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

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

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

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

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

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

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

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

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

definition *common-postfix-p* :: ($'a$ *list*) *set* => $'a$ *list* => *bool*
— Predicate that recognizes the common 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* . *if xss* = {} *then xs* = [] *else ALL xs'* . *xs'* $\in$ *xss* $\longrightarrow$ *suffixeq xs xs'*

definition *l-c-p-pred* :: $'a$ *list set* $\Rightarrow$ $'a$ *list* => *bool*
— Predicate that recognizes the longest common postfix of a set of lists
where
l-c-p-pred $\equiv \lambda$ *xss xs* . *common-postfix-p xss xs* $\wedge$ (*ALL xs'* . *common-postfix-p xss xs'* $\longrightarrow$ *suffixeq xs' xs*)

definition *l-c-p*:: $'a$ *list set* $\Rightarrow$ $'a$ *list*
— The longest common postfix of a set of lists
where
l-c-p $\equiv \lambda$ *xss* . *THE xs* . *l-c-p-pred xss xs*

lemma *l-c-p-ok*: *l-c-p-pred xss* (*l-c-p xss*)
— Proof that the definition of the longest common 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*))) $\longrightarrow$ *suffixeq xs* (*l-c-p ls*)

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

lemma *l-c-p-longest*: *common-postfix-p xss xs* $\longrightarrow$ *suffixeq xs* (*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 ~~/*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* ⇒ *client*

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

type-synonym *hist* = *request list*

— Type of histories of requests.

datatype *ALM-action* =

— The actions of the ALM automaton

Invoke client request

| *Commit client nat hist*

| *Switch client nat hist request*

| *Initialize nat hist*

| *Linearize nat hist*

| *Abort nat*

datatype *phase* = *Sleep* | *Pending* | *Ready* | *Aborted*

— Executions phases of a client

definition *linearizations* :: *request set* ⇒ *hist set*

— The possible linearizations of a set of requests

where

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

definition *postfix-all* :: *hist* ⇒ *hist set* ⇒ *hist set*

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

where

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

definition

ALM-asig :: *nat* ⇒ *nat* ⇒ *ALM-action signature*

— The action signature of ALM automata

— Input actions, output actions, and internal actions

where

$ALM\text{-}asig \equiv \lambda\ id1\ id2\ .\ ($
$\{act\ .\ \exists\ c\ r\ h\ .$
$\quad act = Invoke\ c\ r \mid act = Switch\ c\ id1\ h\ r\},$
$\{act\ .\ \exists\ c\ h\ r\ id'\ .$
$\quad id1 <= id' \wedge id' < id2 \wedge act = Commit\ c\ id'\ h$
$\quad \mid act = Switch\ c\ id2\ h\ r\},$
$\{act\ .\ \exists\ h\ .$
$\quad act = Abort\ id1$
$\quad \mid act = Linearize\ id1\ h$

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

record *ALM-state* =

— The state of the ALM automata

pending :: *client* ⇒ *request*

— Associates a pending request to a client process

initHists :: *hist set*

— The set of init histories submitted by clients

phase :: *client* ⇒ *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* ⇒ *request set*

— the set of requests that have been invoked but that are not yet in the hist parameter

where

$pendingReqs \equiv \lambda\ s\ .\ \{r\ .\ \exists\ \ c\ .$
$\quad r = pending\ s\ c$
$\quad \wedge\ r \notin set\ (hist\ s)$
$\quad \wedge\ phase\ s\ c \in \{Pending,\ Aborted\}\}$

definition *initValidReqs* :: *ALM-state* ⇒ *request set*

— any request that appears in an init hist after the longest common prefix or that is pending

where

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

definition

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

— the transitions of the ALM automaton

where

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

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

$\quad |Linearize\ i\ h \Rightarrow$
$\quad\quad initialized\ s \wedge \neg\ aborted\ s$
$\quad\quad \wedge\ h \in postfix\text{-}all\ (hist\ s)\ (linearizations\ (pendingReqs\ s\ ))$

$\wedge\ s' = s(\!|hist := h|\!)$

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

$|Abort\ i \Rightarrow$
$\neg\ aborted\ s \wedge\ (\exists\ c\ .\ phase\ s\ c \neq Sleep)$
$\wedge\ s' = s(\!|aborted{:=}\ True|\!)$

$|Commit\ c\ i\ h \Rightarrow$
$phase\ s\ c = Pending \wedge pending\ s\ c \in set\ (hist\ s)$
$\wedge\ h = dropWhile\ (\lambda\ r\ .\ r \neq pending\ s\ c)\ (hist\ s)$
$\wedge\ s' = s\ (\!|phase := (phase\ s)(c := Ready)|\!)$

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

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

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

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

type-synonym *compo-state* = *ALM-state* × *ALM-state*

definition *composeALMs* :: *nat* ⇒ *nat* ⇒ (*ALM-action*, *compo-state*) *ioa*
— the composition of two ALMs
where
composeALMs ≡ λ *id1 id2* .
hide (*ALM-ioa 0 id1* || *ALM-ioa id1 id2*)
{*act* . *EX c tr r* . *act* = *Switch c id1 tr r*}

end

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

theory *CompositionCorrectness*
imports *ALM*
begin

declare *split-if-asm* [*split*]
declare *Let-def* [*simp*]

4.1 A case split useful in the proofs

definition *in-trans-cases-fun* :: *nat* => *nat* => (*ALM-state* ∗ *ALM-state*) => (*ALM-state* ∗ *ALM-state*) => *bool*
— Helper function used to decompose proofs
where
in-trans-cases-fun == % *id1 id2 s t* .
(*EX ca ra*. (*fst s*, *Invoke ca ra*, *fst t*) : *ALM-trans 0 id1* & (*snd s*, *Invoke ca ra*, *snd t*) : *ALM-trans id1 id2*)
| (*EX ca h ra*. (*fst s*, *Switch ca id1 h ra*, *fst t*) : *ALM-trans 0 id1* & (*snd s*, *Switch ca id1 h ra*, *snd t*) : *ALM-trans id1 id2*)
| (*EX c id′ h*. *fst t* = *fst s* & (*snd s*, *Commit c id′ h*, *snd t*) : *ALM-trans id1 id2* & *id1* <= *id′* & *id′* < *id2*)
| (*EX c h r*. *fst t* = *fst s* & (*snd s*, *Switch c id2 h r*, *snd t*) : *ALM-trans id1 id2*)
| (*EX h* . *fst t* = *fst s* & (*snd s*, *Linearize id1 h*, *snd t*) : *ALM-trans id1 id2*)
| (*fst t* = *fst s* & (*snd s*, *Abort id1*, *snd t*) : *ALM-trans id1 id2*)
| (*EX h*. *fst t* = *fst s* & (*snd s*, *Initialize id1 h*, *snd t*) : *ALM-trans id1 id2*)
| (*EX ca ta ra*. (*fst s*, *Switch ca 0 ta ra*, *fst t*) : *ALM-trans 0 id1* & *snd t* = *snd s*)
| (*EX ca id′ h*. (*fst s*, *Commit ca id′ h*, *fst t*) : *ALM-trans 0 id1* & *snd t* = *snd s* & *id′* < *id1*)
| (*EX h* . (*fst s*, *Linearize 0 h*, *fst t*) : *ALM-trans 0 id1* & *snd t* = *snd s*)
| (*EX h*. (*fst s*, *Initialize 0 h*, *fst t*) : *ALM-trans 0 id1* & *snd t* = *snd s*)

