The IMAP CmRDT

Tim Jungnickel, Lennart Oldenburg, Matthias Loibl

August 16, 2018

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

1 Preface 1

1.1 The IMAP CmRDT 2

1.2 Proof Guide 2

2 IMAP-CRDT Definitions 3

3 Commutativity of IMAP Commands 5

4 Proof Helpers 7

5 Independence of IMAP Commands 11

6 Convergence of the IMAP-CRDT 23

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 $\mathcal{ID}$). 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 $\mathcal{ID}$ 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 CRDT

1: **payload map** \( u : \mathcal{N} \rightarrow \mathcal{P}(\mathcal{ID}) \times \mathcal{P}(\mathcal{M}) \) \( \triangleright \{ \text{foldername } f \mapsto (\{ \text{tag } t \}, \{ \text{msg } m \}), \ldots \} \)

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: \( u(f) \mapsto (u(f)_1 \cup \{m\}, u(f)_2 \cup \{m\}) \)

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.

```plaintext
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 ⇒ ('id set × 'id set)

definition op-elem :: ('id, 'a) operation ⇒ 'a where
  op-elem oper ≡ case oper of
  Create i e ⇒ e |
  Delete is e ⇒ e |
  Append i e ⇒ e |
  Expunge e mo i ⇒ e |
  Store e mo i ⇒ e

definition interpret-op :: ('id, 'a) operation ⇒ ('id, 'a) state ⇒ ('id, 'a) state
(((-) [0] 1000) where
  interpret-op oper state ≡
  let metadata = fst (state (op-elem oper));
  files = snd (state (op-elem oper));
  after = case oper of
    Create i e ⇒ (metadata ∪ {i}, files) |
    Delete is e ⇒ (metadata − is, files − is) |
    Append i e ⇒ (metadata, files ∪ {i}) |
    Expunge e mo i ⇒ (metadata ∪ {i}, files − {mo}) |
    Store e mo i ⇒ (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.

