

Abortable Linearizable Modules

Rachid Guerraoui Viktor Kuncak Giuliano Losa

May 27, 2015

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* $\sim\sim$ /src/HOL/Library/Sublist

begin

definition *common-postfix-p* :: ('a list) set => 'a list => bool

— Predicate that recognizes the common postfix of a set of lists

— The common postfix of the empty set is the empty list

where

common-postfix-p $\equiv \lambda xss\ xs . \text{if } xss = \{\} \text{ then } xs = [] \text{ else } \text{ALL } xs' . xs' \in xss \longrightarrow \text{suffixeq } xs\ xs'$

definition *l-c-p-pred* :: 'a list set => 'a list => bool

— Predicate that recognizes the longest common postfix of a set of lists

where

l-c-p-pred $\equiv \lambda xss\ xs . \text{common-postfix-p } xss\ xs \wedge (\text{ALL } xs' . \text{common-postfix-p } xss\ xs' \longrightarrow \text{suffixeq } xs'\ xs)$

definition *l-c-p* :: 'a list set => 'a list

— The longest common postfix of a set of lists

where

l-c-p $\equiv \lambda xss . \text{THE } xs . \text{l-c-p-pred } xss\ xs$

lemma *l-c-p-ok*: *l-c-p-pred* *xss* (*l-c-p* *xss*)

— Proof that the definition of the longest common postfix of a set of lists is consistent

lemma *l-c-p-lemma*:

— A useful lemma

$(ls \neq \{\} \wedge (\forall l \in ls . (\exists l' . l = l' @ xs))) \longrightarrow \text{suffixeq } xs\ (\text{l-c-p } ls)$

lemma *l-c-p-common-postfix*: *common-postfix-p* *xss* (*l-c-p* *xss*)

using *l-c-p-ok*[of *xss*] **by** (*auto simp add:l-c-p-pred-def*)

lemma *l-c-p-longest*: *common-postfix-p* *xss* *xs* $\longrightarrow \text{suffixeq } xs\ (\text{l-c-p } xss)$

using *l-c-p-ok*[of *xss*] **by** (*auto simp add:l-c-p-pred-def*)

end

3 The ALM Automata specification

theory *ALM*

imports $\sim\sim$ /src/HOL/HOLCF/IOA/meta-theory/IOA *LCP*

begin

typedecl *client*

— A non-empty set of clients

typedecl *data*

— Data contained in requests

datatype *request* =

— A request is composed of a sender and data

Req client data

definition *request-snd* :: *request* \Rightarrow *client*

where *request-snd* $\equiv \lambda r. \text{case } r \text{ of } \text{Req } c \rightarrow c$

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* \Rightarrow *hist set*

— The possible linearizations of a set of requests

where

linearizations $\equiv \lambda reqs. \{h. \text{set } h \subseteq reqs \wedge \text{distinct } h\}$

definition *postfix-all* :: *hist* \Rightarrow *hist set* \Rightarrow *hist set*

— appends to the right the first argument to every member of the history set

where

postfix-all $\equiv \lambda h hs. \{h'. \exists h''. h' = h'' @ h \wedge h'' \in hs\}$

definition

ALM-asig :: *nat* \Rightarrow *nat* \Rightarrow *ALM-action signature*

— The action signature of ALM automata

— Input actions, output actions, and internal actions

where

ALM-asig $\equiv \lambda id1 id2. ($

$\{act. \exists c r h.$

$act = \text{Invoke } c r \mid act = \text{Switch } c id1 h r\},$

$\{act. \exists c h r id'.$

$id1 \leq id' \wedge id' < id2 \wedge act = \text{Commit } c id' h$

$\mid act = \text{Switch } c id2 h r\},$

$\{act. \exists h.$

$act = \text{Abort } id1$

$\mid act = \text{Linearize } id1 h$

| $act = Initialize\ id1\ h\}$)

record *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 $\equiv \lambda s . \{r . \exists c .$

$r = pending\ s\ c$

$\wedge r \notin set\ (hist\ s)$

$\wedge phase\ s\ c \in \{Pending, Aborted\}\}$

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 $\equiv \lambda s . \{r .$

$(r \in pendingReqs\ s \vee (\exists h \in initHists\ s . r \in set\ h))$

$\wedge r \notin set\ (l-c-p\ (initHists\ s))\}$

definition

ALM-trans :: *nat* \Rightarrow *nat* \Rightarrow (*ALM-action*, *ALM-state*)*transition set*

— the transitions of the ALM automaton

where

ALM-trans $\equiv \lambda id1\ id2 . \{trans .$

$let\ s = fst\ trans; s' = snd\ (snd\ trans); a = fst\ (snd\ trans)\ in$

$case\ a\ of\ Invoke\ c\ r \Rightarrow$

$if\ phase\ s\ c = Ready \wedge request-snd\ r = c \wedge r \notin set\ (hist\ s)$

$then\ s' = s[pending := (pending\ s)(c := r),$

$phase := (phase\ s)(c := Pending)]$

$else\ s' = s$

| *Linearize* *i* *h* \Rightarrow

$initialized\ s \wedge \neg aborted\ s$

$\wedge h \in postfix-all\ (hist\ s)\ (linearizations\ (pendingReqs\ s))$

$$\wedge s' = s(\text{hist} := h)$$

|Initialize i $h \Rightarrow$

$$\begin{aligned}
& (\exists c . \text{phase } s \ c \neq \text{Sleep}) \wedge \neg \text{aborted } s \wedge \neg \text{initialized } s \\
& \wedge h \in \text{postfix-all } (l\text{-c-p } (\text{initHists } s)) (\text{linearizations } (\text{initValidReqs } s)) \\
& \wedge s' = s(\text{hist} := h, \text{initialized} := \text{True})
\end{aligned}$$

|Abort $i \Rightarrow$

$$\begin{aligned}
& \neg \text{aborted } s \wedge (\exists c . \text{phase } s \ c \neq \text{Sleep}) \\
& \wedge s' = s(\text{aborted} := \text{True})
\end{aligned}$$

|Commit c i $h \Rightarrow$

$$\begin{aligned}
& \text{phase } s \ c = \text{Pending} \wedge \text{pending } s \ c \in \text{set } (\text{hist } s) \\
& \wedge h = \text{dropWhile } (\lambda r . r \neq \text{pending } s \ c) (\text{hist } s) \\
& \wedge s' = s(\text{phase} := (\text{phase } s)(c := \text{Ready}))
\end{aligned}$$

|Switch c i h $r \Rightarrow$

$$\begin{aligned}
& \text{if } i = \text{id1} \\
& \text{then if phase } s \ c = \text{Sleep} \\
& \quad \text{then } s' = s(\text{initHists} := \{h\} \cup (\text{initHists } s), \\
& \quad \quad \text{phase} := (\text{phase } s)(c := \text{Pending}), \\
& \quad \quad \text{pending} := (\text{pending } s)(c := r)) \\
& \quad \text{else } s' = s \\
& \text{else if } i = \text{id2} \\
& \quad \text{then aborted } s \\
& \quad \quad \wedge \text{phase } s \ c = \text{Pending} \wedge r = \text{pending } s \ c \\
& \quad \quad \wedge (\text{if initialized } s \\
& \quad \quad \quad \text{then } (h \in \text{postfix-all } (\text{hist } s) (\text{linearizations } (\text{pendingReqs } s))) \\
& \quad \quad \quad \text{else } (h \in \text{postfix-all } (l\text{-c-p } (\text{initHists } s)) (\text{linearizations } (\text{initValidReqs } s)))) \\
& \quad \quad \wedge s' = s(\text{phase} := (\text{phase } s)(c := \text{Aborted})) \\
& \quad \text{else False } \}
\end{aligned}$$

definition $ALM\text{-start} :: \text{nat} \Rightarrow ALM\text{-state set}$

— the set of start states

where

$$\begin{aligned}
ALM\text{-start} & \equiv \lambda id . \{s . \\
& \quad \forall c . \text{phase } s \ c = (\text{if } id \neq 0 \text{ then Sleep else Ready}) \\
& \quad \wedge \text{hist } s = [] \\
& \quad \wedge \neg \text{aborted } s \\
& \quad \wedge (\text{if } id \neq 0 \text{ then } \neg \text{initialized } s \text{ else initialized } s) \\
& \quad \wedge \text{initHists } s = \{\} \}
\end{aligned}$$

definition $ALM\text{-ioa} :: \text{nat} \Rightarrow \text{nat} \Rightarrow (ALM\text{-action}, ALM\text{-state})\text{ioa}$

— The ALM automaton

where

$$\begin{aligned}
ALM\text{-ioa} & \equiv \lambda (id1 :: \text{nat}) id2 . \\
& \quad (ALM\text{-asig } id1 \ id2, \\
& \quad \quad ALM\text{-start } id1,
\end{aligned}$$

ALM-trans id1 id2,
 $\{\}, \{\}$)

type-synonym *compo-state* = *ALM-state* \times *ALM-state*

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

where

composeALMs $\equiv \lambda \text{ id1 id2 .}$
 $\text{hide } (ALM\text{-ioa } 0 \text{ id1} \parallel ALM\text{-ioa id1 id2})$
 $\{act . EX \text{ c tr r . } act = Switch \text{ c id1 tr r}\}$

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

| ((fst s, Abort 0, fst t) : ALM-trans 0 id1 & snd t = snd s)

lemma *composeALMsE*:

— A rule for decomposing proofs

assumes $id1 \sim= 0$ **and** $id1 < id2$ **and** $in-trans-comp:s - (a::ALM-action) - composeALMs$
 $id1 id2 -> t$

shows *decomp*: $in-trans-cases-fun id1 id2 s t$

proof —

from $in-trans-comp$ **and** $\langle id1 \sim= 0 \rangle$ **and** $\langle id1 < id2 \rangle$

have $a : \{act . EX c r h id' . 0 \leq id' \ \& \ id' < id2 \ \& \ ($

$act = Invoke \ c \ r$

| $act : \{Switch \ c \ 0 \ h \ r, Switch \ c \ id1 \ h \ r, Switch \ c \ id2 \ h \ r\}$

| $act : \{Linearize \ 0 \ h, Linearize \ id1 \ h\}$

| $act : \{Initialize \ 0 \ h, Initialize \ id1 \ h\}$

| $act : \{Abort \ 0, Abort \ id1\}$

| $act : \{Commit \ c \ id' \ h\}$

)} **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** $0 \leq id' \ \& \ id' < id2 \ \& \ a : \{act .$

$act = Invoke \ c \ r$

| $act : \{Switch \ c \ 0 \ h \ r, Switch \ c \ id1 \ h \ r, Switch \ c \ id2 \ h \ r\}$

| $act : \{Linearize \ 0 \ h, Linearize \ id1 \ h\}$

| $act : \{Initialize \ 0 \ h, Initialize \ id1 \ h\}$

| $act : \{Abort \ 0, Abort \ id1\}$

| $act : \{Commit \ c \ id' \ h\}$

} **by** *auto*

moreover from $in-trans-comp$ **and** $\langle id1 \sim= 0 \rangle$ **and** $\langle id1 < id2 \rangle$

have

$(a = Linearize \ 0 \ h \mid a = Abort \ 0 \mid a = Initialize \ 0 \ h \mid a = Switch \ c \ 0 \ h \ r \mid (a = Commit \ c \ id' \ h \ \& \ id' < id1)) \implies ((fst \ s, a, fst \ t) : ALM-trans \ 0 \ id1 \ \& \ snd \ s = snd \ t)$

and

$(a = Linearize \ id1 \ h \mid a = Abort \ id1 \mid a = Initialize \ id1 \ h \mid a = Switch \ c \ id2 \ h \ r \mid (a = Commit \ c \ id' \ h \ \& \ id1 \leq id' \ \& \ id' < id2)) \implies (fst \ s = fst \ t \ \& \ (snd \ s, a, snd \ t) : ALM-trans \ id1 \ id2)$

and

$(a = Switch \ c \ id1 \ h \ r \mid a = Invoke \ c \ r) \implies ((fst \ s, a, fst \ t) : ALM-trans \ 0 \ id1 \ \& \ (snd \ s, a, snd \ t) : ALM-trans \ id1 \ id2)$