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

lemma *composeALMsE*:
— A rule for decomposing proofs
assumes *id1* ~= *0* **and** *id1* < *id2* **and** *in-trans-comp*:*s* −(*a*::*ALM-action*)−−*composeALMs id1 id2*−> *t*
shows *decomp*: *in-trans-cases-fun id1 id2 s t*
proof −
from *in-trans-comp* **and** ‹*id1* ~= *0*› **and** ‹*id1* < *id2*›
have *a* : {*act* . *EX c r h id′* . *0* <= *id′* & *id′* < *id2* & (
act = *Invoke c r*
| *act* : {*Switch c 0 h r*, *Switch c id1 h r*, *Switch c id2 h r*}
| *act* : {*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* <= *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** ‹*id1* ~= *0*› **and** ‹*id1* < *id2*›
have
(*a* = *Linearize 0 h* | *a* = *Abort 0* | *a* = *Initialize 0 h* | *a* = *Switch c 0 h r* | (*a* = *Commit c id′ h* & *id′* < *id1*)) ⟹ ((*fst s*, *a*, *fst t*) : *ALM-trans 0 id1* & *snd s* = *snd t*)
and
(*a* = *Linearize id1 h* | *a* = *Abort id1* | *a* = *Initialize id1 h* | *a* = *Switch c id2 h r* | (*a* = *Commit c id′ h* & *id1* <= *id′* & *id′* < *id2*)) ⟹ (*fst s* = *fst t* & (*snd s*, *a*, *snd t*) : *ALM-trans id1 id2*)
and
(*a* = *Switch c id1 h r* | *a* = *Invoke c r*) ⟹ ((*fst s*, *a*, *fst t*) : *ALM-trans 0 id1* & (*snd s*, *a*, *snd t*) : *ALM-trans id1 id2*)
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* ≠ *0*; *id1* < *id2*; *s* −*a*−−*composeALMs id1 id2*−> *t*; [|*in-trans-cases-fun id1 id2 s t*|] ==> *P*|] ==> *P* **by** (*auto intro*: *composeALMsE*[**where** *s*=*s* **and** *t*=*t* **and** *a*=*a*])
lemma *my-rule2*:[|*0* < *id1*; *id1* < *id2*; *s* −*a*−−*composeALMs id1 id2*−> *t*; [|*in-trans-cases-fun id1 id2 s t*|] ==> *P*|] ==> *P* **by** (*auto intro*: *composeALMsE*[**where**

s=s **and** t=t **and** a=a])

4.2 Invariants of a single ALM instance

definition *P1a* :: (*ALM-state* $*$ *ALM-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*} --> *request-snd* (*pending s1 c*) = *c*

definition *P1b* :: (*ALM-state* $*$ *ALM-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* --> *request-snd* (*pending s2 c*) = *c*

definition *P2* :: (*ALM-state* $*$ *ALM-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-state* $*$ *ALM-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-state* $*$ *ALM-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-state* $*$ *ALM-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-state* $*$ *ALM-state*) $\Rightarrow$ *bool*
— Non-interference accross instances
where
P6 == % *s* . *let s1* = *fst s*; *s2* = *snd s in*
(~ *aborted s1* --> (*ALL c . phase s2 c* = *Sleep*)) & (*ALL c . phase s1 c* ~= *Aborted* = (*phase s2 c* = *Sleep*))

definition *P7* :: (*ALM-state* $*$ *ALM-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
$P7 == \%\ s\ .\ let\ s1 = fst\ s;\ s2 = snd\ s\ in$
$ALL\ c\ .\ phase\ s1\ c = Aborted \wedge \neg\ initialized\ s2 \longrightarrow (pending\ s2\ c = pending\ s1\ c \wedge phase\ s2\ c \in \{Pending,\ Aborted\})$

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

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

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

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

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 buit from appending a set of request pending in ALM 1 to the history of ALM 1
$P12 == \%\ s\ .\ let\ s1 = fst\ s;\ s2 = snd\ s\ in$
$(\exists\ c\ .\ phase\ s2\ c \neq Sleep) \longrightarrow (\exists\ rs\ .\ l\text{-}c\text{-}p\ (initHists\ s2) = rs\ @\ (hist\ s1) \wedge set\ rs \subseteq pendingReqs\ s1 \wedge distinct\ rs)$

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

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

definition $P14 :: (\mathit{ALM\text{-}state} * \mathit{ALM\text{-}state}) \Rightarrow \mathit{bool}$
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
$P14 == \%\ s\ .\ \mathit{let}\ s1 = \mathit{fst}\ s;\ s2 = \mathit{snd}\ s\ \mathit{in}$
$(\mathit{hist}\ s2 \neq [] \vee \mathit{initialized}\ s2) \longrightarrow (\exists\ rs\ .$
$\mathit{hist}\ s2 = rs\ @\ (\mathit{hist}\ s1)$
$\wedge\ \mathit{set}\ rs \cap \mathit{set}\ (\mathit{hist}\ s1) = \{\})$

definition $P15 :: (\mathit{ALM\text{-}state} * \mathit{ALM\text{-}state}) \Rightarrow \mathit{bool}$
where
— A client that hasn't yet invoked ALM 2 has no request commited in ALM 2 except for its pending request
$P15 == \%\ s\ .\ \mathit{let}\ s1 = \mathit{fst}\ s;\ s2 = \mathit{snd}\ s\ \mathit{in}$
$\forall\ r\ .\ \mathit{let}\ c = \mathit{request\text{-}snd}\ r\ \mathit{in}\ \mathit{phase}\ s2\ c = \mathit{Sleep} \wedge r \in \mathit{set}\ (\mathit{hist}\ s2) \longrightarrow (r \in \mathit{set}\ (\mathit{hist}\ s1) \vee r \in \mathit{pendingReqs}\ s1)$

4.4 Proofs of invariance

lemma *invariant-imp*: $[\![\mathit{invariant}\ \mathit{ioa}\ P;\ \forall\ s\ .\ P\ s \longrightarrow Q\ s]\!] \Longrightarrow \mathit{invariant}\ \mathit{ioa}\ Q$
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*: $\forall\ ys\ .\ xs = ys\ @\ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys \longrightarrow dropWhile\ (\lambda\ x'\ .\ x' \neq x)\ xs = zs$
— A useful lemma about truncating histories
proof (*induct xs*, *force*)
fix *a xs*
assume $\forall\ ys\ .\ xs = ys\ @\ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys \longrightarrow dropWhile\ (\lambda x'.\ x' \neq x)\ xs = zs$
show $\forall\ ys\ .\ a\ \#\ xs = ys\ @\ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys \longrightarrow dropWhile\ (\lambda x'.\ x' \neq x)\ (a\ \#\ xs) = zs$
proof (*rule allI*, *rule impI*, *cases a = x*)
fix *ys*
assume $a\ \#\ xs = ys\ @\ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys$ **and** $a = x$
hence $x\ \#\ xs = ys\ @\ zs$ **and** $x \notin set\ ys$ **and** $hd\ zs = x$ **and** $zs \neq []$ **by** *auto*
from $\langle x\ \#\ xs = ys\ @\ zs\rangle$ **and** $\langle x \notin set\ ys\rangle$ **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** $dropWhile\ (\lambda x'.\ x' \neq x)\ (a\ \#\ xs) = zs$ **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 ⟨$a \# xs = ys @ zs$⟩ **and** ⟨$hd\ zs = x$⟩ **and** ⟨$a \neq x$⟩ **obtain** ys' **where** $ys = a \# ys'$ **apply** *clarify* **by** (*metis Cons-eq-append-conv list.sel*(*1*))
moreover with ⟨$x \notin set\ ys$⟩ **have** $x \notin set\ ys'$ **by** *auto*
moreover from ⟨$ys = a \# ys'$⟩ **and** ⟨$a \# xs = ys @ zs$⟩ **have** $xs = ys' @ zs$ **by** *auto*
ultimately show $(\bigwedge ys'. [\![xs = ys' @ zs; x \notin set\ ys']\!] \Longrightarrow thesis) \Longrightarrow thesis$ **by** *auto*
qed
with ⟨$\forall\ ys . xs = ys @ zs \wedge hd\ zs = x \wedge zs \neq [] \wedge x \notin set\ ys \longrightarrow dropWhile\ (\lambda x'. x' \neq x)\ xs = zs$⟩ **and** ⟨$hd\ zs = x$⟩ **and** ⟨$zs \neq []$⟩ **have** $dropWhile\ (\lambda x'. x' \neq x)\ xs = zs$ **by** *auto*
with ⟨$a \neq x$⟩ **show** $dropWhile\ (\lambda x'. x' \neq x)\ (a \# xs) = zs$ **by** *auto*
qed
qed

lemma *P2-invariant*: $[|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P2$
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume $(s1, s2) : starts\text{-}of\ (composeALMs\ id1\ id2)$ **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** $in\text{-}trans\text{-}comp{:}(s1, s2)\ {-act--composeALMs\ id1\ id2->}\ (s1', s2')$
from ⟨$0 < id1$⟩ **and** ⟨$id1 < id2$⟩ **and** *in-trans-comp* **show** $P2\ (s1', s2')$
proof (*rule my-rule2*)
assume $in\text{-}trans\text{-}cases\text{-}fun\ id1\ id2\ (s1, s2)\ (s1', s2')$
thus $P2\ (s1', s2')$ **using** ⟨$P2\ (s1, s2)$⟩ **and** ⟨$0 < id1$⟩ **and** ⟨$id1 < id2$⟩ **apply**(*auto simp add*: *in-trans-cases-fun-def*) **apply** (*auto simp add*: *ALM-trans-def P2-def*) **done**
qed
qed

