Abortable Linearizable Modules

Rachid Guerraoui Viktor Kuncak Giuliano Losa

August 28, 2014

Abstract

We define the Abortable Linearizable Module automaton (ALM for short) and prove its key composition property using the IOA theory of HOLCF. The ALM is at the heart of the Speculative Linearizability framework. This framework simplifies devising correct speculative algorithms by enabling their decomposition into independent modules that can be analyzed and proved correct in isolation. It is particularly useful when working in a distributed environment, where the need to tolerate faults and asynchrony has made current monolithic protocols so intricate that it is no longer tractable to check their correctness. Our theory contains a typical example of a refinement proof in the I/O-automata framework of Lynch and Tuttle.

Contents

1 Introduction 2

2 Definition and properties of the longest common postfix of a set of lists 3

3 The ALM Automata specification 3

4 Proof that the composition of two instances of the ALM automaton behaves like a single instance of the ALM automaton 7
  4.1 A case split useful in the proofs 7
  4.2 Invariants of a single ALM instance 9
  4.3 Invariants of the composition of two ALM instances 9
  4.4 Proofs of invariance 11
  4.5 The refinement proof 29

5 Conclusion 46
1 Introduction

Linearizability [2] is a key design methodology for reasoning about implementations of concurrent abstract data types in both shared memory and message passing systems. It presents the illusion that operations execute sequentially and fault-free, despite the asynchrony and faults that are often present in a concurrent system, especially a distributed one.

However, devising complete linearizable objects is very difficult, especially in the presence of process crashes and asynchrony, requiring complex algorithms (such as Paxos [3]) to work correctly under general circumstances, and often resulting in bad average-case behavior. Concurrent algorithm designers therefore resort to speculation, i.e. to optimizing existing algorithms to handle common scenarios more efficiently. More precisely, a speculative systems has a fall-back mode that works in all situations and several optimization modes, each of which is very efficient in a particular situation but might not work at all in some other situation. By observing its execution, a speculative system speculates about which particular situation it will be subject to and chooses the most efficient mode for that situation. If speculation reveals wrong, a new speculation is made in light of newly available observations. Unfortunately, building speculative system ad-hoc results in protocols so complex that it is no longer tractable to prove their correctness.

We present an I/O-automaton [4] specification, called ALM (a shorthand for Abortable Linearizable Module), which can be used to build a speculative linearizable algorithm out of independent modules that implement the different modes of the speculative algorithm. The ALM is at the heart of the Speculative Linearizability framework [1].

The ALM automaton produces traces that are linearizable with respect to a generic type of object. Moreover, the composition of two instances of the ALM automaton behaves like a single instance. Hence it is guaranteed that the composition of any number of instances of the ALM automaton is linearizable.

The properties stated above greatly simplify the development and analysis of speculative systems: Instead of having to reason about an entanglement of complex protocols, one can devise several modules with the property that, when taken in isolation, each module refines the ALM automaton. Hence complex protocols can be divided into smaller modules that can be analyzed independently of each other. In particular, it allows to optimize an existing protocol by creating separate optimization modules, prove each optimization correct in isolation, and obtain the correctness of the overall protocol from the correctness of the existing one.

In this document we define the ALM automaton and prove the Composition Theorem, which states that the composition of two instances of the ALM automaton behaves as a single instance of the ALM automaton. We use a refinement mapping to establish this fact.
2 Definition and properties of the longest common postfix of a set of lists

theory LCP
imports Main ~~/src/HOL/Library/Sublist
begin

definition common-postfix-p :: ('a list) set => 'a list => bool
— Predicate that recognizes the common prefix of a set of lists
— The common prefix of the empty set is the empty list
where
common-postfix-p ≡ λ xss xs . if xss = {} then xs = [] else ALL xs' . xs' ∈ xss
→ suffixeq xs xs'

definition l-c-p-pred :: 'a list set => 'a list => bool
— Predicate that recognizes the longest common prefix of a set of lists
where
l-c-p-pred ≡ λ xss xs . common-postfix-p xss xs ∧ (ALL xs' . common-postfix-p
xss xs' → suffixeq xs' xs)

definition l-c-p :: 'a list set => 'a list
— The longest common prefix of a set of lists
where
l-c-p ≡ λ xss . THE xs . l-c-p-pred xss xs

lemma l-c-p-ok: l-c-p-pred xss (l-c-p xss)
— Proof that the definition of the longest common prefix of a set of lists is
consistent

lemma l-c-p-lemma:
— A useful lemma
(ls ≠ {} ∧ (∀ l ∈ ls . (∃ l'. l = l' @ xs))) → suffixeq xs (l-c-p ls)

lemma l-c-p-common-postfix: common-postfix-p xss (l-c-p xss)
using l-c-p-ok[af xss] by (auto simp add:l-c-p-pred-def)

lemma l-c-p-longest: common-postfix-p xss xs → suffixeq xs (l-c-p xss)
using l-c-p-ok[af xss] by (auto simp add:l-c-p-pred-def)

end

3 The ALM Automata specification

theory ALM
imports ~~/src/HOL/HOLCF/IOA/meta-theory/IOA LCP
begin

typedecl client
— A non-empty set of clients

**typedef** data
— Data contained in requests

**datatype** request =
— A request is composed of a sender and data
  Req client data

**definition** request-snd :: request ⇒ client
  where request-snd ≡ λ r. case r of Req c - ⇒ c

**type-synonym** hist = request list
— Type of histories of requests.

**datatype** ALM-action =
— The actions of the ALM automaton
  Invoke client request
  | Commit client nat hist
  | Switch client nat hist request
  | Initialize nat hist
  | Linearize nat hist
  | Abort nat

**datatype** phase = Sleep | Pending | Ready | Aborted
— Executions phases of a client

**definition** linearizations :: request set ⇒ hist set
— The possible linearizations of a set of requests
  where
  linearizations ≡ λ reqs . { h . set h ⊆ reqs ∧ distinct h }

**definition** postfix-all :: hist ⇒ hist set ⇒ hist set
— appends to the right the first argument to every member of the history set
  where
  postfix-all ≡ λ h hs . { h′ . ∃ h′′ . h′ = h′′ @ h ∧ h′′ ∈ hs }

**definition** ALM-asig :: nat ⇒ nat ⇒ ALM-action signature
— The action signature of ALM automata
— Input actions, output actions, and internal actions
  where
  ALM-asig ≡ λ id1 id2 . ( { act . ∃ c r h .
    act = Invoke c r | act = Switch c id1 h r },
  { act . ∃ c h r id′ .
    id1 ≤ id′ ∧ id′ < id2 ∧ act = Commit c id′ h
    | act = Switch c id2 h r },
  { act . ∃ h .
    act = Abort id1
    | act = Linearize id1 h } } )
| $act = \text{Initialize id1 h}$ |

**record** \text{ALM-state} =  
— The state of the ALM automata  
pending :: client $\Rightarrow$ request  
— Associates a pending request to a client process  
initHists :: hist set  
— The set of init histories submitted by clients  
phase :: client $\Rightarrow$ phase  
— Associates a phase to a client process  
hist :: hist  
— Represents the chosen linearization of the concurrent history of the current instance only  
aborted :: bool  
initialized :: bool

**definition** pendingReqs :: ALM-state $\Rightarrow$ request set  
— the set of requests that have been invoked but that are not yet in the hist parameter  

where  
pendingReqs $\equiv \lambda s . \{ r . \exists c . $  
$r \equiv \text{pending s c}$  
$r \notin \text{set (hist s)}$  
$\land \text{phase s c} \in \{\text{Pending, Aborted}\}$

**definition** initValidReqs :: ALM-state $\Rightarrow$ request set  
— any request that appears in an init hist after the longest common prefix or that is pending  

where  
initValidReqs $\equiv \lambda s . \{ r . $  
$(r \in \text{pendingReqs s} \lor (\exists h \in \text{initHists s} . r \in \text{set h}))$  
$r \notin \text{set (l-c-p (initHists s))}$

**definition** ALM-trans :: nat $\Rightarrow$ nat $\Rightarrow$ (ALM-action, ALM-state)transition set  
— the transitions of the ALM automaton  

where  
ALM-trans $\equiv \lambda id1 id2 . \{\text{trans} . $  
let $s = \text{fst trans}; \ s' = \text{snd (snd trans)}; \ a = \text{fst (snd trans)}$ in  

case $a$ of Invoke c r $\Rightarrow$  
if phase s c = Ready $\land$ request-snd r = c $\land$ r $\notin$ set (hist s)  
then $s' = s[\text{pending := (pending s}(c := r), $  
phase := (phase s)(c := \text{Pending})]$  
else $s' = s$

| Linearize i h $\Rightarrow$  
initialized s $\land$ $\neg$ aborted s  
$h \in \text{postfix-all (hist s)} \ (\text{linearizations (pendingReqs s s))}$
\[ s' = s[hist := h] \]

|Initialize i h ⇒
\[ (\exists \ c . \ phase \ s \ c \neq \text{Sleep}) \land \neg \aborted s \land \neg \text{initialized} \land h \in \text{postfix-all} (l-c-p (\text{initHists} \ s)) (\text{linearizations} (\text{initValidReqs} \ s)) \land s' = s[hist := h, \text{initialized} := \text{True}] \]

|Abort i ⇒
\[ \neg \aborted s \land (\exists \ c . \ phase \ s \ c \neq \text{Sleep}) \land s' = s[\text{aborted} := \text{True}] \]

|Commit c i h ⇒
\[ \text{phase} \ s \ c = \text{Pending} \land \text{pending} \ s \ c \in \text{set} (\text{hist} \ s) \land h = \text{dropWhile} (\lambda r . r \neq \text{pending} \ s \ c) (\text{hist} \ s) \land s' = s[\text{phase} := (\text{phase} \ s)(c := \text{Ready})] \]

|Switch c i h r ⇒
\[ \text{if } i = \text{id1} \text{ then if phase } s \ c = \text{Sleep} \text{ then } s' = s[ \text{initHists} := \{h\} \cup (\text{initHists} \ s), \text{phase} := (\text{phase} \ s)(c := \text{Pending}), \text{pending} := (\text{pending} \ s)(c := r)] \text{ else } s' = s \text{ else if } i = \text{id2} \text{ then aborted } s \land \text{phase } s \ c = \text{Pending} \land r = \text{pending} \ s \ c \land (\text{if initialized} \ s \text{ then } (h \in \text{postfix-all} (\text{hist} \ s) (\text{linearizations} (\text{pendingReqs} \ s))) \text{ else } (h \in \text{postfix-all} (l-c-p (\text{initHists} \ s)) (\text{linearizations} (\text{initValidReqs} \ s)))) \land s' = s[\text{phase} := (\text{phase} \ s)(c := \text{Aborted})] \text{ else False } \]

definition ALM-start :: nat ⇒ ALM-state set — the set of start states
where
\[ \text{ALM-start} \equiv \lambda \text{id} . \{s . \forall \ c . \ phase \ s \ c = (\text{if id} \neq 0 \text{ then Sleep else Ready}) \land \text{hist} \ s = [] \land \neg \text{aborted} \land (\text{if id} \neq 0 \text{ then } \neg \text{initialized} \ s \text{ else initialized} \ s) \land \text{initHists} \ s = []] \]

definition ALM-ioa :: nat ⇒ nat ⇒ (ALM-action, ALM-state)ioa — The ALM automaton
where
\[ \text{ALM-ioa} \equiv \lambda (\text{id1}::nat) \text{id2} . (\text{ALM-asig} \ \text{id1} \ \text{id2}, \text{ALM-start} \ \text{id1}, \text{id2}) \]
\( ALM\text{-trans} \ id1 \ id2, \)
\{, \}

**type-synonym** compo-state \( = ALM\text{-state} \times ALM\text{-state} \)

**definition** composeALMs :: nat \( \Rightarrow \) nat \( \Rightarrow \) (ALM-action, compo-state) ioa
— the composition of two ALMs

**where**

composeALMs \( \equiv \lambda \ id1 \ id2 . \)

hide \( \) (ALM-ioa 0 id1 \mid| ALM-ioa id1 id2)
{act . EX \ c \ tr \ r . \ act \ = \ Switch \ c \ id1 \ tr \ r} \)

end

4 Proof that the composition of two instances of the ALM automaton behaves like a single instance of the ALM automaton

**theory** CompositionCorrectness
**imports** ALM
begin

declare split-if-asn [split]
declare Let-def [simp]

4.1 A case split useful in the proofs

**definition** in-trans-cases-fun :: nat \( \Rightarrow \) nat \( \Rightarrow \) (ALM-state \* ALM-state) \( \Rightarrow \) (ALM-state \* ALM-state) \( \Rightarrow \) bool
— Helper function used to decompose proofs

**where**

in-trans-cases-fun \( \equiv \ % \ id1 \ id2 \ s \ t . \)

(\( EX \ ca \ ra. \) (fst s, Invoke ca ra, fst t) : ALM-trans 0 id1 \& (snd s, Invoke ca ra, snd t) : ALM-trans id1 id2)
\| (\( EX \ ca \ hr. \) (fst s, Switch ca id1 h ra, fst t) : ALM-trans 0 id1 \& (snd s, Switch ca id1 h ra, snd t) : ALM-trans id1 id2)
\| (\( EX \ c \ id1 \ h. \) fst t = fst s \& (snd s, Commit c id1 h, snd t) : ALM-trans id1 id2 \& id1 \( \leq \) id' \& id' < id2) \)
\| (\( EX \ c \ hr. \) fst t = fst s \& (snd s, Switch c id2 h r, snd t) : ALM-trans id1 id2)
\| (\( EX \ h. \) fst t = fst s \& (snd s, Linearize id1 h, snd t) : ALM-trans id1 id2)
\| (\( fst t = \) fst s \& (snd s, Abort id1, snd t) : ALM-trans id1 id2)
\| (\( EX \ h. \) fst t = fst s \& (snd s, Initialize id1 h, snd t) : ALM-trans id1 id2)
\| (\( EX \ ca \ ta \ ra. \) (fst s, Switch ca 0 ta ra, fst t) : ALM-trans 0 id1 \& snd t = snd s)
\| (\( EX \ ca \ id1 \ h. \) (fst s, Commit ca id1 h, fst t) : ALM-trans 0 id1 \& snd t = snd s)
\| (\( EX \ h. \) (fst s, Linearize 0 h, fst t) : ALM-trans 0 id1 \& snd t = snd s)
\| (\( EX \ h. \) (fst s, Initialize 0 h, fst t) : ALM-trans 0 id1 \& snd t = snd s) \)

7
lemma composeALMsE:
— A rule for decomposing proofs
assumes id1 ~ = 0 and id1 < id2 and in-trans-comp: s − (a::ALM-action) − − composeALMs
id1 id2 − > t
shows decom: in-trans-cases-fun id1 id2 s t
proof −
from in-trans-comp and (id1 ~ = 0) and (id1 < id2)
have a : \{ act . EX c r h id' . 0 <= id' & id' < id2 & ( act = Invoke c r | act = \{ Switch c 0 h r, Switch c id1 h r, Switch c id2 h r \} | act = \{ Initialize 0 h, Initialize id1 h \} | act = \{ Abort 0, Abort id1 \} | act = \{ Commit c id' h \} ) \} by (auto simp add: composeALMs-def trans-of-def hide-def ALM-ioa-def par-def actions-def asig-inputs-def asig-outputs-def asig-internals-def asig-of-def ALM-asig-def)
with this obtain c r h id' where 0 <= id' & id' < id2 & a : \{ act . act = Invoke c r | act = \{ Switch c 0 h r, Switch c id1 h r, Switch c id2 h r \} | act = \{ Initialize 0 h, Initialize id1 h \} | act = \{ Abort 0, Abort id1 \} | act = \{ Commit c id' h \} \} by auto
moreover from in-trans-comp and (id1 ~ = 0) and (id1 < id2)
have (a = Linearize 0 h | a = Abort 0 | a = Initialize 0 h | a = Switch c 0 h r | (a = Commit c id' h & id' < id1)) => ((fst s, a, fst t) : ALM-trans 0 id1 & snd s = snd t)
and (a = Linearize id1 h | a = Abort id1 | a = Initialize id1 h | a = Switch c id2 h r | (a = Commit c id' h & id' <= id' & id' < id2)) => (fst s = fst t & (snd s, a, snd t) : ALM-trans id1 id2)
and (a = Switch c id1 h r | a = Invoke c r) => ((fst s, a, fst t) : ALM-trans 0 id1 & (snd s, a, snd t) : ALM-trans id1 id2)
ultimately show ?thesis unfolding in-trans-cases-fun-def apply simp by (metis linorder-not-less)
qed

lemma my-rule::[|id1 ≠ 0; id1 < id2; s − a−− composeALMs id1 id2 − > t; in-trans-cases-fun id1 id2 s t|] => P|] => P by (auto intro: composeALMsE[where s=s and t=t and a=a])
lemma my-rule2::[|0 < id1; id1 < id2; s − a−− composeALMs id1 id2 − > t; in-trans-cases-fun id1 id2 s t|] => P|] => P by (auto intro: composeALMsE[where
\section{4.2 Invariants of a single ALM instance}