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*)

ultimately show *?thesis unfolding in-trans-cases-fun-def apply simp by (metis linorder-not-less)*

qed

lemma *my-rule*: $[id1 \neq 0; id1 < id2; s - a - composeALMs \ id1 \ id2 -> t; [in-trans-cases-fun \ id1 \ id2 \ s \ t]] \implies P] \implies P$ **by** (*auto intro: composeALMsE[where s=s and t=t and a=a]*)

lemma *my-rule2*: $[0 < id1; id1 < id2; s - a - composeALMs \ id1 \ id2 -> t; [in-trans-cases-fun \ id1 \ id2 \ s \ t]] \implies P] \implies P$ **by** (*auto intro: composeALMsE[where*

$s=s$ and $t=t$ and $a=a]$)

4.2 Invariants of a single ALM instance

definition $P1a :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— In ALM 1, a pending request of client c has client c as sender

$P1a == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $ALL\ c . phase\ s1\ c \in \{Pending, Aborted\} \longrightarrow request\text{-}snd\ (pending\ s1\ c) = c$

definition $P1b :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— In ALM 2, a pending request of client c has client c as sender

$P1b == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $ALL\ c . phase\ s2\ c \neq Sleep \longrightarrow request\text{-}snd\ (pending\ s2\ c) = c$

definition $P2 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$ **where**

$P2 == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $(\forall\ c . phase\ s2\ c = Sleep) \longrightarrow (\neg\ initialized\ s2 \wedge hist\ s2 = [])$

definition $P3 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$ **where**

$P3 == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $\forall\ c . (phase\ s2\ c = Ready \longrightarrow initialized\ s2)$

definition $P4 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— The set of init histories of ALM 2 is empty when no client ever invoked anything

$P4 == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $(\forall\ c . phase\ s2\ c = Sleep) = (initHists\ s2 = \{\})$

definition $P5 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— In ALM 1 a client never sleeps

where

$P5 == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $\forall\ c . phase\ s1\ c \neq Sleep$

4.3 Invariants of the composition of two ALM instances

definition $P6 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— Non-interference accross instances

where

$P6 == \% s . let\ s1 = fst\ s; s2 = snd\ s\ in$
 $(\sim\ aborted\ s1 \longrightarrow (ALL\ c . phase\ s2\ c = Sleep)) \ \&\ (ALL\ c . phase\ s1\ c \sim\ Aborted = (phase\ s2\ c = Sleep))$

definition $P7 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— 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

$P7 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad \text{ALL } c . \text{phase } s1 \ c = \text{Aborted} \wedge \neg \text{initialized } s2 \longrightarrow (\text{pending } s2 \ c =$
 $\text{pending } s1 \ c \wedge \text{phase } s2 \ c \in \{\text{Pending}, \text{Aborted}\})$

definition $P8 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— Init histories of ALM 2 are built from the history of ALM 1 plus pending requests of ALM 1

where

$P8 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad \forall h \in \text{initHists } s2 . h \in \text{postfix-all } (\text{hist } s1) (\text{linearizations } (\text{pendingReqs } s1))$

definition $P9 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— ALM 2 does not abort before ALM 1 aborts

where

$P9 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad \text{aborted } s2 \longrightarrow \text{aborted } s1$

definition $P10 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

— ALM 1 is always initialized and when ALM 2 is not initialized its history is empty

where

$P10 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad \text{initialized } s1 \wedge (\neg \text{initialized } s2 \longrightarrow (\text{hist } s2 = []))$

definition $P11 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— 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

$P11 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad ((\exists c . \text{phase } s2 \ c \neq \text{Sleep}) \wedge \neg \text{initialized } s2) \longrightarrow \text{initValidReqs } s2 \subseteq$
 $\text{pendingReqs } s1$

definition $P12 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— 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

$P12 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\quad (\exists c . \text{phase } s2 \ c \neq \text{Sleep}) \longrightarrow (\exists rs . l\text{-}c\text{-}p (\text{initHists } s2) = rs @ (\text{hist } s1)$
 $\wedge \text{set } rs \subseteq \text{pendingReqs } s1 \wedge \text{distinct } rs)$

definition $P13 :: (ALM\text{-}state * ALM\text{-}state) \Rightarrow bool$

where

— 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

$P13 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$

$((\exists c . \text{phase } s2 \ c \neq \text{Sleep}) \wedge \neg \text{initialized } s2) \longrightarrow \text{postfix-all } (l\text{-c-p } (\text{initHists } s2)) (\text{linearizations } (\text{initValidReqs } s2)) \subseteq \text{postfix-all } (\text{hist } s1) (\text{linearizations } (\text{pendingReqs } s1))$

definition $P14 :: (ALM\text{-state} * ALM\text{-state}) \Rightarrow \text{bool}$

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

$P14 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $(\text{hist } s2 \neq [] \vee \text{initialized } s2) \longrightarrow (\exists rs .$
 $\text{hist } s2 = rs @ (\text{hist } s1)$
 $\wedge \text{set } rs \cap \text{set } (\text{hist } s1) = \{\})$

definition $P15 :: (ALM\text{-state} * ALM\text{-state}) \Rightarrow \text{bool}$

where

— A client that hasn't yet invoked ALM 2 has no request committed in ALM 2 except for its pending request

$P15 == \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in}$
 $\forall r . \text{let } c = \text{request-snd } r \text{ in phase } s2 \ c = \text{Sleep} \wedge r \in \text{set } (\text{hist } s2) \longrightarrow (r$
 $\in \text{set } (\text{hist } s1) \vee r \in \text{pendingReqs } s1)$

4.4 Proofs of invariance

lemma *invariant-imp*: $\llbracket \text{invariant } \text{ioa } P; \forall s . P \ s \longrightarrow Q \ s \rrbracket \Longrightarrow \text{invariant } \text{ioa } Q$
by (*simp add:invariant-def*)

declare *phase.split* [*split*]
declare *phase.split-asm* [*split*]
declare *ALM-action.split* [*split*]
declare *ALM-action.split-asm* [*split*]

lemma *dropWhile-lemma*: $\forall ys . xs = ys @ zs \wedge \text{hd } zs = x \wedge zs \neq [] \wedge x \notin \text{set } ys \longrightarrow \text{dropWhile } (\lambda x' . x' \neq x) \ xs = zs$

— A useful lemma about truncating histories

proof (*induct xs, force*)

fix *a xs*

assume $\forall ys . xs = ys @ zs \wedge \text{hd } zs = x \wedge zs \neq [] \wedge x \notin \text{set } ys \longrightarrow \text{dropWhile } (\lambda x' . x' \neq x) \ xs = zs$

show $\forall ys . a \# xs = ys @ zs \wedge \text{hd } zs = x \wedge zs \neq [] \wedge x \notin \text{set } ys \longrightarrow \text{dropWhile } (\lambda x' . x' \neq x) \ (a \# xs) = zs$

proof (*rule allI, rule impI, cases a = x*)

fix *ys*

assume $a \# xs = ys @ zs \wedge \text{hd } zs = x \wedge zs \neq [] \wedge x \notin \text{set } ys$ **and** $a = x$

hence $x \# xs = ys @ zs$ **and** $x \notin \text{set } ys$ **and** $\text{hd } zs = x$ **and** $zs \neq []$ **by** *auto*

from $\langle x \# xs = ys @ zs \rangle$ **and** $\langle x \notin \text{set } ys \rangle$ **have** $ys = []$ **by** (*metis list.sel(1) hd-append hd-in-set*)

with $\langle a = x \rangle$ **and** $\langle x \# xs = ys @ zs \rangle$ **show** $\text{dropWhile } (\lambda x' . x' \neq x) \ (a \# xs) = zs$ **by** *auto*

next

fix ys
assume $a \# xs = ys @ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys$ **and** $a \neq x$
hence $a \# xs = ys @ zs$ **and** $hd\ zs = x$ **and** $zs \neq []$ **and** $x \notin set\ ys$ **by** *auto*
obtain ys' **where** $xs = ys' @ zs$ **and** $x \notin set\ ys'$
proof –
from $\langle a \# xs = ys @ zs \rangle$ **and** $\langle hd\ zs = x \rangle$ **and** $\langle a \neq x \rangle$ **obtain** ys' **where** $ys = a \# ys'$ **apply** *clarify* **by** (*metis Cons-eq-append-conv list.sel(1)*)
moreover with $\langle x \notin set\ ys \rangle$ **have** $x \notin set\ ys'$ **by** *auto*
moreover from $\langle ys = a \# ys' \rangle$ **and** $\langle a \# xs = ys @ zs \rangle$ **have** $xs = ys' @ zs$
by *auto*
ultimately show $(\bigwedge ys'. [xs = ys' @ zs; x \notin set\ ys'] \implies thesis) \implies thesis$
by *auto*
qed
with $\langle \forall\ ys. xs = ys @ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys \longrightarrow dropWhile\ (\lambda x'. x' \neq x)\ xs = zs \rangle$ **and** $\langle hd\ zs = x \rangle$ **and** $\langle zs \neq [] \rangle$ **have** $dropWhile\ (\lambda x'. x' \neq x)\ xs = zs$ **by** *auto*
with $\langle a \neq x \rangle$ **show** $dropWhile\ (\lambda x'. x' \neq x)\ (a \# xs) = zs$ **by** *auto*
qed
qed

lemma *P2-invariant*: $[id1 < id2; id1 \neq 0] \implies invariant\ (composeALMs\ id1\ id2)\ P2$

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 $\langle 0 < id1 \rangle$ **and** $\langle id1 < id2 \rangle$ **and** *in-trans-comp* **show** $P2\ (s1', s2')$
proof (*rule my-rule2*)
assume *in-trans-cases-fun* $id1\ id2\ (s1, s2)\ (s1', s2')$
thus $P2\ (s1', s2')$ **using** $\langle P2\ (s1, s2) \rangle$ **and** $\langle 0 < id1 \rangle$ **and** $\langle id1 < id2 \rangle$ **apply** (*auto simp add: in-trans-cases-fun-def*) **apply** (*auto simp add: ALM-trans-def P2-def*) **done**
qed
qed

lemma *P5-invariant*: $[id1 < id2; id1 \neq 0] \implies invariant\ (composeALMs\ id1\ id2)\ P5$

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  $s1\ s2\ s1'\ s2'$  act
  assume reachable (composeALMs  $id1\ id2$ ) ( $s1, s2$ ) and  $P5\ (s1, s2)$  and  $0 < id1$ 
and  $id1 < id2$  and in-trans-comp:( $s1, s2$ )  $-act--composeALMs\ id1\ id2->$ 
( $s1', s2'$ )
  from  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  and in-trans-comp show  $P5\ (s1', s2')$ 
  proof (rule my-rule2)
    assume in-trans-cases-fun  $id1\ id2\ (s1, s2)\ (s1', s2')$ 
    thus  $P5\ (s1', s2')$  using  $\langle P5\ (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  ap-
ply(auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def
P5-def) done
  qed
qed

```

```

lemma P6-invariant:  $[|id1 \neq 0 ; id1 < id2|] ==> invariant\ (composeALMs\ id1\ id2)\ P6$ 
proof (rule invariantI, rule-tac [2] impI)
  fix  $s$ 
  assume  $s : starts-of\ (composeALMs\ id1\ id2)$  and  $id1 \neq 0$ 
  thus  $P6\ s$  by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
par-def ALM-start-def P6-def)
next
  fix  $s\ t\ a$ 
  assume  $P6\ s$ 
  assume  $id1 \neq 0$  and  $id1 < id2$  and  $s -a--composeALMs\ id1\ id2->\ t$ 
  thus  $P6\ t$ 
  proof (rule my-rule)
    assume in-trans-cases-fun  $id1\ id2\ s\ t$ 
    thus  $P6\ t$  using  $\langle P6\ s \rangle$  and  $\langle id1 \neq 0 \rangle$  and  $\langle id1 < id2 \rangle$  apply(auto simp add:
in-trans-cases-fun-def) apply (simp-all add: ALM-trans-def P6-def) apply (metis
phase.simps(12) phase.simps(4) phase.simps(5)) apply (metis phase.simps(12)
phase.simps(5)) apply (force simp add: ALM-trans-def P6-def) apply (force simp
add: ALM-trans-def P6-def) apply (force simp add: ALM-trans-def P6-def) ap-
ply (force simp add: ALM-trans-def P6-def) apply (force simp add: ALM-trans-def
P6-def) apply (force simp add: ALM-trans-def P6-def) done
  qed
qed

```