lemma *P5-invariant*: $[|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P5$
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume $(s1, s2) : starts\text{-}of\ (composeALMs\ id1\ id2)$ **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 ‹*0* < *id1*› **and** ‹*id1* < *id2*› **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** ‹*P5* (*s1*, *s2*)› **and** ‹*0* < *id1*› **and** ‹*id1* < *id2*› **apply**(*auto simp add*: *in-trans-cases-fun-def*) **apply** (*auto simp add*: *ALM-trans-def P5-def*) **done**
qed
qed

lemma *P6-invariant*: [|*id1* ≠ *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* ≠ *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* ≠ *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** ‹*P6 s*› **and** ‹*id1* ≠ *0*› **and** ‹*id1* < *id2*› **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*) **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*) **done**
qed
qed

lemma *P9-invariant*: [|*id1* < *id2*; *id1* ≠ *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*⟩ **apply**(*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'*) $\in$ *ALM-trans 0 id1* **and** (*s2*, *Switch ca id1 h ra*, *s2'*) $\in$ *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'*) $\in$ *ALM-trans id1 id2* **and** $id1 \leq id'$ **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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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'*) $\in$ *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*, *Abort 0*, *s1'*) ∈ *ALM-trans 0 id1*
thus *P7* (*s1'*, *s2*) **by** (*auto simp add*: *ALM-trans-def P7-def*)
qed
qed
qed

lemma *P4-invariant*: [|$id1 < id2$; $id1 \neq 0$|] ==> *invariant* (*composeALMs id1 id2*) *P4*
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume (*s1*, *s2*) : *starts-of* (*composeALMs id1 id2*) **and** $0 < id1$
thus *P4* (*s1*, *s2*) **by** (*simp add*: *starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P4-def*)
next
fix *s1 s2 s1' s2' act*
assume *reachable* (*composeALMs id1 id2*) (*s1*, *s2*) **and** *P4* (*s1*, *s2*) **and** $0 < id1$ **and** $id1 < id2$ **and** *in-trans-comp*:(*s1*, *s2*) −*act*−−*composeALMs id1 id2*−> (*s1'*, *s2'*)
have *P6* (*s1*, *s2*)
proof −
from *in-trans-comp* **and** ⟨*reachable* (*composeALMs id1 id2*) (*s1*, *s2*)⟩ **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** *P4* (*s1'*, *s2'*)
proof (*rule my-rule2*)
assume *in-trans-cases-fun id1 id2* (*s1*, *s2*) (*s1'*, *s2'*)
thus *P4* (*s1'*, *s2'*) **using** ⟨*P4* (*s1*, *s2*)⟩ **and** ⟨$0 < id1$⟩ **and** ⟨$id1 < id2$⟩ **apply**(*auto simp add*: *in-trans-cases-fun-def*) **apply** (*auto simp add*: *ALM-trans-def P4-def*) **done**
qed
qed

lemma *P8-invariant*: [|$id1 < id2$; $id1 \neq 0$|] ==> *invariant* (*composeALMs id1 id2*) *P8*
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume (*s1*, *s2*) : *starts-of* (*composeALMs id1 id2*) **and** $0 < id1$
thus *P8* (*s1*, *s2*) **by** (*simp add*: *starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P8-def*)
next
fix *s1 s2 s1' s2' act*
assume *reachable* (*composeALMs id1 id2*) (*s1*, *s2*) **and** *P8* (*s1*, *s2*) **and** $0 < id1$ **and** $id1 < id2$ **and** *in-trans-comp*:(*s1*, *s2*) −*act*−−*composeALMs id1 id2*−> (*s1'*, *s2'*)
have *P6* (*s1*, *s2*) **and** *P10* (*s1*, *s2*) **and** *P5* (*s1*, *s2*) **and** *P4* (*s1*, *s2*)

proof −
from *in-trans-comp* **and** ‹*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* **and** *P5-invariant* **and** *P4-invariant* **show** *P6* (*s1*, *s2*) **and** *P10* (*s1*, *s2*) **and** *P5* (*s1*, *s2*) **and** *P4* (*s1*, *s2*) **unfolding** *invariant-def* **by** *auto*
qed
from ‹*0* < *id1*› **and** ‹*id1* < *id2*› **and** *in-trans-comp* **show** *P8* (*s1'*, *s2'*)
proof (*rule my-rule2*)
assume *in-trans-cases-fun id1 id2* (*s1*, *s2*) (*s1'*, *s2'*)
thus *P8* (*s1'*, *s2'*) **using** ‹*P8* (*s1*, *s2*)› **and** ‹*0* < *id1*› **and** ‹*id1* < *id2*›
proof (*auto simp add*: *in-trans-cases-fun-def*)
fix *ca ra*
assume *P8* (*s1*, *s2*) **and** *0* < *id1* **and** *id1* < *id2* **and** *in-invoke-1*:(*s1*, *Invoke ca ra*, *s1'*) ∈ *ALM-trans 0 id1* **and** *in-invoke-2*:(*s2*, *Invoke ca ra*, *s2'*) ∈ *ALM-trans id1 id2*
show *P8* (*s1'*, *s2'*)
proof (*cases s1'* = *s1*)
assume *s1'* = *s1*
with *in-invoke-2* **and** ‹*P8* (*s1*, *s2*)› **show** *?thesis* **by** (*auto simp add*: *ALM-trans-def P8-def*)
next
assume *s1'* ≠ *s1*
with *in-invoke-1* **have** *pendingReqs s1* ⊆ *pendingReqs s1'* **by** (*force simp add*:*pendingReqs-def ALM-trans-def*)
moreover from *in-invoke-1* **have** *hist s1'* = *hist s1* **by** (*auto simp add*:*ALM-trans-def*)
moreover from *in-invoke-2* **have** *initHists s2'* = *initHists s2* **by** (*auto simp add*:*ALM-trans-def*)
moreover note ‹*P8* (*s1*, *s2*)›
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*, *Switch ca id1 h ra*, *s1'*) ∈ *ALM-trans 0 id1* **and** *in-switch-2*:(*s2*, *Switch ca id1 h ra*, *s2'*) ∈ *ALM-trans id1 id2*
show *P8* (*s1'*, *s2'*)
proof (*auto simp add*:*P8-def*)
fix *h1*
assume *h1* ∈ *initHists s2'*
show *h1* ∈ *postfix-all* (*hist s1'*) (*linearizations* (*pendingReqs s1'*))
proof (*cases h1* ∈ *initHists s2*)
assume *h1* ∈ *initHists s2*
moreover from *in-switch-1* **and** ‹*0* < *id1*› **have** *hist s1'* = *hist s1* **and** *pendingReqs s1'* = *pendingReqs s1* **by** (*auto simp add*:*ALM-trans-def pendingReqs-def*)