\textbf{definition} \( P1a :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where
\hspace{1em} \text{— In ALM 1, a pending request of client } c \text{ has client } c \text{ as sender}
\hspace{1em} \( P1a == \% s . \text{ let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}
\hspace{2em} \forall c . \text{ phase } s1 c \in \{\text{Pending, Aborted} \} \implies \text{request-snd (pending } s1 c) = c \)

\textbf{definition} \( P1b :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where
\hspace{1em} \text{— In ALM 2, a pending request of client } c \text{ has client } c \text{ as sender}
\hspace{1em} \( P1b == \% s . \text{ let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}
\hspace{2em} \forall c . \text{ phase } s2 c \neq \text{Sleep} \implies \text{request-snd (pending } s2 c) = c \)

\textbf{definition} \( P2 :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where \( (\forall c . \text{ phase } s2 c = \text{Sleep}) \implies (\neg \text{initialized } s2 \wedge \text{hist } s2 = []) \)

\textbf{definition} \( P3 :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where \( (\forall c . \text{ phase } s2 c = \text{Ready} \implies \text{initialized } s2) \)

\textbf{definition} \( P4 :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where \( (\forall c . \text{ phase } s1 c \neq \text{Sleep}) \)

\section{4.3 Invariants of the composition of two ALM instances}

\textbf{definition} \( P6 :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where \((\neg \text{aborted } s1 \implies (\forall c . \text{ phase } s2 c = \text{Sleep})) \wedge (\forall c . \text{ phase } s1 c \neq \text{Sleep}) \)

\textbf{definition} \( P7 :: (ALM-state \ast ALM-state) \Rightarrow bool \)
\hspace{1em} where \( \text{— Before initialization of the ALM 2, pending requests are the same as in ALM 1 and no new requests may be accepted (phase is not Ready)} \)
where
\[ P7 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ \forall \ c . \ \text{phase} \ s1 \ c = \text{Aborted} \land \neg \ \text{initialized} \ s2 \ \rightarrow (\text{pending} \ s2 \ c = \text{pending} \ s1 \ c \land \text{phase} \ s2 \ c \in \{\text{Pending}, \text{Aborted}\}) \]

**definition** \( P8 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— Init histories of ALM 2 are built from the history of ALM 1 plus pending requests of ALM 1
where
\[ P8 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ \forall \ h \in \text{initHists} \ s2 . \ h \in \text{postfix-all} (\text{hist} \ s1) (\text{linearizations} (\text{pendingReqs} \ s1)) \]

**definition** \( P9 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— ALM 2 does not abort before ALM 1 aborts
where
\[ P9 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ \text{aborted} \ s2 \ \rightarrow \text{aborted} \ s1 \]

**definition** \( P10 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— ALM 1 is always initialized and when ALM 2 is not initialized its history is empty
where
\[ P10 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ \text{initialized} \ s1 \land (\neg \ \text{initialized} \ s2 \ \rightarrow (\text{hist} \ s2 = [\])) \]

**definition** \( P11 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, any request found in init histories after their longest common prefix is pending in ALM 1
where
\[ P11 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ ((\exists \ c . \ \text{phase} \ s2 \ c \neq \text{Sleep}) \land \neg \ \text{initialized} \ s2) \ \rightarrow \text{initValidReqs} \ s2 \subseteq \text{pendingReqs} \ s1 \]

**definition** \( P12 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, the longest common prefix of the init histories of ALM 2 is built from appending a set of request pending in ALM 1 to the history of ALM 1
where
\[ P12 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[ ((\exists \ c . \ \text{phase} \ s2 \ c \neq \text{Sleep}) \rightarrow (\exists \ rs . \ \text{l-c-p} (\text{initHists} \ s2) = rs \ @ (\text{hist} \ s1) \land \text{set rs} \subseteq \text{pendingReqs} \ s1 \land \text{distinct rs}) \]

**definition** \( P13 :: (\text{ALM-state} \ast \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, any history that may be chosen at initialization is a valid linearization of the concurrent history of ALM 1
where
\[ P13 \iff \% \ s . \ \text{let} \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ \text{in} \]
\[(\exists~c~.~\text{phase}~s_2~c \neq \text{Sleep}) \land \neg \text{initialized}~s_2) \rightarrow \text{postfix-all}~(l-c-p\ (\text{initHists}~s_2))\ (\text{linearizations}~(\text{initValidReqs}~s_2)) \subseteq \text{postfix-all}~(\text{hist}~s_1)\ (\text{linearizations}~(\text{pendingReqs}~s_1))\]

definition \text{P14} :: \((\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool}\)
where
— The history of ALM 1 is a postfix of the history of ALM 2 and requests appearing in ALM 2 after the history of ALM 1 are not in the history of ALM 1
\[\text{P14} = \%~s~.~\text{let}~s_1 = \text{fst}~s;~s_2 = \text{snd}~s\ in\]
\[\text{hist}~s_2 = \text{rs} \odot (\text{hist}~s_1)\]
\[\land \text{set}~\text{rs} \cap \text{set}~(\text{hist}~s_1) = \{\}\]

definition \text{P15} :: \((\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool}\)
where
— A client that hasn’t yet invoked ALM 2 has no request committed in ALM 2 except for its pending request
\[\text{P15} = \%~s~.~\text{let}~s_1 = \text{fst}~s;~s_2 = \text{snd}~s\ in\]
\[\forall~r~.~\text{let}~c = \text{request-snd}~r\ in \text{phase}~s_2~c = \text{Sleep} \land r \in \text{set}~(\text{hist}~s_2) \rightarrow (r \in \text{set}~(\text{hist}~s_1) \lor r \in \text{pendingReqs}~s_1)\]

4.4 Proofs of invariance

lemma \text{invariant-imp}: \([\text{invariant}~\text{ioa}~P;~\forall~s~.~P~s \rightarrow Q~s] \implies \text{invariant}~\text{ioa}~Q\)
by \text{(simp add:invariant-def)}

declare \text{phase.split} [split]
declare \text{phase.split-asm} [split]
declare \text{ALM-action.split} [split]
declare \text{ALM-action.split-asm} [split]

lemma \text{dropWhile-lemma}: \forall~ys~.~xs = ys \odot zs \land \text{hd}~zs = x \land zs \neq [] \land x \notin \text{set}~ys \rightarrow \text{dropWhile}~(\lambda~x'.~x' \neq x)~xs = zs
— A useful lemma about truncating histories
proof (induct \text{xs}, force)
fix \text{a} \text{xs}
assume \forall~ys~.~\text{xs} = ys \odot zs \land \text{hd}~zs = x \land zs \neq [] \land x \notin \text{set}~ys \rightarrow \text{dropWhile}~(\lambda~x'.~x' \neq x)~\text{xs} = zs
show \forall~ys~.~\text{a} \notin \text{xs} = ys \odot zs \land \text{hd}~zs = x \land zs \neq [] \land x \notin \text{set}~ys \rightarrow \text{dropWhile}~(\lambda~x'.~x' \neq x)~(\text{a} \notin \text{xs}) = zs
proof (rule allI, rule impI, cases \text{a} = \text{x})
fix \text{ys}
assume \text{a} \notin \text{xs} = ys \odot zs \land \text{hd}~zs = x \land zs \neq [] \land x \notin \text{set}~ys\ \text{and} \ \text{a} = x
hence \text{x} \notin \text{xs} = ys \odot zs \land \text{x} \notin \text{set}~ys\ \text{and} \ \text{a} = x
by auto
from \text{x} \notin \text{xs} = ys \odot zs\ \text{and} \ \text{x} \notin \text{set}~ys\ \text{have} \ ys = []\ \text{by (metis list.sel(1).hd-append hd-in-set)}
with \text{a} = x\ \text{and} \ \text{x} \notin \text{xs} = ys \odot zs\ \text{show} \ \text{dropWhile}~(\lambda x'.~x' \neq x)~(\text{a} \notin \text{xs}) = zs\ \text{by auto}
next
fix ys
assume a # xs = ys @ zs ∧ hd zs = x ∧ zs ≠ [] ∧ x # set ys and a ≠ x
hence a # xs = ys @ zs and hd zs = x and zs ≠ [] and x # set ys by auto
obtain ys' where xs = ys' @ zs and x # set ys'
proof
from (a # xs = ys @ zs) and (hd zs = x) and (a ≠ x) obtain ys' where ys
= a # ys' apply clarify by (metis Cons-eq-append-conv list.sel(1))
moreover with x # set ys: have x # set ys' by auto
moreover from (ys = a # ys') and (a # xs = ys @ zs) have xs = ys' @ zs
by auto
ultimately show (∀ys'. [xs = ys' @ zs; x # set ys'] ==> thesis) ==> thesis
by auto
qed

lemma P2-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P2
proof (rule invariantI, auto)
fix s1 s2
assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
thus P2 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P2-def)
next
fix s1 s2 s1' s2' act
assume reachable (composeALMs id1 id2) (s1, s2) and P2 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) - act -- composeALMs id1 id2 --> (s1', s2')
from (0 < id1) and (id1 < id2) and in-trans-comp show P2 (s1', s2')
proof (rule my-rule)
assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
thus P2 (s1', s2') using P2 (s1, s2) and (0 < id1) and (id1 < id2) apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P2-def) done
qed
qed

lemma P5-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P5
proof (rule invariantI, auto)
fix s1 s2
assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
thus P5 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P5-def)
next
fix $s_1$ $s_2$ $s_1'$ $s_2'$ act
assume reachable $(\text{composeALMs id1 id2}) (s_1, s_2)$ and $P5 (s_1, s_2)$ and $0 < id1$ and $id1 < id2$ and $\text{in-trans-comp} (s_1, s_2)$ -- $\text{act} -- \text{composeALMs id1 id2} -> (s_1', s_2')$
from $0 < id1$ and $id1 < id2$ and $\text{in-trans-comp}$ show $P5 (s_1', s_2')$
proof (rule my-rule2)
  assume $\text{in-trans-cases-fun} id1 id2 (s_1, s_2) (s_1', s_2')$
  thus $P5 (s_1', s_2')$ using $P5 (s_1, s_2)$ and $0 < id1$ and $id1 < id2$
apply (auto simp add: $\text{in-trans-cases-fun-def}$) apply (auto simp add: $\text{ALM-trans-def}$ $P5$-$\text{def}$) done
qed

lemma $P6$-invariant: $[|id1 \neq 0 ; id1 < id2|] => \text{invariant} (\text{composeALMs id1 id2}) P6$
proof (rule invariantI, rule-tac [2] impI)
  fix $s$
  assume $s : \text{starts-of} (\text{composeALMs id1 id2})$ and $id1 \neq 0$
  thus $P6 s$ by (simp add: $\text{starts-of-def}$ composeALMs-def hide-def $\text{ALM-ioa-def}$ par-def $\text{ALM-start-def}$ $P6$-$\text{def}$)
next
  fix $s$ $t$ $a$
  assume $P6 s$
  assume $id1 \neq 0$ and $id1 < id2$ and $s - a -- \text{composeALMs id1 id2} -> t$
  thus $P6 t$
proof (rule my-rule)
  assume $\text{in-trans-cases-fun} id1 id2 s t$
  thus $P6 t$ using $P6 s$ and $id1 \neq 0$ and $id1 < id2$
apply (auto simp add: $\text{in-trans-cases-fun-def}$) apply (simp-all add: $\text{ALM-trans-def}$ $P6$-$\text{def}$) apply (metis $\text{phase}$ phase $\text{alms}$ $\text{phase}$ phase $\text{alms}$ $\text{phase}$ phase $\text{alms}$ $\text{phase}$ phase $\text{alms}$) apply (force simp add: $\text{ALM-trans-def}$ $P6$-$\text{def}$) apply (force simp add: $\text{ALM-trans-def}$ $P6$-$\text{def}$) apply (force simp add: $\text{ALM-trans-def}$ $P6$-$\text{def}$) apply (force simp add: $\text{ALM-trans-def}$ $P6$-$\text{def}$) done
qed

lemma $P9$-invariant: $[|id1 < id2; id1 \neq 0|] => \text{invariant} (\text{composeALMs id1 id2}) P9$
proof (rule invariantI, auto)
  fix $s_1$ $s_2$
  assume $(s_1, s_2) : \text{starts-of} (\text{composeALMs id1 id2})$
  thus $P9 (s_1, s_2)$ by (simp add: $\text{starts-of-def}$ composeALMs-def hide-def $\text{ALM-ioa-def}$ par-def $\text{ALM-start-def}$ $P9$-$\text{def}$)
next
  fix $s_1$ $s_2$ $s_1'$ $s_2'$ act
  assume reachable $(\text{composeALMs id1 id2}) (s_1, s_2)$ and $P9 (s_1, s_2)$ and $0 < id1$ and $id1 < id2$ and $\text{in-trans-comp} (s_1, s_2)$ -- $\text{act} -- \text{composeALMs id1 id2} -> (s_1', s_2')$
have $P_6 (s_1, s_2)$
proof
  from in-trans-comp and reachable (composeALMs id1 id2) (s_1, s_2) have reachable (composeALMs id1 id2) (s_1', s_2') by (auto intro: reachable.reachable-n)
  with reachable (composeALMs id1 id2) (s_1, s_2) and $0 < id_1$ and $id_1 < id_2$ and $P_6$-invariant show $P_6 (s_1, s_2)$ unfolding invariant-def by auto
qed
from ($0 < id_1$ and ($id_1 < id_2$ and in-trans-comp show $P_9 (s_1', s_2')$
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s_1, s_2) (s_1', s_2')
  thus $P_9 (s_1', s_2')$ using $P_9 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$
apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_9$-def $P_6$-def) done
qed

lemma $P_10$-invariant: [[id_1 < id_2; id_1 \sim 0]] ==> invariant (composeALMs id1 id2) $P_{10}$
proof (rule invariantI, auto)
  fix s_1 s_2
  assume (s_1, s_2) : starts-of (composeALMs id1 id2) and $0 < id_1$
  thus $P_{10} (s_1, s_2)$ by (simp add: starts-of-def composeALMs-def hide-def ALM-iaoa-def par-def ALM-start-def $P_{10}$-def)
next
  fix s_1 s_2 s_1' s_2' act
  assume reachable (composeALMs id1 id2) (s_1, s_2) and $P_{10} (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp:(s_1, s_2) -act--composeALMs id1 id2-- (s_1', s_2')
  from ($0 < id_1$ and ($id_1 < id_2$ and in-trans-comp show $P_{10} (s_1', s_2')$
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s_1, s_2) (s_1', s_2')
  thus $P_{10} (s_1', s_2')$ using $P_{10} (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$
apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_{10}$-def) done
qed

lemma $P_3$-invariant: [[id_1 < id_2; id_1 \neq 0]] ==> invariant (composeALMs id1 id2) $P_3$
proof (rule invariantI, auto)
  fix s_1 s_2
  assume (s_1, s_2) : starts-of (composeALMs id1 id2) and $0 < id_1$
  thus $P_3 (s_1, s_2)$ by (simp add: starts-of-def composeALMs-def hide-def ALM-iaoa-def par-def ALM-start-def $P_3$-def)
next
  fix s_1 s_2 s_1' s_2' act
  assume reachable (composeALMs id1 id2) (s_1, s_2) and $P_3 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp:(s_1, s_2) -act--composeALMs id1 id2-- (s_1', s_2')
have $P_{10}$ ($s_1$, $s_2$)