```plaintext
definition valid-behaviours :: ('id, 'a) state ⇒ 'id × ('id, 'a) operation ⇒ bool where
  valid-behaviours state msg ≡
  case msg of
    (i, Create j e) ⇒ i = j |
    (i, Delete is e) ⇒ is = fst (state e) ∪ snd (state e) |
    (i, Append j e) ⇒ i = j |
    (i, Expunge e mo j) ⇒ i = j ∧ mo ∈ snd (state e) |
```

4
\[(i, \text{Store} \ e \ \text{mo} \ j) \Rightarrow i = j \land \text{mo} \in \text{snd} \ (\text{state} \ e)\]

locale imap = network-with-constrained-ops \cdot 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 \(i \notin \text{is}\)
  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 (auto simp add: interpret-op-def kleisli-def, fastforce)

lemma (in imap) delete-append-commute:
  assumes \(i \notin \text{is}\)
  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 \(i \notin \text{is}\)
  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 /∈ is
  shows ⟨Delete is e1⟩ ▷ ⟨Store e2 mo i⟩ = ⟨Store e2 mo i⟩ ▷ ⟨Delete is e1⟩
using assms by (auto simp add: interpret-op-def kleisli-def op-elem-def, fastforce)

lemma (in imap) append-append-commute:
  shows ⟨Append i1 e1⟩ ▷ ⟨Append i2 e2⟩ = ⟨Append i2 e2⟩ ▷ ⟨Append i1 e1⟩
by (auto simp add: interpret-op-def kleisli-def op-elem-def)

lemma (in imap) append-expurge-commute:
  assumes i1 ≠ mo
  shows ((⟨Append i1 e1⟩ ▷ ⟨Expurge e2 mo i2⟩) = (⟨Expurge e2 mo i2⟩ ▷ ⟨Append i1 e1⟩)
proof
  fix x
  show ((⟨Append i1 e1⟩ ▷ ⟨Expurge e2 mo i2⟩) x = (⟨Expurge e2 mo i2⟩ ▷ ⟨Append i1 e1⟩) x
    using assms by (auto simp add: interpret-op-def kleisli-def op-elem-def)
qed

lemma (in imap) append-store-commute:
  assumes i1 ≠ mo
  shows ((⟨Append i1 e1⟩ ▷ ⟨Store e2 mo i2⟩) = (⟨Store e2 mo i2⟩ ▷ ⟨Append i1 e1⟩)
proof
  fix x
  show ((⟨Append i1 e1⟩ ▷ ⟨Store e2 mo i2⟩) x = (⟨Store e2 mo i2⟩ ▷ ⟨Append i1 e1⟩) x
    using assms by (auto simp add: interpret-op-def kleisli-def op-elem-def)
qed

lemma (in imap) expunge-expurge-commute:
  shows ((⟨Expurge e1 mo1 i1⟩ ▷ ⟨Expurge e2 mo2 i2⟩) = (⟨Expurge e2 mo2 i2⟩ ▷ ⟨Expurge e1 mo1 i1⟩)
proof
  fix x
  show ((⟨Expurge e1 mo1 i1⟩ ▷ ⟨Expurge e2 mo2 i2⟩) x
    = (⟨Expurge e2 mo2 i2⟩ ▷ ⟨Expurge e1 mo1 i1⟩) x
    by (auto simp add: interpret-op-def kleisli-def op-elem-def) qed

lemma (in imap) expunge-store-commute:
  assumes i1 ≠ mo2 and i2 ≠ mo1
  shows ((⟨Expurge e1 mo1 i1⟩ ▷ ⟨Store e2 mo2 i2⟩) = (⟨Store e2 mo2 i2⟩ ▷ ⟨Expurge e1 mo1 i1⟩)
proof
  fix x
  show ((⟨Expurge e1 mo1 i1⟩ ▷ ⟨Store e2 mo2 i2⟩) x = (⟨Store e2 mo2 i2⟩ ▷ ⟨Expurge e1 mo1 i1⟩) 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

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)
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 (λx. 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

— 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 ≠ e' ==> 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 ≠ e' ==> 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 ≠ e' ==> 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 ≠ e' ==> 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 /∈ set (added-ids [Deliver (i, Expunge e mo i2)]) e
  shows i1 ≠ i2
  using assms by simp

lemma (in imap) apply-operations-added-ids:
  assumes es prefix of j
  and apply-operations es = Some f
  shows fst (f x) ⊆ set (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

10
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

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 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

subsection {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—dependent

imports
IMAP—def
IMAP—proof—helpers

begin

lemma (in imap) Broadcast-Expunge-Deliver-prefix-closed:
assumes \( xs \in \text{prefix of } j \)
shows \( \text{Deliver}(\text{mo}, \text{Append } mo \ e) \in set \ xs \lor \) 
(\( \exists \text{mo}2. \text{Deliver}(\text{mo}, \text{Store } mo2 \ mo) \in set \ xs \))
proof |
obtain y where apply-operations \( xs = \text{Some } y \)
using assms broadcast-only-valid-msgs by blast
moreover hence \( mo \in \text{snd}(y \ e) \)
using broadcast-only-valid-msgs[of \( xs \text{ (i, Expunge e mo i) j } \)]
valid-behaviours-def[of \( y \text{ (i, Expunge e mo i) } \)] 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 \( xs \in \text{prefix of } j \)
shows \( \text{Deliver}(\text{mo}, \text{Append } mo \ e) \in set \ xs \lor \) 
(\( \exists \text{mo}2. \text{Deliver}(\text{mo}, \text{Store } mo2 \ mo) \in set \ xs \))
proof |
obtain y where apply-operations \( xs = \text{Some } y \)
using assms broadcast-only-valid-msgs by blast
moreover hence \( mo \in \text{snd}(y \ e) \)
using broadcast-only-valid-msgs[of \( xs \text{ (i, Store e mo i) j } \)]
valid-behaviours-def[of \( y \text{ (i, Store e mo i) } \)] 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 \( xs \text{ prefix of } j \)
and \( i \in \text{set (added-ids } xs \ e) \)