moreover note ‹*P8* (*s1*, *s2*)›
ultimately show *h1* ∈ *postfix-all* (*hist s1'*) (*linearizations* (*pendingReqs s1'*)) **by** (*auto simp add*:*P8-def*)
next
assume *h1* ∉ *initHists s2*
with ‹*h1* ∈ *initHists s2'*› **and** *in-switch-2* **have** *h1* = *h* **by** (*auto simp add*:*ALM-trans-def*)
with *in-switch-1* **and** ‹*0* < *id1*› **and** ‹*P10* (*s1*, *s2*)› **have** *h1* ∈ *postfix-all* (*hist s1*) (*linearizations* (*pendingReqs s1*)) **by** (*auto simp add*:*ALM-trans-def P10-def*)
moreover from *in-switch-1* **and** ‹*0* < *id1*› **have** *hist s1'* = *hist s1* **and** *pendingReqs s1'* = *pendingReqs s1* **by** (*auto simp add*:*ALM-trans-def pendingReqs-def*)
ultimately show *?thesis* **by** *auto*
qed
qed
next
fix *c id' h*
assume *P8* (*s1*, *s2*) **and** *0* < *id1* **and** (*s2*, *Commit c id' h*, *s2'*) ∈ *ALM-trans id1 id2* **and** *id1* ≤ *id'* **and** *id'* < *id2*
thus *P8* (*s1*, *s2'*) **by** (*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*, *Switch c id2 h r*, *s2'*) ∈ *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** (*s2*, *Linearize id1 h*, *s2'*) ∈ *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** (*s2*, *Initialize id1 h*, *s2'*) ∈ *ALM-trans id1 id2*
thus *P8* (*s1*, *s2'*) **by** (*auto simp add*: *ALM-trans-def P8-def*)
next
fix *ca ta ra*
assume *P8* (*s1*, *s2*) **and** *0* < *id1* **and** *id1* < *id2* **and** (*s1*, *Switch ca 0 ta ra*, *s1'*) ∈ *ALM-trans 0 id1*
thus *P8* (*s1'*, *s2*) **using** ‹*P5* (*s1*, *s2*)› **by** (*auto simp add*: *ALM-trans-def P8-def P5-def*)
next
fix *ca id' h*
assume *P8* (*s1*, *s2*) **and** *in-commit-1*:(*s1*, *Commit ca id' h*, *s1'*) ∈ *ALM-trans 0 id1*
from *in-commit-1* **have** *pendingReqs s1'* = *pendingReqs s1* **and** *hist s1'* = *hist s1* **by** (*auto simp add*:*pendingReqs-def ALM-trans-def*)
with ‹*P8* (*s1*, *s2*)› **show** *P8* (*s1'*, *s2*) **by** (*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** ⟨*P6* (*s1*, *s2*)⟩ **and** ⟨*P4* (*s1*, *s2*)⟩ **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** ⟨*P10* (*s1*, *s2*)⟩ **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*|] ==> *invariant* (*composeALMs id1 id2*) *P12*
proof *clarify*
assume *id1* < *id2* **and** *0* < *id1*
with *P8-invariant* **and** *P4-invariant* **have** *invariant* (*composeALMs id1 id2*) (λ (*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 ⟨*P4* (*s1*, *s2*)⟩ **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 ⟨*initHists s2* $\neq$ {}⟩ **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-c-p-common-postfix*[*of initHists s2*] **and** ‹$h \in$ *initHists s2*› **obtain** h'' **where** $h = h''$ @ (*l-c-p* (*initHists s2*)) **by** (*auto simp add*:*common-postfix-p-def suffixeq-def*)
moreover note ‹*l-c-p* (*initHists s2*) = *rs* @ *hist s1*›
ultimately show *?thesis* **by** *auto*
qed
moreover have *distinct rs*
proof –
from ‹*initHists s2* $\neq$ {}› **obtain** *h* **where** $h \in$ *initHists s2* **by** *auto*
with *initHists-prop* **obtain** h' **where** $h = h'$ @ (*hist s1*) **and** *distinct* h' **by** *auto*
with *l-c-p-common-postfix*[*of initHists s2*] **and** ‹$h \in$ *initHists s2*› **and** ‹*l-c-p* (*initHists s2*) = *rs* @ *hist s1*› **obtain** h'' **where** $h' = h''$ @ *rs* **apply** (*auto simp add*:*common-postfix-p-def suffixeq-def*) **by** (*metis* ‹$h = h'$ @ *hist s1*› *append-assoc append-same-eq*)
with ‹*distinct* h'› **show** *?thesis* **by** *auto*
qed
ultimately show $\exists$ *rs*. *l-c-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-imp*)
qed

lemma *P11-invariant*: [|*id1* < *id2*; *id1* $\neq$ *0*|] ==> *invariant* (*composeALMs id1 id2*) *P11*
proof *clarify*
assume *id1* < *id2* **and** *0* < *id1*
with *P8-invariant* **and** *P12-invariant* **and** *P6-invariant* **and** *P7-invariant* **have** *invariant* (*composeALMs id1 id2*) (λ (*s1*, *s2*) . *P8* (*s1*, *s2*) $\wedge$ *P12* (*s1*, *s2*) $\wedge$ *P6* (*s1*, *s2*) $\wedge$ *P7* (*s1*, *s2*)) **by** (*auto simp add*:*invariant-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-def initValidReqs-def*, *auto*)
fix *x c h*
assume *phase s2 c* $\neq$ *Sleep*
with ‹*P12* (*s1*, *s2*)› **and** ‹*P8* (*s1*, *s2*)› **have** *initHists-prop*:$\forall$ $h \in$ *initHists s2* . ($\exists$ h' . $h = h'$ @ (*hist s1*) $\wedge$ *set* $h' \subseteq$ *pendingReqs s1*) **and** *lcp-prop*:$\exists$ *rs* . *l-c-p* (*initHists s2*) = *rs* @ (*hist s1*) **by** (*auto simp add*:*P12-def P8-def postfix-all-def linearizations-def*)
assume $x \notin$ *set* (*l-c-p* (*initHists s2*)) **and** $h \in$ *initHists s2* **and** $x \in$ *set h*
from *initHists-prop* **and** ‹$h \in$ *initHists s2*› **obtain** h' **where** $h = h'$ @ (*hist s1*) **and** *set* $h' \subseteq$ *pendingReqs s1* **by** *auto*
moreover from *lcp-prop* **obtain** *rs* **where** *l-c-p* (*initHists s2*) = *rs* @ (*hist s1*) **by** *auto*
moreover note ‹$x \notin$ *set* (*l-c-p* (*initHists s2*))› **and** ‹$x \in$ *set h*›

ultimately have *x* ∈ *set h′* **by** *auto*
with ⟨*set h′* ⊆ *pendingReqs s1*⟩ **show** *x* ∈ *pendingReqs s1* **by** *auto*
next
fix *x c h*
assume *phase s2 c* ≠ *Sleep* **and** ¬ *initialized s2*
with ⟨*P12* (*s1*, *s2*)⟩ **have** *lcp-prop*:∃ *rs* . *l-c-p* (*initHists s2*) = *rs* @ (*hist s1*) **by** (*auto simp add*:*P12-def P8-def postfix-all-def linearizations-def*)
assume *x* ∉ *set* (*l-c-p* (*initHists s2*)) **and** *x* ∈ *pendingReqs s2*
from ⟨*x* ∉ *set* (*l-c-p* (*initHists s2*))⟩ **and** *lcp-prop* **have** *x* ∉ *set* (*hist s1*) **by** *auto*
moreover obtain *c′* **where** *phase s1 c′* = *Aborted* **and** *x* = *pending s1 c′*
proof −
from ⟨*x* ∈ *pendingReqs s2*⟩ **and** ⟨*P6* (*s1*, *s2*)⟩ **obtain** *c′* **where** *phase s1 c′* = *Aborted* **and** *x* = *pending s2 c′* **by** (*force simp add*:*pendingReqs-def P6-def*)
moreover with ⟨¬ *initialized s2*⟩ **and** ⟨*P7* (*s1*, *s2*)⟩ **have** *x* = *pending s1 c′* **by** (*auto simp add*:*P7-def*)
ultimately show (⋀*c′*. ⟦*phase s1 c′* = *Aborted*; *x* = *pending s1 c′*⟧ ⟹ *thesis*) ⟹ *thesis* **by** *auto*
qed
ultimately show *x* ∈ *pendingReqs s1* **by** (*auto simp add*:*pendingReqs-def*)
qed
qed
ultimately show *?thesis* **by** (*auto intro*:*invariant-imp*)
qed