proof -
from in-trans-comp and reachable (composeALMs $id_1$ $id_2$) ($s_1$, $s_2$); have reachable (composeALMs $id_1$ $id_2$) ($s_1'$, $s_2'$) by (auto intro: reachable.reachable-n)
with reachable (composeALMs $id_1$ $id_2$) ($s_1$, $s_2$) and $0 < id_1$ and $id_1 < id_2$ and $P_{10}$-invariant show $P_{10}$ ($s_1$, $s_2$) unfolding invariant-def by auto
qed
from ($0 < id_1$) and ($id_1 < id_2$) and in-trans-comp show $P_3$ ($s_1'$, $s_2'$)
proof (rule my-rule2)
assume in-trans-cases-fun $id_1$ $id_2$ ($s_1$, $s_2$) ($s_1'$, $s_2'$)
thus $P_3$ ($s_1'$, $s_2'$) using ($P_3$ ($s_1$, $s_2$)) and ($P_{10}$ ($s_1$, $s_2$)) and ($0 < id_1$) and ($id_1 < id_2$) apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_3$-def $P_{10}$-def) done
qed

lemma $P_7$-invariant: [$|id_1 < id_2; id_1 \neq 0|] \Rightarrow$ invariant (composeALMs $id_1$ $id_2$) $P_7$

proof (rule invariantI, auto)
fix $s_1$ $s_2$
assume ($s_1$, $s_2$) : starts-of (composeALMs $id_1$ $id_2$) and $0 < id_1$
thus $P_7$ ($s_1$, $s_2$) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def $P_7$-def)
next
fix $s_1$ $s_2$ $s_1'$ $s_2'$ act
assume reachable (composeALMs $id_1$ $id_2$) ($s_1$, $s_2$) and $P_7$ ($s_1$, $s_2$) and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp ($s_1$, $s_2$) act -- composeALMs $id_1$ $id_2$ -- ($s_1'$, $s_2'$)
have $P_6$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$)
proof -
from in-trans-comp and reachable (composeALMs $id_1$ $id_2$) ($s_1$, $s_2$); have reachable (composeALMs $id_1$ $id_2$) ($s_1'$, $s_2'$) by (auto intro: reachable.reachable-n)
with reachable (composeALMs $id_1$ $id_2$) ($s_1$, $s_2$) and ($0 < id_1$) and ($id_1 < id_2$) and $P_6$-invariant and $P_{10}$-invariant show $P_6$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$) unfolding invariant-def by auto
qed
from ($0 < id_1$) and ($id_1 < id_2$) and in-trans-comp show $P_7$ ($s_1'$, $s_2'$)
proof (rule my-rule2)
assume in-trans-cases-fun $id_1$ $id_2$ ($s_1$, $s_2$) ($s_1'$, $s_2'$)
thus $P_7$ ($s_1'$, $s_2'$) using ($P_7$ ($s_1$, $s_2$)) and ($P_6$ ($s_1$, $s_2$)) and ($0 < id_1$) and ($id_1 < id_2$)
proof (auto simp add: in-trans-cases-fun-def)
fix $ca$ $ra$
assume $P_7$ ($s_1$, $s_2$) and $P_6$ ($s_1$, $s_2$) and $0 < id_1$ and $id_1 < id_2$ and ($s_1$, $s_1'$) $\in$ ALM-trans $0$ $id_1$ and ($s_2$, $s_2'$) $\in$ ALM-trans $id_1$ $id_2$
thus $P_7$ ($s_1'$, $s_2'$) by (auto simp add: ALM-trans-def $P_7$-def)
next
fix $ca$ $h$ $ra$
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s1, Switch ca \(\text{id1} \) ra, \(s1') \in \text{ALM-trans} 0 \text{id1} \) and (s2, Switch ca \(\text{id1} \) ra, \(s2') \in \text{ALM-trans} \text{id1} \text{id2}

thus P7 (\(s1', s2'\)) by (auto simp add: ALM-trans-def P7-def P6-def)

next
fix \(c \text{id'} h\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and (s2, Commit c \(\text{id'} h\), \(s2') \in \text{ALM-trans} \text{id1} \text{id2} \) and \(\text{id1} \leq \text{id'} \) and \(\text{id'} < \text{id2}\)
thus P7 (s1, s2') using (P10 (s1, s2)) by (auto simp add: ALM-trans-def P7-def P10-def)

next
fix \(c h r\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s2, Switch c \(\text{id2} h r\), (s2') \in \text{ALM-trans} \text{id1} \text{id2}

thus P7 (s1, s2') by (auto simp add: ALM-trans-def P7-def)

next
fix \(h\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s2, Initialize \(\text{id1} h\), \(s2') \in \text{ALM-trans} \text{id1} \text{id2}

thus P7 (s1, s2') by (simp add: ALM-trans-def P7-def)

next
fix \(h\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s2, Initialize \(\text{id1} h\), \(s2') \in \text{ALM-trans} \text{id1} \text{id2}

thus P7 (s1, s2') by (auto simp add: ALM-trans-def P7-def)

next
fix \(c a t a r\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s1, Switch ca \(0 t a r\), \(s1') \in \text{ALM-trans} 0 \text{id1}

thus P7 (\(s1', s2\)) by (auto simp add: ALM-trans-def P7-def)

next
fix \(c a t a r\)
assume P7 (s1, s2) and P6 (s1, s2) and \(\text{id1} < \text{id2} \) and (s1, Commit ca \(\text{id'} h\), \(s1') \in \text{ALM-trans} 0 \text{id1} \) and \(\text{id'} < \text{id1}\)

thus P7 (\(s1', s2\)) by (auto simp add: ALM-trans-def P7-def)

next
fix \(h\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s1, Linearize \(0 h\), \(s1') \in \text{ALM-trans} 0 \text{id1}

thus P7 (\(s1', s2\)) by (auto simp add: ALM-trans-def P7-def)

next
fix \(h\)
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s1, Initialize \(0 h\), \(s1') \in \text{ALM-trans} 0 \text{id1}

thus P7 (\(s1', s2\)) by (auto simp add: ALM-trans-def P7-def)

next
assume P7 (s1, s2) and P6 (s1, s2) and \(0 < \text{id1} \) and \(\text{id1} < \text{id2} \) and (s2, Abort \(\text{id1}\), \(s2') \in \text{ALM-trans} \text{id1} \text{id2}

thus P7 (s1, s2') by (auto simp add: ALM-trans-def P7-def)
next
  assume P7 (s1, s2) and P6 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Abort 0, s1') ∈ ALM-trans 0 id1
  thus P7 (s1', s2) by (auto simp add: ALM-trans-def P7-def)
qed
qed

lemma P4-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P4
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P4 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P4-def)
  qed
  qed

next
  fix s1 s2 s1' s2' act
  assume reachable (composeALMs id1 id2) (s1, s2) and P4 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) − act − composeALMs id1 id2 −> (s1', s2')
  have P6 (s1, s2)
  proof −
  from in-trans-comp and reachable (composeALMs id1 id2) (s1, s2) and P4 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) − act − composeALMs id1 id2 −> (s1', s2')
  have P6 (s1, s2) unfolding invariant-def by auto
  qed
  from (0 < id1) and (id1 < id2) and in-trans-comp show P4 (s1', s2')
  proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
  thus P4 (s1', s2') using P4 (s1, s2); and (0 < id1) and (id1 < id2) apply(auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P4-def) done
  qed
  qed

lemma P8-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P8
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P8 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P8-def)
  qed
  qed

next
  fix s1 s2 s1' s2' act
  assume reachable (composeALMs id1 id2) (s1, s2) and P8 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) − act − composeALMs id1 id2 −> (s1', s2')
  have P6 (s1, s2) and P10 (s1, s2) and P5 (s1, s2) and P4 (s1, s2)
proof

from in-trans-comp and \( \langle \text{reachable (composeALMs id1 id2)} (s1, s2) \rangle \) have reachable (composeALMs id1 id2) \( (s1', s2') \) by (auto intro: reachable.reachable-n)

with \( \langle \text{reachable (composeALMs id1 id2)} (s1, s2) \rangle \) and \( 0 < id1 \) and \( id1 < id2 \) and P6-invariant and P10-invariant and P5-invariant and P4-invariant show P6 \( (s1, s2) \) and P10 \( (s1, s2) \) and P5 \( (s1, s2) \) and P4 \( (s1, s2) \) unfolding invariant-def by auto

qed

from \( 0 < id1 \) and \( id1 < id2 \) and in-trans-comp show P8 \( (s1', s2') \)

proof (rule my-rule2)

assume in-trans-cases-fun id1 id2 \( (s1, s2) \) \( (s1', s2') \)

thus P8 \( (s1', s2') \) using \( \langle P8 (s1, s2) \rangle \) and \( 0 < id1 \) and \( id1 < id2 \)

proof (auto simp add: in-trans-cases-fun-def)

fix ca ra

assume P8 \( (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and in-involve-1: \( \langle s1, \text{Invoke ca ra, s1'} \rangle \in \text{ALM-trans 0 id1 and in-involve-2:} \( (s2, \text{Invoke ca ra, s2'}) \in \text{ALM-trans id1 id2} \)

show P8 \( (s1', s2') \)

proof (cases \( s1' = s1 \))

assume \( s1' = s1 \)

with in-involve-2 and \( \langle P8 (s1, s2) \rangle \) show \( \? \text{thesis} \) by (auto simp add: ALM-trans-def P8-def)

next

assume \( s1' \neq s1 \)

with in-involve-1 have pendingReqs s1 \( \subseteq \) pendingReqs s1' by (force simp add: pendingReqs-def ALM-trans-def)

moreover from in-involve-1 have hist s1' = hist s1 by (auto simp add: ALM-trans-def)

moreover from in-involve-2 have initHists s2' = initHists s2 by (auto simp add: ALM-trans-def)

moreover note \( \langle P8 (s1, s2) \rangle \)

ultimately show \( \? \text{thesis} \) by (auto simp add: ALM-trans-def P8-def linearizations-def postfix-all-def)

qed

next

fix ca h ra

assume P8 \( (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and in-switch-1: \( \langle s1, \text{Switch ca id1 h ra, s1'} \rangle \in \text{ALM-trans 0 id1 and in-switch-2:} \( (s2, \text{Switch ca id1 h ra, s2'}) \in \text{ALM-trans id1 id2} \)

show P8 \( (s1', s2') \)

proof (auto simp add: P8-def)

fix h1

assume h1 \( \in \text{initHists s2'} \)

show h1 \( \in \text{postfix-all (hist s1')} \) (linearizations (pendingReqs s1'))

proof (cases h1 \( \in \text{initHists s2} \))

assume h1 \( \in \text{initHists s2} \)

moreover from in-switch-1 and \( 0 < id1 \) have hist s1' = hist s1 and pendingReqs s1' = pendingReqs s1 by (auto simp add: ALM-trans-def pendingReqs-def)

18
moreover note \( \langle P8 (s_1, s_2) \rangle \)
ultimately show \( h_1 \in \text{postfix-all} (\text{hist } s_1') \) (linearizations (pendingReqs \( s_1' \))) by (auto simp add: P8-def)

next
assume \( h_1 \notin \text{initHists } s_2 \)
with \( h_1 \in \text{initHists } s_2' \) and in-switch-2 have \( h_1 = h \) by (auto simp add: ALM-trans-def)
with in-switch-1 and \( 0 < id_1 \) and \( (P10 (s_1, s_2)) \) have \( h_1 \in \text{postfix-all} (\text{hist } s_1) \) (linearizations (pendingReqs \( s_1 \))) by (auto simp add: ALM-trans-def P10-def)
moreover from in-switch-1 and \( 0 < id_1 \) have \( \text{hist } s_1' = \text{hist } s_1 \) and pendingReqs \( s_1' = \text{pendingReqs } s_1 \) by (auto simp add: ALM-trans-def pendingReqs-def)
ultimately show \( \text{thesis} \) by auto
qed
qed

next
fix \( c \ id' \ h \)
assume \( P8 (s_1, s_2) \) and \( 0 < id_1 \) and \( (s_2, \text{Commit } c \ id' \ h, s_2') \in \text{ALM-trans} \)
\( id_1 \ id_2 \) and \( id_1 \leq id' \) and \( id' < id_2 \)
thus \( P8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( c h r \)
assume \( P8 (s_1, s_2) \) and \( 0 < id_1 \) and \( id_1 < id_2 \) and \( (s_2, \text{Switch } c \ id_2 \ h \ r, s_2') \in \text{ALM-trans} \)
\( id_1 \ id_2 \)
thus \( P8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( h \)
assume \( P8 (s_1, s_2) \) and \( 0 < id_1 \) and \( id_1 < id_2 \) and \( (s_2, \text{Linearize } id_1 \ h, s_2') \in \text{ALM-trans} \)
\( id_1 \ id_2 \)
thus \( P8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( h \)
assume \( P8 (s_1, s_2) \) and \( 0 < id_1 \) and \( id_1 < id_2 \) and \( (s_2, \text{Initialize } id_1 \ h, s_2') \in \text{ALM-trans} \)
\( id_1 \ id_2 \)
thus \( P8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( ca \ ta \ ra \)
assume \( P8 (s_1, s_2) \) and \( 0 < id_1 \) and \( id_1 < id_2 \) and \( (s_1, \text{Switch } ca \ 0 \ ta \ ra, s_1') \in \text{ALM-trans} \)
\( 0 \ id_1 \)
thus \( P8 (s_1', s_2) \) using \( (P5 (s_1, s_2)) \) by (auto simp add: ALM-trans-def P8-def P5-def)
next
fix \( ca \ id' \ h \)
assume \( P8 (s_1, s_2) \) and in-commit-1:(\( s_1, \text{Commit } ca \ id' \ h, s_1' \) \in \text{ALM-trans} \)
\( 0 \ id_1 \)
from in-commit-1 have pendingReqs \( s_1' = \text{pendingReqs } s_1 \) and \( \text{hist } s_1' = \text{hist } s_1 \) by (auto simp add: pendingReqs-def ALM-trans-def)
with \( (P8 (s_1, s_2)) \) show \( P8 (s_1', s_2) \) by (auto simp add: ALM-trans-def P8-def pendingReqs-def)
next
    fix h
    assume \( P8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Linearize } 0 \ h, s_1') \in \text{ALM-trans } 0 \ id1 \)
    thus \( P8 (s_1', s_2) \) using \( P6 (s_1, s_2) \) and \( P4 (s_1, s_2) \) by (auto simp add: ALM-trans-def P8-def P6-def P4-def)
next
    assume \( P8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_2, \text{Abort } id1, s_2') \in \text{ALM-trans } id1 \ id2 \)
    thus \( P8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
    assume \( P8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Initialize } 0 \ h, s_1') \in \text{ALM-trans } 0 \ id1 \)
    thus \( P8 (s_1', s_2) \) using \( P10 (s_1, s_2) \) by (auto simp add: ALM-trans-def P8-def P10-def)
next
    assume \( P8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Abort } 0, s_1') \in \text{ALM-trans } 0 \ id1 \)
    thus \( P8 (s_1', s_2) \) by (auto simp add: ALM-trans-def P8-def pendingReqs-def)
qed