```

lemma P9-invariant:  $[|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P9$ 
proof (rule invariantI, auto)
  fix  $s1\ s2$ 
  assume  $(s1, s2) : starts-of\ (composeALMs\ id1\ id2)$ 
  thus  $P9\ (s1, s2)$  by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
par-def ALM-start-def P9-def)
next
  fix  $s1\ s2\ s1'\ s2'$  act
  assume reachable (composeALMs  $id1\ id2$ ) ( $s1, s2$ ) and  $P9\ (s1, s2)$  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)⟩ have
  reachable (composeALMs id1 id2) (s1', s2') by (auto intro: reachable.reachable-n)
  with ⟨reachable (composeALMs id1 id2) (s1, s2)⟩ and ⟨0 < id1⟩ and ⟨id1 <
  id2⟩ and P6-invariant show P6 (s1, s2) unfolding invariant-def by auto
qed
from ⟨0 < id1⟩ and ⟨id1 < id2⟩ and in-trans-comp show P9 (s1', s2')
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
  thus P9 (s1', s2') using ⟨P9 (s1, s2)⟩ and ⟨P6 (s1, s2)⟩ and ⟨0 < id1⟩ and
  ⟨id1 < id2⟩ apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add:
  ALM-trans-def P9-def P6-def) done
qed
qed

lemma P10-invariant: [|id1 < id2; id1 ~ = 0|] ==> invariant (composeALMs
id1 id2) P10
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P10 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
  par-def ALM-start-def P10-def)
next
  fix s1 s2 s1' s2' act
  assume reachable (composeALMs id1 id2) (s1, s2) and P10 (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 P10 (s1', s2')
  proof (rule my-rule2)
    assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
    thus P10 (s1', s2') using ⟨P10 (s1, s2)⟩ and ⟨0 < id1⟩ and ⟨id1 < id2⟩ ap-
    ply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def
    P10-def) done
  qed
qed

lemma P3-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1
id2) P3
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P3 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
  par-def ALM-start-def P3-def)
next
  fix s1 s2 s1' s2' act
  assume reachable (composeALMs id1 id2) (s1, s2) and P3 (s1, s2) and 0 <
  id1 and id1 < id2 and in-trans-comp:(s1, s2) -act--composeALMs id1 id2->
  (s1', s2')

```

```

have P10 (s1, s2)
proof -
  from in-trans-comp and ⟨reachable (composeALMs id1 id2) (s1, s2)⟩ have
  reachable (composeALMs id1 id2) (s1', s2') by (auto intro: reachable.reachable-n)
  with ⟨reachable (composeALMs id1 id2) (s1, s2)⟩ and ⟨0 < id1⟩ and ⟨id1 <
  id2⟩ and P10-invariant show P10 (s1, s2) unfolding invariant-def by auto
qed
from ⟨0 < id1⟩ and ⟨id1 < id2⟩ and in-trans-comp show P3 (s1', s2')
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
  thus P3 (s1', s2') using ⟨P3 (s1, s2)⟩ and ⟨P10 (s1, s2)⟩ and ⟨0 < id1⟩ and
  ⟨id1 < id2⟩ apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add:
  ALM-trans-def P3-def P10-def) done
qed
qed

lemma P7-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1
id2) P7
proof (rule invariantI, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P7 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
  par-def ALM-start-def P7-def)
next
  fix s1 s2 s1' s2' act
  assume reachable (composeALMs id1 id2) (s1, s2) and P7 (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)
  proof -
    from in-trans-comp and ⟨reachable (composeALMs id1 id2) (s1, s2)⟩ have
    reachable (composeALMs id1 id2) (s1', s2') by (auto intro: reachable.reachable-n)
    with ⟨reachable (composeALMs id1 id2) (s1, s2)⟩ and ⟨0 < id1⟩ and ⟨id1 <
    id2⟩ and P6-invariant and P10-invariant show P6 (s1, s2) and P10 (s1, s2)
    unfolding invariant-def by auto
  qed
  from ⟨0 < id1⟩ and ⟨id1 < id2⟩ and in-trans-comp show P7 (s1', s2')
  proof (rule my-rule2)
    assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
    thus P7 (s1', s2') using ⟨P7 (s1, s2)⟩ and ⟨P6 (s1, s2)⟩ and ⟨0 < id1⟩ and
    ⟨id1 < id2⟩
    proof (auto simp add: in-trans-cases-fun-def)
      fix ca ra
      assume P7 (s1, s2) and P6 (s1, s2) and 0 < id1 and id1 < id2 and (s1,
      Invoke ca ra, s1') ∈ ALM-trans 0 id1 and (s2, Invoke ca ra, s2') ∈ ALM-trans
      id1 id2
      thus P7 (s1', s2') by (auto simp add: ALM-trans-def P7-def)
    next
      fix ca h ra

```

```

    assume P7 (s1, s2) and P6 (s1, s2) and 0 < id1 and id1 < id2 and (s1,
Switch ca id1 h ra, s1') ∈ ALM-trans 0 id1 and (s2, Switch ca id1 h ra, s2') ∈
ALM-trans id1 id2
    thus P7 (s1', s2') by (auto simp add: ALM-trans-def P7-def P6-def)
next
    fix c id' h
    assume P7 (s1, s2) and P6 (s1, s2) and 0 < id1 and (s2, Commit c id'
h, s2') ∈ ALM-trans id1 id2 and id1 ≤ id' and id' < 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 < id1 and id1 < id2 and (s2,
Switch c id2 h r, s2') ∈ ALM-trans id1 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 < id1 and id1 < id2 and (s2,
Linearize id1 h, s2') ∈ ALM-trans id1 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 < id1 and id1 < id2 and (s2,
Initialize id1 h, s2') ∈ ALM-trans id1 id2
    thus P7 (s1, s2') by (auto simp add: ALM-trans-def P7-def)
next
    fix ca ta ra
    assume P7 (s1, s2) and P6 (s1, s2) and 0 < id1 and id1 < id2 and (s1,
Switch ca 0 ta ra, s1') ∈ ALM-trans 0 id1
    thus P7 (s1', s2) by (auto simp add: ALM-trans-def P7-def)
next
    fix ca id' h
    assume P7 (s1, s2) and P6 (s1, s2) and id1 < id2 and (s1, Commit ca
id' h, s1') ∈ ALM-trans 0 id1 and id' < 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 < id1 and id1 < id2 and (s1,
Linearize 0 h, s1') ∈ ALM-trans 0 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 < id1 and id1 < id2 and (s1,
Initialize 0 h, s1') ∈ ALM-trans 0 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 < id1 and id1 < id2 and (s2,
Abort id1, s2') ∈ ALM-trans id1 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  $(s1, \text{Abort } 0, s1') \in \text{ALM-trans } 0 \text{ id1}$ 
  thus  $P7(s1', s2)$  by (auto simp add: ALM-trans-def P7-def)
qed
qed
qed

lemma  $P4\text{-invariant}$ :  $[id1 < id2; id1 \neq 0] \implies \text{invariant } (\text{composeALMs } id1 \text{ id2}) P4$ 
proof (rule invariantI, auto)
  fix  $s1 \ s2$ 
  assume  $(s1, s2) : \text{starts-of } (\text{composeALMs } id1 \text{ id2})$  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)
next
  fix  $s1 \ s2 \ s1' \ s2' \ act$ 
  assume  $\text{reachable } (\text{composeALMs } id1 \text{ id2}) (s1, s2)$  and  $P4(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $\text{in-trans-comp}:(s1, s2) - \text{act} - \text{composeALMs } id1 \text{ id2} -> (s1', s2')$ 
  have  $P6(s1, s2)$ 
  proof -
    from  $\text{in-trans-comp}$  and  $\langle \text{reachable } (\text{composeALMs } id1 \text{ id2}) (s1, s2) \rangle$  have  $\text{reachable } (\text{composeALMs } id1 \text{ id2}) (s1', s2')$  by (auto intro: reachable.reachable-n)
    with  $\langle \text{reachable } (\text{composeALMs } id1 \text{ id2}) (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  and  $P6\text{-invariant}$  show  $P6(s1, s2)$  unfolding invariant-def by auto
  qed
  from  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  and  $\text{in-trans-comp}$  show  $P4(s1', s2')$ 
  proof (rule my-rule2)
    assume  $\text{in-trans-cases-fun } id1 \text{ id2 } (s1, s2) (s1', s2')$ 
    thus  $P4(s1', s2')$  using  $\langle P4(s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P4-def) done
  qed
qed
qed

lemma  $P8\text{-invariant}$ :  $[id1 < id2; id1 \neq 0] \implies \text{invariant } (\text{composeALMs } id1 \text{ id2}) P8$ 
proof (rule invariantI, auto)
  fix  $s1 \ s2$ 
  assume  $(s1, s2) : \text{starts-of } (\text{composeALMs } id1 \text{ id2})$  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)
next
  fix  $s1 \ s2 \ s1' \ s2' \ act$ 
  assume  $\text{reachable } (\text{composeALMs } id1 \text{ id2}) (s1, s2)$  and  $P8(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $\text{in-trans-comp}:(s1, s2) - \text{act} - \text{composeALMs } id1 \text{ id2} -> (s1', s2')$ 
  have  $P6(s1, s2)$  and  $P10(s1, s2)$  and  $P5(s1, s2)$  and  $P4(s1, s2)$ 

```

proof –

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

with $\langle \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, \ s2) \rangle$ **and** $\langle 0 < id1 \rangle$ **and** $\langle id1 < id2 \rangle$ **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 $\langle 0 < id1 \rangle$ **and** $\langle id1 < id2 \rangle$ **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** $\langle 0 < id1 \rangle$ **and** $\langle id1 < id2 \rangle$

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

fix *ca ra*

assume *P8* $(s1, \ s2)$ **and** $0 < id1$ **and** $id1 < id2$ **and** *in-invoke-1*: $(s1, \ \text{Invoke } ca \ ra, \ s1') \in \text{ALM-trans } 0 \ id1$ **and** *in-invoke-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-invoke-2* **and** $\langle P8 \ (s1, \ s2) \rangle$ **show** *?thesis* **by** *(auto simp add: ALM-trans-def P8-def)*

next

assume $s1' \neq s1$

with *in-invoke-1* **have** *pendingReqs* $s1 \subseteq \text{pendingReqs } s1'$ **by** *(force simp add: pendingReqs-def ALM-trans-def)*

moreover from *in-invoke-1* **have** *hist* $s1' = \text{hist } s1$ **by** *(auto simp add: ALM-trans-def)*

moreover from *in-invoke-2* **have** *initHists* $s2' = \text{initHists } s2$ **by** *(auto simp add: ALM-trans-def)*

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

ultimately show *?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*: $(s1, \ \text{Switch } ca \ id1 \ h \ ra, \ s1') \in \text{ALM-trans } 0 \ id1$ **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 $h1 \in \text{postfix-all } (\text{hist } s1') \ (\text{linearizations } (\text{pendingReqs } s1'))$

proof *(cases h1 ∈ initHists s2)*

assume $h1 \in \text{initHists } s2$

moreover from *in-switch-1* **and** $\langle 0 < id1 \rangle$ **have** *hist* $s1' = \text{hist } s1$ **and** *pendingReqs* $s1' = \text{pendingReqs } s1$ **by** *(auto simp add: ALM-trans-def pendingReqs-def)*

moreover note $\langle P8 (s1, s2) \rangle$
ultimately show $h1 \in \text{postfix-all } (\text{hist } s1') (\text{linearizations } (\text{pendingReqs } s1'))$ **by** $(\text{auto simp add: P8-def})$
next
assume $h1 \notin \text{initHists } s2$
with $\langle h1 \in \text{initHists } s2' \rangle$ **and** in-switch-2 **have** $h1 = h$ **by** $(\text{auto simp add: ALM-trans-def})$
with in-switch-1 **and** $\langle 0 < id1 \rangle$ **and** $\langle P10 (s1, s2) \rangle$ **have** $h1 \in \text{postfix-all } (\text{hist } s1) (\text{linearizations } (\text{pendingReqs } s1'))$ **by** $(\text{auto simp add: ALM-trans-def P10-def})$
moreover from in-switch-1 **and** $\langle 0 < id1 \rangle$ **have** $\text{hist } s1' = \text{hist } s1$ **and** $\text{pendingReqs } s1' = \text{pendingReqs } s1$ **by** $(\text{auto simp add: ALM-trans-def pendingReqs-def})$
ultimately show $?thesis$ **by** auto
qed
qed
next
fix $c \text{ id}' h$
assume $P8 (s1, s2)$ **and** $0 < id1$ **and** $(s2, \text{Commit } c \text{ id}' h, s2') \in \text{ALM-trans } id1 \text{ id2}$ **and** $id1 \leq id'$ **and** $id' < id2$
thus $P8 (s1, s2')$ **by** $(\text{auto simp add: ALM-trans-def P8-def})$
next
fix $c h r$
assume $P8 (s1, s2)$ **and** $0 < id1$ **and** $id1 < id2$ **and** $(s2, \text{Switch } c \text{ id2 } h r, s2') \in \text{ALM-trans } id1 \text{ id2}$
thus $P8 (s1, s2')$ **by** $(\text{auto simp add: ALM-trans-def P8-def})$
next
fix h
assume $P8 (s1, s2)$ **and** $0 < id1$ **and** $id1 < id2$ **and** $(s2, \text{Linearize } id1 h, s2') \in \text{ALM-trans } id1 \text{ id2}$
thus $P8 (s1, s2')$ **by** $(\text{auto simp add: ALM-trans-def P8-def})$
next
fix h
assume $P8 (s1, s2)$ **and** $0 < id1$ **and** $id1 < id2$ **and** $(s2, \text{Initialize } id1 h, s2') \in \text{ALM-trans } id1 \text{ id2}$
thus $P8 (s1, s2')$ **by** $(\text{auto simp add: ALM-trans-def P8-def})$
next
fix $ca \text{ ta } ra$
assume $P8 (s1, s2)$ **and** $0 < id1$ **and** $id1 < id2$ **and** $(s1, \text{Switch } ca \text{ } 0 \text{ ta } ra, s1') \in \text{ALM-trans } 0 \text{ id1}$
thus $P8 (s1', s2)$ **using** $\langle P5 (s1, s2) \rangle$ **by** $(\text{auto simp add: ALM-trans-def P8-def P5-def})$
next
fix $ca \text{ id}' h$
assume $P8 (s1, s2)$ **and** $\text{in-commit-1}:(s1, \text{Commit } ca \text{ id}' h, s1') \in \text{ALM-trans } 0 \text{ id1}$
from in-commit-1 **have** $\text{pendingReqs } s1' = \text{pendingReqs } s1$ **and** $\text{hist } s1' = \text{hist } s1$ **by** $(\text{auto simp add: pendingReqs-def ALM-trans-def})$
with $\langle P8 (s1, s2) \rangle$ **show** $P8 (s1', s2)$ **by** $(\text{auto simp add: ALM-trans-def P8-def pendingReqs-def})$

