Abortable Linearizable Modules

Rachid Guerraoui    Viktor Kuncak    Giuliano Losa

September 19, 2015

Abstract

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

Contents

1 Introduction 2
2 Definition and properties of the longest common postfix of a set of lists 3
3 The ALM Automata specification 3
4 Proof that the composition of two instances of the ALM automaton behaves like a single instance of the ALM automaton 7
  4.1 A case split useful in the proofs 7
  4.2 Invariants of a single ALM instance 9
  4.3 Invariants of the composition of two ALM instances 9
  4.4 Proofs of invariance 11
  4.5 The refinement proof 29
5 Conclusion 46
1 Introduction

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

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

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

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

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

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

theory LCP
imports Main ~~/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 ≡ λ xss xs . if xss = {} then xs = [] else ALL xs’ . xs’ ∈ xss
→ suffixeq xs xs’

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

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

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

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

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

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

end

3 The ALM Automata specification

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

typedef client

3
— A non-empty set of clients
typedcl data
— Data contained in requests
datatype request =
— A request is composed of a sender and data
  Req client data
definition request-snd :: request ⇒ client
  where request-snd ≡ λ r. case r of Req c - ⇒ c
type-synonym hist = request list
— Type of histories of requests.
datatype ALM-action =
— The actions of the ALM automaton
  Invoke client request
  | Commit client nat hist
  | Switch client nat hist request
  | Initialize nat hist
  | Linearize nat hist
  | Abort nat
datatype phase = Sleep | Pending | Ready | Aborted
— Executions phases of a client
definition linearizations :: request set ⇒ hist set
— The possible linearizations of a set of requests
  where
  linearizations ≡ λ reqs . {h . set h ⊆ reqs ∧ distinct h}
definition postfix-all :: hist ⇒ hist set ⇒ hist set
— appends to the right the first argument to every member of the history set
  where
  postfix-all ≡ λ h hs . {h′. ∃ h′′. h′ = h′′ @ h ∧ h′′ ∈ hs}
definition ALM-asig :: nat ⇒ nat ⇒ ALM-action signature
— The action signature of ALM automata
— Input actions, output actions, and internal actions
  where
  ALM-asig ≡ λ id1 id2 . (act . ∃ c r h .
  {act . ∃ c r h .
    act = Invoke c r | act = Switch c id1 h r},
  {act . ∃ c h r id′. id1 <= id′ ∧ id′ < id2 ∧ act = Commit c id′ h
    | act = Switch c id2 h r},
  {act . ∃ h .
    act = Abort id1
    | act = Linearize id1 h}
\[
| \text{act} = \text{Initialize id1 h} \}
\]

**record** \text{ALM-state} =

— The state of the ALM automata

\text{pending} :: \text{client} \Rightarrow \text{request}

— Associates a pending request to a client process

\text{initHists} :: \text{hist set}

— The set of init histories submitted by clients

\text{phase} :: \text{client} \Rightarrow \text{phase}

— Associates a phase to a client process

\text{hist} :: \text{hist}

— Represents the chosen linearization of the concurrent history of the current instance only

\text{aborted} :: \text{bool}

\text{initialized} :: \text{bool}

**definition** \text{pendingReqs} :: \text{ALM-state} \Rightarrow \text{request set}

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

\text{where}

\text{pendingReqs} \equiv \lambda s . \{ r . \exists c .

r = \text{pending } s \ c

\land r \notin \text{set } (\text{hist } s)

\land \text{phase } s \ c \in \{ \text{Pending, Aborted} \} \}

**definition** \text{initValidReqs} :: \text{ALM-state} \Rightarrow \text{request set}

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

\text{where}

\text{initValidReqs} \equiv \lambda s . \{ r .