shows \( \text{Deliver}(i, \text{Create } i \ e) \in set \ xs \lor \) 
(\( \exists \text{mo}. \text{Deliver}(i, \text{Expunge e mo i}) \in set \ xs \))
using assms proof (induct \( xs \text{ rule: rev-induct, clarsimp} \)
case (snoc \( x \ xs \)) thus \(?\)case
proof (cases \( x, \text{force} \)
case \( X: \text{(Deliver } e') \)
moreover obtain \( a b \text{ where } e' = (a, b) \text{ by force} \)
ultimately show \(?\)thesis
using snoc apply (case-tac \( b; \text{clarify} \)
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

12
in-set-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

lemma (in imap) Broadcast-Deliver-prefix-closed:
assumes xs @ [Broadcast (r, Delete i x 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-ids assms)
qed

lemma (in imap) concurrent-create-delete-independent-technical:
assumes i ∈ is
and zs prefix of j
and (i, Create i e) ∈ set (node-deliver-messages zs)
and (ir, Delete is e) ∈ set (node-deliver-messages zs)
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-elm-to-carriers prefix-of-appendD)
have f4: ∀ mo. Deliver (i, Expunge e mo i) ∉ set pre using f1 P

13
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 \( \forall m o . \text{Deliver} (i, \text{Store e mo i}) \notin \text{set pre} \) 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 \( \text{thesis} \) using \( f2 \) \( f3 \) \( f4 \) \( P \) events-in-local-order hb-deliver by blast

qed

lemma (in imap) concurrent-store-expunge-independent-technical:
assumes \( x s \) prefix of \( j \)
  and \((i, \text{Store e mo i}) \in \text{set (node-deliver-messages xs)}\)
  and \((r, \text{Expunge e i r}) \in \text{set (node-deliver-messages xs)}\)
shows \( \text{hb} (i, \text{Store e mo i}) (r, \text{Expunge e i r}) \)

proof –
  obtain \( \text{pre k where} \ P ; \text{pre} @ [(\text{Broadcast}) (r, \text{Expunge e i r})] \) prefix of \( k \)
  using \( \text{assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast} \)
  moreover hence \( f1 : \text{Deliver} (i, \text{Append i e}) \in \text{set pre} \) \( \forall \)
  \((\exists \text{mo2} . \text{Deliver} (i, \text{Store e mo2 i}) \in \text{set pre})\)
  using \( \text{Broadcast-Expunge-Deliver-prefix-closed assms(1) by auto} \)
  hence \( f2 : \text{Deliver} (i, \text{Append i e}) \notin \text{set (history k)} \)
  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 \( \text{mo2 :: 'a where} \)
  \( \text{Deliver} (i, \text{Store e mo2 i}) \in \text{set (history k)} \) using \( f2 \)
  using \( P \) prefix-elem-to-carriers by blast
  hence \( \text{Deliver} (i, \text{Store e mo i}) \in \text{set (history k)} \) using \( \text{assms f1 f2 P} \)
  by (metis fst-conv msg-id-unique network.delivery-has-a-cause network-axioms

  prefix-msg-in-history)
  then show \( \text{thesis} \)
  using \( \text{hb.intro (2) events-in-local-order f1 f2 P} \)
  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 \( x s \) prefix of \( j \)
  and \((i, \text{Store e1 mo2 i}) \in \text{set (node-deliver-messages xs)}\)
  and \((r, \text{Expunge e mo r}) \in \text{set (node-deliver-messages xs)}\)
shows \( \text{mo2} \neq r \)

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

  qed
by moura

obtain pre :: nat ⇒ ('a × ('a, 'b) operation) event ⇒ ('a × ('a, 'b) operation) event list

where pre: ∀ k op₁. (∃ op₂. op₂ ⊤ [op₁] prefix of k) = ( ∀ k op₁ ⊤ [op₁] prefix of k)

by moura

hence ∃: ∀ e n. e ∉ set (history n) ∨ pre n e ⊤ [e] prefix of n

using events-before-exist by simp

hence ∃: ∀ e n. e ∉ set (history n) ∨ pre n e ⊤ [e] prefix of n

prefix of oid (i, Store e1 mo2 i)

using oid assms(1) assms(2) prefix-msg-in-history by blast

have ∃: ∀ e n. e ∉ set (history n) ∨ pre n e ⊤ [e] prefix of n

by (metis (no-types) oid f1 fst-conv msg-id-unique old.proc.inject operation.distinct(15))

have ∃: ∀ e n. e ∉ set (history n) ∨ pre n e ⊤ [e] prefix of n

Deliver (r, Append r e1) ∉ set (history (oid (i, Store e1 mo2 i)))

by (metis (no-types) oid f1 fst-conv msg-id-unique old.proc.inject operation.distinct(19))

thus ?thesis using oid 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 ∈ is

and zs prefix of j

and (i, Store e mo i) ∈ set (node-deliver-messages zs)

and (ir, Delete is e) ∈ set (node-deliver-messages zs)

shows hb (i, Store e mo i) (ir, Delete is e)

proof –

have f1: Deliver (i, Store e mo i) ∈ set (history j) using assms prefix-msg-in-history by auto

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(1) by auto

have f3: Deliver (i, Create i e) ∉ 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: Deliver (i, Append i e) ∉ 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 ∀ mo . Deliver (i, Expunge e mo i) ∉ 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 Deliver (i, Store e mo i) ∈ 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 events-in-local-order hb-deliver by blast