lemma \( P12\)-invariant: \([id1 < id2; id1 \neq 0]\) == invariant (composeALMs id1 id2) \( P12 \)
proof clariry
    assume \( id1 < id2 \) and \( 0 < id1 \)
    with \( P8\)-invariant and \( P4\)-invariant have invariant (composeALMs id1 id2) \((\lambda (s_1, s_2). P8 (s_1, s_2) \land P4 (s_1, s_2))\) by (auto simp add: invariant-def)
moreover have \( \forall s. \ P8 s \land P4 s \rightarrow P12 s \)
proof auto
    fix \( s_1 s_2 \)
    assume \( P8 (s_1, s_2) \) and \( P4 (s_1, s_2) \)
    hence \( \text{initHists-prop:}\exists h \in \text{initHists } s_2. (\exists h'. h = h' @ (\text{hist } s_1) \land \text{set } h' \subseteq \text{pendingReqs } s_1 \land \text{distinct } h') \) by (auto simp add: P8-def postfix-all-def linearizations-def)
    show \( P12 (s_1, s_2) \)
proof (simp add:P12-def, rule impI)
    assume \( \exists c. \text{ phase } s_2 c \neq \text{Sleep} \)
    with \( P4 (s_1, s_2) \) have \( \text{initHists } s_2 \neq {} \) by (auto simp add:P4-def)
    with l-c-p-lemma[of \( \text{initHists } s_2 \) hist \( s_1 \) ] and \( \text{initHists-prop} \)
    obtain \( rs \) where \( l-c-p \ (\text{initHists } s_2) = rs \circ \text{hist } s_1 \) by (auto simp add: suffixeq-def)
moreover have \( \text{set } rs \subseteq \text{pendingReqs } s_1 \)
proof -
    from \( \text{initHists } s_2 \neq {} \) obtain \( h \) where \( h \in \text{initHists } s_2 \) by auto
    with \( \text{initHists-prop} \) obtain \( h' \) where \( h = h' @ (\text{hist } s_1) \land \text{set } h' \subseteq \text{pendingReqs } s_1 \) by auto
qed
moreover from \( l-c-p\text{-}common\text{-}postfix\text{[of]} \text{initHists } s2 \) and \( (h \in \text{initHists } s2) \) obtain \( h'' \) where \( h = h'' \circ (l-c-p \text{ (initHists } s2)) \) by (auto simp add:common-postfix-p-def suffixeq-def)

moreover note \( (l-c-p \text{ (initHists } s2) = \text{rs } @ \text{hist } s1) \)
ultimately show \( \text{thesis} \) by auto
qed
moreover have \( \text{distinct } \text{rs} \)
proof -
  from \((\text{initHists } s2 \neq \{\})\) obtain \( h \) where \( h \in \text{initHists } s2 \) by auto
with \( \text{initHists-prop} \) obtain \( h' \) where \( h = h' \circ (\text{hist } s1) \) and \( \text{distinct } h' \)
by auto
with \( l-c-p\text{-}common\text{-}postfix\text{[of]} \text{initHists } s2 \) and \( (h \in \text{initHists } s2) \) and \( (l-c-p \text{ (initHists } s2) = \text{rs } @ \text{hist } s1) \) obtain \( h'' \) where \( h'' = h'' \circ \text{rs} \) apply (auto simp add:common-postfix-p-def suffixeq-def) by (metis \( h = h' \circ (\text{hist } s1) \) append-assoc append.same-eq)
with \( \text{distinct } h' \) show \( \text{thesis} \) by auto
qed
qed
ultimately show \( \text{thesis} \) by (auto intro:invariant-imp)
qed

lemma \( P11\text{-}invariant: \left[ \text{id1} < \text{id2}; \text{id1} \neq 0 \right] \implies \text{invariant} \ (\text{composeALMs } \text{id1} \ \text{id2}) \ P11 \)
proof clarify
  assume \( \text{id1} < \text{id2} \) and \( 0 < \text{id1} \)
  with \( P8\text{-}invariant \) and \( P12\text{-}invariant \) and \( P6\text{-}invariant \) and \( P7\text{-}invariant \) have
  invariant \( (\text{composeALMs } \text{id1} \ \text{id2}) \ (\lambda \text{ (s1, s2)} . \ P8 \ (\text{s1, s2}) \land P12 \ (\text{s1, s2}) \land P6 \ (\text{s1, s2}) \land P7 \ (\text{s1, s2})) \) by (auto simp add:invariant-def)
  moreover have \( \forall \text{ s } \ . \ P8 \text{ s } \land P12 \text{ s } \land P6 \text{ s } \land P7 \text{ s } \longrightarrow P11 \text{ s} \)
proof auto
  fix \( \text{s1 s2} \)
  assume \( P8 \ (\text{s1, s2}) \) and \( P12 \ (\text{s1, s2}) \) and \( P6 \ (\text{s1, s2}) \) and \( P7 \ (\text{s1, s2}) \)
  show \( P11 \ (\text{s1, s2}) \)
  proof (simp add:P11-def initValidReqs-def, auto)
    fix \( x \ c \ h \)
    assume phase \( s2 \ c = \text{Sleep} \)
    with \( P12 \ (\text{s1, s2}) \) and \( P8 \ (\text{s1, s2}) \) have \( \text{initHists-prop} \forall \text{ h } \in \text{initHists } s2 \ . \ (\exists \text{ h'} . \ h = h' \circ (\text{hist } s1) \land \text{set } h' \subseteq \text{pendingReqs } s1) \) and \( \text{lcp-prop} \exists \text{ rs } . \ l-c-p \ (\text{initHists } s2) = \text{rs } @ (\text{hist } s1) \) by (auto simp add:P12-def P8-def postfix-all-def linearizations-def)
    assume \( x \notin \text{ set } (l-c-p \ (\text{initHists } s2)) \) and \( h \in \text{initHists } s2 \) and \( x \in \text{set } h \)
    from \( \text{initHists-prop} \) and \( h \in \text{initHists } s2 \) obtain \( h' \) where \( h = h' \circ (\text{hist } s1) \) and \( \text{set } h' \subseteq \text{pendingReqs } s1 \) by auto
    moreover from \( \text{lcp-prop} \) obtain \( rs \) where \( l-c-p \ (\text{initHists } s2) = \text{rs } @ (\text{hist } s1) \) by auto
    moreover note \( x \notin \text{ set } (l-c-p \ (\text{initHists } s2)) \) and \( \forall \text{ x } \in \text{set } h \)

ultimately have \( x \in \text{set } h \) by auto

with \( \{set h \subseteq \text{pendingReqs } s1\} \) show \( x \in \text{pendingReqs } s1 \) by auto

next

fix \( x \) c h

assume phase \( s2 \) c \( \neq \) Sleep and \( \neg \) initialized \( s2 \)

with \( \{P12 (s1, s2)\} \) have lcp-prop:3 rs . \( l\cdot c\cdot p \) (initHists s2) = rs @ (hist s1) by (auto simp add:P12-def P8-def postfix-all-def linearizations-def)

assume \( x \notin \text{set } l\cdot c\cdot p \) (initHists s2) \( \) and \( x \in \text{pendingReqs } s2 \)

from \( x \notin \text{set } l\cdot c\cdot p \) (initHists s2)) \( \) and lcp-prop have \( x \notin \text{set } (\text{hist } s1) \) by auto

moreover obtain \( c' \) where phase \( s1 \) c' = Aborted and \( x = \text{pending } s1 \) c'

proof –

from \( x \in \text{pendingReqs } s2 \) \( \) and \( \{P6 (s1, s2)\} \) obtain \( c' \) where phase \( s1 \) c' = Aborted and \( x = \text{pending } s2 \) c' by (force simp add:pendingReqs-def P6-def)

moreover with \( \neg \) initialized \( s2 \) \( \) and \( \{P7 (s1, s2)\} \) have \( x = \text{pending } s1 \) c' by (auto simp add:P7-def)

ultimately show \( (\forall c'. (\text{phase } s1 c' = \text{Aborted}; x = \text{pending } s1 c')) \Longrightarrow \text{thesis} \) by auto

qed ultimately show \( x \in \text{pendingReqs } s1 \) by (auto simp add:pendingReqs-def)

qed

qed ultimately show \( ?\text{thesis} \) by (auto intro:invariant-imp)

qed

lemma P1a-invariant: \( [\mid id1 < id2; id1 \neq 0\mid] \Longrightarrow \text{invariant } (\text{composeALMs } id1 id2) P1a\)

proof (rule invariantI, auto)

fix \( s1 \) \( s2 \)

assume \( (s1, s2) : \text{starts-of } (\text{composeALMs } id1 id2) \) \( \) and \( 0 < id1 \)

thus P1a \( (s1, s2) \) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1a-def)

next

fix \( s1 \) \( s2 \) \( s1' \) \( s2' \) act

assume reachable \( (\text{composeALMs } id1 id2) (s1, s2) \) \( \) and \( P1a (s1, s2) \) \( \) and \( 0 < id1 \) \( \) and \( id1 < id2 \) \( \) and \( \text{in-trans-comp}(s1, s2) - act - - \text{composeALMs } id1 id2 - > (s1', s2') \)

have \( P5 (s1, s2) \)

proof –

from \( \text{in-trans-comp} \) \( \) and \( \text{reachable } (\text{composeALMs } id1 id2) (s1, s2) \) \( \) have reachable \( (\text{composeALMs } id1 id2) (s1', s2') \) by (auto intro: reachable.reachable-n)

with \( \text{reachable } (\text{composeALMs } id1 id2) (s1, s2) \) \( \) and \( 0 < id1 \) \( \) and \( id1 < id2 \) \( \) and \( \text{P5-invariant} \) show \( P5 (s1, s2) \) unfolding invariant-def by auto

qed

from \( 0 < id1 \) \( \) and \( \{id1 < id2\} \) \( \) and \( \text{in-trans-comp} \) \( \) show \( P1a (s1', s2') \)

proof (rule my-rule2)

assume \( \text{in-trans-cases-fun } id1 id2 (s1, s2) (s1', s2') \)

thus \( P1a (s1', s2') \) using \( \{P1a (s1, s2)\} \) \( \) \( \) and \( P5 (s1, s2) \) \( \) \( \) and \( 0 < id1 \) \( \) and \( \{id1 < id2\} \)

and \( \{id1 < id2\} \) apply(auto simp add: in-trans-cases-fun-def) apply (auto simp
lemma P1b-invariant: ||id1 < id2; id1 ≠ 0|| ===> invariant (composeALMs id1 id2) P1b
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
thus P1b (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1b-def)
next
  fix s1 s2 s1’ s2’ act
  assume reachable (composeALMs id1 id2) (s1, s2) and P1b (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp: (s1, s2) — act — composeALMs id1 id2 —> (s1’, s2’)
  have P1a (s1, s2)
    proof
      from in-trans-comp and reachable (composeALMs id1 id2) (s1, s2) have reachable (composeALMs id1 id2) (s1’, s2’) by (auto intro: reachablereachable-n)
      with reachable (composeALMs id1 id2) (s1, s2) and 0 < id1 and id1 < id2 and P1a-invariant show P1a (s1, s2)
    qed
  qed
  from 0 < id1 and id1 < id2 and in-trans-comp show P1b (s1’, s2’)
  proof (rule my-rule2)
    assume in-trans-cases-fun id1 id2 (s1, s2) (s1’, s2’)
    thus P1b (s1’, s2’) using P1b (s1, s2) and P1a (s1, s2) and 0 < id1 and id1 < id2 apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P1a-def P5-def)
  qed
  qed

lemma P13-invariant: ||id1 < id2; id1 ≠ 0|| ===> invariant (composeALMs id1 id2) P13
proof clarify
  assume id1 < id2 and 0 < id1
  with P11-invariant and P12-invariant have invariant (composeALMs id1 id2) \( \lambda (s1, s2) . P11 (s1, s2) \land P12 (s1, s2) \) by (auto simp add:invariant-def)
  moreover have \( \forall s . P11 s \land P12 s \implies P13 s \)
  proof auto
    fix s1 s2
    assume P11 (s1, s2) and P12 (s1, s2)
    show P13 (s1, s2)
      proof (simp add:P13-def, rule impl)
        assume \( \exists \ c . \ phase s2 c \neq \text{Sleep} \) \land \neg \text{initialized} s2
        with P12 (s1, s2) and P11 (s1, s2) obtain rs where initValidReqs-prop:initValidReqs s2 \subseteq pendingReqs s1 and l-c-p (initHists s2) = rs \otimes \text{hist s1} and set rs \subseteq pendingReqs s1 and distinct rs by (auto simp add:P12-def P11-def postfix-all-def linearizations-def)
moreover from \( l \cdot c \cdot p \) (initHists \( s_2 \)) = rs \oplus (\text{hist} \ s_1) \ have \ initValidReqs \ s_2
\cap \ 
set \ rs = \{\} \ by \ (auto \ simp \ add: initValidReqs-def)

ultimately show postfix-all \( l \cdot c \cdot p \) (initHists \( s_2 \)) (linearizations (initValidReqs \( s_2 \)) \subseteq \ 
postfix-all \( \text{hist} \ s_1 \) \ (linearizations \ pendingReqs \( s_1 \)) \ by \ (force \ simp \ add: \ 
postfix-all-def \ linearizations-def)

qed

qed

ultimately show ?thesis by \ (auto \ intro: \ invariant-imp)

qed

lemma \( P_{14} \)-invariant: \( [[id_1 < id_2; id_1 \neq 0]] \Longrightarrow \ invariant \ (composeALMs \ id_1 \ id_2) \)

proof \ (\text{rule \ invariantI, \ auto})

fix \( s_1, s_2 \)

assume \( (s_1, s_2): \ \text{starts-of} \ (\text{composeALMs} \ id_1 \ id_2) \ \text{and} \ 0 < id_1 \)

thus \( P_{14} \ (s_1, s_2) \) \ by \ (simp \ add: \ \text{starts-of-def} \ \text{composeALMs-def} \ \text{hide-def} \ \text{ALM-iaa-def} \ \text{par-def} \ \text{ALM-start-def} \ P_{14-def})

next

fix \( s_1, s_2, s_1', s_2' \) \ act

assume \( \text{reachable} \ (\text{composeALMs} \ id_1 \ id_2) \ (s_1, s_2) \ \text{and} \ P_{14} \ (s_1, s_2) \ \text{and} \ 0 < \ id_1 \ \text{and} \ id_1 < id_2 \ \text{and} \ \text{in-trans-comp} (s_1, s_2) \ \act \ \text{-- composeALMs} \ id_1 \ id_2 \ \text{--} \ (s_1', s_2') \)

have \( P_6 \ (s_1, s_2) \ \text{and} \ P_{13} \ (s_1, s_2) \ \text{and} \ P_{10} \ (s_1, s_2) \ \text{and} \ P_2 \ (s_1, s_2) \ \text{and} \ P_4 \ (s_1, s_2) \)

proof

from \( \text{in-trans-comp} \ \text{and} \ \text{reachable} \ (\text{composeALMs} \ id_1 \ id_2) \ (s_1, s_2) \) \ have \ reachable \ (\text{composeALMs} \ id_1 \ id_2) \ (s_1', s_2') \ by \ (auto \ intro: \ reachablereachable-n)

with \( \text{reachable} \ (\text{composeALMs} \ id_1 \ id_2) \ (s_1, s_2) \) \ \text{and} \ 0 < id_1 \ \text{and} \ id_1 < id_2 \ \text{and} \ P_6 \text{-invariant \ and} \ P_{13} \text{-invariant \ and} \ P_{10} \text{-invariant \ and} \ P_4 \text{-invariant}

and \( P_2 \text{-invariant \ show} \ P_6 \ (s_1, s_2) \ \text{and} \ P_{13} \ (s_1, s_2) \ \text{and} \ P_{10} \ (s_1, s_2) \ \text{and} \ P_2 \ (s_1, s_2) \ \text{and} \ P_4 \ (s_1, s_2) \) \ unfolding \ \text{invariant-def} \ by \ auto

qed

from \( 0 < id_1 \ \text{and} \ id_1 < id_2 \ \text{and} \ \text{in-trans-comp \ show} \ P_{14} \ (s_1', s_2') \)

proof \ (\text{rule \ my-rule2})

assume \( \text{in-trans-cases-fun} \ id_1 \ id_2 \ (s_1, s_2) \ (s_1', s_2') \)

thus \( P_{14} \ (s_1', s_2') \ \text{using} \ (P_{14} \ (s_1, s_2): \ \text{and} \ 0 < id_1 \ \text{and} \ id_1 < id_2) \)

proof \ (\text{auto \ simp \ add: \ \text{in-trans-cases-fun-def})

fix \( ca \ ra \)

assume \( P_{14} \ (s_1, s_2) \ \text{and} \ 0 < id_1 \ \text{and} \ id_1 < id_2 \ \text{and} \ (s_1, \ \text{Invoke} \ ca \ ra, \ s_1') \in \ \text{ALM-trans} \ 0 \ id_1 \ \text{and} \ (s_2, \ \text{Invoke} \ ca \ ra, \ s_2') \in \ \text{ALM-trans} \ id_1 \ id_2 \)

thus \( P_{14} \ (s_1', s_2') \) \ by \ (auto \ simp \ add: \ \text{ALM-trans-def} \ P_{14-def})

next

fix \( ca \ h \ ra \)

assume \( P_{14} \ (s_1, s_2) \ \text{and} \ 0 < id_1 \ \text{and} \ id_1 < id_2 \ \text{and} \ (s_1, \ \text{Switch} \ ca \ id_1 \ h \ ra, \ s_1') \in \ \text{ALM-trans} \ 0 \ id_1 \ \text{and} \ (s_2, \ \text{Switch} \ ca \ id_1 \ h \ ra, \ s_2') \in \ \text{ALM-trans} \ id_1 \ id_2 \)

thus \( P_{14} \ (s_1', s_2') \) \ by \ (auto \ simp \ add: \ \text{ALM-trans-def} \ P_{14-def})

next

fix \( c \ \text{id}', h \)