```

next
  fix h
    assume  $P8(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, Linearize\ 0\ h, s1') \in ALM-trans\ 0\ id1$ 
    thus  $P8(s1', s2)$  using  $\langle P6(s1, s2) \rangle$  and  $\langle P4(s1, s2) \rangle$  by (auto simp add: ALM-trans-def P8-def P6-def P4-def)
  next
    assume  $P8(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, Abort\ id1, s2') \in ALM-trans\ id1\ id2$ 
    thus  $P8(s1, s2')$  by (auto simp add: ALM-trans-def P8-def)
  next
    fix h
      assume  $P8(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, Initialize\ 0\ h, s1') \in ALM-trans\ 0\ id1$ 
      thus  $P8(s1', s2)$  using  $\langle P10(s1, s2) \rangle$  by (auto simp add: ALM-trans-def P8-def P10-def)
    next
      assume  $P8(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, Abort\ 0, s1') \in ALM-trans\ 0\ id1$ 
      thus  $P8(s1', s2)$  by (auto simp add: ALM-trans-def P8-def pendingReqs-def)
    qed
  qed
qed

```

lemma *P12-invariant*: $[id1 < id2; id1 \neq 0] \implies invariant\ (composeALMs\ id1\ id2)\ P12$

proof *clarify*

```

  assume  $id1 < id2$  and  $0 < id1$ 
  with P8-invariant and P4-invariant have invariant (composeALMs id1 id2) ( $\lambda (s1, s2) . P8(s1, s2) \wedge P4(s1, s2)$ ) by (auto simp add: invariant-def)
  moreover have  $\forall s . P8\ s \wedge P4\ s \longrightarrow P12\ s$ 
  proof auto
    fix s1 s2
    assume  $P8(s1, s2)$  and  $P4(s1, s2)$ 
    hence initHists-prop:  $\forall h \in initHists\ s2 . (\exists h' . h = h' @ (hist\ s1) \wedge set\ h' \subseteq pendingReqs\ s1 \wedge distinct\ h')$  by (auto simp add: P8-def postfix-all-def linearizations-def)
    show P12 (s1, s2)
    proof (simp add: P12-def, rule impI)
      assume  $\exists c . phase\ s2\ c \neq Sleep$ 
      with  $\langle P4(s1, s2) \rangle$  have  $initHists\ s2 \neq \{\}$  by (auto simp add: P4-def)
      with l-c-p-lemma[of  $initHists\ s2\ hist\ s1$ ] and initHists-prop
        obtain rs where  $l-c-p\ (initHists\ s2) = rs @ hist\ s1$  by (auto simp add: suffixeq-def)
      moreover have  $set\ rs \subseteq pendingReqs\ s1$ 
      proof -
        from  $\langle initHists\ s2 \neq \{\} \rangle$  obtain h where  $h \in initHists\ s2$  by auto
        with initHists-prop obtain h' where  $h = h' @ (hist\ s1) \wedge set\ h' \subseteq pendingReqs\ s1$  by auto
      
```

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

lemma $P11\text{-}invariant: [|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P11$

proof $clarify$

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

ultimately have $x \in \text{set } h'$ by *auto*
 with $\langle \text{set } h' \subseteq \text{pendingReqs } s1 \rangle$ show $x \in \text{pendingReqs } s1$ by *auto*
 next
 fix $x \ c \ h$
 assume $\text{phase } s2 \ c \neq \text{Sleep}$ and $\neg \text{initialized } s2$
 with $\langle P12 \ (s1, s2) \rangle$ have $\text{lcp-prop} : \exists \ rs \ . \ \text{l-c-p } (\text{initHists } s2) = rs \ @ \ (\text{hist } s1)$ by $(\text{auto simp add:P12-def P8-def postfix-all-def linearizations-def})$
 assume $x \notin \text{set } (\text{l-c-p } (\text{initHists } s2))$ and $x \in \text{pendingReqs } s2$
 from $\langle x \notin \text{set } (\text{l-c-p } (\text{initHists } s2)) \rangle$ and lcp-prop have $x \notin \text{set } (\text{hist } s1)$ by *auto*
 moreover obtain c' where $\text{phase } s1 \ c' = \text{Aborted}$ and $x = \text{pending } s1 \ c'$
 proof –
 from $\langle x \in \text{pendingReqs } s2 \rangle$ and $\langle P6 \ (s1, s2) \rangle$ obtain c' where $\text{phase } s1 \ c' = \text{Aborted}$ and $x = \text{pending } s2 \ c'$ by $(\text{force simp add:pendingReqs-def P6-def})$
 moreover with $\langle \neg \text{initialized } s2 \rangle$ and $\langle P7 \ (s1, s2) \rangle$ have $x = \text{pending } s1 \ c'$ by $(\text{auto simp add:P7-def})$
 ultimately show $(\bigwedge c'. \llbracket \text{phase } s1 \ c' = \text{Aborted}; x = \text{pending } s1 \ c' \rrbracket \implies \text{thesis}) \implies \text{thesis}$ by *auto*
 qed
 ultimately show $x \in \text{pendingReqs } s1$ by $(\text{auto simp add:pendingReqs-def})$
 qed
 qed
 ultimately show $?thesis$ by $(\text{auto intro:invariant-imp})$
 qed

lemma *P1a-invariant*: $[[id1 < id2; id1 \neq 0]] \implies \text{invariant } (\text{composeALMs } id1 \ id2)$ *P1a*
proof $(\text{rule invariantI, auto})$
 fix $s1 \ s2$
 assume $(s1, s2) : \text{starts-of } (\text{composeALMs } id1 \ id2)$ and $0 < id1$
 thus *P1a* $(s1, s2)$ by $(\text{simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1a-def})$
 next
 fix $s1 \ s2 \ s1' \ s2' \ act$
 assume $\text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, s2)$ and *P1a* $(s1, s2)$ and $0 < id1$ and $id1 < id2$ and $\text{in-trans-comp} : (s1, s2) \text{ --act-- } \text{composeALMs } id1 \ id2 \text{ --}(s1', s2')$
 have *P5* $(s1, s2)$
 proof –
 from in-trans-comp and $\langle \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, s2) \rangle$ have $\text{reachable } (\text{composeALMs } id1 \ id2) \ (s1', s2')$ by $(\text{auto intro: reachable.reachable-n})$
 with $\langle \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, s2) \rangle$ and $\langle 0 < id1 \rangle$ and $\langle id1 < id2 \rangle$ and *P5-invariant* show *P5* $(s1, s2)$ unfolding *invariant-def* by *auto*
 qed
 from $\langle 0 < id1 \rangle$ and $\langle id1 < id2 \rangle$ and 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 $\langle P1a \ (s1, s2) \rangle$ and $\langle P5 \ (s1, s2) \rangle$ and $\langle 0 < id1 \rangle$ and $\langle id1 < id2 \rangle$ apply $(\text{auto simp add: in-trans-cases-fun-def})$ apply (auto simp)

```

add: ALM-trans-def P1a-def P5-def) done
qed
qed

lemma P1b-invariant:  $[[id1 < id2; id1 \neq 0]] \implies invariant (composeALMs id1 id2) P1b$ 
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  $\langle reachable (composeALMs id1 id2) (s1, s2) \rangle$  have
 $reachable (composeALMs id1 id2) (s1', s2')$  by (auto intro: reachable.reachable-n)
    with  $\langle reachable (composeALMs id1 id2) (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$ 
    and  $P1a-invariant$  show  $P1a (s1, s2)$  unfolding invariant-def by auto
  qed
  from  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  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  $\langle P1b (s1, s2) \rangle$  and  $\langle P1a (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$ 
    and  $\langle id1 < id2 \rangle$  apply (auto simp add: in-trans-cases-fun-def) apply (auto simp
add: ALM-trans-def P1b-def P1a-def) done
  qed
qed

lemma P13-invariant:  $[[id1 < id2; id1 \neq 0]] \implies invariant (composeALMs id1 id2) P13$ 
proof clarify
  assume  $id1 < id2$  and  $0 < id1$ 
  with  $P11-invariant$  and  $P12-invariant$  have  $invariant (composeALMs id1 id2)$ 
 $(\lambda (s1, s2) . P11 (s1, s2) \wedge P12 (s1, s2))$  by (auto simp add: invariant-def)
  moreover have  $\forall s . P11 s \wedge P12 s \longrightarrow P13 s$ 
  proof auto
    fix s1 s2
    assume  $P11 (s1, s2)$  and  $P12 (s1, s2)$ 
    show  $P13 (s1, s2)$ 
    proof (simp add: P13-def, rule impI)
      assume  $(\exists c . phase s2 c \neq Sleep) \wedge \neg initialized s2$ 
      with  $\langle P12 (s1, s2) \rangle$  and  $\langle P11 (s1, s2) \rangle$  obtain  $rs$  where  $initValidReqs-prop: initValidReqs$ 
 $s2 \subseteq pendingReqs s1$  and  $l-c-p (initHists s2) = rs @ (hist s1)$  and  $set rs \subseteq$ 
 $pendingReqs s1$  and  $distinct rs$  by (auto simp add: P12-def P11-def postfix-all-def
linearizations-def)

```