qed
lemma (in imap) concurrent-append-delete-independent-technical:
assumes i ∈ is
and xs prefix of j
and (i, Append i e) ∈ set (node-deliver-messages xs)
and (ir, Delete is e) ∈ set (node-deliver-messages xs)
shows hh (i, Append i e) (ir, Delete is e)

proof –
obtain pre k where P: pre k [Broadcast (ir, Delete is e)] prefix of k
using assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast

hence f1: 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(1) by auto

hence Deliver (i, Append i e) ∈ set pre using assms P f1
by (metis (no-types, hide-lams) delivery-has-a-cause events-in-local-order fst-conv
hb-broadcast-exists 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, Append i e) ∈ set (node-deliver-messages xs)
and (r, Expunge e mo r) ∈ set (node-deliver-messages xs)
shows hh (i, Append i e) (r, Expunge e mo r)

proof –
obtain pre k where P: pre k [Broadcast (r, Expunge e mo r)] prefix of k
using assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast

hence f1: Deliver (mo, Append mo e) ∈ set pre ∨
(∃ mo . Deliver (mo, Expunge mo i) ∈ set pre) ∨
(∃ mo . Deliver (mo, Store mo i) ∈ set pre)

using Broadcast-Expunge-Deliver-prefix-closed assms(1) by auto

hence (∀ mo2 . Deliver (mo, Store mo2 mo) ∉ set pre) using P assms

proof –
have Deliver (mo, Append mo e) ∈ set (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-carrers prefix-of-appendD)

qed

thus ?thesis
using hh.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, Append i e) ∈ set (node-deliver-messages xs)
and (r, Store e mo r) ∈ set (node-deliver-messages xs)
shows $hb\ (i,\ Append \ i\ e)\ (r,\ Store\ e\ mo\ r)$

proof –

obtain $pre\ k$ where $pre:\ pre@[(\text{Broadcast}\ (r,\ Store\ e\ mo\ r))]\ prefix\ of\ k$

using assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast

moreover hence $f1:\ \text{Deliver}\ (mo,\ Append\ mo\ e)\ \in\ set\ pre$ ∨

($\exists\ mo2.\ \text{Deliver}\ (mo,\ Store\ e\ mo2\ mo)\ \in\ set\ pre$)

using Broadcast-Store-Deliver-prefix-closed assms(1) by auto

have $f2:\ \text{Deliver}\ (i,\ Append\ i\ e)\ \in\ set\ (\text{history}\ j)$

by (meson assms network.prefix-msg-in-history network-axioms)

then show $?thesis$ using assms

proof –

have $f1:\ \forall e.\ e\ \notin\ set\ pre\ \lor\ e\ \in\ set\ (\text{history}\ k)$

using assms prefix-elem-to-carriers by blast

have $f2:\ \text{Deliver}\ (i,\ Expunge\ e\ mo\ i)\ \in\ set\ (\text{history}\ j)$

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-expunge-delete-independent-technical:

assumes $i\ \in\ is$

and $xs\ prefix\ of\ j$

and $(i,\ Expunge\ e\ mo\ i)\ \in\ set\ (\text{node-deliver-messages}\ xs)$

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

shows $hb\ (i,\ Expunge\ e\ mo\ i)\ (ir,\ Delete\ is\ e)$

proof –

obtain $pre\ k$ where $pre:\ pre@[(\text{Broadcast}\ (ir,\ Delete\ is\ e))]\ prefix\ of\ k$

using assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast

moreover hence $A:\ \text{Deliver}\ (i,\ Create\ i\ e)\ \in\ set\ pre$ ∨

$\text{Deliver}\ (i,\ Append\ i\ e)\ \in\ set\ pre$ ∨

($\exists\ mo.\ \text{Deliver}\ (i,\ Expunge\ e\ mo\ i)\ \in\ set\ pre$) ∨

($\exists\ mo.\ \text{Deliver}\ (i,\ Store\ e\ mo\ i)\ \in\ set\ pre$)

using Broadcast-Deliver-prefix-closed assms(1) by auto

hence $\text{Deliver}\ (i,\ Expunge\ e\ mo\ i)\ \in\ set\ pre$ using assms

proof –

have $f1:\ \forall e.\ e\ \notin\ set\ pre\ \lor\ e\ \in\ set\ (\text{history}\ k)$

using assms prefix-elem-to-carriers by blast

have $f2:\ \text{Deliver}\ (i,\ Expunge\ e\ mo\ i)\ \in\ set\ (\text{history}\ j)$

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

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

assumes $xs\ prefix\ of\ j$

and $(i,\ Store\ e\ mo\ i)\ \in\ set\ (\text{node-deliver-messages}\ xs)$

and $(r,\ Store\ e\ i\ r)\ \in\ set\ (\text{node-deliver-messages}\ xs)$

shows $hb\ (i,\ Store\ e\ mo\ i)\ (r,\ Store\ e\ i\ r)$

proof –


obtain \( pre \ k \) where \( P \): \( \text{pre}@[\text{Broadcast}(r, \text{Store} \ e \ i \ r)] \) prefix of \( k \)
using \( \text{assms} \) \( \text{delivery-has-a-cause} \) \( \text{events-before-exist} \) \( \text{prefix-msg-in-history} \) by \( \text{blast} \)

hence \( f1 : \forall e. e \notin \text{set} \ pre \ \lor e \in \text{set} \ (\text{history} \ k) \)
using \( \text{prefix-elem-to-carriers} \) by \( \text{blast} \)

have \( f2 : \text{Deliver}(i, \text{Append} \ e \ i) \in \text{set} \ pre \ \lor (\exists m02 . \text{Deliver}(i, \text{Store} \ m02 \ i) \in \text{set} \ pre) \)
using \( \text{Broadcast}-\text{Store}-\text{Deliver-prefix-closed} \) \( \text{assms}(1) \) \( P \) by \( \text{auto} \)

hence \( \text{Deliver}(i, \text{Store} \ m02 \ i) \in \text{set} \ \text{using} \ \text{Broadcast-Store-Deliver-prefix-closed assms} \)

then show \( \text{thesis} \)