24
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and ($s_2$, Commit $c$ id\'$ h$, $s_2'$) $\in$ ALM-trans
\begin{align*}
&\text{next} \\
&\text{fix } c h r \\
&\text{assume } P_{14} (s_1, s_2') \text{ by (auto simp add: ALM-trans-def $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{thus } P_{14} (s_1, s_2') \text{ by (auto simp add: ALM-trans-def $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } h \\
&\text{assume } P_{14} (s_1, s_2') \text{ by (auto simp add: ALM-trans-def $P_{14}$-def linearizations-def postfix-all-def pendingReqs-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } h \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } 0 < id_1 \text{ and } id_1 < id_2 \text{ and } (s_2, \text{Initialize id}_1 h, s_2') \in \text{ALM-trans id}_1 id_2 \\
&\text{thus } P_{14} (s_1, s_2') \text{ using } P_{13} (s_1, s_2) \text{ apply (auto simp add: ALM-trans-def $P_{14}$-def $P_{13}$-def linearizations-def postfix-all-def pendingReqs-def)} \text{ prefer 2 apply force apply blast done}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } 0 < id_1 \text{ and } id_1 < id_2 \text{ and } (s_2, \text{Abort id}_1, s_2') \in \text{ALM-trans id}_1 id_2 \\
&\text{thus } P_{14} (s_1, s_2') \text{ by (auto simp add: ALM-trans-def $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } c a t a r a \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } 0 < id_1 \text{ and } id_1 < id_2 \text{ and } (s_1, \text{Switch ca 0 to ra}, s_1') \in \text{ALM-trans 0 id}_1 id_1 \\
&\text{thus } P_{14} (s_1', s_2) \text{ by (auto simp add: ALM-trans-def $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } c a \text{ id}' h \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } id_1 < id_2 \text{ and } (s_1, \text{Commit ca id}' h, s_1') \in \text{ALM-trans 0 id}_1 \text{ and } id' < id_1 \\
&\text{thus } P_{14} (s_1', s_2) \text{ by (auto simp add: ALM-trans-def $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } h \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } 0 < id_1 \text{ and } id_1 < id_2 \text{ and } \text{in-lin}:(s_1, \text{Linearize 0 h}, s_1') \in \text{ALM-trans 0 id}_1 \\
&\text{from } \text{in-lin have } \text{initialized s}_2 \text{ and hist s}_2 = [] \text{ using } P_6 (s_1, s_2), \text{ and } P_2 (s_1, s_2) \text{ and } P_10 (s_1, s_2) \text{ and } P_2 (s_1, s_2) \text{ by (auto simp add: ALM-trans-def $P_{14}$-def $P_6$-def $P_{10}$-def $P_2$-def)}
\end{align*}
\begin{align*}
&\text{thus } P_{14} (s_1', s_2) \text{ by (auto simp add: $P_{14}$-def)}
\end{align*}
\begin{align*}
&\text{next} \\
&\text{fix } h \\
&\text{assume } P_{14} (s_1, s_2) \text{ and } 0 < id_1 \text{ and } id_1 < id_2 \text{ and } (s_1, \text{Initialize 0 h}, s_1') \in \text{ALM-trans 0 id}_1 \\
&\text{thus } P_{14} (s_1', s_2) \text{ using } P_{10} (s_1, s_2) \text{ by (auto simp add: ALM-trans-def $P_{14}$-def $P_{10}$-def)}
\end{align*}
next
assume \(P14\ (s_1, s_2)\) and \(0 < id_1\) and \(id_1 < id_2\) and \((s_1, \text{Abort} 0, s_1')\) ∈ ALM-trans 0 id_1
thus \(P14\ (s_1', s_2)\) by (auto simp add: ALM-trans-def P14-def)
qed

lemma P15-invariant: \([[id_1 < id_2]; id_1 \neq 0]] \implies \text{invariant (composeALMs id1 id2) P15}\\
proof (rule invariantI, auto)
  fix s1 s2
  assume \((s_1, s_2)\) : starts-of (composeALMs id1 id2) and \(0 < id_1\)
  thus \(P15\ (s_1, s_2)\) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P15-def)

next
fix s1 s2 s1' s2' act
assume reachable (composeALMs id1 id2) \((s_1, s_2)\) and \(P15\ (s_1, s_2)\) and \(0 < id_1\) and \(id_1 < id_2\) and in-trans-comp:\((s_1, s_2)\) — act — composeALMs id1 id2 —> \((s_1', s_2')\)
have \(P13\ (s_1, s_2)\) and \(P1b\ (s_1, s_2)\) and \(P6\ (s_1, s_2)\) and \(P1a\ (s_1, s_2)\) and \(P5\ (s_1, s_2)\) and \(P10\ (s_1, s_2)\)
proof —
  from in-trans-comp and reachable (composeALMs id1 id2) \((s_1, s_2)\) and \(P13\ (s_1, s_2)\) and \(P1b\ (s_1, s_2)\) and \(P6\ (s_1, s_2)\) and \(P1a\ (s_1, s_2)\) and \(P5\ (s_1, s_2)\) and \(P10\ (s_1, s_2)\) unfolding invariant-def by auto
  qed
from \((0 < id_1)\) and \((id_1 < id_2)\) and in-trans-comp show \(P15\ (s_1', s_2')\)
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 \((s_1, s_2)\) \((s_1', s_2')\)
  thus \(P15\ (s_1', s_2')\) using \(P15\ (s_1, s_2)\) and \(0 < id_1\) and \((id_1 < id_2)\)
proof (auto simp add: in-trans-cases-fun-def)
  fix ca ra
  assume \(P15\ (s_1, s_2)\) and in-involve: \((s_1, \text{Invoke} ca ra, s_1')\) ∈ ALM-trans 0 id1 and in-involve2: \((s_2, \text{Invoke} ca ra, s_2')\) ∈ ALM-trans id1 id2
  show \(P15\ (s_1', s_2')\)
proof —
  \{ assume \(s_1' = s_1\)
  with \(P15\ (s_1, s_2)\) and in-involve1 and in-involve2 and \(0 < id_1\) and \(id_1 < id_2\)
  have \(\text{thesis}\) by (auto simp add: ALM-trans-def P15-def)
  \} note case1 = this
  \{ assume \(s_1' \neq s_1\)
  with in-involve1 and in-involve2 and \(P6\ (s_1, s_2)\) have \(s_2' = s_2\) apply (auto simp add: ALM-trans-def P6-def) by (metis phase.simps(12) phase.simps(4))

26
with \((s_1' \neq s_1)\) and \((P15\ (s_1, s_2))\) and in-involve1 have \(?thesis\) by \((force simpt add:P15-def ALM-trans-def pendingReqs-def)\)
}

note case2 = this

from case1 and case2 show \(?thesis\) by auto

qed

next

fix ca h ru

assume \(P15\ (s_1, s_2)\) and \(0 < id1\) and \(id1 < id2\) and \((s_1, Switch\ ca\ id1\ h\ ru, s_1')\) \(\in\ ALM-trans\ 0\ id1\) and \((s_2, Switch\ ca\ id1\ h\ ru, s_2')\) \(\in\ ALM-trans\ id1\ id2\)

thus \(P15\ (s_1', s_2')\) by \((auto\ simpt add: ALM-trans-def P15-def pendingReqs-def)\)

next

fix c id' h

assume \(P15\ (s_1, s_2)\) and \(0 < id1\) and \((s_2, Commit\ c\ id'\ h, s_2')\) \(\in\ ALM-trans\ id1\ id2\) and \(id1 < id'\) and \(id' < id2\)

thus \(P15\ (s_1, s_2')\) by \((auto\ simpt add: ALM-trans-def P15-def)\)

next

fix c h r

assume \(P15\ (s_1, s_2)\) and \(0 < id1\) and \(id1 < id2\) and \((s_2, Switch\ c\ id2\ h, r, s_2')\) \(\in\ ALM-trans\ id1\ id2\)

thus \(P15\ (s_1, s_2')\) by \((auto\ simpt add: ALM-trans-def P15-def)\)

next

fix h

assume \(in-lin:(s_2, Linearize\ id1\ h, s_2')\) \(\in\ ALM-trans\ id1\ id2\)

show \(P15\ (s_1, s_2')\)

proof \((auto\ simpt add:P15-def)\)

fix r

assume phase s_2' (request-snd r) = Sleep and r \in set (hist s_2') and r \notin pendingReqs s_1

show r \in set (hist s_1)

proof -

from phase s_2' (request-snd r) = Sleep and in-lin have phase s_2 (request-snd r) = Sleep by \((auto\ simpt add: ALM-trans-def)\)

with \((P1b\ (s_1, s_2))\) have r \notin pendingReqs s_2 by \((auto\ simpt add: pendingReqs-def P1b-def)\)

with in-lin and \(r \in set (hist s_2')\) have r \in set (hist s_2) by \((auto\ simpt add: ALM-trans-def postfix-all-def linearizations-def)\)

with \((phase s_2 (request-snd r) = Sleep)\) and \((P15\ (s_1, s_2))\) and \(r \notin pendingReqs s_1\) show \(?thesis\) by \((auto\ simpt add:P15-def)\)

qed

qed

next

assume \(P15\ (s_1, s_2)\) and \(0 < id1\) and \(id1 < id2\) and \((s_2, Abort\ id1, s_2')\) \(\in\ ALM-trans\ id1\ id2\)

thus \(P15\ (s_1, s_2')\) by \((auto\ simpt add: ALM-trans-def P15-def)\)

next

fix h

assume \(in-init:(s_2, Initialize\ id1\ h, s_2')\) \(\in\ ALM-trans\ id1\ id2\)

show \(P15\ (s_1, s_2')\)
proof (auto simp add:P15-def)
  fix r
  assume phase s2' (request-snd r) = Sleep and r ∈ set (hist s2') and r ∉ pendingReqs s1
  show r ∈ set (hist s1)
  proof
    from in-init and ⟨P13 (s1, s2)⟩ have hist s2' ∈ postfix-all (hist s1) (linearizations (pendingReqs s1)) by
    (auto simp add:ALM-trans-def P13-def)
    with ⟨r ∈ set (hist s2')⟩ have r ∈ set (hist s1) ∨ r ∈ pendingReqs s1 by
    (auto simp add:postfix-all-def linearizations-def)
    with ⟨r ∉ pendingReqs s1⟩ show ?thesis by auto
  qed
next
  fix cata ra
  assume ⟨s1, Switch ca 0 ta ra, s1'⟩ ∈ ALM-trans 0 id1
  hence s1' = s1 using ⟨P5 (s1, s2)⟩ by
  (auto simp add: ALM-trans-def P5-def)
  thus P15 (s1', s2) using ⟨P15 (s1, s2)⟩ by auto
next
  fix ca id' h
  assume P15 (s1, s2) and id1 < id2 and (s1, Commit ca id' h, s1') ∈ ALM-trans 0 id1 and id' < id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
next
  fix h
  assume P15 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Linearize 0 h, s1') ∈ ALM-trans 0 id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def postfix-all-def)
next
  fix h
  assume ⟨s1, Initialize 0 h, s1'⟩ ∈ ALM-trans 0 id1
  hence s1' = s1 using ⟨P10 (s1, s2)⟩ by (auto simp add: ALM-trans-def P10-def)
  thus P15 (s1', s2) using ⟨P15 (s1, s2)⟩ by auto
next
  assume P15 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Abort 0, s1') ∈ ALM-trans 0 id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
qed
qed

4.5 The refinement proof

definition ref-mapping :: (ALM-state ∗ ALM-state) => ALM-state
  — The refinement mapping between the composition of two ALMs and a single
ALM

where
ref-mapping ≡ λ (s1, s2).

\( \text{pending} = \lambda c. (\text{if phase } s1 \neq \text{Aborted then pending } s1 \ c \ \text{else pending } s2 \ c), \)
initHists = { },

phase = \( \lambda c. (\text{if phase } s1 \neq \text{Aborted then phase } s1 \ c \ \text{else phase } s2 \ c), \)
hist = (if hist s2 = [] then hist s1 else hist s2),

\( \text{aborted} = \text{aborted } s2, \)
initialized = True.

\[ (|\text{pending}| = \lambda c. (\text{if phase } s1 \ c \neq \text{Aborted then pending } s1 \ c \ \text{else pending } s2 \ c)) \]

initHists = {},

phase = \( \lambda c. (\text{if phase } s1 \ c \neq \text{Aborted then phase } s1 \ c \ \text{else phase } s2 \ c), \)
hist = (if hist s2 = [] then hist s1 else hist s2),

\( \text{aborted} = \text{aborted } s2, \)
initialized = True.

\[ \text{theorem composition: } [|\text{id1} \neq 0; \text{id1} < \text{id2}|] \implies ((\text{composeALMs } \text{id1} \ \text{id2}) = < | \ (\text{ALM}-\text{ioa } \text{id2})) \]
— The composition theorem

\( \text{proof } – \)
assume \( \text{id1} \neq 0 \text{ and } \text{id1} < \text{id2} \)
show \( \text{composeALMs } \text{id1} \ \text{id2} = < | \ \text{ALM}-\text{ioa } \text{0 } \text{id2} \)
proof (simp add: \text{iioa-implements-def, rule conjI, rule-tac[2] conjI})
show same-input-sig:inp (composeALMs id1 id2) = inp (\text{ALM}-\text{ioa } \text{0 } \text{id2})
— First we show that both automata have the same input and output signature

using \( \text{id1} \neq 0 \text{ and } \text{id1} < \text{id2}; \text{by} \) (simp add: composeALMs-def hide-def hide-asig-def \text{ALM}-\text{ioa}-def \text{asig-inputs-def} \text{asig-outputs-def} \text{asig-of-def} \text{ALM}-\text{asig-def} \text{par-def \text{asig-comp-def}, auto})
from \( \text{id1} \neq 0 \text{ and } \text{id1} < \text{id2}; \)
show same-output-sig:out (composeALMs id1 id2) = out (\text{ALM}-\text{ioa } \text{0 } \text{id2})
— Then we show that output signatures match
by (simp add: \text{asig-inputs-def} \text{asig-outputs-def} \text{asig-of-def} \text{composeALMs-def} hide-def hide-asig-def \text{ALM}-\text{ioa}-def \text{ALM}-\text{asig-def} \text{par-def \text{asig-comp-def}, auto})
show traces (composeALMs id1 id2) <= traces (\text{ALM}-\text{ioa } \text{0 } \text{id2})
— Finally we show trace inclusion
proof (rule trace-inclusion \[\text{where } f=\text{ref-mapping}] )
— We use the mapping \text{ref-mapping}, defined before
from same-input-sig and same-output-sig show ext (composeALMs id1 id2) = ext (\text{ALM}-\text{ioa } \text{0 } \text{id2})
— First we show that they have the same external signature
by (simp add: \text{externals-def})
next
show is-ref-map \text{ref-mapping} (composeALMs id1 id2) (\text{ALM}-\text{ioa } \text{0 } \text{id2})
— Then we show that \text{ref-mapping-comp} is a refinement mapping
apply (simp add: \text{is-ref-map-def}, auto, rename-tac \text{s1} \text{s2}) prefer 2 apply (rename-tac \text{s1} \text{s2} \text{s1'} \text{s2'} act)
proof –
— First we show that start states correspond
fix \( \text{s1} \text{s2} \)
assume (s1, s2) : starts-of (composeALMs id1 id2)
thus ref-mapping (s1, s2) : starts-of (\text{ALM}-\text{ioa } \text{0 } \text{id2}) using \( \text{id1} \neq 0 \text{ and } \text{id1} < \text{id2}; \text{by} \) (simp add: \text{ALM}-\text{ioa-def} \text{ALM}-\text{start-def} \text{starts-of-def} composeALMs-def hide-def par-def \text{ref-mapping-def})
next
— Then we show the main property of a refinement mapping

\[ \text{fix } s1 \ s2 \ s1' \ s2' \ \text{act} \]
\[ \text{assume reachable:reachable (composeALMs id1 id2) (s1, s2) and in-trans-comp:(s1, s2) -- act-- composeALMs id1 id2--> (s1', s2')} \]