(r \in \text{pendingReqs } s \ \lor (\exists h \in \text{initHists } s . r \in \text{set } h))

\land r \notin \text{set } (\text{l-c-p } (\text{initHists } s)) \}

**definition** \text{ALM-trans} :: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{ALM-action}, \text{ALM-state}) \text{transition set}

— the transitions of the ALM automaton

\text{where}

\text{ALM-trans} \equiv \lambda id1 id2 . \{ \text{trans} .

let s = \text{fst } \text{trans}; s' = \text{snd } (\text{snd } \text{trans}); a = \text{fst } (\text{snd } \text{trans}) \text{ in}

\text{case } a \text{ of Invoke } c \ r \Rightarrow

\text{if phase } s \ c = \text{Ready} \land \text{request-snd } r = c \land r \notin \text{set } (\text{hist } s)

\text{then } s' = s[\text{pending} := (\text{pending } s)(c := r),

\text{phase} := (\text{phase } s)(c := \text{Pending})]

\text{else } s' = s

| \text{Linearize } i \ h \Rightarrow

\text{initialized } s \land \neg \text{aborted } s

\land h \in \text{postfix-all } (\text{hist } s) \ (\text{linearizations } (\text{pendingReqs } s)) \}

5
\[ s' = s[\text{hist} := h] \]

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

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

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

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

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

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

6
\textbf{type-synonym} \textit{compo-state} = \textit{ALM-state} \times \textit{ALM-state}

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

\textbf{where}

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

\textbf{end}

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

\textbf{theory} \textit{CompositionCorrectness}
\textbf{imports} \textit{ALM}
\textbf{begin}

\textbf{declare} \textit{split-if asm} [\textit{split}]
\textbf{declare} \textit{Let-def} [\textit{simp}]

4.1 A case split useful in the proofs

\textbf{definition} \textit{in-trans-cases-fun} :: \textit{nat} \Rightarrow \textit{nat} \Rightarrow (\textit{ALM-state} \times \textit{ALM-state}) \Rightarrow bool
— Helper function used to decompose proofs

\textbf{where}

\textit{in-trans-cases-fun} \equiv \% \textit{id1 \ id2} \textit{s t} .
\begin{align*}
&(\text{EX ca ra} . (\text{fst s, Invoke ca ra}, \text{fst t}) : \text{ALM-trans 0 id1} \& (\text{snd s, Invoke ca ra}, \\
&\text{snd t}) : \text{ALM-trans id1 id2}) \\
&\text{\mid (EX ca h ra} . (\text{fst s, Switch ca id1 h ra}, \text{fst t}) : \text{ALM-trans 0 id1} \& (\text{snd s, Switch ca id1 h ra, snd t}) : \text{ALM-trans id1 id2}) \\
&\text{\mid (EX c id' h. \text{fst t = fst s} \& (\text{snd s, Commit c id' h, snd t}) : \text{ALM-trans id1 id2} \\
&\text{& id1} \leq\text{ id' \& id' < id2}) \\
&\text{\mid (EX c h r. \text{fst t = fst s} \& (\text{snd s, Switch c id2 h r, snd t}) : \text{ALM-trans id1 id2})} \\
&\text{\mid (EX h. \text{fst t = fst s} \& (\text{snd s, Linearize id1 h, snd t}) : \text{ALM-trans id1 id2})} \\
&\text{\mid (fst t = fst s \& (snd s, Abort id1, snd t) : \text{ALM-trans id1 id2})} \\
&\text{\mid (EX h. \text{fst t = fst s} \& (\text{snd s, Initialize id1 h, snd t}) : \text{ALM-trans id1 id2})} \\
&\text{\mid (EX ca ta ra. (\text{fst s, Switch ca 0 ta ra, fst t}) : \text{ALM-trans 0 id1} \& \text{snd t} = \text{snd s})} \\
&\text{\mid (EX ca id' h. (\text{fst s, Commit ca id' h, fst t}) : \text{ALM-trans 0 id1} \& \text{snd t} = \text{snd s \& id' < id1})} \\
&\text{\mid (EX h. (\text{fst s, Linearize 0 h, fst t}) : \text{ALM-trans 0 id1} \& \text{snd t} = \text{snd s})} \\
&\text{\mid (EX h. (\text{fst s, Initialize 0 h, fst t}) : \text{ALM-trans 0 id1} \& \text{snd t} = \text{snd s})}
\end{align*}