```

moreover from  $\langle l\text{-}c\text{-}p \text{ (initHists } s2) = rs \text{ @ (hist } s1) \rangle$  have  $\text{initValidReqs } s2$ 
 $\cap \text{set } rs = \{\}$  by  $(\text{auto simp add: initValidReqs-def})$ 
ultimately show  $\text{postfix-all (l-c-p (initHists } s2)) (linearizations (initValidReqs$ 
 $s2)) \subseteq \text{postfix-all (hist } s1) (linearizations (pendingReqs } s1))$  by  $(\text{force simp add:}$ 
 $\text{postfix-all-def linearizations-def})$ 
qed
qed
ultimately show  $?thesis$  by  $(\text{auto intro: invariant-imp})$ 
qed

lemma  $P14\text{-invariant: } [[id1 < id2; id1 \neq 0]] \implies \text{invariant (composeALMs } id1$ 
 $id2) P14$ 
proof  $(\text{rule invariantI, auto})$ 
fix  $s1 s2$ 
assume  $(s1, s2) : \text{starts-of (composeALMs } id1 id2)$  and  $0 < id1$ 
thus  $P14 (s1, s2)$  by  $(\text{simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def}$ 
 $\text{par-def ALM-start-def } P14\text{-def})$ 
next
fix  $s1 s2 s1' s2' \text{ act}$ 
assume  $\text{reachable (composeALMs } id1 id2) (s1, s2)$  and  $P14 (s1, s2)$  and  $0 <$ 
 $id1$  and  $id1 < id2$  and  $\text{in-trans-comp: (s1, s2) - act -- composeALMs } id1 id2 ->$ 
 $(s1', s2')$ 
have  $P6 (s1, s2)$  and  $P13 (s1, s2)$  and  $P10 (s1, s2)$  and  $P2 (s1, s2)$  and  $P4$ 
 $(s1, s2)$ 
proof  $-$ 
from  $\text{in-trans-comp}$  and  $\langle \text{reachable (composeALMs } id1 id2) (s1, s2) \rangle$  have
 $\text{reachable (composeALMs } id1 id2) (s1', s2')$  by  $(\text{auto intro: reachable.reachable-n})$ 
with  $\langle \text{reachable (composeALMs } id1 id2) (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 <$ 
 $id2 \rangle$  and  $P6\text{-invariant}$  and  $P13\text{-invariant}$  and  $P10\text{-invariant}$  and  $P4\text{-invariant}$ 
and  $P2\text{-invariant}$  show  $P6 (s1, s2)$  and  $P13 (s1, s2)$  and  $P10 (s1, s2)$  and  $P2$ 
 $(s1, s2)$  and  $P4 (s1, s2)$  unfolding invariant-def by auto
qed
from  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  and  $\text{in-trans-comp}$  show  $P14 (s1', s2')$ 
proof  $(\text{rule my-rule2})$ 
assume  $\text{in-trans-cases-fun } id1 id2 (s1, s2) (s1', s2')$ 
thus  $P14 (s1', s2')$  using  $\langle P14 (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$ 
proof  $(\text{auto simp add: in-trans-cases-fun-def})$ 
fix  $ca ra$ 
assume  $P14 (s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, \text{Invoke } ca \text{ } ra,$ 
 $s1') \in \text{ALM-trans } 0 id1$  and  $(s2, \text{Invoke } ca \text{ } ra, s2') \in \text{ALM-trans } id1 id2$ 
thus  $P14 (s1', s2')$  by  $(\text{auto simp add: ALM-trans-def } P14\text{-def})$ 
next
fix  $ca h ra$ 
assume  $P14 (s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, \text{Switch } ca \text{ } id1$ 
 $h \text{ } ra, s1') \in \text{ALM-trans } 0 id1$  and  $(s2, \text{Switch } ca \text{ } id1 h \text{ } ra, s2') \in \text{ALM-trans } id1$ 
 $id2$ 
thus  $P14 (s1', s2')$  by  $(\text{auto simp add: ALM-trans-def } P14\text{-def})$ 
next
fix  $c id' h$ 

```



```

    assume  $P14(s1, s2)$  and  $0 < id1$  and  $(s2, \text{Commit } c \text{ id}' h, s2') \in \text{ALM-trans}$ 
 $id1 \text{ id2}$  and  $id1 \leq id'$  and  $id' < id2$ 
    thus  $P14(s1, s2')$  by (auto simp add: ALM-trans-def P14-def)
  next
    fix  $c \ h \ r$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Switch } c \text{ id2 } h$ 
 $r, s2') \in \text{ALM-trans } id1 \text{ id2}$ 
    thus  $P14(s1, s2')$  by (auto simp add: ALM-trans-def P14-def)
  next
    fix  $h$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Linearize } id1 \ h,$ 
 $s2') \in \text{ALM-trans } id1 \text{ id2}$ 
    thus  $P14(s1, s2')$  by (auto simp add: ALM-trans-def P14-def linearizations-def
    postfix-all-def pendingReqs-def)
  next
    fix  $h$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Initialize } id1 \ h,$ 
 $s2') \in \text{ALM-trans } id1 \text{ id2}$ 
    thus  $P14(s1, s2')$  using  $\langle P13(s1, s2) \rangle$  apply (auto simp add: ALM-trans-def
    P14-def P13-def linearizations-def postfix-all-def pendingReqs-def) prefer 2 apply
    force apply blast done
  next
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Abort } id1, s2') \in \text{ALM-trans } id1 \text{ id2}$ 
    thus  $P14(s1, s2')$  by (auto simp add: ALM-trans-def P14-def)
  next
    fix  $ca \ ta \ ra$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, \text{Switch } ca \ 0 \ ta$ 
 $ra, s1') \in \text{ALM-trans } 0 \ id1$ 
    thus  $P14(s1', s2)$  by (auto simp add: ALM-trans-def P14-def)
  next
    fix  $ca \ id' \ h$ 
    assume  $P14(s1, s2)$  and  $id1 < id2$  and  $(s1, \text{Commit } ca \ id' \ h, s1') \in$ 
 $\text{ALM-trans } 0 \ id1$  and  $id' < id1$ 
    thus  $P14(s1', s2)$  by (auto simp add: ALM-trans-def P14-def)
  next
    fix  $h$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $\text{in-lin}:(s1, \text{Linearize}$ 
 $0 \ h, s1') \in \text{ALM-trans } 0 \ id1$ 
    from  $\text{in-lin}$  have  $\neg \text{initialized } s2$  and  $\text{hist } s2 = []$  using  $\langle P6(s1, s2) \rangle$  and  $\langle P2$ 
 $(s1, s2) \rangle$  and  $\langle P10(s1, s2) \rangle$  and  $\langle P2(s1, s2) \rangle$  by (auto simp add: ALM-trans-def
    P14-def P6-def P10-def P2-def P2-def)
    thus  $P14(s1', s2)$  by (auto simp add: P14-def)
  next
    fix  $h$ 
    assume  $P14(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, \text{Initialize } 0 \ h,$ 
 $s1') \in \text{ALM-trans } 0 \ id1$ 
    thus  $P14(s1', s2)$  using  $\langle P10(s1, s2) \rangle$  by (auto simp add: ALM-trans-def
    P14-def P10-def)

```

```

next
  assume  $P14\ (s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, Abort\ 0, s1') \in$ 
   $ALM-trans\ 0\ id1$ 
  thus  $P14\ (s1', s2)$  by (auto simp add:  $ALM-trans-def\ P14-def$ )
qed
qed
qed

lemma  $P15-invariant: [|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1$ 
 $id2)\ P15$ 
proof (rule invariantI, auto)
  fix  $s1\ s2$ 
  assume  $(s1, s2) : starts-of\ (composeALMs\ id1\ id2)$  and  $0 < id1$ 
  thus  $P15\ (s1, s2)$  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  $reachable\ (composeALMs\ id1\ id2)\ (s1, s2)$  and  $P15\ (s1, s2)$  and  $0 <$ 
 $id1$  and  $id1 < id2$  and  $in-trans-comp:(s1, s2) - act -> composeALMs\ id1\ id2 ->$ 
 $(s1', s2')$ 
  have  $P13\ (s1, s2)$  and  $P1b\ (s1, s2)$  and  $P6\ (s1, s2)$  and  $P1a\ (s1, s2)$  and
 $P5\ (s1, s2)$  and  $P10\ (s1, s2)$ 
  proof -
    from  $in-trans-comp$  and  $\langle reachable\ (composeALMs\ id1\ id2)\ (s1, s2) \rangle$  have
 $reachable\ (composeALMs\ id1\ id2)\ (s1', s2')$  by (auto intro:  $reachable.reachable-n$ )
    with  $\langle reachable\ (composeALMs\ id1\ id2)\ (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 <$ 
 $id2 \rangle$  and  $P13-invariant$  and  $P1b-invariant$  and  $P1a-invariant$  and  $P6-invariant$ 
    and  $P5-invariant$  and  $P10-invariant$  show  $P13\ (s1, s2)$  and  $P1b\ (s1, s2)$  and
 $P6\ (s1, s2)$  and  $P1a\ (s1, s2)$  and  $P5\ (s1, s2)$  and  $P10\ (s1, s2)$  unfolding
 $invariant-def$  by auto
  qed
  from  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$  and  $in-trans-comp$  show  $P15\ (s1', s2')$ 
proof (rule my-rule2)
  assume  $in-trans-cases-fun\ id1\ id2\ (s1, s2)\ (s1', s2')$ 
  thus  $P15\ (s1', s2')$  using  $\langle P15\ (s1, s2) \rangle$  and  $\langle 0 < id1 \rangle$  and  $\langle id1 < id2 \rangle$ 
proof (auto simp add:  $in-trans-cases-fun-def$ )
  fix  $ca\ ra$ 
  assume  $P15\ (s1, s2)$  and  $in-invoke1:(s1, Invoke\ ca\ ra, s1') \in ALM-trans\ 0$ 
 $id1$  and  $in-invoke2:(s2, Invoke\ ca\ ra, s2') \in ALM-trans\ id1\ id2$ 
  show  $P15\ (s1', s2')$ 
  proof -
    { assume  $s1' = s1$ 
      with  $\langle P15\ (s1, s2) \rangle$  and  $in-invoke1$  and  $in-invoke2$  and  $\langle 0 < id1 \rangle$  and
 $\langle id1 < id2 \rangle$ 
      have ?thesis by (auto simp add:  $ALM-trans-def\ P15-def$ )
    } note case1 = this
    { assume  $s1' \neq s1$ 
      with  $in-invoke1$  and  $in-invoke2$  and  $\langle P6\ (s1, s2) \rangle$  have  $s2' = s2$  apply
 $(auto\ simp\ add: ALM-trans-def\ P6-def)$  by (metis  $phase.simps(12)\ phase.simps(4)$ )
    }
  qed

```

```

    with  $\langle s1' \neq s1 \rangle$  and  $\langle P15(s1, s2) \rangle$  and in-invoke1 have ?thesis by (force
simp add:P15-def ALM-trans-def pendingReqs-def)
  } note case2 = this
  from case1 and case2 show ?thesis by auto
qed
next
fix ca h ra
  assume  $P15(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s1, \text{Switch } ca \ id1 \ h \ ra, s1') \in \text{ALM-trans } 0 \ id1$  and  $(s2, \text{Switch } ca \ id1 \ h \ ra, s2') \in \text{ALM-trans } id1 \ id2$ 
  thus  $P15(s1', s2')$  by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
next
fix c id' h
  assume  $P15(s1, s2)$  and  $0 < id1$  and  $(s2, \text{Commit } c \ id' \ h, s2') \in \text{ALM-trans } id1 \ id2$  and  $id1 \leq id'$  and  $id' < id2$ 
  thus  $P15(s1, s2')$  by (auto simp add: ALM-trans-def P15-def)
next
fix c h r
  assume  $P15(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Switch } c \ id2 \ h \ r, s2') \in \text{ALM-trans } id1 \ id2$ 
  thus  $P15(s1, s2')$  by (auto simp add: ALM-trans-def P15-def)
next
fix h
  assume in-lin: $(s2, \text{Linearize } id1 \ h, s2') \in \text{ALM-trans } id1 \ id2$ 
  show  $P15(s1, s2')$ 
  proof (auto simp add:P15-def)
    fix r
    assume  $\text{phase } s2'(\text{request-snd } r) = \text{Sleep}$  and  $r \in \text{set } (\text{hist } s2')$  and  $r \notin \text{pendingReqs } s1$ 
    show  $r \in \text{set } (\text{hist } s1)$ 
    proof -
      from  $\langle \text{phase } s2'(\text{request-snd } r) = \text{Sleep} \rangle$  and in-lin have  $\text{phase } s2(\text{request-snd } r) = \text{Sleep}$  by (auto simp add:ALM-trans-def)
      with  $\langle P1b(s1, s2) \rangle$  have  $r \notin \text{pendingReqs } s2$  by (auto simp add:pendingReqs-def P1b-def)
      with in-lin and  $\langle r \in \text{set } (\text{hist } s2') \rangle$  have  $r \in \text{set } (\text{hist } s2)$  by (auto simp add:ALM-trans-def postfix-all-def linearizations-def)
      with  $\langle \text{phase } s2(\text{request-snd } r) = \text{Sleep} \rangle$  and  $\langle P15(s1, s2) \rangle$  and  $\langle r \notin \text{pendingReqs } s1 \rangle$  show ?thesis by (auto simp add:P15-def)
    qed
  qed
next
assume  $P15(s1, s2)$  and  $0 < id1$  and  $id1 < id2$  and  $(s2, \text{Abort } id1, s2') \in \text{ALM-trans } id1 \ id2$ 
  thus  $P15(s1, s2')$  by (auto simp add: ALM-trans-def P15-def)
next
fix h
  assume in-init: $(s2, \text{Initialize } id1 \ h, s2') \in \text{ALM-trans } id1 \ id2$ 
  show  $P15(s1, s2')$ 

```