We make the invariants available for later use

\[ \text{have P6 (s1, s2) and P6 (s1', s2') and P9 (s1, s2) and P7 (s1, s2) and P10 (s1, s2) and P4 (s1, s2) and P5 (s1, s2) and P13 (s1, s2) and P1a (s1, s2) and P14 (s1, s2) and P14 (s1', s2') and P15 (s1, s2) and P2 (s1, s2) and P3 (s1, s2)} \]
\[ \text{proof --} \]
\[ \text{from reachable and in-trans-comp have reachable (composeALMs id1 id2) (s1', s2') by (rule reachablereachable-n)} \]
\[ \text{with P6-invariant and P9-invariant and P2-invariant and P7-invariant and P10-invariant and P4-invariant and P5-invariant and P13-invariant and P1a-invariant and P14-invariant and P15-invariant and P3-invariant (id1 \neq 0) and (id1 < id2) and reachable} \]
\[ \text{show P6 (s1, s2) and P6 (s1', s2') and P9 (s1, s2) and P7 (s1, s2) and P10 (s1, s2) and P4 (s1, s2) and P5 (s1, s2) and P13 (s1, s2) and P1a (s1, s2) and P14 (s1, s2) and P14 (s1', s2') and P15 (s1, s2) and P2 (s1, s2) and P3 (s1, s2) by (auto simp add: invariant-def)} \]
\[ \text{qed} \]
\[ \text{let ?t = ref-mapping (s1, s2)} \]
\[ \text{let ?t' = ref-mapping (s1', s2')} \]
\[ \text{show EX ex. move (ALM-iao 0 id2) ex ?t act ?t'} \]
— the main part of the proof
\[ \text{proof (simp add: move-def, auto)} \]
\[ \text{assume act : ext (ALM-iao 0 id2)} \]
\[ \text{hence act : \{act . EX c r . act = Invoke c r | (EX t . act = Switch c t 0 t r)} Un \{act . EX c r . (EX id' . 0 <= id' \& id' < id2 \& act = Commit c id' tr) \| (EX r . act = Switch c id2 tr r)} by (auto simp add: ALM-iao-def ALM-asig-def externals-def asig-inputs-def asig-outputs-def asig-0-def)
\[ \text{with in-trans-comp show EX ex. is-exec-frag (ALM-iao 0 id2) (?t, ex) \& Finite ex \& laststate (?t, ex) = ?t' \& mk-trace (ALM-iao 0 id2) \& \text{ex} = [act!]} \]
— If act is an external action of the composition, then there must be an execution of the spec with matching states and forming trace "act"
\[ \text{apply auto} \]
\[ \text{proof --} \]
\[ \text{fix c r} \]
\[ \text{assume in-invoke:(s1, s2) -- Invoke c r -- composeALMs id1 id2--> (s1', s2')} \]
— If the current action is Invoke
\[ \text{show EX ex. is-exec-frag (ALM-iao 0 id2) (?t, ex) \& Finite ex \& laststate (?t, ex) = ?t' \& mk-trace (ALM-iao 0 id2) \& \text{ex} = [Invoke c r!]} \]
\[ \text{proof --} \]
\[ \text{let ?ex = [(Invoke c r, ?t')]!} \]
\[ \text{have Finite ?ex by auto} \]
\[ \text{moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)} \]

30
\textbf{moreover have} \texttt{mk-trace (ALM-\textit{ioa} 0 id2) (\?ex) = [Invoke c \textit{r}]}}

\begin{itemize}
\item \textbf{by} \texttt{(simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-\textit{ioa}\-def ALM-asig-def)}
\end{itemize}

\textbf{moreover have} \texttt{is-exec-frag (ALM-\textit{ioa} 0 id2) (\texttt{\?t, \?ex})}

\textbf{proof} ~

\begin{itemize}
\item \textbf{assum} \texttt{s1' \neq s1 \& s2' \neq s2}
\item \texttt{contradiction}
\end{itemize}

\textbf{with} \texttt{in-involve \& \texttt{id1 \neq 0 \& id1 < id2 \& P6 (s1', s2')}}

\textbf{have} \texttt{thesis apply (simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def)
  ALM-\textit{ioa}\-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def
  asig-of-def) apply(auto simp add:ALM-trans-def P6-def) done}

\textbf{moreover}

\begin{itemize}
\item \texttt{assume s1' = s1 \& s2' = s2}
\item \texttt{with in-involve have pre-s1: (phase s1 c = Ready \& request-snd r
  c \& r \notin \texttt{set (hist s1)}) \& pre-s2: (phase s2 c = Ready \& request-snd r = c
  c \& r \notin \texttt{set (hist s2)}) using [[hypsubst-thin]] apply (auto simp add: is-exec-frag-def
  composeALMs-def trans-of-def hide-def ALM-\textit{ioa}\-def ALM-asig-def par-def actions-def
  asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp-all add:ALM-trans-def)
  apply (drule-tac[\!\!] arg-cong[where \texttt{f = phase}]) apply simp-all apply (metis
  phase.simps(8) fun-upd-idem-iff) apply (metis phase.simps(8) fun-upd-idem-iff
  (\texttt{\?t}) done}
\item \texttt{hence (phase \texttt{\?t c = Ready \& request-snd r = c \& r \notin \texttt{set (hist
  \texttt{\?t})}) using (P14 (s1, s2)) by (auto simp add:ref-mapping-def P14-def)
  hence thesis using (id1 \neq 0) \& (s1' = s1) \& (s2' = s2) apply (simp add: is-exec-frag-def
  composeALMs-def trans-of-def hide-def ALM-\textit{ioa}\-def ALM-asig-def par-def actions-def
  asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp-all add:ALM-trans-def
  force) done}
\item \texttt{moreover}
\item \texttt{assume s1' \neq s1 \& s2' = s2}
\item \texttt{with in-involve have pre-s1:phase s1 c = Ready \& request-snd r = c \& r
  \notin \texttt{set (hist s1)} \& trans-s1: s1' = s1\{pending := (pending s1)(c := r), phase :=
  (phase s1)(c := Pending)} apply (simp-all add: is-exec-frag-def composeALMs-def
  trans-of-def hide-def ALM-\textit{ioa}\-def ALM-asig-def par-def actions-def asig-outputs-def
  asig-inputs-def asig-internals-def asig-of-def) apply(simp-all add:ALM-trans-def
  ref-mapping-def) done
\item \texttt{have pre-t: phase \?t c = Ready \& request-snd r = c \& r \notin \texttt{set (hist
  \?t)}
  proof ~
  \item \textbf{from} pre-s1 \textbf{have} phase \?t c = Ready \& request-snd r = c \textbf{by}
  \begin{itemize}
  \item \texttt{(auto simp add:ref-mapping-def)}
  \end{itemize}
  \textbf{moreover have} \texttt{r \notin \texttt{set (hist \?t)}
  proof (cases hist s2 = [])}
\end{itemize}
assume hist s2 = []
with pre-s1 show ?thesis by (auto simp add:ref-mapping-def)
next
assume hist s2 ≠ []
show r ∉ set (hist ?t)
proof auto
  assume r ∈ set (hist ?t)
  with ⟨hist s2 ≠ []⟩ have r ∈ set (hist s2) by (auto simp add:ref-mapping-def)
moreover from pre-s1 and ⟨P6 (s1, s2): have phase s2
(request-snd r) = Sleep by (force simp add:P6-def)
moreover note ⟨P15 (s1, s2):
ultimately have r ∈ set (hist s1) ∨ r ∈ pendingReqs s1 by
  (auto simp add:P15-def)
with pre-s1 have r ∈ pendingReqs s1 by auto
with ⟨P1a (s1, s2): and pre-s1 show False by (auto simp add:pendingReqs-def P1a-def)
qed
qed
moreover from pre-s1 and trans-s1 and ⟨s2′ = s2: have trans-t: ?t' = ?t⟨pending := (pending ?t)(c := r), phase := (phase ?t)(c := Pending)]⟩ by
(auto simp add:ref-mapping-def fun-eq-iff)
ultimately have ?thesis apply (simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def
asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply (simp add:ALM-trans-def) done
}
moreover
{
  assume s1' = s1 and s2' ≠ s2

  with in-involve and ⟨id1 ≠ 0⟩ have pre-s2: phase s2 c = Ready & request-snd r = c & r ∉ set (hist s2) and trans-s2: s2' = s2⟨pending := (pending s2)(c := r), phase := (phase s2)(c := Pending)]⟩ apply (simp-all add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def)
apply (simp-all add:ALM-trans-def ref-mapping-def) done
from pre-s2 and ⟨P6 (s1, s2): have aborted-s1-c:phase s1 c = Aborted by (auto simp add: P6-def)
  with pre-s2 and ⟨P3 (s1, s2): and ⟨P14 (s1, s2): have pre-t:phase
  ?t c = Ready & request-snd r = c & r ∉ set (hist ?t) apply (auto simp add: fun-eq-iff ref-mapping-def P3-def P14-def) done
  moreover have trans-t: ?t' = ?t⟨pending := (pending ?t)(c := r), phase := (phase ?t)(c := Pending)] using aborted-s1-c and ⟨s1' = s1: and trans-s2 apply (force simp add: fun-eq-iff ref-mapping-def) done
ultimately have ?thesis apply (simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def

asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp add:ALM-trans-def)
done

ultimately show ?thesis by auto
qed
ultimately show ?thesis by (auto intro: exI[where x=?ex])
qed

next
fix c r h
assume in-switch:(s1, s2)−Switch c 0 h r−−composeALMs id1 id2−>

— If we get a switch 0 input (nothing happens)

show EX ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) & Finite ex & laststate
(?t, ex) = ?t' & mk-trace (ALM-ioa 0 id2)$ex = [Switch c 0 h r!]

proof —

let ?ex = [(Switch c 0 h r, ?t')!]

have Finite ?ex by auto
moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
moreover have mk-trace (ALM-ioa 0 id2)$?ex = [Switch c 0 h r!]
by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def ALM-ioa-def ALM-asig-def)
moreover have is-exec-frag (ALM-ioa 0 id2) (?t, ?ex)

proof —

from in-switch and ⟨id1 ≠ 0⟩ and ⟨id1 < id2⟩ and ⟨P5 (s1, s2)⟩ have s1' = s1 and s2' = s2 and ∨ c . phase s1 c ≠ Sleep apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(simp-all add: ALM-trans-def P5-def)
done

hence ?t = ?t' and ∨ c . phase ?t c ≠ Sleep using ⟨P6 (s1, s2)⟩
by (auto simp add:ref-mapping-def P6-def)
thus ?thesis by (simp add:is-exec-frag-def ALM-ioa-def trans-of-def ALM-trans-def)
qed
ultimately show ?thesis by (auto intro: exI[where x=?ex])
qed

next
fix c h r
assume in-switch:(s1, s2)−Switch c id2 h r−−composeALMs id1 id2−>

— The case when the system switches to a third, new, instance

show EX ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) &
Finite ex & laststate (?t, ex) = ?t' & mk-trace (ALM-ioa 0 id2)$ex = [Switch c id2 h r!]

proof —

let ?ex = [(Switch c id2 h r, ?t')!]

have Finite ?ex by auto
moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
moreover have \( mk\)-trace (\( ALM\)-ioa 0 id2) \( \forall\) (\( ?t \) \& \( ?ex \)) = \[Switch c \ id2 \ h \ r\] by (simp add: \( mk\)-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def \( ALM\)-ioa-def \( ALM\)-asig-def)

moreover have is-exec-frag (\( ALM\)-ioa 0 id2) (?t, ?ex)

proof –

from in-switch and \( id1 < id2 \) have \( s1' = s1 \) apply (simp-all add: composeALMs-def trans-of-def hide-def \( \text{par-def actions-def asig-outputs-def asig-internals-def asig-of-def ALM\text{-}ioa-def ALM\text{-}asig-def}) \) done

from \( id1 \neq 0 \) and \( id1 < id2 \) in-switch have \( \text{pre\text{-}s2:aborted} \ s2 \ & \ \text{phase} \ s2 \ c = \text{Pending} \ & \ r = \text{pending} \ s2 \ c \ & \ \text{(if initialized} \ s2 \ \text{then} \ h \in \text{postfix-all (hist s2) (linearizations (pendingReqs s2)))) \) else \( h : \text{postfix-all (l-c-p (initHists s2)) (linearizations (initValidReqs s2))} \) and \( \text{trans\text{-}s2:} \ s2' = s2 \{ \text{phase := (phase s2)(c := Aborted)} \} \) apply (simp-all add: composeALMs-def trans-of-def hide-def \( \text{par-def actions-def asig-outputs-def asig-internals-def asig-of-def ALM\text{-}ioa-def ALM\text{-}asig-def}) \) apply(auto simp add:ALM-trans-def) done

from pre-s2 have \( \text{s1\text{-}aborted:phase} \ s1 \ c = \text{Aborted using} \ \langle \text{P6} (s1, s2) \rangle \) apply(auto simp add: P6-def) done

have \( \text{pre\text{-}t:aborted} \ ?t \ & \ \text{phase} \ ?t \ c = \text{Pending} \ & \ \text{initialized} \ ?t \ & \ h : \text{postfix-all (hist} ?t \rangle \ \text{(linearizations (pendingReqs} ?t)) \} \ & \ r = \text{pending} \ ?t \ c

proof –

from \( \text{s1\text{-}aborted} \ \text{and} \ \text{pre\text{-}s2} \ ?t \ & \ \text{pending} \ ?t \ c = r \)

and \( \text{phase} \ ?t \ c = \text{Pending} \ \text{and} \ \text{initialized} \ ?t \ \text{by} \ \text{(auto simp add: ref-mapping-def fun-eq-iff)}

moreover have \( h : \text{postfix-all (hist} \ ?t \rangle \ \text{(linearizations (pendingReqs} ?t))\)

proof –

from pre-s2 have \( \text{(if initialized} \ s2 \ \text{then} \ h : \text{postfix-all (hist} s2) \ \text{(linearizations (pendingReqs} s2)) \} \ \text{else} \ h : \text{postfix-all (l-c-p (initHists s2)) (linearizations (initValidReqs s2))} \) by auto

thus \( \text{?thesis} \)

proof auto

assume case1-1;\text{initialized} \ s2 \ \text{and case1-2:h : postfix-all (hist} s2) \ \text{(linearizations (pendingReqs} s2))

hence \text{suffixeq (hist} s1) \ (hist} s2) \ \text{using} \ \langle \text{P14} (s1, s2) \rangle \ \text{by} \ \text{(auto simp add:P14-def suffixeq-def)}

show \( h \in \text{postfix-all (hist} \ ?t \rangle \ \text{(linearizations (pendingReqs} ?t))\)

proof –

have \( \text{hist} \ ?t = \text{hist} s2 \)

proof (cases \( \text{hist} s2 = [] \))

assume \( \text{hist} s2 = [] \)

show \( \text{hist} \ ?t = \text{hist} s2 \)

proof –

from \( \text{hist} s2 = [] \) and \( \text{suffixeq (hist} s1) \ (hist} s2) \ \text{have} \ \text{hist} s1 = [] \ \text{by} \ \text{(auto simp add:suffixeq-def)}\)

with \( \text{hist} s2 = [] \) show \( \text{hist} \ ?t = \text{hist} s2 \ \text{by} \ \text{(auto simp add: ref-mapping-def)}\)

qed

next
assume \( \text{hist } s2 \neq [] \)

thus \( \text{hist } \ ?t \ = \ \text{hist } s2 \) by (simp add:ref-mapping-def)

qed

moreover have \( \text{pendingReqs } s2 \leq \text{pendingReqs } ?t \)

proof (simp add: pendingReqs-def, clarify)

fix \( c \)

assume \( \text{pending } s2 \ c \notin \text{set } (\text{hist } s2) \) and \( \text{phase } s2 \ c = \text{Pending} \lor \text{phase } s2 \ c = \text{Aborted} \)

moreover with \( \langle P6 \ (s1, s2) \rangle \) have \( \text{phase } s1 \ c = \text{Aborted} \)

by (auto simp add:P6-def)

moreover note case1-2

ultimately show \( \exists \ c. \ \text{pending } s2 \ c \ = \ \text{pending } ?t \ c \) and \( \text{phase } s2 \ c \notin \text{set } (\text{hist } ?t) \) and \( \text{phase } ?t \ c = \text{Pending} \lor \text{phase } ?t \ c = \text{Aborted} \)

apply (simp add:ref-mapping-def suffixeq-def) by (metis prefixeq-Nil prefixeq-def self-append-conv2)

qed

moreover note case2-1

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

by (auto simp add: linearizations-def postfix-all-def)

qed

next

assume case2-1: \( \neg \text{initialized } s2 \) and case2-2: \( \text{h : postfix-all (l-c-p (initHists s2))} \)