7
lemma composeALMsE:

— A rule for decomposing proofs

assumes id1 < id2 and id1 < id2 and in-trans-comp:s—(a::ALM-action)——composeALMs id1 id2—> t

shows decom: in-trans-cases-fun id1 id2 s t

proof –
from in-trans-comp and (id1 < id2) and (id1 < id2)

have a : {act . EX c r h id'. 0 <= id' & id' < id2 & (a = 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 < id2) and (id1 < id2)

have
(a = Linearize 0 h | a = Abort 0 | a = Initialize 0 h | a = Switch c 0 h r | (a = Commit c id' h & id' < id1)) ==> ((fst s, a, fst t) : ALM-trans 0 id1 & snd s = snd t)

and
(a = Linearize id1 h | a = Abort id1 | a = Initialize id1 h | a = Switch c id2 h r | (a = Commit c id' h & id' < id2)) ==> (fst s = fst t & (snd s, a, snd t) : ALM-trans id1 id2)

and
(a = Switch c id1 h r | a = Invoke c r) ==> ((fst s, a, fst t) : ALM-trans 0 id1 & (snd s, a, snd t) : ALM-trans id1 id2)


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

qed
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

\[ P1a == \% s . let s1 = fst s; s2 = snd s in \]
\[ \forall c . phase s1 c \in \{\text{Pending, Aborted}\} \rightarrow request-snd (\text{pending} s1 c) = c \]

**definition** \( P1b :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- In ALM 2, a pending request of client \( c \) has client \( c \) as sender

\[ P1b == \% s . let s1 = fst s; s2 = snd s in \]
\[ \forall c . phase s2 c \neq \text{Sleep} \rightarrow request-snd (\text{pending} s2 c) = c \]

**definition** \( P2 :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- \( P2 == \% s . let s1 = fst s; s2 = snd s in \)

\[ (\forall c . phase s2 c = \text{Sleep}) \rightarrow (\neg \text{initialized} s2 \land \text{hist} s2 = []) \]

**definition** \( P3 :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- \( P3 == \% s . let s1 = fst s; s2 = snd s in \)

\[ \forall c . (\text{phase} s2 c = \text{Ready} \rightarrow \text{initialized} s2) \]

**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

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

**definition** \( P5 :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- In ALM 1, a client never sleeps

\[ P5 == \% s . let s1 = fst s; s2 = snd s in \]
\[ \forall c . phase s1 c \neq \text{Sleep} \]

4.3 Invariants of the composition of two ALM instances

**definition** \( P6 :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- Non-interference across instances

\[ P6 == \% s . let s1 = fst s; s2 = snd s in \]
\[ (\neg \text{aborted} s1 \rightarrow (\forall c . phase s2 c = \text{Sleep})) \land (\forall c . phase s1 c = \text{Aborted} \rightarrow (\text{phase} s2 c = \text{Sleep})) \]

**definition** \( P7 :: (ALM-state * ALM-state) \Rightarrow bool \)

**where**
- 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
\[ P7 \equiv \% s. \ let s1 = fst s; s2 = snd s in \]
\[ \forall c . \ phase s1 c = Aborted \land \neg initialized s2 \rightarrow (pending s2 c = pending s1 c \land phase s2 c \in \{Pending, Aborted\}) \]

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

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

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

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

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

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

\textbf{definition} \text{P14} :: (\text{ALM-state } * \text{ALM-state}) \Rightarrow \text{bool} \\
\textbf{where} \\
\text{— 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 \equiv \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in} \\
(hist s2 \neq [\ ] \lor \text{initialized } s2) \rightarrow (\exists rs . \\
\text{hist } s2 = rs @ (\text{hist } s1) \\
\lor \text{set } rs \cap \text{set } (\text{hist } s1) = \{\})