```

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
qed
next
  fix ca ta 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
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

$ref_mapping \equiv \lambda (s1, s2) .$

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

$phase = \lambda c. (if\ phase\ s1\ c \neq Aborted\ then\ phase\ s1\ c\ else\ phase\ s2\ c),$

$hist = (if\ hist\ s2 = []\ then\ hist\ s1\ else\ hist\ s2),$

$aborted = aborted\ s2,$

$initialized = True)$

theorem *composition*: $[id1 \neq 0; id1 < id2] ==> ((composeALMs\ id1\ id2) =<|$
 $(ALM-ioa\ 0\ id2))$

— The composition theorem

proof —

assume $id1 \neq 0$ **and** $id1 < id2$

show $composeALMs\ id1\ id2 =<| ALM-ioa\ 0\ id2$

proof (*simp add: ioa-implements-def, rule conjI, rule-tac[2] conjI*)

show $same-input-sig:inp\ (composeALMs\ id1\ id2) = inp\ (ALM-ioa\ 0\ id2)$

— First we show that both automata have the same input and output signature

using $\langle id1 \neq 0 \rangle$ **and** $\langle id1 < id2 \rangle$ **by** (*simp add: composeALMs-def hide-def*
 $hide-asig-def ALM-ioa-def asig-inputs-def asig-outputs-def asig-of-def ALM-asig-def$
 $par-def asig-comp-def, auto$)

from $\langle id1 \neq 0 \rangle$ **and** $\langle id1 < id2 \rangle$

show $same-output-sig:out\ (composeALMs\ id1\ id2) = out\ (ALM-ioa\ 0\ id2)$

— Then we show that output signatures match

by (*simp add: asig-inputs-def asig-outputs-def asig-of-def composeALMs-def*
 $hide-def hide-asig-def ALM-ioa-def ALM-asig-def par-def asig-comp-def, auto$)

show $traces\ (composeALMs\ id1\ id2) <= traces\ (ALM-ioa\ 0\ id2)$

— Finally we show trace inclusion

proof (*rule trace-inclusion[where f=ref-mapping]*)

— We use the mapping *ref-mapping*, defined before

from $same-input-sig$ **and** $same-output-sig$ **show** $ext\ (composeALMs\ id1\ id2)$
 $= ext\ (ALM-ioa\ 0\ id2)$

— First we show that they have the same external signature

by (*simp add: externals-def*)

next

show $is-ref-map\ ref-mapping\ (composeALMs\ id1\ id2)\ (ALM-ioa\ 0\ id2)$

— Then we show that *ref-mapping-comp* is a refinement mapping

apply (*simp add: is-ref-map-def, auto, rename-tac s1 s2*) **prefer 2 apply**
 $(rename-tac\ s1\ s2\ s1'\ s2'\ act)$

proof —

— First we show that start states correspond

fix $s1\ s2$

assume $(s1, s2) : starts-of\ (composeALMs\ id1\ id2)$

thus $ref-mapping\ (s1, s2) : starts-of\ (ALM-ioa\ 0\ id2)$ **using** $\langle id1 \neq$
 $0 \rangle$ **and** $\langle id1 < id2 \rangle$ **by** (*simp add: ALM-ioa-def ALM-start-def starts-of-def*
 $composeALMs-def hide-def par-def ref-mapping-def$)

next

— Then we show the main property of a refinement mapping
fix $s1\ s2\ s1'\ s2'\ act$
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

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 $P3\ (s1,\ s2)$

proof —

from $reachable$ **and** $in-trans-comp$ **have** $reachable\ (composeALMs\ id1\ id2)$
 $(s1',\ s2')$ **by** (rule $reachable.reachable-n$)

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 $\langle id1 \neq 0 \rangle$
and $\langle id1 < id2 \rangle$ **and** $reachable$

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)

qed

let $?t = ref_mapping\ (s1,\ s2)$

let $?t' = ref_mapping\ (s1',\ s2')$

show $EX\ ex.\ move\ (ALM-ioa\ 0\ id2)\ ex\ ?t\ act\ ?t'$

— the main part of the proof

proof (simp add: move-def, auto)

assume $act : ext\ (ALM-ioa\ 0\ id2)$

hence $act : \{act . EX\ c\ r . act = Invoke\ c\ r \mid (EX\ t . act = Switch\ c\ 0\ t\ r)\}$ $Un\ \{act . EX\ c\ tr . (EX\ id' . 0 \leq id' \ \&\ id' < id2 \ \&\ act = Commit\ c\ id'\ tr)\}$
 $\mid (EX\ r . act = Switch\ c\ id2\ tr\ r)\}$ **by** (auto simp add: ALM-ioa-def ALM-asig-def
externals-def asig-inputs-def asig-outputs-def asig-of-def)

with $in-trans-comp$ **show** $EX\ ex.\ is-exec-frag\ (ALM-ioa\ 0\ id2)\ (?t,\ ex)\ \&$
 $Finite\ ex\ \&\ laststate\ (?t,\ ex) = ?t' \ \&\ mk-trace\ (ALM-ioa\ 0\ id2)\ \$ex = [act!]$

— 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"

apply auto

proof —

fix $c\ r$

assume $in-invoke:(s1,\ s2) -Invoke\ c\ r--composeALMs\ id1\ id2->\ (s1',\ s2')$

— If the current action is Invoke

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

proof —

let $?ex = [(Invoke\ c\ r,\ ?t')!]$

have $Finite\ ?ex$ **by** auto

moreover **have** $laststate\ (?t,\ ?ex) = ?t'$ **by** (simp add: laststate-def)

moreover have $mk\text{-}trace\ (ALM\text{-}ioa\ 0\ id2)\$(?ex) = [Invoke\ c\ r!]$
by ($simp\ add: mk\text{-}trace\text{-}def\ external\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}of\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def$)

moreover have $is\text{-}exec\text{-}frag\ (ALM\text{-}ioa\ 0\ id2)\ (?t, ?ex)$
proof –
{
 assume $s1' \neq s1 \ \& \ s2' \neq s2$
 — contradiction
 with $in\text{-}invoke$ **and** $\langle id1 \neq 0 \rangle$ **and** $\langle id1 < id2 \rangle$ **and** $\langle P6\ (s1', s2') \rangle$ **have**
 $?thesis$ **apply** ($simp\ add: is\text{-}exec\text{-}frag\text{-}def\ composeALMs\text{-}def\ trans\text{-}of\text{-}def\ hide\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def\ par\text{-}def\ actions\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}internals\text{-}def\ asig\text{-}of\text{-}def$) **apply**($auto\ simp\ add: ALM\text{-}trans\text{-}def\ P6\text{-}def$) **done**
}
moreover
{
 assume $s1' = s1$ **and** $s2' = s2$
 with $in\text{-}invoke$ **have** $pre\text{-}s1: \sim (phase\ s1\ c = Ready \ \& \ request\text{-}snd\ r = c \ \& \ r \notin set\ (hist\ s1))$ **and** $pre\text{-}s2: \sim (phase\ s2\ c = Ready \ \& \ request\text{-}snd\ r = c \ \& \ r \notin set\ (hist\ s2))$ **using** $[[hyps\text{-}subst\text{-}thin]]$ **apply** ($auto\ simp\ add: is\text{-}exec\text{-}frag\text{-}def\ composeALMs\text{-}def\ trans\text{-}of\text{-}def\ hide\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def\ par\text{-}def\ actions\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}internals\text{-}def\ asig\text{-}of\text{-}def$) **apply**($simp\text{-}all\ add: ALM\text{-}trans\text{-}def$)
apply ($drule\text{-}tac[!]\ arg\text{-}cong[where\ f = phase]$) **apply** $simp\text{-}all$ **apply** ($metis\ phase.simps(8)\ fun\text{-}upd\text{-}idem\text{-}iff$) **apply** ($metis\ phase.simps(8)\ fun\text{-}upd\text{-}idem\text{-}iff$)
apply ($metis\ phase.simps(8)\ fun\text{-}upd\text{-}idem\text{-}iff$) **apply** ($metis\ phase.simps(8)\ fun\text{-}upd\text{-}idem\text{-}iff$)
done
 hence $\sim (phase\ ?t\ c = Ready \ \& \ request\text{-}snd\ r = c \ \& \ r \notin set\ (hist\ ?t))$ **using** $\langle P14\ (s1, s2) \rangle$ **by** ($auto\ simp\ add: ref\text{-}mapping\text{-}def\ P14\text{-}def$)
 hence $?thesis$ **using** $\langle id1 \neq 0 \rangle$ **and** $\langle s1' = s1 \rangle$ **and** $\langle s2' = s2 \rangle$ **apply**
 ($simp\ add: is\text{-}exec\text{-}frag\text{-}def\ composeALMs\text{-}def\ trans\text{-}of\text{-}def\ hide\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def\ par\text{-}def\ actions\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}internals\text{-}def\ asig\text{-}of\text{-}def$) **apply**($simp\text{-}all\ add: ALM\text{-}trans\text{-}def$) **apply** $force$ **done**
}
moreover
{
 assume $s1' \neq s1$ **and** $s2' = s2$
 with $in\text{-}invoke$ **have** $pre\text{-}s1: phase\ s1\ c = Ready \ \& \ request\text{-}snd\ r = c \ \& \ r \notin set\ (hist\ s1)$ **and** $trans\text{-}s1: s1' = s1 \ (\backslash pending := (pending\ s1)(c := r), phase := (phase\ s1)(c := Pending))$ **apply** ($simp\text{-}all\ add: is\text{-}exec\text{-}frag\text{-}def\ composeALMs\text{-}def\ trans\text{-}of\text{-}def\ hide\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def\ par\text{-}def\ actions\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}internals\text{-}def\ asig\text{-}of\text{-}def$) **apply**($simp\text{-}all\ add: ALM\text{-}trans\text{-}def\ ref\text{-}mapping\text{-}def$) **done**
 have $pre\text{-}t: phase\ ?t\ c = Ready \ \& \ request\text{-}snd\ r = c \ \& \ r \notin set\ (hist\ ?t)$
proof –
 from $pre\text{-}s1$ **have** $phase\ ?t\ c = Ready \ \& \ request\text{-}snd\ r = c$ **by**
 ($auto\ simp\ add: ref\text{-}mapping\text{-}def$)
 moreover have $r \notin set\ (hist\ ?t)$
 proof ($cases\ hist\ s2 = []$)

```

    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
  ultimately show ?thesis by auto
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-invoke 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->
(s1', s2')
    — 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 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 ≠ 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->
(s1', s2')
    — 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 $mk\text{-}trace\ (ALM\text{-}ioa\ 0\ id2)\$(?ex) = [Switch\ c\ id2\ h\ r!]$
by ($simp\ add: mk\text{-}trace\text{-}def\ external\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}of\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def$)

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

proof –

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

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

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

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

proof –

from $s1\text{-}aborted$ **and** $pre\text{-}s2$ **have** $aborted\ ?t$ & $pending\ ?t\ c = r$ **and** $phase\ ?t\ c = Pending$ **and** $initialized\ ?t$ **by** ($auto\ simp\ add: ref\text{-}mapping\text{-}def\ fun\text{-}eq\text{-}iff$)

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

proof –

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

thus $?thesis$

proof $auto$

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

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

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

proof –

have $hist\ ?t = hist\ s2$

proof ($cases\ hist\ s2 = []$)

assume $hist\ s2 = []$

show $hist\ ?t = hist\ s2$

proof –

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

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

qed

next