(\( \text{linearizations (initValidReqs s2)} \) )

from case2-1 and :P10 (s1, s2) have \( \text{hist } s2 \ = \ [] \) by (auto simp add:P10-def)

have \( \ h : \text{postfix-all } (\text{hist } s1) \)

(\( \text{linearizations } (\text{pendingReqs } s1) \) )

proof

from pre-s2 have \( \text{phase } s2 \ c \neq \text{Sleep} \) by auto

moreover note \( \langle P13 \ (s1, s2) \rangle \) and case2-1 and case2-2

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

by (auto simp add:P13-def)

qed

moreover from \( \langle \text{hist } s2 = [] \rangle \) have \( \text{hist } ?t = \text{hist } s1 \) by (auto simp add:P10-def ref-mapping-def)

moreover have \( \text{pendingReqs } ?t = \text{pendingReqs } s1 \)

proof auto

fix \( r \)

assume \( r \in \text{pendingReqs } ?t \)

with this obtain \( c' \ where \ r = \text{pending } ?t \ c' \) and \( r \notin \text{set } (\text{hist } ?t) \) and \( \text{phase } ?t \ c' \in \{\text{Pending, Aborted}\} \) by (auto simp add:pendingReqs-def)

show \( r \in \text{pendingReqs } s1 \)

proof (cases phase s1 c' = Aborted)

assume \( \text{phase } s1 \ c' = \text{Aborted} \)

with \( \langle \text{phase } ?t \ c' \in \{\text{Pending, Aborted}\} \rangle \) and \( \langle r = \text{pending } ?t \ c' \rangle \) have \( \text{phase } s2 \ c' \in \{\text{Pending, Aborted}\} \) and \( r = \text{pending } s2 \ c' \) by (auto simp add:ref-mapping-def)

with \( \langle P6 \ (s1, s2) \rangle \) and case2-1 and \( \langle P7 \ (s1, s2) \rangle \) and \( \langle \text{hist } ?t = \text{hist } s1 \rangle \) and \( \langle r \notin \text{set } (\text{hist } ?t) \rangle \) have \( \text{phase } s1 \ c' = \text{Aborted} \) and \( r = \text{pending } s1 \ c' \) and \( r \notin \text{set } (\text{hist } s1) \)

apply (auto simp add: P6-def P7-def) apply force apply force done
thus ?thesis by (auto simp add: pendingReqs-def)

next
  assume phase s1 c' ≠ Aborted
  with | r = pending ?t c' | and | r ∉ set (hist ?t) | and | phase ?t c' ∈ {Pending, Aborted} | and | hist ?t = hist s1 | show ?thesis by (auto simp add:ref-mapping-def pendingReqs-def)
  qed
next
  fix r
  assume r ∈ pendingReqs s1
  with this obtain c where
    r = pending s1 c
    and phase s1 c ∈ {Pending, Aborted}
    and r ∉ set (hist s1)
  by (auto simp add: pendingReqs-def P7-def)
  with | hist s2 = [] | and | ¬ initialized s2 | and | P7 (s1, s2) |
  show r ∈ pendingReqs ?t by (auto simp add:ref-mapping-def pendingReqs-def)
  qed
ultimately show ?thesis by (auto simp add: postfix-all-def linearizations-def)
  qed
ultimately show ?thesis by auto
  qed
moreover have trans-t;?t′ = ?t(phase := (phase ?t)(c := Aborted))]
  using s1-aborted and (s1′ = s1) and trans-s2 by (auto simp add:ref-mapping-def fun-eq-iff)
apply(simp add:ALM-trans-def) done

ultimately show ?thesis by (auto intro: exI[where x=ex])
  qed
next
  fix c h id'
  assume in-commit:(s1, s2)–Commit c id' h–composeALMs id1 id2–→ (s1', s2') and id' < id2
  — Case when the composition commits a request
  show ∃ ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) ∧ Finite ex ∧ laststate (?t, ex) = ?t' ∧ mk-trace (ALM-ioa 0 id2)·ex = [Commit c id' h]!

  proof
    let ?ex = [(Commit c id' h, ?t')]
    have Finite ?ex by auto
    moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
    moreover have mk-trace (ALM-ioa 0 id2)$(?ex) = [Commit c id' h] using |id' < id2| by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def)
    moreover have is-exec-frag (ALM-ioa 0 id2) (?t, ?ex)
    proof —
\[
\begin{aligned}
\{ & \text{assume id' < idl} \\
& \text{with in-commit have s2' = s2 and pre-s1:phase s1 c = Pending} \\
& \quad \land \text{pending s1 c \in set } \{\text{hist s1}\} \land h = \text{dropWhile } (\lambda r . r \neq \text{pending } \text{?t} \text{ c}) (\text{hist s1}) \text{ and trans-s1:s1' = s1 } (\text{phase := } (\text{phase s1})(c := \text{Ready})) \text{ apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done} \\
& \quad \text{from pre-s1 have s1-not-aborted-c:phase s1 c \neq \text{Aborted by auto} has} \\
& \quad \text{have pre-t:phase ?t c = Pending & pending ?t c \in set } \{\text{hist ?t}\} \land h \\
& \quad \text{= dropWhile } (\lambda r . r \neq \text{pending } \text{?t} \text{ c}) (\text{hist ?t}) \\
& \quad \text{proof } (\text{cases hist s2 = []}) \\
& \quad \text{assume hist s2 = []} \\
& \quad \text{with pre-s1 and } \text{phase s1 c \neq \text{Aborted}} \text{ show } ?\text{thesis by (auto simp add: ref-mapping-def)} \\
& \quad \text{next hist s2 \neq []} \\
& \quad \text{hence initialized s2 using (P10 (s1, s2)) by (auto simp add:P10-def) has} \\
& \quad \text{from pre-s1 and } \text{phase s1 c \neq \text{Aborted: have phase ?t c = Pending} and pending ?t c = pending s1 c and pending s1 c \in set } \{\text{hist s1}\} \text{ by (auto simp add:ref-mapping-def)} \\
& \quad \text{moreover have pending ?t c \in set } \{\text{hist ?t}\} \\
& \quad \text{proof --} \\
& \quad \text{from } \text{initialized s2} \text{ and } (P14 (s1, s2)) \text{ obtain rs3 where hist s2 = rs3 } @ (\text{hist s1}) \text{ by (auto simp add:P14-def) has} \\
& \quad \text{with } \text{pending s1 c \in set } \{\text{hist s1}\} \text{ and } \text{hist s2 = rs3 } @ (\text{hist s1}) \text{ and pending ?t c = pending s1 c} \text{ show } \text{pending ?t c \in set } \{\text{hist ?t}\} \text{ by (auto simp add:ref-mapping-def suffixeq-def)} \\
& \quad \text{qed} \\
& \quad \text{moreover have h = dropWhile } (\lambda r . r \neq \text{pending } ?t c) (\text{hist ?t}) \\
& \quad \text{proof --} \\
& \quad \text{from } \text{pending s1 c \in set } \{\text{hist s1}\} \text{ obtain rs1 rs2 where hist s1 = rs2 } @ rs1 \text{ and hd rs1 = pending s1 c and rs1 \neq [] and pending s1 c \notin set rs2 by (metis list.sel(1) in-set-conj-decomp-first list.simps(3)) has} \\
& \quad \text{with } \text{pending ?t c = pending s1 c} \text{ and } \text{dropWhile-lemma[of hist s1 rs1 pending s1 c] and pre-s1 have h = rs1 by auto has} \\
& \quad \text{moreover have dropWhile } (\lambda r . r \neq \text{pending } ?t c) (\text{hist ?t}) = \text{rs1} \\
& \quad \text{proof --} \\
& \quad \text{from } \text{initialized s2} \text{ and } (P14 (s1, s2)) \text{ obtain rs3 where hist s2 = rs3 } @ (\text{hist s1}) \text{ and set rs3 } \cap \text{ set } \{\text{hist s1}\} = {} \text{ by (auto simp add:P14-def) has} \\
& \quad \text{with } \text{pending s1 c } \in \text{ set } \{\text{hist s1}\} \text{ and } \text{hist s1 = rs2 } @ rs1 \text{ have hist s2 = rs3 } @ rs2 \cap rs1 \text{ and pending s1 c } \notin \text{ set rs3 by auto has} \\
& \quad \text{with } \text{pending s1 c } \notin \text{ set rs2} \text{ obtain rs4 where hist s2 = rs4 } @ rs1 \text{ and pending s1 c } \notin \text{ set rs4 by auto has} \\
& \quad \text{with } \text{hd rs1 = pending s1 c and rs1 } \neq [] \text{ and dropWhile-lemma[of hist s2 rs1 pending s1 c] have dropWhile } (\lambda r . r \neq \text{pending s1 c}) (\text{hist s2}) = \text{rs1 by auto has} \\
& \quad \text{thus } ?\text{thesis using } \text{hist s2 } \neq [] \text{ and } \text{pending ?t c = pending }
\end{aligned}
\]
s1 c: by (auto simp add:ref-mapping-def)
    qed
    ultimately show ?thesis by auto
    qed
    ultimately show ?thesis by auto
    qed

moreover from ⟨s2′ = s2⟩ and s1-not-aborted-c and trans-s1
    have trans-t:?t′ = ?t {(phase := (phase ?t)(c := Ready))} by (simp add:fun-eq-iff ref-mapping-def)
    ultimately have ?thesis using ⟨id1 < id2⟩ apply (simp add:
is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def
par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp add:ALM-trans-def) done

moreover {
    assume id1 ≤ id′
    with in-commit have s1′ = s1 and pre-s2:phase s2 c = Pending ∧ pending s2 c ∈ set (hist s2) ∧ h = dropWhile (λ r . r ≠ pending s2 c) (hist s2) and trans-s2:s2′ = s2 {(phase := (phase s2)(c := Ready))} apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def
asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done
    from pre-s2 and ⟨P6 (s1, s2)⟩ have facts:aborted s1 & phase s1 c = Aborted & hist s2 ≠ [] by (force simp add:P6-def)
    with pre-s2 have pre-t:phase ?t c = Pending ∧ pending ?t c ∈ set (hist ?t) ∧ h = dropWhile (λ r . r ≠ pending ?t c) (hist ?t) by (auto simp add:ref-mapping-def)
    moreover from ⟨s1′ = s1⟩ and facts and trans-s2 have
trans-t: ?t′ = ?t {(phase := (phase ?t)(c := Ready))} by (auto simp add:fun-eq-iff ref-mapping-def)
    ultimately have ?thesis using ⟨id1 < id2⟩ apply (simp add:
is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def
par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp add:ALM-trans-def) done
  }
  ultimately show ?thesis using ⟨id′ < id2⟩ by force
    qed
    ultimately show ?thesis by (auto intro: extI[where x=?ex])
    qed
    qed

— We finished the case when the composition takes an action that is in
the external signature of the spec

next
  assume act ∉ ext (ALM-ioa 0 id2)
  — Now the case when the composition takes an action that is not in
the external signature of the spec
  with in-trans-comp and ⟨id1 < id2⟩ and ⟨id1 ≠ 0⟩ have act : {act . act = Abort 0 | act = Abort id1 | (EX c r h . act = Linearize 0 h | act =
Linearize id1 h | act = Switch c id1 h r | act = Initialize 0 h | act = Initialize id1 h) by (auto simp add: composeALMs-def hide-def hide-asig-def ALM-ioa-def ALM-asig-def externals-def asig-inputs-def asig-outputs-def asig-internals-def asig-of-def trans-of-def par-def actions-def)

with in-trans-comp show ∃ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) ∧ Finite ex ∧ laststate (?t, ex) = ?t' ∧ mk-trace (ALM-ioa 0 id2)-ex = nil proof auto
  assume in-abort: (s1, s2) — Abort 0 — composeALMs id1 id2 → (s1', s2')
  — The case where the first Abstract aborts
  moreover with id1 ≠ 0 and id1 < id2 and P6 (s1, s2) and P2 (s1, s2) have ∀ c . phase s1 c = Aborted and hist s2 = [] and ∃ c . phase s2 c = Sleep apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply (auto simp add: fun-eq-iff ALM-trans-def ref-mapping-def P6-def P2-def) done thus ?thesis
  proof simp
    let ?ex = nil
    have Finite ?ex by auto
    moreover have laststate (?t, ?ex) = ?t by (simp add: laststate-def)
    moreover have mk-trace (ALM-ioa 0 id2)-ex = nil using id1 < id2 by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def)
    moreover have is-exec-frag (ALM-ioa 0 id2) (?t, ?ex) by (auto simp add: is-exec-frag-def)
    ultimately show ∃ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) ∧ Finite ex ∧ laststate (?t, ex) = ?t ∧ mk-trace (ALM-ioa 0 id2)-ex = nil by (auto intro: exI[where x=?ex])
  qed

next
  assume in-abort: (s1, s2) — Abort id1 — composeALMs id1 id2 → (s1', s2')
  — The case where the second ALM aborts
  show ?thesis
  proof –
    let ?ex = [(Abort 0, ?t')!]
    have Finite ?ex by auto
    moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
    moreover have mk-trace (ALM-ioa 0 id2)-ex = nil by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def)
    moreover have is-exec-frag (ALM-ioa 0 id2) (?t, ?ex) by (auto simp add: is-exec-frag-def)
    proof –
      from in-abort and id1 ≠ 0 have s1' = s1 and pre-s2: aborted s2 & (∃ c . phase s2 c ≠ Sleep) and trans-s2:s2' = s2[aborted:= True] apply (simp-all

from pre-s2 and \(P6(s1, s2)\) have pre-t: ~ aborted ?t \& (\(\exists c \) . phase ?t c \# Sleep) apply (force simp add: ref-mapping-def P6-def) done 

moreover from trans-s2 and \(\langle s1' = s1 \rangle\) have trans-t: ?t' = ?t\(|\{\text{aborted} := \text{True}\}\) by (auto simp add: fun-eq-iff ref-mapping-def) 


qed 

ultimately show ?thesis by (auto intro: exI [where \(x = \text{ex}\)]) 

qed 

next 

fix \(h\) 

assume in-lin: (\(s1, s2\) \(\rightarrow\) Linearize 0 \(h\) \(\rightarrow\) composeALMs id1 id2 \(\rightarrow\) \(s1', s2'\) 