lemma *P1a-invariant*: [|*id1* < *id2*; *id1* ≠ *0*|] ==> *invariant* (*composeALMs id1 id2*) *P1a*
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume (*s1*, *s2*) : *starts-of* (*composeALMs id1 id2*) **and** *0* < *id1*
thus *P1a* (*s1*, *s2*) **by** (*simp add*: *starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1a-def*)
next
fix *s1 s2 s1′ s2′ act*
assume *reachable* (*composeALMs id1 id2*) (*s1*, *s2*) **and** *P1a* (*s1*, *s2*) **and** *0* < *id1* **and** *id1* < *id2* **and** *in-trans-comp*:(*s1*, *s2*) −*act*−−*composeALMs id1 id2*−> (*s1′*, *s2′*)
have *P5* (*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** *P5-invariant* **show** *P5* (*s1*, *s2*) **unfolding** *invariant-def* **by** *auto*
qed
from ⟨*0* < *id1*⟩ **and** ⟨*id1* < *id2*⟩ **and** *in-trans-comp* **show** *P1a* (*s1′*, *s2′*)
proof (*rule my-rule2*)
assume *in-trans-cases-fun id1 id2* (*s1*, *s2*) (*s1′*, *s2′*)
thus *P1a* (*s1′*, *s2′*) **using** ⟨*P1a* (*s1*, *s2*)⟩ **and** ⟨*P5* (*s1*, *s2*)⟩ **and** ⟨*0* < *id1*⟩ **and** ⟨*id1* < *id2*⟩ **apply**(*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*: $[|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P1b$
proof (*rule invariantI*, *auto*)
  fix *s1 s2*
  assume $(s1, s2)$: *starts-of* (*composeALMs id1 id2*) **and** $0 < id1$
  thus *P1b* (*s1*, *s2*) **by** (*simp add*: *starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1b-def*)
next
  fix *s1 s2 s1' s2' act*
  assume *reachable* (*composeALMs id1 id2*) (*s1*, *s2*) **and** *P1b* (*s1*, *s2*) **and** $0 < id1$ **and** $id1 < id2$ **and** *in-trans-comp*:(*s1*, *s2*) −*act*−−*composeALMs id1 id2*−> (*s1'*, *s2'*)
  have *P1a* (*s1*, *s2*)
  proof −
    from *in-trans-comp* **and** ‹*reachable* (*composeALMs id1 id2*) (*s1*, *s2*)› **have** *reachable* (*composeALMs id1 id2*) (*s1'*, *s2'*) **by** (*auto intro*: *reachable.reachable-n*)
    with ‹*reachable* (*composeALMs id1 id2*) (*s1*, *s2*)› **and** ‹$0 < id1$› **and** ‹$id1 < id2$› **and** *P1a-invariant* **show** *P1a* (*s1*, *s2*) **unfolding** *invariant-def* **by** *auto*
  qed
  from ‹$0 < id1$› **and** ‹$id1 < id2$› **and** *in-trans-comp* **show** *P1b* (*s1'*, *s2'*)
  proof (*rule my-rule2*)
    assume *in-trans-cases-fun id1 id2* (*s1*, *s2*) (*s1'*, *s2'*)
    thus *P1b* (*s1'*, *s2'*) **using** ‹*P1b* (*s1*, *s2*)› **and** ‹*P1a* (*s1*, *s2*)› **and** ‹$0 < id1$› **and** ‹$id1 < id2$› **apply**(*auto simp add*: *in-trans-cases-fun-def*) **apply** (*auto simp add*: *ALM-trans-def P1b-def P1a-def*) **done**
  qed
qed

lemma *P13-invariant*: $[|id1 < id2; id1 \neq 0|] ==> invariant\ (composeALMs\ id1\ id2)\ P13$
proof *clarify*
  assume $id1 < id2$ **and** $0 < id1$
  with *P11-invariant* **and** *P12-invariant* **have** *invariant* (*composeALMs id1 id2*) (λ (*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 ‹*P12* (*s1*, *s2*)› **and** ‹*P11* (*s1*, *s2*)› **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 ‹*l-c-p* (*initHists s2*) = *rs* @ (*hist s1*)› **have** *initValidReqs s2* ∩ *set rs* = {} **by** (*auto simp add*:*initValidReqs-def*)
ultimately show *postfix-all* (*l-c-p* (*initHists s2*)) (*linearizations* (*initValidReqs s2*)) ⊆ *postfix-all* (*hist s1*) (*linearizations* (*pendingReqs s1*)) **by** (*force simp add*: *postfix-all-def linearizations-def*)
qed
qed
ultimately show *?thesis* **by** (*auto intro*:*invariant-imp*)
qed

lemma *P14-invariant*: [|*id1* < *id2*; *id1* ≠ *0*|] ==> *invariant* (*composeALMs id1 id2*) *P14*
proof (*rule invariantI*, *auto*)
fix *s1 s2*
assume (*s1*, *s2*) : *starts-of* (*composeALMs id1 id2*) **and** *0* < *id1*
thus *P14* (*s1*, *s2*) **by** (*simp add*: *starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P14-def*)
next
fix *s1 s2 s1′ s2′ act*
assume *reachable* (*composeALMs id1 id2*) (*s1*, *s2*) **and** *P14* (*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** *P13* (*s1*, *s2*) **and** *P10* (*s1*, *s2*) **and** *P2* (*s1*, *s2*) **and** *P4* (*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** *P13-invariant* **and** *P10-invariant* **and** *P4-invariant* **and** *P2-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 ‹*0* < *id1*› **and** ‹*id1* < *id2*› **and** *in-trans-comp* **show** *P14* (*s1′*, *s2′*)
proof (*rule my-rule2*)
assume *in-trans-cases-fun id1 id2* (*s1*, *s2*) (*s1′*, *s2′*)
thus *P14* (*s1′*, *s2′*) **using** ‹*P14* (*s1*, *s2*)› **and** ‹*0* < *id1*› **and** ‹*id1* < *id2*›
proof (*auto simp add*: *in-trans-cases-fun-def*)
fix *ca ra*
assume *P14* (*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 *P14* (*s1′*, *s2′*) **by** (*auto simp add*: *ALM-trans-def P14-def*)
next
fix *ca h ra*
assume *P14* (*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 *P14* (*s1′*, *s2′*) **by** (*auto simp add*: *ALM-trans-def P14-def*)
next
fix *c id′ h*

assume *P14* (*s1*, *s2*) **and** *0* < *id1* **and** (*s2*, *Commit c id′ h*, *s2′*) ∈ *ALM-trans id1 id2* **and** *id1* ≤ *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*, *Switch c id2 h r*, *s2′*) ∈ *ALM-trans id1 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*, *Linearize id1 h*, *s2′*) ∈ *ALM-trans id1 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*, *Initialize id1 h*, *s2′*) ∈ *ALM-trans id1 id2*
thus *P14* (*s1*, *s2′*) **using** ‹*P13* (*s1*, *s2*)› **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*, *Abort id1*, *s2′*) ∈ *ALM-trans id1 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*, *Switch ca 0 ta ra*, *s1′*) ∈ *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*, *Commit ca id′ h*, *s1′*) ∈ *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** *in-lin*:(*s1*, *Linearize 0 h*, *s1′*) ∈ *ALM-trans 0 id1*
from *in-lin* **have** ¬ *initialized s2* **and** *hist s2* = [] **using** ‹*P6* (*s1*, *s2*)› **and** ‹*P2* (*s1*, *s2*)› **and** ‹*P10* (*s1*, *s2*)› **and** ‹*P2* (*s1*, *s2*)› **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*, *Initialize 0 h*, *s1′*) ∈ *ALM-trans 0 id1*
thus *P14* (*s1′*, *s2*) **using** ‹*P10* (*s1*, *s2*)›**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′*) ∈ *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* ≠ *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** ‹*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** *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 ‹*0* < *id1*› **and** ‹*id1* < *id2*› **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** ‹*P15* (*s1*, *s2*)› **and** ‹*0* < *id1*› **and** ‹*id1* < *id2*›
proof (*auto simp add*: *in-trans-cases-fun-def*)
fix *ca ra*
assume *P15* (*s1*, *s2*) **and** *in-invoke1*:(*s1*, *Invoke ca ra*, *s1′*) ∈ *ALM-trans 0 id1* **and** *in-invoke2*:(*s2*, *Invoke ca ra*, *s2′*) ∈ *ALM-trans id1 id2*
show *P15* (*s1′*, *s2′*)
proof −
{ assume *s1′* = *s1*
with ‹*P15* (*s1*, *s2*)› **and** *in-invoke1* **and** *in-invoke2* **and** ‹*0* < *id1*› **and** ‹*id1* < *id2*›
have *?thesis* **by** (*auto simp add*:*ALM-trans-def P15-def*)
} note *case1* = *this*
{ assume *s1′* ≠ *s1*
with *in-invoke1* **and** *in-invoke2* **and** ‹*P6* (*s1*, *s2*)› **have** *s2′* = *s2* **apply** (*auto simp add*:*ALM-trans-def P6-def*) **by** (*metis phase.simps*(*12*) *phase.simps*(*4*))