```

    assume hist s2 ≠ []
    thus hist ?t = hist s2 by (simp add:ref-mapping-def)
  qed
  moreover have pendingReqs s2 ≤ pendingReqs ?t
  proof (simp add: pendingReqs-def, clarify)
    fix c
    assume pending s2 c ∉ set (hist s2) and phase s2 c =
    Pending ∨ phase s2 c = Aborted
    moreover with ⟨P6 (s1, s2)⟩ have phase s1 c = Aborted
  by (auto simp add:P6-def)
    moreover note ⟨suffixeq (hist s1) (hist s2)⟩
    ultimately show ∃ ca. pending s2 c = pending ?t ca ∧
    pending s2 c ∉ set (hist ?t) ∧ (phase ?t ca = Pending ∨ phase ?t ca = Aborted)
  apply (simp add:ref-mapping-def suffixeq-def) by (metis prefixeq-Nil prefixeq-def
    self-append-conv2)
  qed
  moreover note case1-2
  ultimately show ?thesis by (auto simp add: linearizations-def
    postfix-all-def)
  qed
  next
    assume case2-1:¬ initialized s2 and case2-2:h : postfix-all (l-c-p
    (initHists s2)) (linearizations (initValidReqs s2))
    from case2-1 and ⟨P10 (s1, s2)⟩ have hist s2 = [] by (auto
    simp add:P10-def)
    have h : postfix-all (hist s1) (linearizations (pendingReqs s1))
    proof -
      from pre-s2 have phase s2 c ≠ Sleep by auto
      moreover note ⟨P13 (s1, s2)⟩ and case2-1 and case2-2
      ultimately show ?thesis by (auto simp add:P13-def)
    qed
    moreover from ⟨hist s2 = []⟩ have hist ?t = hist s1 by (auto
    simp add:P10-def ref-mapping-def)
    moreover have pendingReqs ?t = pendingReqs s1
    proof auto
      fix r
      assume r ∈ pendingReqs ?t
      with this obtain c' where r = pending ?t c' and r ∉ set (hist
      ?t) and phase ?t c' ∈ {Pending, Aborted} by (auto simp add:pendingReqs-def)
      show r ∈ pendingReqs s1
      proof (cases phase s1 c' = Aborted)
        assume phase s1 c' = Aborted
        with ⟨phase ?t c' ∈ {Pending, Aborted}⟩ and ⟨r = pending ?t
        c'⟩ have phase s2 c' ∈ {Pending, Aborted} and r = pending s2 c' by (auto simp
        add:ref-mapping-def)
        with ⟨P6 (s1, s2)⟩ and case2-1 and ⟨P7 (s1, s2)⟩ and
        ⟨hist ?t = hist s1⟩ and ⟨r ∉ set (hist ?t)⟩ have phase s1 c' = Aborted and r =
        pending s1 c' and r ∉ set (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)
    with ⟨hist s2 = []⟩ and ⟨¬ initialized s2⟩ and ⟨P7 (s1, s2)⟩ show
r ∈ pendingReqs ?t by (auto simp add:ref-mapping-def pendingReqs-def P7-def)
  qed
  ultimately show ?thesis by (auto simp add: postfix-all-def
linearizations-def)
qed
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)
ultimately show ?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
qed
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 –

```

```

{
  assume  $id' < id1$ 
  with in-commit have  $s2' = s2$  and  $pre-s1:phase\ s1\ c = Pending$ 
 $\wedge pending\ s1\ c \in set\ (hist\ s1) \wedge h = dropWhile\ (\lambda\ r.\ r \neq pending\ s1\ c)\ (hist\ s1)$ 
  and  $trans-s1:s1' = s1\ (\llbracket phase := (phase\ s1)(c := Ready) \rrbracket)$  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-s1 have  $s1\text{-not-aborted-}c:phase\ s1\ c \neq Aborted$  by auto
  have  $pre-t:phase\ ?t\ c = Pending \ \&\ pending\ ?t\ c \in set\ (hist\ ?t) \wedge h$ 
 $= dropWhile\ (\lambda\ r.\ r \neq pending\ ?t\ c)\ (hist\ ?t)$ 
  proof (cases  $hist\ s2 = []$ )
    assume  $hist\ s2 = []$ 
    with pre-s1 and  $\langle phase\ s1\ c \neq Aborted \rangle$  show ?thesis by (auto
    simp add: ref-mapping-def)
  next
    assume  $hist\ s2 \neq []$ 
    hence initialized  $s2$  using  $\langle P10\ (s1, s2) \rangle$  by (auto simp add: P10-def)
    from pre-s1 and  $\langle phase\ s1\ c \neq Aborted \rangle$  have  $phase\ ?t\ c = Pending$ 
    and  $pending\ ?t\ c = pending\ s1\ c$  and  $pending\ s1\ c \in set\ (hist\ s1)$  by (auto simp
    add: ref-mapping-def)
    moreover have  $pending\ ?t\ c \in set\ (hist\ ?t)$ 
    proof -
      from  $\langle initialized\ s2 \rangle$  and  $\langle P14\ (s1, s2) \rangle$  obtain  $rs3$  where  $hist$ 
 $s2 = rs3\ @\ (hist\ s1)$  by (auto simp add: P14-def)
      with  $\langle pending\ s1\ c \in set\ (hist\ s1) \rangle$  and  $\langle hist\ s2 = rs3\ @\ (hist\ s1) \rangle$ 
      and  $\langle pending\ ?t\ c = pending\ s1\ c \rangle$  show  $pending\ ?t\ c \in set\ (hist\ ?t)$  by (auto
      simp add: ref-mapping-def suffixeq-def)
    qed
    moreover have  $h = dropWhile\ (\lambda\ r.\ r \neq pending\ ?t\ c)\ (hist\ ?t)$ 
    proof -
      from  $\langle pending\ s1\ c \in set\ (hist\ s1) \rangle$  obtain  $rs1\ rs2$  where  $hist$ 
 $s1 = rs2\ @\ rs1$  and  $hd\ rs1 = pending\ s1\ c$  and  $rs1 \neq []$  and  $pending\ s1\ c \notin set$ 
 $rs2$  by (metis list.sel(1) in-set-conv-decomp-first list.simps(3))
      with  $\langle pending\ ?t\ c = pending\ s1\ c \rangle$  and dropWhile-lemma[of  $hist$ 
 $s1\ rs1\ pending\ s1\ c$ ] and pre-s1 have  $h = rs1$  by auto
      moreover have  $dropWhile\ (\lambda\ r.\ r \neq pending\ ?t\ c)\ (hist\ ?t) =$ 
 $rs1$ 
      proof -
        from  $\langle initialized\ s2 \rangle$  and  $\langle P14\ (s1, s2) \rangle$  obtain  $rs3$  where  $hist$ 
 $s2 = rs3\ @\ (hist\ s1)$  and  $set\ rs3 \cap set\ (hist\ s1) = \{\}$  by (auto simp add: P14-def)
        with  $\langle pending\ s1\ c \in set\ (hist\ s1) \rangle$  and  $\langle hist\ s1 = rs2\ @\ rs1 \rangle$ 
        have  $hist\ s2 = rs3\ @\ rs2\ @\ rs1$  and  $pending\ s1\ c \notin set\ rs3$  by auto
        with  $\langle pending\ s1\ c \notin set\ rs2 \rangle$  obtain  $rs4$  where  $hist\ s2 = rs4$ 
 $@\ rs1$  and  $pending\ s1\ c \notin set\ rs4$  by auto
        with  $\langle hd\ rs1 = pending\ s1\ c \rangle$  and  $\langle rs1 \neq [] \rangle$  and dropWhile-lemma[of  $hist$ 
 $s2\ rs1\ pending\ s1\ c$ ] have  $dropWhile\ (\lambda\ r.\ r \neq pending\ s1\ c)\ (hist\ s2) = rs1$ 
        by auto
      thus ?thesis using  $\langle hist\ s2 \neq [] \rangle$  and  $\langle pending\ ?t\ c = pending$ 

```

$s1\ c\rangle$ **by** (*auto simp add:ref-mapping-def*)
 qed
 ultimately show *?thesis* **by** *auto*
 qed
 ultimately show *?thesis* **by** *auto*
 qed
 moreover from $\langle s2' = s2 \rangle$ **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** $\langle id1 < id2 \rangle$ **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 $id1 \leq id'$
with *in-commit* **have** $s1' = s1$ **and** *pre-s2:phase s2 c = Pending*
 \wedge *pending s2 c* \in *set (hist s2) \wedge h = dropWhile ($\lambda r . r \neq$ 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** $\langle P6\ (s1, s2) \rangle$ **have** *facts:aborted s1 & phase s1 c*
 $=$ *Aborted & hist s2 $\neq \square$* **by** (*force simp add:P6-def*)
with *pre-s2* **have** *pre-t:phase ?t c = Pending \wedge pending ?t c* \in
set (hist ?t) \wedge h = dropWhile ($\lambda r . r \neq$ pending ?t c) (hist ?t) **by** (*auto simp*
add:ref-mapping-def)
moreover from $\langle s1' = s1 \rangle$ **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** $\langle id1 < id2 \rangle$ **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** $\langle id' < id2 \rangle$ **by** *force*
qed
ultimately show *?thesis* **by** (*auto intro: exI[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 $act \notin ext\ (ALM-ioa\ 0\ id2)$
 — Now the case when the composition takes an action that is not in the
 external signature of the spec
with *in-trans-comp* **and** $\langle id1 < id2 \rangle$ **and** $\langle id1 \neq 0 \rangle$ **have** $act : \{act$
 $.\ act = Abort\ 0 \mid act = Abort\ id1 \mid (EX\ c\ r\ h . act = Linearize\ 0\ h \mid act =$

$Linearize\ id1\ h \mid act = Switch\ c\ id1\ h\ r \mid act = Initialize\ 0\ h \mid act = Initialize\ id1\ h\}$ **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** $\exists ex. is-exec-frag\ (ALM-ioa\ 0\ id2)\ (?t, ex) \wedge Finite\ ex \wedge laststate\ (?t, ex) = ?t' \wedge mk-trace\ (ALM-ioa\ 0\ id2) \cdot ex = nil$

proof auto

assume in-abort: $(s1, s2) - Abort\ 0 - composeALMs\ id1\ id2 \longrightarrow (s1', s2')$

— The case where the first Aabstract aborts

moreover with $\langle id1 \neq 0 \rangle$ **and** $\langle id1 < id2 \rangle$ **and** $\langle P6\ (s1, s2) \rangle$ **and** $\langle P2\ (s1, s2) \rangle$ **have** $\forall c. phase\ s1\ c \neq Aborted$ **and** $hist\ s2 = []$ **and** $\forall c. phase\ s2\ c = Sleep$

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**

moreover note $\langle id1 \neq 0 \rangle$

ultimately have $?t' = ?t$ **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) **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 $mk-trace\ (ALM-ioa\ 0\ id2) \cdot ?ex = nil$ **using** $\langle id1 < id2 \rangle$ **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 $\exists ex. is-exec-frag\ (ALM-ioa\ 0\ id2)\ (?t, ex) \wedge Finite\ ex \wedge laststate\ (?t, ex) = ?t \wedge mk-trace\ (ALM-ioa\ 0\ id2) \cdot ex = nil$ **by** (auto intro: exI[where $x=?ex$])

qed

next

assume in-abort: $(s1, s2) - Abort\ id1 - composeALMs\ id1\ id2 \longrightarrow (s1', s2')$

— 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 $mk-trace\ (ALM-ioa\ 0\ id2) \cdot ?ex = nil$ **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 $(ALM-ioa\ 0\ id2)\ (?t, ?ex)$

proof —

from in-abort **and** $\langle id1 \neq 0 \rangle$ **have** $s1' = s1$ **and** $pre-s2: \sim aborted\ s2 \ \& (\exists c. phase\ s2\ c \neq Sleep)$ **and** $trans-s2: s2' = s2 \mid aborted := True$ **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  $\langle P6 (s1, s2) \rangle$  have pre-t:~ aborted ?t &  $(\exists c .$ 
phase ?t c  $\neq$  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( $\backslash$ aborted:= True) by (auto simp add: fun-eq-iff ref-mapping-def)
    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(simp add:ALM-trans-def)
done
    qed
    ultimately show ?thesis by (auto intro: exI[where x=?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 ?ex = [(Linearize 0 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 = 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)
      proof —
        from in-lin and  $\langle id1 \neq 0 \rangle$  have s2' = s2 and pre-s1:initialized
s1 & ~ aborted s1 & h  $\in$  postfix-all (hist s1) (linearizations (pendingReqs s1))
and trans-s1:s1' = s1( $\backslash$ hist := h, initialized := True) 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
        have pre-t:initialized ?t & ~ aborted ?t & h  $\in$  postfix-all (hist ?t)
(linearizations (pendingReqs ?t))
        proof —
          from pre-s1 have ~ aborted s1 by auto
          with  $\langle P9 (s1, s2) \rangle$  have ~ aborted ?t and initialized ?t by (auto
simp add:ref-mapping-def P9-def)
          moreover have h  $\in$  postfix-all (hist ?t) (linearizations (pendingReqs
?t))
          proof —
            from  $\langle \neg$  aborted s1  $\rangle$  have hist ?t = hist s1 using  $\langle P6 (s1, s2) \rangle$ 
and  $\langle P2 (s1, s2) \rangle$  by (auto simp add:P6-def P2-def ref-mapping-def)
            moreover have pendingReqs s1  $\subseteq$  pendingReqs ?t
            proof auto
              fix x