— If the composition executes Linearize 0 

show ?thesis 

proof 

let \(\text{ex} = [(\text{Linearize 0} h, ?t')]\] 

have Finite \(\text{ex}\) by auto 

moreover have laststate (\(?, \text{ex}\)) = ?t' by (simp add: laststate-def) 

moreover have mk-trace (\(\text{ALM-}ioa\ 0\ id2\)) \(\cdot\) \(\text{ex} = \text{nil}\) by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def) 

moreover have is-exec-frag (\(\text{ALM-}ioa\ 0\ id2\) (?t, \(\text{ex}\)) by (simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply (auto simp add: ALM-trans-def) done 

have pre-t: initialized ?t \& ~ aborted ?t \& \(h \in \text{postfix-all}(\text{hist} ?t)\) (linearizations (pendingReqs ?t)) by (simp add: linearizations (pendingReqs ?t)) 

proof 

from pre-s1 have ~ aborted \(s1\) by auto 

with \(P9(s1, s2)\) have ~ aborted ?t and initialized ?t by (auto simp add: ref-mapping-def P9-def) 

moreover have \(h \in \text{postfix-all}(\text{hist} ?t)\) (linearizations (pendingReqs ?t)) by (auto simp add: linearizations (pendingReqs ?t)) 

proof 

from \(\neg\) aborted \(s1\) have hist ?t = hist \(s1\) using \(P6(s1, s2)\) and \(P2(s1, s2)\) by (auto simp add: P6-def P2-def ref-mapping-def) 

moreover have pendingReqs \(s1 \subseteq\) pendingReqs ?t by (auto simp add: pendingReqs_def) 

proof 

fix \(x\)
assume \( x \in \text{pendingReqs} \ s1 \)

moreover note \( \neg \text{aborted} \ s1 \) and \( \langle P6 (s1, s2) \rangle \)

ultimately obtain \( c \) where \( x = \text{pending} \ s1 \ c \) and \( \text{phase} \ s1 \ c \notin \text{set} \ (\text{hist} \ s1) \) by \( \text{(auto simp add: pendingReqs-def P6-def)} \)

thus \( x \in \text{pendingReqs} \ ?t \) using \( \langle \text{hist} \ ?t = \text{hist} \ s1 \rangle \) by \( \text{(force simp add: ref-mapping-def pendingReqs-def)} \)

qed

moreover from \( \text{pre-s1} \) have \( h \in \text{postfix-all} \ (\text{hist} \ s1) \) \( \text{(linearizations (pendingReqs} \ s1)) \) by auto

ultimately show \( ?\text{thesis} \) by \( \text{(auto simp add: postfix-all-def linearizations-def)} \)

qed

ultimately show \( ?\text{thesis} \) by \( \text{(auto simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def)} \) \( \text{apply(auto simp add:ALM-trans-def)} \)
done

ultimate show \( ?\text{thesis} \) by \( \text{(auto intro: exI[where x=?ex])} \)

next

fix \( h \)

assume \( \text{in-lin}(s1, s2) - \text{Linearize id1} h - \text{composeALMs} \ id1 id2 \to (s1', s2') \)

— If the composition executes \text{Linearize} \ id1

let \( ?ex = [(\text{Linearize} \ id1 \ h, \ ?t')!] \)

have \( \text{Finite} ?ex \) by auto

moreover have \( \text{laststate} (\ ?t, \ ?ex) = \ ?t' \) by \( \text{(simp add: laststate-def)} \)

moreover have \( \text{mk-trace} \ (\text{ALM-ioa 0 id2}) \ ? ex = \text{nil} \) by \( \text{(simp add: mk-trace-def externals-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def)} \)

moreover have \( \text{is-exec-frag} \ (\text{ALM-ioa 0 id2}) \ (?t, \ ?ex) \)

proof —

from \( \text{in-lin and (id1 \neq 0)} \) have \( s1' = s1 \) and \( \text{pre-s2: initialized} \ s2 \)
∧ ¬ aborted s2 ∧ h ∈ postfix-all (hist s2) (linearizations (pendingReqs s2)) and
trans-s2: s2′ = s2[hist := h] apply (simp-all add: composeALMs-def trans-of-def
hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def
ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done

have pre-t:initialized ?t ∧ ¬ aborted ?t ∧ h ∈ postfix-all (hist ?t)
(linearizations (pendingReqs ?t))
proof –
  have ¬ aborted ?t and initialized ?t using pre-s2 by (auto simp
add:ref-mapping-def)
  moreover have h ∈ postfix-all (hist ?t) (linearizations (pendingReqs
?t))
proof –
  from pre-s2 have initialized s2 by auto
  hence suffixeq (hist s1) (hist s2) using :P14 (s1, s2) by (auto
simp add:P14-def suffixeq-def)
  hence hist ?t = hist s2 by (auto simp add:ref-mapping-def)
  moreover have pendingReqs s2 ⊆ pendingReqs ?t
proof auto
  fix x
  assume x ∈ pendingReqs s2
  from this obtain c where x = pending s2 c and phase
s2 c ∈ {Pending, Aborted} and pending s2 c \notin set (hist s2) by (auto simp
add:pendingReqs-def)
  with :P6 (s1, s2); and hist ?t = hist s2; show x ∈ pendingReqs
?t by (force simp add:ref-mapping-def P6-def pendingReqs-def)
qed
moreover from pre-s2 have h ∈ postfix-all (hist s2) (linearizations
(pendingReqs s2)) by auto
ultimately show ?thesis by (auto simp add:postfix-all-def
linearizations-def)
qed
ultimately show ?thesis by auto
qed
moreover have trans-t: ?t′ = ?t(hist := h)
proof –
  from pre-s2 and trans-s2 have initialized s2′ by auto
  hence suffixeq (hist s1′) (hist s2′) using :P14 (s1′, s2′) by (auto
simp add:P14-def suffixeq-def)
  hence hist ?t′ = hist s2′ by (auto simp add:ref-mapping-def)
  with trans-s2 and (s1′ = s1) show ?thesis by (auto simp
add:ref-mapping-def fun-eq-iff)
qed
ultimately show ?thesis apply (simp add: is-exec-frag-def composeALMs-def
trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def
asig-inputs-def asig-internals-def asig-of-def) apply(auto simp add:ALM-trans-def) done
qed
ultimately show ?thesis by (auto intro: exI[where x=?ex])
next
  fix $c \ast h$
  assume in-switch: $(s1, s2) \Rightarrow \text{Switch } c \ast id1 \ast h \ast \text{composeALMs } id1 \ast id2 \Rightarrow$

  $(s1', s2')$
  — If the composition switches internally
  show $?thesis$
  proof  
  let $?ex = \text{nil}$ by auto
  moreover have laststate $(?t, $?ex) = ?t$ by (simp add: laststate-def)
  moreover have mk-trace $(ALM-\text{ioa } 0 \ast id2) \ast $?ex = \text{nil}$ by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def)
  moreover have is-exec-frag $(ALM-\text{ioa } 0 \ast id2) (\text{?t, } $?ex)$ by (auto simp add:is-exec-frag-def)
  moreover have $?t' = ?t$
  proof  
  from in-switch and $(id1 \neq 0)$ have pre-s1: aborted $s1 \ast \text{phase } s1 \ast c = \text{Pending} \ast r = \text{pending } s1 \ast c \ast (\text{if initialized } s1 \text{ then } (h \in \text{postfix-all } \text{hist } s1) \ast (\text{linearizations } (\text{pendingReqs } s1))) \ast \text{else } (h : \text{postfix-all } (t-c-p \ast (\text{initHists } s1))) \ast \text{and } \text{trans-s1: } s1' = s1 \ast (\text{phase := phase } s1 \ast (c := \text{Aborted}))$ apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-\text{ioa-def ALM-asig-def}) apply (auto simp add:ALM-trans-def) done
  have pre-s2: phase $s2 \ast c = \text{Sleep}$ and \text{trans-s2: } $s2' = s2 \langle \text{initHists := } \{h\} \cup \text{(initHists } s2)\rangle, \text{phase := (phase } s2\ast (c := \text{Pending}), \text{pending := (pending } s2\ast (c := r))\rangle$
  proof  
  from pre-s1 have phase $s1 \ast c = \text{Pending}$ by auto
  with $\langle \text{P6 } (s1, s2) \rangle$ have phase $s2 \ast c = \text{Sleep}$ apply (simp add:P6-def)
  by (metis phase,simps(10))
  with in-switch and $(id1 \neq 0)$ and $(id1 < id2)$ show phase $s2 \ast c = \text{Sleep}$ and $s2' = s2 \langle \text{initHists := } \{h\} \cup \text{(initHists } s2)\rangle, \text{phase := (phase } s2\ast (c := \text{Pending}), \text{pending := (pending } s2\ast (c := r))\rangle$ apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-\text{ioa-def ALM-asig-def}) apply (auto simp add:ALM-trans-def P6-def) done
  qed
  from pre-s1 and pre-s2 and trans-s1 and trans-s2 and (P1a $(s1, s2)$): have pending $?t \ast c = \text{pending } ?t' \ast c \& \text{initHists } ?t = \text{initHists } ?t' \& \text{hist } ?t = \text{hist } ?t' \& \text{aborted } ?t = \text{aborted } ?t' \& \text{phase } ?t' \ast c = \text{phase } ?t \ast c$ by (simp add:ref-mapping-def fun-eq-iff P1a-def)
  moreover note pre-s1 and pre-s2 and trans-s1 and trans-s2
  ultimately show $?thesis$ by (force simp add:ref-mapping-def fun-eq-iff)
  qed
  ultimately show $?thesis$ by (auto intro: exI[where $x=\text{?ex}$])
  qed
  next
  fix $h$
assume in-initialize: \((s_1, s_2)\) \(\rightarrow \) Initialize \(0 \ h \ \\text{composeALMs} \ id1 \ id2 \rightarrow (s_1', s_2')\)

hence \(\text{False}\) using \(\langle P10 \ (s_1, s_2) \rangle\) apply \((\text{simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def})\) apply \((\text{auto simp add: ALM-trans-def P10-def})\) done

thus \(?\text{thesis}\) by \(\text{auto}\)

next
fix \(h\)
assume in-initialize: \((s_1, s_2)\) \(\rightarrow \) Initialize \(id1 \ h \ \\text{composeALMs} \ id1 \ id2 \rightarrow (s_1', s_2')\)

— If the second ALM of the composition initializes

let \(?\text{ex} = [\langle \text{Linearize} \ id1 \ h, \ ?t'\rangle]\)

have \(\text{Finite} \ ?\text{ex}\) by \(\text{auto}\)

moreover have laststate \((?t, ?\text{ex}) = ?t'\) by \((\text{simp add: laststate-def})\)

moreover have \(\text{mk-trace} \ (\text{ALM-ioa} \ 0 \ id2) \cdot ?\text{ex} = \text{nil}\) by \((\text{simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def})\)

moreover have \(\text{is-exec-frag} \ (\text{ALM-ioa} \ 0 \ id2) \ (?t, ?\text{ex})\)

proof

from in-initialize and \(\text{id1} \neq 0\) have \(s_1' = s_1\) and \(\text{pre-s2}:(\exists \ c . \ \text{phase} s_2 c \neq \text{Sleep})\) \(\land \) \(\neg \text{aborted} s_2 \ \land \ \neg \text{initialized} s_2 \ \land \ h \in \text{postfix-all} \ ((l-c-p \ (\text{initHists} s_2)) \ (\text{linearizations} \ (\text{initValidReqs} s_2)))\) \(\land \) \(\text{trans-s2}:s_2' = s_2(\text{hist} := h, \ \text{initialized} := \text{True})\) apply \((\text{simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def})\) apply \((\text{auto simp add: ALM-trans-def})\) done

have \(\text{pre-t:initialized} \ ?t \ \land \ \neg \text{aborted} \ ?t \ \land \ h \in \text{postfix-all} \ (\text{hist} \ ?t)\)

(proof

from \(\text{pre-s2}\) have \(\text{initialized} \ ?t \ \land \ \neg \text{aborted} \ ?t\) by \((\text{auto simp add: ref-mapping-def})\)

moreover have \(h \in \text{postfix-all} \ (\text{hist} \ ?t)\) \((\text{linearizations} \ (\text{pendingReqs} \ ?t))\)

(proof

from \(\text{pre-s2}\) have \(h \in \text{postfix-all} \ ((l-c-p \ (\text{initHists} s_2)) \ (\text{linearizations} \ (\text{initValidReqs} s_2)))\) \(\land \ \neg \text{initialized} s_2 \ \land \ \exists \ c . \ \text{phase} s_2 c \neq \text{Sleep}\) by \(\text{auto}\)

with \(\langle P13 \ (s_1, s_2) \rangle\) have \(h \in \text{postfix-all} \ (\text{hist} s_1)\) \((\text{linearizations} \ (\text{pendingReqs} s_1))\) by \((\text{auto simp add:P13-def})\)

moreover from \(\neg \text{initialized} s_2\) and \(\langle P10 \ (s_1, s_2) \rangle\) have \(\text{hist} \ ?t = \text{hist} s_1\) by \(\text{(auto simp add:ref-mapping-def P10-def})\)

moreover have \(\text{pendingReqs} s_1 \subseteq \text{pendingReqs} \ ?t\)

proof \(\text{auto}\)

fix \(x\)

assume \(x \in \text{pendingReqs} s_1\)

from this obtain \(c\) where \(x = \text{pending} s_1 \ c \ \text{and phase} s_1 c \in \{\text{Pending, Aborted}\} \ \text{and pending} s_1 c \notin \text{set} (\text{hist} s_1)\) by \((\text{auto simp add:pendingReqs-def})\)

show \(x \in \text{pendingReqs} \ ?t\)

proof \(\text{(cases phase} s_1 c = \text{Pending})\)
assume phase s1 c = Pending

with ⟨x = pending s1 c⟩ and ⟨pending s1 c \notin set (hist s1)⟩ and ⟨hist ?t = hist s1⟩

show thesis by (force simp add:ref-mapping-def pendingReqs-def)

next

assume phase s1 c \neq Pending

with ⟨phase s1 c \in \{Pending, Aborted\}⟩ have phase s1 c = Aborted by auto

with ⟨\neg initialized s2⟩ and ⟨P6 (s1, s2)⟩ and ⟨P7 (s1, s2)⟩ have pending s2 c = pending s1 c and phase s2 c \in \{Pending, Aborted\} by (auto simp add:P6-def P7-def)

with ⟨x = pending s1 c⟩ and ⟨pending s1 c \notin set (hist s1)⟩ and ⟨hist ?t = hist s1⟩ and ⟨P6 (s1, s2)⟩ show thesis by (auto simp add:ref-mapping-def pendingReqs-def P6-def)

qed

qed

ultimately show thesis by (auto simp add:postfix-all-def linearizations-def)

qed

ultimately show thesis by auto

qed

moreover have trans-t : ?t' = ?t[hist := h]

proof -

from pre-s2 have \exists c . phase s2 c \neq Sleep by auto

with trans-s2 have initialized s2' and \exists c . phase s2' c \neq Sleep by auto

hence suffixeq (hist s1') (hist s2') using ⟨P14 (s1', s2')⟩ by (auto simp add:P14-def suffixeq-def)

hence hist ?t' = hist s2' by (auto simp add:ref-mapping-def)

with trans-s2 and ⟨s1' = s1⟩ show thesis by (auto simp add:ref-mapping-def fun-eq-iff)

qed


done

qed

ultimately show thesis by (auto intro: exI[where x=?ex])

qed

qed

qed

end
5 Conclusion

In this document we have defined the ALM automaton (a shorthand for Aboratable Linearizable Modules) and we have proved that the composition of two instances of the ALM automaton behaves like a single instance of the ALM automaton. This theorem justifies the compositional proof technique presented in [1].

References