with ‹*s1′* ≠ *s1*› **and** ‹*P15* (*s1*, *s2*)› **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*, *Switch ca id1 h ra*, *s1′*) ∈ *ALM-trans 0 id1* **and** (*s2*, *Switch ca id1 h ra*, *s2′*) ∈ *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*, *Commit c id′ h*, *s2′*) ∈ *ALM-trans id1 id2* **and** *id1* ≤ *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*, *Switch c id2 h r*, *s2′*) ∈ *ALM-trans id1 id2*
thus *P15* (*s1*, *s2′*) **by** (*auto simp add*: *ALM-trans-def P15-def*)
next
fix *h*
assume *in-lin*:(*s2*, *Linearize id1 h*, *s2′*) ∈ *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 ‹*phase s2′* (*request-snd r*) = *Sleep*› **and** *in-lin* **have** *phase s2* (*request-snd r*) = *Sleep* **by** (*auto simp add*:*ALM-trans-def*)
with ‹*P1b* (*s1*, *s2*)› **have** *r* ∉ *pendingReqs s2* **by** (*auto simp add*:*pendingReqs-def P1b-def*)
with *in-lin* **and** ‹*r* ∈ *set* (*hist s2′*)› **have** *r* ∈ *set* (*hist s2*) **by** (*auto simp add*:*ALM-trans-def postfix-all-def linearizations-def*)
with ‹*phase s2* (*request-snd r*) = *Sleep*› **and** ‹*P15* (*s1*, *s2*)› **and** ‹*r* ∉ *pendingReqs s1*› **show** *?thesis* **by** (*auto simp add*:*P15-def*)
qed
qed
next
assume *P15* (*s1*, *s2*) **and** *0* < *id1* **and** *id1* < *id2* **and** (*s2*, *Abort id1*, *s2′*) ∈ *ALM-trans id1 id2*
thus *P15* (*s1*, *s2′*) **by** (*auto simp add*: *ALM-trans-def P15-def*)
next
fix *h*
assume *in-init*:(*s2*, *Initialize id1 h*, *s2′*) ∈ *ALM-trans id1 id2*
show *P15* (*s1*, *s2′*)

proof (*auto simp add*:*P15-def*)
fix *r*
assume *phase s2′* (*request-snd r*) = *Sleep* **and** $r \in$ *set* (*hist s2′*) **and** $r \notin$ *pendingReqs s1*
show $r \in$ *set* (*hist s1*)
proof −
from *in-init* **and** ‹*P13* (*s1*, *s2*)›
have *hist s2′* $\in$ *postfix-all* (*hist s1*) (*linearizations* (*pendingReqs s1*)) **by** (*auto simp add*:*ALM-trans-def P13-def*)
with ‹$r \in$ *set* (*hist s2′*)› **have** $r \in$ *set* (*hist s1*) $\lor$ $r \in$ *pendingReqs s1* **by** (*auto simp add*: *postfix-all-def linearizations-def*)
with ‹$r \notin$ *pendingReqs s1*› **show** *?thesis* **by** *auto*
qed
qed
next
fix *ca ta ra*
assume (*s1*, *Switch ca 0 ta ra*, *s1′*) $\in$ *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′*) $\in$ *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′*) $\in$ *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′*) $\in$ *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′*) $\in$ *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* = λ*c*. (*if phase s1 c* $\neq$ *Aborted then pending s1 c else pending s2 c*),
    initHists = {},
    phase = λ*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 ⟨*id1* $\neq$ *0*⟩ **and** ⟨*id1* < *id2*⟩ **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 ⟨*id1* $\neq$ *0*⟩ **and** ⟨*id1* < *id2*⟩
    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** ⟨*id1* $\neq$ *0*⟩ **and** ⟨*id1* < *id2*⟩ **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* ‹*id1* ≠ *0*› **and** ‹*id1* < *id2*› **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* | (*EX t* . *act* = *Switch c 0 t r*)} *Un* {*act* . *EX c tr* . (*EX id′* . *0* <= *id′* & *id′* < *id2* & *act* = *Commit c id′ tr*) | (*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-trace* (*ALM-ioa 0 id2*)$(*?ex*) = [*Invoke c 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 –
{
assume *s1'* ≠ *s1* & *s2'* ≠ *s2*
— contradiction
with *in-invoke* **and** ‹*id1* ≠ *0*› **and** ‹*id1* < *id2*› **and** ‹*P6* (*s1'*, *s2'*)› **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**(*auto simp add*:*ALM-trans-def P6-def*) **done**
}
moreover
{
assume *s1'* = *s1* **and** *s2'* = *s2*
with *in-invoke* **have** *pre-s1*:~(*phase s1 c* = *Ready* & *request-snd r* = *c* & *r* ∉ *set* (*hist s1*)) **and** *pre-s2*:~(*phase s2 c* = *Ready* & *request-snd r* = *c* & *r* ∉ *set* (*hist s2*)) **using** [[*hypsubst-thin*]] **apply** (*auto simp add*: *is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def*) **apply**(*simp-all add*:*ALM-trans-def*) **apply** (*drule-tac*[!] *arg-cong*[**where** *f* = *phase*]) **apply** *simp-all* **apply** (*metis phase.simps*(*8*) *fun-upd-idem-iff*) **apply** (*metis phase.simps*(*8*) *fun-upd-idem-iff*) **apply** (*metis phase.simps*(*8*) *fun-upd-idem-iff*) **apply** (*metis phase.simps*(*8*) *fun-upd-idem-iff*) **done**
hence ~(*phase ?t c* = *Ready* & *request-snd r* = *c* & *r* ∉ *set* (*hist ?t*)) **using** ‹*P14* (*s1*, *s2*)› **by** (*auto simp add*:*ref-mapping-def P14-def*)
hence *?thesis* **using** ‹*id1* ≠ *0*› **and** ‹*s1'* = *s1*› **and** ‹*s2'* = *s2*› **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-all add*:*ALM-trans-def*) **apply** *force* **done**
}
moreover
{
assume *s1'* ≠ *s1* **and** *s2'* = *s2*
with *in-invoke* **have** *pre-s1*:*phase s1 c* = *Ready* & *request-snd r* = *c* & *r* ∉ *set* (*hist s1*) **and** *trans-s1*: *s1'* = *s1*⦇*pending* := (*pending s1*)(*c* := *r*), *phase* := (*phase s1*)(*c* := *Pending*)⦈ **apply** (*simp-all add*: *is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-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**
have *pre-t*: *phase ?t c* = *Ready* & *request-snd r* = *c* & *r* ∉ *set* (*hist ?t*)
proof –
from *pre-s1* **have** *phase ?t c* = *Ready* & *request-snd r* = *c* **by** (*auto simp add*:*ref-mapping-def*)
moreover have *r* ∉ *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-trace* (*ALM-ioa 0 id2*)$(*?ex*) = [*Switch c id2 h r*!] **by** (*simp add*: *mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def*)

moreover have *is-exec-frag* (*ALM-ioa 0 id2*) (*?t*, *?ex*)
proof −
from *in-switch* **and** ‹*id1* < *id2*› **have** *s1′* = *s1* **apply** (*simp-all add*: *composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def*) **done**
from ‹*id1* ≠ *0*› **and** ‹*id1* < *id2*› *in-switch* **have** *pre-s2*:*aborted s2* & *phase s2 c* = *Pending* & *r* = *pending s2 c* & (*if initialized s2 then* (*h* ∈ *postfix-all* (*hist s2*) (*linearizations* (*pendingReqs s2*))) *else* (*h* : *postfix-all* (*l-c-p* (*initHists s2*)) (*linearizations* (*initValidReqs s2*)))) **and** *trans-s2*: *s2′* = *s2*⦇*phase* := (*phase s2*)(*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**
from *pre-s2* **have** *s1-aborted*:*phase s1 c* = *Aborted* **using** ‹*P6* (*s1*, *s2*)› **apply**(*auto simp add*: *P6-def*) **done**
have *pre-t*:*aborted ?t* & *phase ?t c* = *Pending* & *initialized ?t* & *h* : *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*)) & *r* = *pending ?t c*
proof −
from *s1-aborted* **and** *pre-s2* **have** *aborted ?t* & *pending ?t c* = *r* **and** *phase ?t c* = *Pending* **and** *initialized ?t* **by** (*auto simp add*: *ref-mapping-def fun-eq-iff*)
moreover have *h* : *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))
proof −
from *pre-s2* **have** (*if initialized s2 then* (*h* : *postfix-all* (*hist s2*) (*linearizations* (*pendingReqs s2*))) *else* (*h* : *postfix-all* (*l-c-p* (*initHists s2*)) (*linearizations* (*initValidReqs s2*)))) **by** *auto*
thus *?thesis*
proof *auto*
assume *case1-1*:*initialized s2* **and** *case1-2*:*h* : *postfix-all* (*hist s2*) (*linearizations* (*pendingReqs s2*))
hence *suffixeq* (*hist s1*) (*hist s2*) **using** ‹*P14* (*s1*, *s2*)› **by** (*auto simp add*:*P14-def suffixeq-def*)
show *h* ∈ *postfix-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 ‹*hist s2* = []› **and** ‹*suffixeq* (*hist s1*) (*hist s2*)› **have** *hist s1* = [] **by** (*auto simp add*:*suffixeq-def*)
with ‹*hist s2* = []› **show** *hist ?t* = *hist s2* **by** (*auto simp add*: *ref-mapping-def*)
qed
next