using \( \text{hb.intro}(2) \) \( \text{events-in-local-order} \) \( P \) by \( \text{blast} \)

qed

lemma \( \text{(in imap)} \) \( \text{expunge-delete-tag-causality} \):
assumes \( i \in \text{is} \)
and \( \text{zs \ prefix \ of \ j} \)
and \( (i, \text{Expunge} \ e1 \ m0 \ i) \in \text{set} \ (\text{node-deliver-messages} \ \text{xs}) \)
and \( (ir, \text{Delete} \ is \ e2) \in \text{set} \ (\text{node-deliver-messages} \ \text{xs}) \)
and \( \text{prefix@}[\text{Broadcast}(ir, \text{Delete} \ is \ e2)] \) \( \text{prefix \ of \ k} \)
shows \( \text{Deliver}(i, \text{Expunge} \ e2 \ m0 \ i) \in \text{set} \ (\text{history} \ k) \)

proof –

have \( f1 : \text{Deliver}(i, \text{Append} \ e2 \ i) \notin \text{set} \ (\text{history} \ k) \) using \( \text{assms} \)
by \( \text{(metis \ \text{fst-conv} \ \text{msg-id-unique} \ \text{network.delivery-has-a-cause} \ \text{network-axioms \ old.prod.inject)} \)

operation \( \text{distinct}(15) \) \( \text{prefix-msg-in-history} \)

have \( f2 : \text{Deliver}(i, \text{Create} \ i \ e2) \notin \text{set} \ (\text{history} \ k) \) using \( \text{assms} \)
by \( \text{(metis \ \text{fst-conv} \ \text{msg-id-unique} \ \text{network.delivery-has-a-cause} \ \text{network-axioms \ old.prod.inject)} \)

operation \( \text{distinct}(5) \) \( \text{prefix-msg-in-history} \)

have \( f3 : \forall m0. \text{Deliver}(i, \text{Store} \ e2 \ m0 \ i) \notin \text{set} \ (\text{history} \ k) \) using \( \text{assms} \)
by \( \text{(metis \ \text{Pair-inject} \ \text{fst-conv} \ \text{msg-id-unique} \ \text{network.delivery-has-a-cause} \ \text{network-axioms} \ \text{operation_simps}(25) \ \text{prefix-msg-in-history})} \)

hence \( \exists m01. \text{Deliver}(i, \text{Expunge} \ e2 \ m01 \ i) \in \text{set} \ (\text{history} \ k) \) using \( \text{assms} \) \( f1 \) \( f2 \)
by \( \text{(meson \ \text{imap.Broadcast-Deliver-prefix-closed} \ \text{imap-axioms} \ \text{node-histories.prefix-of-appendD} \ \text{node-histories-axioms} \ \text{prefix-elem-to-carriers})} \)

then obtain \( m01 : \ \text{'}a \ \text{where} \)
\( \text{Deliver}(i, \text{Expunge} \ e2 \ m01 \ i) \in \text{set} \ (\text{history} \ k) \) by \( \text{blast} \)

then show \( \text{thesis} \) using \( \text{assms} \) \( f1 \) \( f2 \) \( f3 \)
by \( \text{(metis \ \text{fst-conv} \ \text{msg-id-unique} \ \text{network.delivery-has-a-cause} \ \text{network-axioms \ old.prod.inject)} \)

operation \( \text{inject}(4) \) \( \text{prefix-msg-in-history} \)

qed

lemma \( \text{(in imap)} \) \( \text{expunge-delete-ids-imply-messages-same} \):
assumes \( i \in \text{is} \)
and \( \text{zs \ prefix \ of \ j} \)
and \( (i, \text{Expunge} \ e1 \ m0 \ i) \in \text{set} \ (\text{node-deliver-messages} \ \text{xs}) \)
and \( (ir, \text{Delete} \ is \ e2) \in \text{set} \ (\text{node-deliver-messages} \ \text{xs}) \)
shows \( e1 = e2 \)
lemma \((\text{in imap})\) store-delete-ids-imply-messages-same:
assumes \(i \in \text{is} \)
and \(xs \) prefix of \(j \)
and \((i, \text{Store e1 mo i}) \in \text{set} (\text{node-deliver-messages} \; xs) \)
and \((ir, \text{Delete is e2}) \in \text{set} (\text{node-deliver-messages} \; xs) \)
shows \(e1 = e2 \)

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

qed

lemma \((\text{in imap})\) create-delete-ids-imply-messages-same:
assumes \(i \in \text{is} \)
and \(xs \) prefix of \(j \)
and \((i, \text{Create } i e1) \in \text{set (node-deliver-messages } xs)\)
and \((ir, \text{Delete is } e2) \in \text{set (node-deliver-messages } xs)\)
shows \(e1 = e2\)

\textbf{proof –}

\textbf{obtain} \(pre k\) \textbf{where} \(P:: \text{pre@[Broadcast (ir, Delete is e2)] prefix of } k\)

\textbf{using} \(\text{assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast}\)

\textbf{have} \(f1:: \text{Deliver (i, Append i e2) } \notin \text{set (history } k)\)

\textbf{by (metis \text{assms(2)} \text{assms(3)} delivery-has-a-cause \text{fst-conv network.msg-id-unique network.prefix-msg-in-history network-axioms operation.distinct(3)} prod.inject)}

\textbf{have} \(f2:: \forall \text{ mo. Deliver (i, Expunge e2 mo i) } \notin \text{set (history } k)\)

\textbf{by (metis \text{assms(2)} \text{assms(3)} \text{fst-conv msg-id-unique network.delivery-has-a-cause network-axioms}}

\textit{old.prod.inject operation.distinct(5) prefix-msg-in-history)}

\textbf{have} \(f3:: \forall \text{ mo. Deliver (i, Store e2 mo i) } \notin \text{set (history } k)\)

\textbf{by (metis \text{Pair-inject assms(2) assms(3) delivery-has-a-cause \text{fst-conv msg-id-unique operation.distinct(7)} prefix-msg-in-history)}

\textbf{hence} \(\text{Deliver (i, Create i e2) } \in \text{set pre using assms } P f2 f1 \text{ imap-axioms}\)

\textbf{by (meson imap.Broadcast-Deliver-prefix-closed prefix-elem-to-carrors prefix-of-appendD)}

\textbf{then show} \(?\text{thesis using } f1 f2 f3\)

