imap*) *concurrent-append-delete-independent:*

**assumes**  $\neg hb (i, \text{Append } i \ e1) (ir, \text{Delete } is \ e2)$

**and**  $\neg hb (ir, \text{Delete } is \ e2) (i, \text{Append } i \ e1)$

**and** *xs prefix of j*

**and**  $(i, \text{Append } i \ e1) \in \text{set } (\text{node-deliver-messages } xs)$

**and**  $(ir, \text{Delete } is \ e2) \in \text{set } (\text{node-deliver-messages } xs)$

**shows**  $i \notin is$

**using** *assms append-delete-ids-imply-messages-same concurrent-append-delete-independent-technical*

**by** *fastforce*

**corollary** (in *imap*) *concurrent-append-expunge-independent*:

**assumes**  $\neg hb (i, Append\ i\ e1) (r, Expunge\ e2\ mo\ r)$   
**and**  $\neg hb (r, Expunge\ e2\ mo\ r) (i, Append\ i\ e1)$   
**and** *xs prefix of j*  
**and**  $(i, Append\ i\ e1) \in set\ (node-deliver-messages\ xs)$   
**and**  $(r, Expunge\ e2\ mo\ r) \in set\ (node-deliver-messages\ xs)$   
**shows**  $i \neq mo$

**using** *assms append-expunge-ids-imply-messages-same concurrent-append-expunge-independent-technical*

**by** *fastforce*

**corollary** (in *imap*) *concurrent-append-store-independent*:

**assumes**  $\neg hb (i, Append\ i\ e1) (r, Store\ e2\ mo\ r)$   
**and**  $\neg hb (r, Store\ e2\ mo\ r) (i, Append\ i\ e1)$   
**and** *xs prefix of j*  
**and**  $(i, Append\ i\ e1) \in set\ (node-deliver-messages\ xs)$   
**and**  $(r, Store\ e2\ mo\ r) \in set\ (node-deliver-messages\ xs)$   
**shows**  $i \neq mo$

**using** *assms append-store-ids-imply-messages-same concurrent-append-store-independent-technical*

**by** *fastforce*

**corollary** (in *imap*) *concurrent-expunge-delete-independent*:

**assumes**  $\neg hb (i, Expunge\ e1\ mo\ i) (ir, Delete\ is\ e2)$   
**and**  $\neg hb (ir, Delete\ is\ e2) (i, Expunge\ e1\ mo\ i)$   
**and** *xs prefix of j*  
**and**  $(i, Expunge\ e1\ mo\ i) \in set\ (node-deliver-messages\ xs)$   
**and**  $(ir, Delete\ is\ e2) \in set\ (node-deliver-messages\ xs)$   
**shows**  $i \notin is$

**using** *assms expunge-delete-ids-imply-messages-same concurrent-expunge-delete-independent-technical*

**by** *fastforce*

**corollary** (in *imap*) *concurrent-store-delete-independent*:

**assumes**  $\neg hb (i, Store\ e1\ mo\ i) (ir, Delete\ is\ e2)$   
**and**  $\neg hb (ir, Delete\ is\ e2) (i, Store\ e1\ mo\ i)$   
**and** *xs prefix of j*  
**and**  $(i, Store\ e1\ mo\ i) \in set\ (node-deliver-messages\ xs)$   
**and**  $(ir, Delete\ is\ e2) \in set\ (node-deliver-messages\ xs)$   
**shows**  $i \notin is$

**using** *assms store-delete-ids-imply-messages-same concurrent-store-delete-independent-technical*

**by** *fastforce*

**corollary** (in *imap*) *concurrent-store-expunge-independent*:

**assumes**  $\neg hb (i, Store\ e1\ mo\ i) (r, Expunge\ e2\ mo2\ r)$   
**and**  $\neg hb (r, Expunge\ e2\ mo2\ r) (i, Store\ e1\ mo\ i)$   
**and** *xs prefix of j*

**and**  $(i, \text{Store } e1 \text{ mo } i) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(r, \text{Expunge } e2 \text{ mo2 } r) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $i \neq \text{mo2} \wedge r \neq \text{mo}$   
**using** *assms expunge-store-ids-imply-messages-same concurrent-store-expunge-independent-technical2*

*concurrent-store-expunge-independent-technical* **by** *metis*

**corollary** (**in** *imap*) *concurrent-store-store-independent*:

**assumes**  $\neg \text{hb } (i, \text{Store } e1 \text{ mo } i) (r, \text{Store } e2 \text{ mo2 } r)$   
**and**  $\neg \text{hb } (r, \text{Store } e2 \text{ mo2 } r) (i, \text{Store } e1 \text{ mo } i)$   
**and** *xs prefix of j*  
**and**  $(i, \text{Store } e1 \text{ mo } i) \in \text{set } (\text{node-deliver-messages } xs)$   
**and**  $(r, \text{Store } e2 \text{ mo2 } r) \in \text{set } (\text{node-deliver-messages } xs)$   
**shows**  $i \neq \text{mo2} \wedge r \neq \text{mo}$   
**using** *assms store-store-ids-imply-messages-same concurrent-store-store-independent-technical*