assume *hist s2* $\neq$ []
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* $\notin$ *set* (*hist s2*) **and** *phase s2 c* = *Pending* $\vee$ *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 $\exists$ *ca*. *pending s2 c* = *pending ?t ca* $\wedge$ *pending s2 c* $\notin$ *set* (*hist ?t*) $\wedge$ (*phase ?t ca* = *Pending* $\vee$ *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*:$\neg$ *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* $\neq$ *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* $\in$ *pendingReqs ?t*
with *this* **obtain** *c'* **where** *r* = *pending ?t c'* **and** *r* $\notin$ *set* (*hist ?t*) **and** *phase ?t c'* $\in$ {*Pending*, *Aborted*} **by** (*auto simp add*:*pendingReqs-def*)
show *r* $\in$ *pendingReqs s1*
proof (*cases phase s1 c'* = *Aborted*)
assume *phase s1 c'* = *Aborted*
with ‹*phase ?t c'* $\in$ {*Pending*, *Aborted*}› **and** ‹*r* = *pending ?t c'*› **have** *phase s2 c'* $\in$ {*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* $\notin$ *set* (*hist ?t*)› **have** *phase s1 c'* = *Aborted* **and** *r* = *pending s1 c'* **and** *r* $\notin$ *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* $c' \neq$ *Aborted*
with ‹r = *pending ?t* c'› **and** ‹$r \notin$ *set* (*hist ?t*)› **and** ‹*phase ?t* $c' \in$ {*Pending*, *Aborted*}› **and** ‹*hist ?t* = *hist s1*› **show** *?thesis* **by** (*auto simp add*:*ref-mapping-def pendingReqs-def*)
qed
next
fix r
assume $r \in$ *pendingReqs s1*
with *this* **obtain** c **where** r = *pending s1 c* **and** *phase s1 c* $\in$ {*Pending*, *Aborted*} **and** $r \notin$ *set* (*hist s1*) **by** (*auto simp add*:*pendingReqs-def*)
with ‹*hist s2* = []› **and** ‹$\neg$ *initialized s2*› **and** ‹*P7* (*s1*, *s2*)› **show** $r \in$ *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*:$s1' = s1\ (|phase := (phase\ s1)(c := Ready)|)$ **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-s1* **have** *s1-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** ‹$phase\ s1\ c \neq Aborted$› **show** *?thesis* **by** (*auto simp add*: *ref-mapping-def*)

next

assume $hist\ s2 \neq []$

hence *initialized s2* **using** ‹$P10\ (s1, s2)$› **by** (*auto simp add*:*P10-def*)

from *pre-s1* **and** ‹$phase\ s1\ c \neq Aborted$› **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 ‹*initialized s2*› **and** ‹$P14\ (s1, s2)$› **obtain** *rs3* **where** $hist\ s2 = rs3\ @\ (hist\ s1)$ **by** (*auto simp add*:*P14-def*)

with ‹$pending\ s1\ c \in set\ (hist\ s1)$› **and** ‹$hist\ s2 = rs3\ @\ (hist\ s1)$› **and** ‹$pending\ ?t\ c = pending\ s1\ c$› **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 ‹$pending\ s1\ c \in set\ (hist\ s1)$› **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 ‹$pending\ ?t\ c = pending\ s1\ c$› **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 ‹*initialized s2*› **and** ‹$P14\ (s1, s2)$› **obtain** *rs3* **where** $hist\ s2 = rs3\ @\ (hist\ s1)$ **and** $set\ rs3 \cap set\ (hist\ s1) = \{\}$ **by** (*auto simp add*:*P14-def*)

with ‹$pending\ s1\ c \in set\ (hist\ s1)$› **and** ‹$hist\ s1 = rs2\ @\ rs1$› **have** $hist\ s2 = rs3\ @\ rs2\ @\ rs1$ **and** $pending\ s1\ c \notin set\ rs3$ **by** *auto*

with ‹$pending\ s1\ c \notin set\ rs2$› **obtain** *rs4* **where** $hist\ s2 = rs4\ @\ rs1$ **and** $pending\ s1\ c \notin set\ rs4$ **by** *auto*

with ‹$hd\ rs1 = pending\ s1\ c$› **and** ‹$rs1 \neq []$› **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** ‹$hist\ s2 \neq []$› **and** ‹$pending\ ?t\ c = pending$

s1 c⟩ **by** (*auto simp add*:*ref-mapping-def*)
qed
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
moreover from ⟨*s2′ = s2*⟩ **and** *s1-not-aborted-c* **and** *trans-s1* **have** *trans-t*:*?t′ = ?t* (|*phase* := (*phase ?t*)(*c* := *Ready*)|) **by** (*simp add*:*fun-eq-iff ref-mapping-def*)
ultimately have *?thesis* **using** ⟨*id1 < id2*⟩ **apply** (*simp add*: *is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def*) **apply**(*simp add*:*ALM-trans-def*) **done**
}
moreover
{
assume *id1* ≤ *id′*
with *in-commit* **have** *s1′ = s1* **and** *pre-s2*:*phase s2 c = Pending* ∧ *pending s2 c* ∈ *set* (*hist s2*) ∧ *h = dropWhile* (λ *r* . *r* ≠ *pending s2 c*) (*hist s2*) **and** *trans-s2*:*s2′ = s2* (|*phase* := (*phase s2*)(*c* := *Ready*)|) **apply** (*simp-all add*: *composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def*) **apply**(*auto simp add*:*ALM-trans-def*) **done**
from *pre-s2* **and** ⟨*P6* (*s1*, *s2*)⟩ **have** *facts*:*aborted s1* & *phase s1 c = Aborted* & *hist s2* ≠ [] **by** (*force simp add*:*P6-def*)
with *pre-s2* **have** *pre-t*:*phase ?t c = Pending* ∧ *pending ?t c* ∈ *set* (*hist ?t*) ∧ *h = dropWhile* (λ *r* . *r* ≠ *pending ?t c*) (*hist ?t*) **by** (*auto simp add*:*ref-mapping-def*)
moreover from ⟨*s1′ = s1*⟩ **and** *facts* **and** *trans-s2* **have** *trans-t*:*?t′ = ?t* (|*phase* := (*phase ?t*)(*c* := *Ready*)|) **by** (*auto simp add*:*fun-eq-iff ref-mapping-def*)
ultimately have *?thesis* **using** ⟨*id1 < id2*⟩ **apply** (*simp add*: *is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def*) **apply**(*simp add*:*ALM-trans-def*) **done**
}
ultimately show *?thesis* **using** ⟨*id′ < id2*⟩ **by** *force*
qed
ultimately show *?thesis* **by** (*auto intro*: *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* ∉ *ext* (*ALM-ioa 0 id2*)
— Now the case when the composition takes an action that is not in the external signature of the spec
with *in-trans-comp* **and** ⟨*id1 < id2*⟩ **and** ⟨*id1* ≠ *0*⟩ **have** *act* : {*act* . *act = Abort 0* | *act = Abort id1* | (*EX c r h* . *act = Linearize 0 h* | *act =*