\textbf{by (metis \text{no-types, lifting) P assms(2) assms(3) delivery-has-a-cause \text{fst-conv msg-id-unique}}

\textit{node-histories.prefix-of-appendD node-histories-axioms old.prod.inject operation.inject(1)}

\textit{prefix-elem-to-carrors prefix-msg-in-history)}

\textbf{qed}

\textbf{lemma} \((\text{in imap}) \text{ append-delete-ids-imply-messages-same:}\)

\textbf{assumes} \(i \in \text{is}\)

\textbf{and} \(xs \text{ prefix of } j\)

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

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

\textbf{shows} \(e1 = e2\)

\textbf{proof –}

\textbf{obtain} \(pre k\) \textbf{where} \(P:: \text{pre@[Broadcast (ir, Delete is e2)] prefix of } k\)

\textbf{using} \(\text{assms delivery-has-a-cause events-before-exist prefix-msg-in-history by blast}\)

\textbf{hence} \(f1:: \forall e. e \in \text{set pre } \implies e \in \text{set (history } k)\) \textbf{using prefix-elem-to-carrors by blast}

\textbf{have} \(f2:: \text{Deliver (i, Create i e2) } \notin \text{set pre using P f1}\)

\textbf{by (metis \text{assms(2)} \text{assms(3)} \text{fst-conv msg-id-unique network.delivery-has-a-cause network-axioms}}

\textit{old.prod.inject operation.distinct(3) prefix-msg-in-history)}

\textbf{moreover have} \(D1:: \forall \text{ mo. Deliver (i, Expunge e2 mo i) } \notin \text{set pre using P f1}\)

\textbf{by (metis \text{Pair-inject assms(2) assms(3) \text{fst-conv msg-id-unique network.delivery-has-a-cause network-axioms}}}

\textit{operation.distinct(15) prefix-msg-in-history)}

\textbf{moreover have} \(D2:: \forall \text{ mo. Deliver (i, Store e2 mo i) } \notin \text{set pre using P f1}\)

\textbf{by (metis \text{Pair-inject assms(2) assms(3) \text{fst-conv msg-id-unique network.delivery-has-a-cause network-axioms}}}

\textit{operation.simps(23) prefix-msg-in-history)}

\textbf{moreover hence} \(\text{Deliver (i, Append i e2) } \in \text{set pre}\)
using \( P \ D1 \ D2 \ f2 \ \text{assms}(1) \) \text{Broadcast-Deliver-prefix-closed by blast}

moreover have \( \text{Deliver} \ (i, \text{Append} \ i \ e1) \in \text{set} \ \text{(history} \ j) \)

using \( \text{assms}(2) \ \text{assms}(3) \) \text{prefix-msg-in-history by blast}

ultimately show \( ?\text{thesis using assms} \)

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) \text{append-expunge-ids-imply-messages-same}:

assumes \( i = \text{mo} \)

and \( \text{xs prefix of} \ j \)

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

and \( (r, \text{Expunge} \ e2 \ \text{mo} \ r) \in \text{set} \ \text{(node-deliver-messages} \ \text{xs}) \)

shows \( e1 = e2 \)

proof –

obtain \( \text{pre k where} \ \text{pre: pre} @[\text{Broadcast} \ (r, \text{Expunge} \ e2 \ \text{mo} \ r)] \ \text{prefix of} \ k \)

using \( \text{assms} \) \text{delivery-has-a-cause events-before-exist prefix-msg-in-history by blast}

moreover hence \( \text{Deliver} \ (\text{mo}, \text{Append} \ \text{mo} \ e2) \in \text{set} \ \text{pre} \lor \)

\( (\exists \ \text{mo2} . \text{Deliver} \ (\text{mo}, \text{Store} \ e2 \ \text{mo2} \ \text{mo}) \in \text{set} \ \text{pre} \)

using \( \text{Broadcast-Expunge-Deliver-prefix-closed assms}(1) \)

by (meson imap.Broadcast-Deliver-prefix-closed imap-axioms)

hence \( \text{Deliver} \ (i, \text{Append} \ i \ e2) \in \text{set} \ \text{pre} \) using \( \text{assms} \)

by (metis \( \text{no-types}, \text{lifting} \) \text{pre delivery-has-a-cause} fst-conv \( \text{hb-broadcast-exists}1 \)

msg-id-unique network \( \text{hb-deliver network prefix-msg-in-history network-axioms node-histories.events-in-local-order node-histories-axioms operation.distinct}(17) \)

prod.inject)

moreover have \( \text{Deliver} \ (i, \text{Append} \ i \ e1) \in \text{set} \ \text{(history} \ j) \)

using \( \text{assms}(2) \ \text{assms}(3) \) \text{prefix-msg-in-history by blast}

ultimately show \( ?\text{thesis} \)

by (metis \( \text{no-types}, \text{lifting} \) \text{fst-conv network delivery-has-a-cause network.msg-id-unique network-axioms operation.inject}(3) \text{ prefix-elem-to-carriers prefix-of-appendD prod.inject})

qed

lemma (in imap) \text{append-store-ids-imply-messages-same}:

assumes \( i = \text{mo} \)

and \( \text{xs prefix of} \ j \)

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

and \( (r, \text{Store} \ e2 \ \text{mo} \ r) \in \text{set} \ \text{(node-deliver-messages} \ \text{xs}) \)

shows \( e1 = e2 \)

proof –

obtain \( \text{pre k where} \ P; \ \text{pre}@[\text{Broadcast} \ (r, \text{Store} \ e2 \ \text{mo} \ r)] \ \text{prefix of} \ k \)

using \( \text{assms} \) \text{delivery-has-a-cause events-before-exist prefix-msg-in-history by blast}

moreover hence \( A: \text{Deliver} \ (\text{mo}, \text{Append} \ \text{mo} \ e2) \in \text{set} \ \text{pre} \lor \)