```



```

      assume  $x \in \text{pendingReqs } s1$ 
      moreover note  $\langle \neg \text{aborted } s1 \rangle$  and  $\langle P6 (s1, s2) \rangle$ 
      ultimately obtain  $c$  where  $x = \text{pending } s1 \ c$  and phase  $s1$ 
 $c = \text{Pending}$  and  $\text{pending } s1 \ c \notin \text{set } (\text{hist } s1)$  by (auto simp add: pendingReqs-def
 $P6\text{-def}$ )
      thus  $x \in \text{pendingReqs } ?t$  using  $\langle \text{hist } ?t = \text{hist } s1 \rangle$  by (force simp
add: ref-mapping-def pendingReqs-def)
    qed
    moreover from pre-s1 have  $h \in \text{postfix-all } (\text{hist } s1)$  (linearizations
(pendingReqs  $s1$ )) 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:  $?t' = ?t \setminus \{ \text{hist} := h, \text{initialized} := \text{True} \}$ 
proof -
  have  $\text{hist } ?t' = \text{hist } s1'$ 
  proof -
    from pre-s1 have  $\sim \text{aborted } s1$  by auto
    with  $\langle P6 (s1, s2) \rangle$  and  $\langle P2 (s1, s2) \rangle$  have  $\text{hist } s2 = []$  by (auto
simp add: P6-def P2-def)
    with  $\langle s2' = s2 \rangle$  show ?thesis by (auto simp add: ref-mapping-def)
  qed
  with trans-s1 have  $\text{hist } ?t' = h$  by auto
  thus ?thesis using  $\langle s2' = s2 \rangle$  and trans-s1 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  $x = ?ex$ ])
qed
next
  fix  $h$ 
  assume in-lin:  $(s1, s2) \text{ --Linearize id1 } h \text{ --composeALMs id1 id2 --> } (s1', s2')$ 
  — If the composition executes Linearize id1
  let  $?ex = [( \text{Linearize id1 } h, ?t' )]$ 
  have Finite ?ex by auto
  moreover have laststate  $(?t, ?ex) = ?t'$  by (simp add: laststate-def)
  moreover have mk-trace  $(\text{ALM-ioa } 0 \text{ id2}). ?ex = \text{nil}$  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  $(\text{ALM-ioa } 0 \text{ id2}) (?t, ?ex)$ 
  proof -
    from in-lin and  $\langle id1 \neq 0 \rangle$  have  $s1' = s1$  and pre-s2: initialized  $s2$ 

```

$\wedge \neg \text{aborted } s2 \wedge h \in \text{postfix-all } (\text{hist } s2) (\text{linearizations } (\text{pendingReqs } s2))$ **and**
 $\text{trans-s2}: s2' = s2 \langle \text{hist} := h \rangle$ **apply** ($\text{simp-all add: composeALMs-def trans-of-def}$
 $\text{hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def}$
 $\text{ALM-ioa-def ALM-asig-def}$) **apply**($\text{auto simp add: ALM-trans-def}$) **done**
have $\text{pre-t:initialized } ?t \wedge \neg \text{aborted } ?t \wedge h \in \text{postfix-all } (\text{hist } ?t)$
 $(\text{linearizations } (\text{pendingReqs } ?t))$
proof –
have $\neg \text{aborted } ?t$ **and** $\text{initialized } ?t$ **using** pre-s2 **by** (auto simp
 $\text{add:ref-mapping-def}$)
moreover have $h \in \text{postfix-all } (\text{hist } ?t) (\text{linearizations } (\text{pendingReqs}$
 $?t))$
proof –
from pre-s2 **have** $\text{initialized } s2$ **by** auto
hence $\text{suffixeq } (\text{hist } s1) (\text{hist } s2)$ **using** $\langle P14 \ (s1, s2) \rangle$ **by** (auto
 $\text{simp add:P14-def suffixeq-def}$)
hence $\text{hist } ?t = \text{hist } s2$ **by** ($\text{auto simp add:ref-mapping-def}$)
moreover have $\text{pendingReqs } s2 \subseteq \text{pendingReqs } ?t$
proof auto
fix x
assume $x \in \text{pendingReqs } s2$
from this obtain c **where** $x = \text{pending } s2 \ c$ **and** phase
 $s2 \ c \in \{\text{Pending}, \text{Aborted}\}$ **and** $\text{pending } s2 \ c \notin \text{set } (\text{hist } s2)$ **by** (auto simp
 $\text{add:pendingReqs-def}$)
with $\langle P6 \ (s1, s2) \rangle$ **and** $\langle \text{hist } ?t = \text{hist } s2 \rangle$ **show** $x \in \text{pendingReqs}$
 $?t$ **by** ($\text{force simp add:ref-mapping-def P6-def pendingReqs-def}$)
qed
moreover from pre-s2 **have** $h \in \text{postfix-all } (\text{hist } s2) (\text{linearizations}$
 $(\text{pendingReqs } s2))$ **by** auto
ultimately show $?thesis$ **by** ($\text{auto simp add:postfix-all-def}$
 $\text{linearizations-def}$)
qed
ultimately show $?thesis$ **by** auto
qed
moreover have $\text{trans-t}: ?t' = ?t \langle \text{hist} := h \rangle$
proof –
from pre-s2 **and** trans-s2 **have** $\text{initialized } s2'$ **by** auto
hence $\text{suffixeq } (\text{hist } s1') (\text{hist } s2')$ **using** $\langle P14 \ (s1', s2') \rangle$ **by** (auto
 $\text{simp add:P14-def suffixeq-def}$)
hence $\text{hist } ?t' = \text{hist } s2'$ **by** ($\text{auto simp add:ref-mapping-def}$)
with trans-s2 **and** $\langle s1' = s1 \rangle$ **show** $?thesis$ **by** (auto simp
 $\text{add:ref-mapping-def fun-eq-iff}$)
qed
ultimately show $?thesis$ **apply** ($\text{simp add: is-exec-frag-def composeALMs-def}$
 $\text{trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def}$
 $\text{asig-inputs-def asig-internals-def asig-of-def}$) **apply**($\text{auto simp add:ALM-trans-def}$)
done
qed
ultimately show $?thesis$ **by** ($\text{auto intro: exI}[\text{where } x=?ex]$)

```

next
  fix c r h
  assume in-switch:(s1, s2) - Switch c id1 h r - composeALMs id1 id2 →
    (s1', s2')
    — If the composition switches internally
  show ?thesis
  proof -
    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 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)
    moreover have ?t' = ?t
    proof -
      from in-switch and ⟨id1 ≠ 0⟩ have pre-s1:aborted s1 ∧ phase
s1 c = Pending ∧ r = pending s1 c ∧ (if initialized s1 then (h ∈ postfix-all (hist
s1) (linearizations (pendingReqs s1))) else (h : postfix-all (l-c-p (initHists s1))
(linearizations (initValidReqs s1)))) and trans-s1: s1' = s1 (|phase := (phase s1)(c
:= 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-ioa-def
ALM-asig-def) apply (auto simp add: ALM-trans-def) done
      have pre-s2:phase s2 c = Sleep and trans-s2: s2' = s2 (|initHists
:= {h} ∪ (initHists s2), phase := (phase s2)(c := Pending), pending := (pending
s2)(c := r)|)
      proof -
        from pre-s1 have phase s1 c = Pending by auto
        with ⟨P6 (s1, s2)⟩ have phase s2 c = Sleep apply (simp add: P6-def)
      by (metis phase.simps(10))
      with in-switch and ⟨id1 ≠ 0⟩ and ⟨id1 < id2⟩ show phase s2 c = Sleep
and s2' = s2 (|initHists := {h} ∪ (initHists s2), phase := (phase s2)(c := Pend-
ing), pending := (pending s2)(c := r)|) 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 P6-def)
      done
    qed
    from pre-s1 and pre-s2 and trans-s1 and trans-s2 and ⟨P1a
(s1, s2)⟩ have pending ?t c = pending ?t' c & initHists ?t = initHists ?t' & hist
?t = hist ?t' & aborted ?t = aborted ?t' ∧ phase ?t' c = phase ?t 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=?ex])
qed
next
fix h

```

```

    assume in-initialize:(s1, s2) - Initialize 0 h - composeALMs id1 id2 →
(s1', s2')
    hence False using ⟨P10 (s1, s2)⟩ 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 P10-def)
done
    thus ?thesis by auto
next
    fix h
    assume in-initialize:(s1, s2) - Initialize id1 h - composeALMs id1 id2 →
(s1', s2')
    — If the second ALM of the composition initializes
    let ?ex = [(Linearize id1 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 = 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)
    proof —
    from in-initialize and ⟨id1 ≠ 0⟩ have s1' = s1 and pre-s2:(∃ c . phase
s2 c ≠ Sleep) ∧ ¬ aborted s2 ∧ ¬ initialized s2 ∧ h ∈ postfix-all (l-c-p (initHists
s2)) (linearizations (initValidReqs s2)) and trans-s2:s2' = s2(hist := h, initialized
:= True) 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
    have pre-t:initialized ?t ∧ ¬ aborted ?t ∧ h ∈ postfix-all (hist ?t)
(linearizations (pendingReqs ?t))
    proof —
    from pre-s2 have initialized ?t ∧ ¬ aborted ?t by (auto simp
add:ref-mapping-def)
    moreover have h ∈ postfix-all (hist ?t) (linearizations (pendingReqs
?t))
    proof —
    from pre-s2 have h ∈ postfix-all (l-c-p (initHists s2)) (linearizations
(initValidReqs s2)) and ¬ initialized s2 and ∃ c . phase s2 c ≠ Sleep by auto
    with ⟨P13 (s1, s2)⟩ have h ∈ postfix-all (hist s1) (linearizations
(pendingReqs s1)) by (auto simp add: P13-def)
    moreover from ⟨¬ initialized s2⟩ and ⟨P10 (s1, s2)⟩ have hist ?t
= hist s1 by (auto simp add:ref-mapping-def P10-def)
    moreover have pendingReqs s1 ⊆ pendingReqs ?t
    proof auto
    fix x
    assume x ∈ pendingReqs s1
    from this obtain c where x = pending s1 c and phase
s1 c ∈ {Pending, Aborted} and pending s1 c ∉ set (hist s1) by (auto simp
add:pendingReqs-def)
    show x ∈ pendingReqs ?t
    proof (cases phase s1 c = Pending)

```

```

      assume phase s1 c = Pending
      with ⟨x = pending s1 c⟩ and ⟨pending s1 c ∉ set (hist s1)⟩ and ⟨hist
?t = hist s1⟩ show ?thesis by (force simp add:ref-mapping-def pendingReqs-def)
    next
      assume phase s1 c ≠ Pending
      with ⟨phase s1 c ∈ {Pending, Aborted}⟩ have phase s1 c =
Aborted by auto
      with ⟨¬ initialized s2⟩ and ⟨P6 (s1, s2)⟩ and ⟨P7 (s1, s2)⟩ have
pending s2 c = pending s1 c and phase s2 c ∈ {Pending, Aborted} by (auto simp
add:P6-def P7-def)
      with ⟨x = pending s1 c⟩ and ⟨pending s1 c ∉ 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 ∃ c . phase s2 c ≠ Sleep by auto
    with trans-s2 have initialized s2' and ∃ c . phase s2' c ≠ 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
  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])
  qed
  qed
  qed
  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

- [1] R. Guerraoui, V. Kuncak, and G. Losa. Speculative linearizability. Technical report, EPFL, 2011. Accepted for publication at PLDI 2012, available at <http://lara.epfl.ch/w/slin>.
- [2] M. P. Herlihy and J. M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.
- [3] L. Lamport and K. Marzullo. The part-time parliament. *ACM Transactions on Computer Systems*, 16:133–169, 1998.
- [4] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.