Linearize id1 h | *act* = *Switch c id1 h r* | *act* = *Initialize 0 h* | *act* = *Initialize id1 h*)} **by** (*auto simp add*: *composeALMs-def hide-def hide-asig-def ALM-ioa-def ALM-asig-def externals-def asig-inputs-def asig-outputs-def asig-internals-def asig-of-def trans-of-def par-def actions-def*)
with *in-trans-comp* **show** ∃ *ex*. *is-exec-frag* (*ALM-ioa 0 id2*) (*?t*, *ex*) ∧ *Finite ex* ∧ *laststate* (*?t*, *ex*) = *?t′* ∧ *mk-trace* (*ALM-ioa 0 id2*)·*ex* = *nil*
proof *auto*
assume *in-abort*:(*s1*, *s2*) −*Abort 0*−*composeALMs id1 id2*⟶ (*s1′*, *s2′*)
— The case where the first Abastract aborts
moreover **with** ‹*id1* ≠ *0*› **and** ‹*id1* < *id2*› **and** ‹*P6* (*s1*, *s2*)› **and** ‹*P2* (*s1*, *s2*)› **have** ∀ *c* . *phase s1 c* ≠ *Aborted* **and** *hist s2* = [] **and** ∀ *c* . *phase s2 c* = *Sleep*
apply (*simp-all add*: *composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def*)
apply(*auto simp add*:*fun-eq-iff ALM-trans-def ref-mapping-def P6-def P2-def*) **done**
moreover **note** ‹*id1* ≠ *0*›
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*)·*?ex* = *nil* **using** ‹*id1* < *id2*› **by** (*simp add*: *mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def*)
moreover **have** *is-exec-frag* (*ALM-ioa 0 id2*) (*?t*, *?ex*) **by** (*auto simp add*:*is-exec-frag-def*)
ultimately **show** ∃ *ex*. *is-exec-frag* (*ALM-ioa 0 id2*) (*?t*, *ex*) ∧ *Finite ex* ∧ *laststate* (*?t*, *ex*) = *?t* ∧ *mk-trace* (*ALM-ioa 0 id2*)·*ex* = *nil* **by** (*auto intro*: *exI*[**where** *x*=*?ex*])
qed
next
assume *in-abort*:(*s1*, *s2*) −*Abort id1*−*composeALMs id1 id2*⟶ (*s1′*, *s2′*)
— The case where the second ALM aborts
show *?thesis*
proof −
let *?ex* = [(*Abort 0*, *?t′*)!]
have *Finite ?ex* **by** *auto*
moreover **have** *laststate* (*?t*, *?ex*) = *?t′* **by** (*simp add*: *laststate-def*)
moreover **have** *mk-trace* (*ALM-ioa 0 id2*)·*?ex* = *nil* **by** (*simp add*: *mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def*)
moreover **have** *is-exec-frag* (*ALM-ioa 0 id2*) (*?t*, *?ex*)
proof −
from *in-abort* **and** ‹*id1* ≠ *0*› **have** *s1′* = *s1* **and** *pre-s2*:~ *aborted s2* & (∃ *c* . *phase s2 c* ≠ *Sleep*) **and** *trans-s2*:*s2′* = *s2*(|*aborted*:= *True*|) **apply** (*simp-all*

add: *composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def*) **apply**(*auto simp add*:*ALM-trans-def*) **done**

from *pre-s2* **and** ‹*P6* (*s1*, *s2*)› **have** *pre-t*:~ *aborted ?t* & (∃ *c* . *phase ?t c* ≠ *Sleep*) **apply** (*force simp add*:*ref-mapping-def P6-def*) **done**

moreover from *trans-s2* **and** ‹*s1′* = *s1*› **have** *trans-t*:*?t′* = *?t*(|*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*) −*Linearize 0 h*−*composeALMs id1 id2*⟶ (*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** ‹*id1* ≠ *0*› **have** *s2′* = *s2* **and** *pre-s1*:*initialized s1* & ~ *aborted s1* & *h* ∈ *postfix-all* (*hist s1*) (*linearizations* (*pendingReqs s1*)) **and** *trans-s1*:*s1′* = *s1*(|*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-s1* **have** ~ *aborted s1* **by** *auto*

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

moreover have *h* ∈ *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))

proof −

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

moreover have *pendingReqs s1* ⊆ *pendingReqs ?t*

proof *auto*

fix *x*

assume $x \in$ *pendingReqs s1*
moreover note ‹¬ *aborted s1*› **and** ‹*P6* (*s1* ,*s2*)›
ultimately obtain *c* **where** x = *pending s1 c* **and** *phase s1 c* = *Pending* **and** *pending s1 c* $\notin$ *set* (*hist s1*) **by** (*auto simp add*:*pendingReqs-def P6-def*)
thus $x \in$ *pendingReqs ?t* **using** ‹*hist ?t* = *hist s1*› **by** (*force simp add*:*ref-mapping-def pendingReqs-def*)
qed
moreover from *pre-s1* **have** $h \in$ *postfix-all* (*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*(|*hist* := *h*, *initialized* := *True*|)
proof −
have *hist ?t′* = *hist s1′*
proof −
from *pre-s1* **have** ~ *aborted s1* **by** *auto*
with ‹*P6* (*s1*, *s2*)› **and** ‹*P2* (*s1*, *s2*)› **have** *hist s2* = [] **by** (*auto simp add*:*P6-def P2-def*)
with ‹*s2′* = *s2*› **show** *?thesis* **by** (*auto simp add*:*ref-mapping-def*)
qed
with *trans-s1* **have** *hist ?t′* = *h* **by** *auto*
thus *?thesis* **using** ‹*s2′* = *s2*› **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*) −*Linearize id1 h*−*composeALMs id1 id2*⟶ (*s1′*, *s2′*)
— If the composition executes Linearize id1
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-lin* **and** ‹*id1* ≠ *0*› **have** *s1′* = *s1* **and** *pre-s2*: *initialized s2*

$\wedge \neg$ *aborted s2* $\wedge$ $h \in$ *postfix-all* (*hist s2*) (*linearizations* (*pendingReqs s2*)) **and** *trans-s2*: *s2'* = *s2*(|*hist* := *h*|) **apply** (*simp-all add*: *composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def*) **apply**(*auto simp add*:*ALM-trans-def*) **done**

have *pre-t*:*initialized ?t* $\wedge \neg$ *aborted ?t* $\wedge$ $h \in$ *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))

proof –

have $\neg$ *aborted ?t* **and** *initialized ?t* **using** *pre-s2* **by** (*auto simp add*:*ref-mapping-def*)

moreover have $h \in$ *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))

proof –

from *pre-s2* **have** *initialized s2* **by** *auto*

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

hence *hist ?t* = *hist s2* **by** (*auto simp add*:*ref-mapping-def*)

moreover have *pendingReqs s2* $\subseteq$ *pendingReqs ?t*

proof *auto*

fix *x*

assume $x \in$ *pendingReqs s2*

from *this* **obtain** *c* **where** *x* = *pending s2 c* **and** *phase s2 c* $\in$ {*Pending*, *Aborted*} **and** *pending s2 c* $\notin$ *set* (*hist s2*) **by** (*auto simp add*:*pendingReqs-def*)

with ⟨*P6* (*s1*, *s2*)⟩ **and** ⟨*hist ?t* = *hist s2*⟩ **show** $x \in$ *pendingReqs ?t* **by** (*force simp add*:*ref-mapping-def P6-def pendingReqs-def*)

qed

moreover from *pre-s2* **have** $h \in$ *postfix-all* (*hist s2*) (*linearizations* (*pendingReqs s2*)) **by** *auto*

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

qed

ultimately show *?thesis* **by** *auto*

qed

moreover have *trans-t*: *?t'* = *?t*(|*hist* := *h*|)

proof –

from *pre-s2* **and** *trans-s2* **have** *initialized s2'* **by** *auto*

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

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

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

qed

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

qed

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

next
fix *c 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* := *Pending*), *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*$\longrightarrow$ $(s1', s2')$
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*$\longrightarrow$ $(s1', s2')$
— 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*)$\cdot$*?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-initialize* **and** ⟨$id1 \neq 0$⟩ **have** $s1' = s1$ **and** *pre-s2*:($\exists$ *c* . *phase s2 c* $\neq$ *Sleep*) $\wedge$ $\neg$ *aborted s2* $\wedge$ $\neg$ *initialized s2* $\wedge$ $h \in$ *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* $\wedge$ $\neg$ *aborted ?t* $\wedge$ $h \in$ *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))
proof $-$
from *pre-s2* **have** *initialized ?t* $\wedge$ $\neg$ *aborted ?t* **by** (*auto simp add*:*ref-mapping-def*)
moreover have $h \in$ *postfix-all* (*hist ?t*) (*linearizations* (*pendingReqs ?t*))
proof $-$
from *pre-s2* **have** $h \in$ *postfix-all* (*l-c-p* (*initHists s2*)) (*linearizations* (*initValidReqs s2*)) **and** $\neg$ *initialized s2* **and** $\exists$ *c* . *phase s2 c* $\neq$ *Sleep* **by** *auto*
with ⟨*P13* $(s1, s2)$⟩ **have** $h \in$ *postfix-all* (*hist s1*) (*linearizations* (*pendingReqs s1*)) **by** (*auto simp add*:*P13-def*)
moreover from ⟨$\neg$ *initialized s2*⟩ **and** ⟨*P10* $(s1, s2)$⟩ **have** *hist ?t* = *hist s1* **by** (*auto simp add*:*ref-mapping-def P10-def*)
moreover have *pendingReqs s1* $\subseteq$ *pendingReqs ?t*
proof *auto*
fix *x*
assume $x \in$ *pendingReqs s1*
from *this* **obtain** *c* **where** *x* = *pending s1 c* **and** *phase s1 c* $\in$ {*Pending*, *Aborted*} **and** *pending s1 c* $\notin$ *set* (*hist s1*) **by** (*auto simp add*:*pendingReqs-def*)
show $x \in$ *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.