\( (\exists \ \text{mo2} . \text{Deliver} \ (\text{mo}, \text{Store} \ e2 \ \text{mo2} \ \text{mo}) \in \text{set} \ \text{pre} \)

using \( \text{Broadcast-Store-Deliver-prefix-closed assms}(1) \)

by (meson imap.Broadcast-Deliver-prefix-closed imap-axioms)

have \( \text{fl: Deliver} \ (i, \text{Append} \ i \ e1) \in \text{set} \ \text{(history} \ j) \)

using \( \text{assms}(2) \ \text{assms}(3) \) \text{prefix-msg-in-history by blast}

hence \( \text{Deliver} \ (i, \text{Append} \ i \ e2) \in \text{set} \ \text{pre} \) using \( \text{assms} \) \( P \ A \)
by (metis Pair-inject assms(1) P delivery-has-a-cause fst-conv msg-id-unique
operation.simps(23) prefix-elem-to-carriers prefix-of-appendD)
then show \textbf{?thesis} using \textbf{fl}
by (metis P delivery-has-a-cause fst-conv msg-id-unique
node-histories.prefix-of-appendD node-histories-axioms operation.inject(3)
prefix-elem-to-carriers prod.inject)
qed

\textbf{lemma} (in \textbf{imap}) \textbf{expunge-store-ids-imply-messages-same}:
assumes \textbf{xs prefix of j}
and \textbf{(i, Store e1 mo i) \in set (node-deliver-messages xs)}
and \textbf{(r, Expunge e2 i r) \in set (node-deliver-messages xs)}
shows \textbf{e1 = e2}
proof –
obtain pre k where \textbf{P; pre \emptyset[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 \textbf{pprefix : pre prefix of k}
using \textbf{P by blast}
have \textbf{A: Deliver (i, Append i e2) \in set pre \lor}
(\exists \textbf{mo2} \cdot Deliver (i, Store e2 mo2 i) \in set pre)
using Broadcast-Expunge-Deliver-prefix-closed assms(1) \textbf{P by blast}
have \textbf{Deliver (i, Store e2 mo i) \in set pre using assms A P}
proof –
obtain op1 :: 'a \times ('a, 'b) operation \Rightarrow \textbf{nat where}
\textbf{fl1: Broadcast (i, Store e1 mo i) \in 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 \textbf{fl1 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 \textbf{Deliver (i, Store e1 mo i) \in set (history j)}
using assms(1) assms(2) prefix-msg-in-history by blast
ultimately show \textbf{?thesis using assms P}
by (metis delivery-has-a-cause fst-conv msg-id-unique operation.inject(5)
prefix-elem-to-carriers prefix-of-appendD prod.inject)
qed

\textbf{lemma} (in \textbf{imap}) \textbf{store-store-ids-imply-messages-same}:
assumes \textbf{xs prefix of j}
and \textbf{(i, Store e1 mo i) \in set (node-deliver-messages xs)}
and \textbf{(r, Store e2 i r) \in set (node-deliver-messages xs)}
shows \textbf{e1 = e2}
proof –
obtain pre k where \textbf{P; pre \emptyset[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 \textbf{A: Deliver (i, Append i e2) \in set pre \lor}
(\exists \textbf{mo2} \cdot Deliver (i, Store e2 mo2 i) \in set pre)
using Broadcast-Store-Deliver-prefix-closed assms(1) by blast
have $\forall e.\ e \notin set\ pre \land e \in set\ (\text{history}\ k)$

using $P$ prefix-elem-to-carriers by auto

hence Deliver $(i,\ \text{Store}\ e_2\ \text{mo}\ i) \in set\ pre$

by (metis $A\ \text{assms}(1)\ \text{assms}(2)\ \text{delivery}-\text{has-a-cause}\ \text{fst-conv}\ \text{msg-id-unique}\\ 
\text{operation}.\text{distinct}(17)\ \text{operation}.\text{inject}(5)\ \text{prefix}-\text{msg-in-history}\ \text{prod}.\text{inject})$

moreover have Deliver $(i,\ \text{Store}\ e_1\ \text{mo}\ i) \in set\ (\text{history}\ j)$

using $\text{assms}(1)\ \text{assms}(2)\ \text{prefix}-\text{msg-in-history}\ \text{by}\ \text{auto}$

ultimately show $?\text{thesis}\ using\ \text{assms}$

by (metis Pair-inject delivery-has-a-cause msg-id-unique operation.\text{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

$\text{IMAP-\text{proof}}$

imports

$\text{IMAP-def}$

$\text{IMAP-\text{proof-commute}}$

$\text{IMAP-\text{proof-helpers}}$

$\text{IMAP-\text{proof-independent}}$

begin

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

assumes $\neg\ hb\ (i,\ \text{Create}\ i\ e_1)\ (ir,\ \text{Delete}\ is\ e_2)$

and $\neg\ hb\ (ir,\ \text{Delete}\ is\ e_2)\ (i,\ \text{Create}\ i\ e_1)$

and $zs$ prefix of $j$

and $(i,\ \text{Create}\ i\ e_1) \in set\ \text{(node-deliver-messages}\ xs)$

and $(ir,\ \text{Delete}\ is\ e_2) \in set\ \text{(node-deliver-messages}\ xs)$

shows $i \notin is$

using $\text{assms create-delete-ids-\text{imply}-messages-same}\ concurrent-create-delete-independent-technical$

by fastforce

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

assumes $\neg\ hb\ (i,\ \text{Append}\ i\ e_1)\ (ir,\ \text{Delete}\ is\ e_2)$

and $\neg\ hb\ (ir,\ \text{Delete}\ is\ e_2)\ (i,\ \text{Append}\ i\ e_1)$

and $zs$ prefix of $j$

and $(i,\ \text{Append}\ i\ e_1) \in set\ \text{(node-deliver-messages}\ xs)$

and $(ir,\ \text{Delete}\ is\ e_2) \in set\ \text{(node-deliver-messages}\ xs)$

shows $i \notin is$

using $\text{assms append-delete-ids-\text{imply}-messages-same}\ concurrent-append-delete-independent-technical$

by fastforce
corollary (in imap) concurrent-append-expunge-independent:
assumes ¬hb (i, Append i e1) (r, Expunge e2 mo r)
and ¬hb (r, Expunge e2 mo r) (i, Append i e1)
and xs prefix of j
and (i, Append i e1) ∈ set (node-deliver-messages xs)
and (r, Expunge e2 mo r) ∈ set (node-deliver-messages xs)
shows i ≠ mo
using assms append-expunge-ids-imply-messages-same concurrent-append-expunge-independent-technical
by fastforce

corollary (in imap) concurrent-append-store-independent:
assumes ¬hb (i, Append i e1) (r, Store e2 mo r)
and ¬hb (r, Store e2 mo r) (i, Append i e1)
and xs prefix of j
and (i, Append i e1) ∈ set (node-deliver-messages xs)
and (r, Store e2 mo r) ∈ set (node-deliver-messages xs)
shows i ≠ mo
using assms append-store-ids-imply-messages-same concurrent-append-store-independent-technical
by fastforce

corollary (in imap) concurrent-expunge-delete-independent:
assumes ¬hb (i, Expunge e1 mo i) (ir, Delete is e2)
and ¬hb (ir, Delete is e2) (i, Expunge e1 mo i)
and xs prefix of j
and (i, Expunge e1 mo i) ∈ set (node-deliver-messages xs)
and (ir, Delete is e2) ∈ set (node-deliver-messages xs)
shows i /∈ is
using assms expunge-delete-ids-imply-messages-same concurrent-expunge-delete-independent-technical
by fastforce

corollary (in imap) concurrent-store-delete-independent:
assumes ¬hb (i, Store e1 mo i) (ir, Delete is e2)
and ¬hb (ir, Delete is e2) (i, Store e1 mo i)
and xs prefix of j
and (i, Store e1 mo i) ∈ set (node-deliver-messages xs)
and (ir, Delete is e2) ∈ set (node-deliver-messages xs)
shows i /∈ is
using assms store-delete-ids-imply-messages-same concurrent-store-delete-independent-technical
by fastforce

corollary (in imap) concurrent-store-expunge-independent:
assumes ¬hb (i, Store e1 mo i) (r, Expunge e2 mo2 r)
and ¬hb (r, Expunge e2 mo2 r) (i, Store e1 mo i)
and xs prefix of j

24
and \((i, \text{Store } e1 \text{ mo } i) \in \text{set (node-deliver-messages } xs)\)
and \((r, \text{Expunge } e2 \text{ mo } 2 \text{ r}) \in \text{set (node-deliver-messages } xs)\)
shows \(i \neq \text{mo2} \land r \neq \text{mo}\)
using assms expunge-store-ids-imply-messages-same concurrent-store-expunge-independent-technical2

**corollary (in imap) concurrent-store-store-independent:**
assumes \(\neg \text{hb} (i, \text{Store } e1 \text{ mo } i) \land \neg \text{hb} (r, \text{Store } e2 \text{ mo } 2 \text{ r})\)
and \(\neg \text{hb} (r, \text{Store } e2 \text{ mo } 2 \text{ r}) \land \neg \text{hb} (i, \text{Store } e1 \text{ mo } i)\)
and \(xs \text{ prefix of } j\)
and \((i, \text{Store } e1 \text{ mo } i) \in \text{set (node-deliver-messages } xs)\)
and \((r, \text{Store } e2 \text{ mo } 2 \text{ r}) \in \text{set (node-deliver-messages } xs)\)
shows \(i \neq \text{mo2} \land r \neq \text{mo}\)
using assms store-store-ids-imply-messages-same concurrent-store-store-independent-technical

by **metis**

**lemma (in imap) concurrent-operations-commute:**
assumes \(xs \text{ prefix of } i\)
shows \(\text{hb}.\text{concurrent-ops-commute (node-deliver-messages } xs)\)
proof \(\quad \)
\{ \fix a b x y \\
\quad \text{assume } (a, b) \in \text{set (node-deliver-messages } xs) \\
\quad (x, y) \in \text{set (node-deliver-messages } xs) \\
\quad \text{hb}.\text{concurrent } (a, b) (x, y) \\
\quad \text{hence } \text{interp-msg} (a, b) \triangleright \text{ interp-msg } (x, y) = \text{ interp-msg } (x, y) \triangleright \text{ interp-msg } (a, b) \\
\quad \text{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)} \\
\quad \text{using assms prefix-contains-msg apply (metis (full-types))} \\
\quad \text{create-id-valid create-delete-commute concurrent-create-delete-independent)} \\
\quad \text{using assms prefix-contains-msg apply (metis (full-types))} \\
\quad \text{create-id-valid create-delete-commute concurrent-create-delete-independent)} \\
\quad \text{using assms prefix-contains-msg apply (metis)} \\
\quad \text{append-id-valid append-delete-ids-imply-messages-same concurrent-append-delete-independent-technical delete-append-commute)} \\
\quad \text{using assms prefix-contains-msg apply (metis)} \\
\quad \text{concurrent-expunge-delete-independent expunge-id-valid imap.delete-expunge-commute imap-axioms)} \\
\quad \text{using assms prefix-contains-msg apply (metis concurrent-store-delete-independent delete-store-commute store-id-valid)} \\
\quad \text{using assms prefix-contains-msg apply (metis append-id-valid append-delete-ids-imply-messages-same concurrent-append-delete-independent-technical delete-append-commute)} \\
\quad \text{using assms prefix-contains-msg apply (metis append-id-valid expunge-id-valid append-expunge-ids-imply-messages-same)} \}

25
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)

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

\lambda ops. \exists xs i. xs prefix of i \land node-deliver-messages xs = ops \lambda x.\{\},\{}

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