**by** *metis*

**lemma** (**in** *imap*) *concurrent-operations-commute*:

**assumes** *xs prefix of i*  
**shows** *hb.concurrent-ops-commute (node-deliver-messages xs)*

**proof** –

**{ fix** *a b x y*  
**assume**  $(a, b) \in \text{set } (\text{node-deliver-messages } xs)$   
 $(x, y) \in \text{set } (\text{node-deliver-messages } xs)$   
 $\text{hb.concurrent } (a, b) (x, y)$   
**hence**  $\text{interp-msg } (a, b) \triangleright \text{interp-msg } (x, y) = \text{interp-msg } (x, y) \triangleright \text{interp-msg } (a, b)$   
**apply**(*unfold interp-msg-def, case-tac b; case-tac y;*  
*simp add: create-create-commute delete-delete-commute append-append-commute*  
*create-append-commute create-expunge-commute create-store-commute*  
*expunge-expunge-commute hb.concurrent-def*)  
**using** *assms prefix-contains-msg apply (metis (full-types)*  
*create-id-valid create-delete-commute concurrent-create-delete-independent)*  
**using** *assms prefix-contains-msg apply (metis (full-types)*  
*create-id-valid create-delete-commute concurrent-create-delete-independent)*  
**using** *assms prefix-contains-msg apply (metis*  
*append-id-valid append-delete-ids-imply-messages-same*  
*concurrent-append-delete-independent-technical delete-append-commute)*  
**using** *assms prefix-contains-msg apply (metis*  
*concurrent-expunge-delete-independent expunge-id-valid imap.delete-expunge-commute*  
*imap-axioms)*  
**using** *assms prefix-contains-msg apply (metis*  
*concurrent-store-delete-independent delete-store-commute store-id-valid)*  
**using** *assms prefix-contains-msg apply (metis*  
*append-id-valid append-delete-ids-imply-messages-same*  
*concurrent-append-delete-independent-technical delete-append-commute)*  
**using** *assms prefix-contains-msg apply (metis*  
*append-id-valid expunge-id-valid append-expunge-ids-imply-messages-same*

```

    concurrent-append-expunge-independent-technical append-expunge-commute)
using assms prefix-contains-msg apply (metis
  append-id-valid append-store-commute concurrent-append-store-independent store-id-valid)
using assms prefix-contains-msg apply (metis
  concurrent-expunge-delete-independent expunge-id-valid delete-expunge-commute)
using assms prefix-contains-msg apply (metis
  append-expunge-commute append-id-valid concurrent-append-expunge-independent
  expunge-id-valid)
using assms prefix-contains-msg apply (metis
  expunge-id-valid expunge-store-commute imap.concurrent-store-expunge-independent
  imap-axioms store-id-valid)
using assms prefix-contains-msg apply (metis
  concurrent-store-delete-independent delete-store-commute store-id-valid)
using assms prefix-contains-msg apply (metis
  append-id-valid append-store-commute imap.concurrent-append-store-independent imap-axioms
  store-id-valid)
using assms prefix-contains-msg apply (metis
  expunge-id-valid expunge-store-commute imap.concurrent-store-expunge-independent
  imap-axioms store-id-valid)
using assms prefix-contains-msg by (metis concurrent-store-store-independent store-id-valid
  store-store-commute)
} thus ?thesis
  by(fastforce simp: hb.concurrent-ops-commute-def)
qed

theorem (in imap) convergence:
  assumes set (node-deliver-messages xs) = set (node-deliver-messages ys)
  and xs prefix of i
  and ys prefix of j
  shows apply-operations xs = apply-operations ys
  using assms by(auto simp add: apply-operations-def intro: hb.convergence-ext
  concurrent-operations-commute node-deliver-messages-distinct hb-consistent-prefix)

context imap begin

sublocale sec: strong-eventual-consistency weak-hb hb interp-msg
  λops.∃xs i. xs prefix of i ∧ node-deliver-messages xs = ops λx.({},{}))
  apply(standard; clarsimp simp add: hb-consistent-prefix node-deliver-messages-distinct
  concurrent-operations-commute)
  apply(metis (no-types, lifting) apply-operations-def bind.bind-lunit not-None-eq
  hb.apply-operations-Snoc kleisli-def apply-operations-never-fails interp-msg-def)
  using drop-last-message apply blast
  done

end
end

```

## References

- [1] V. B. F. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford. A framework for establishing Strong Eventual Consistency for Conflict-free Replicated Datatypes. *Archive of Formal Proofs*, 2017. <http://isa-afp.org/entries/CRDT.html>.
- [2] V. B. F. Gomes, M. Kleppmann, D. P. Mulligan, and A. R. Beresford. Verifying Strong Eventual Consistency in Distributed Systems. *ArXiv e-prints*, 2017.
- [3] T. Jungnickel, L. Oldenburg, and M. Loibl. Designing a Planetary-Scale IMAP Service with Conflict-free Replicated Data Types. In *21th International Conference on Principles of Distributed Systems (OPODIS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- [4] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical report, 2011.
- [5] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free Replicated Data Types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS'11*, pages 386–400, 2011.