\textbf{definition} \text{P15} :: (\text{ALM-state } * \text{ALM-state}) \Rightarrow \text{bool} \\
\textbf{where} \\
\text{— A client that hasn’t yet invoked ALM 2 has no request committed in ALM 2 except for its pending request} \\
P15 \equiv \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in} \\
\forall r . \text{let } c = \text{request-snd } r \text{ in phase } s2 c = \text{Sleep} \land r \in \text{set } (\text{hist } s2) \rightarrow (r \in \text{set } (\text{hist } s1) \lor r \in \text{pendingReqs } s1)

\textbf{4.4 Proofs of invariance}

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

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

\textbf{lemma} \text{dropWhile-lemlmma}: \forall ys . xs = ys @ zs \land \text{hd } zs = x \land zs \neq [\ ] \land x \notin \text{set } ys \rightarrow \text{dropWhile } (\lambda x'. x' \neq x) \text{ xs } = zs

\text{— A useful lemma about truncating histories} \\
\textbf{proof} (\text{induct } xs, \text{force}) \\
\text{fix } a \text{ xs} \\
\text{assume } \forall ys . xs = ys @ zs \land \text{hd } zs = x \land zs \neq [\ ] \land x \notin \text{set } ys \rightarrow \text{dropWhile } (\lambda x'. x' \neq x) \text{ xs } = zs \\
\text{show } \forall ys . a \neq xs = ys @ zs \land \text{hd } zs = x \land zs \neq [\ ] \land x \notin \text{set } ys \rightarrow \text{dropWhile } (\lambda x'. x' \neq x) (a \neq xs) = zs \\
\text{proof} (\text{rule allI, rule impI, cases } a = x) \\
\text{fix } ys \\
\text{assume } a \neq xs = ys @ zs \land \text{hd } zs = x \land zs \neq [\ ] \land x \notin \text{set } ys \text{ and } a = x \\
\text{hence } x \neq xs = ys @ zs \text{ and } x \notin \text{set } ys \text{ and } xs = x \text{ and } zs \neq [\ ] \text{ by auto} \\
\text{from } x \neq xs = ys @ zs \text{ and } x \notin \text{set } ys \text{ have } ys = [\ ] \text{ by (metis list.sel(1))} \\
\text{hd-append hd-in-set}) \\
\text{with } (a = x); \text{ and } (x \neq xs = ys @ zs) \text{ show } \text{dropWhile } (\lambda x'. x' \neq x) (a \neq xs) = zs \text{ by auto} \\
\text{next}

11
fix \( y s \)
assume \( a \neq x s = y s @ z s \land h d z s = x \land z s \neq [] \land x \notin \text{set } y s \) and \( a = x \)
hence \( a = y s \neq z s \land h d z s = x \land z s \neq [] \land x \notin \text{set } y s \) by auto
obtain \( y s' \) where \( x s = y s' @ z s \) and \( x \notin \text{set } y s' \)
proof
  from \( (a \neq x s = y s @ z s) \land (h d z s = x) \) and \( (a = x) \) obtain \( y s' \) where \( y s = a \neq y s' \) apply clarify by (metis Cons-eq-append-conv list.sel(1))
  moreover with \( x \notin \text{set } y s \) have \( x \notin \text{set } y s' \) by auto
  moreover from \( (y s = a \neq y s') \) and \( (a \neq x s = y s @ z s) \) have \( x s = y s' @ z s \) by auto
  ultimately show \((\forall y s'. [x s = y s' @ z s; x \notin \text{set } y s'] \Longrightarrow \text{thesis}) \Longrightarrow \text{thesis}\)
  by auto
qed

lemma \( P 2\)-invariant: \([|id 1 < id 2; id 1 \neq 0|] \Longrightarrow \text{invariant } (\text{composeALMs } id 1 id 2) P 2\)
proof (rule invariant1, auto)
  fix \( s 1, s 2 \)
  assume \((s 1, s 2) : \text{starts-of } (\text{composeALMs } id 1 id 2) \) and \( 0 < id 1 \)
  thus \( P 2 (s 1, s 2) \) by (simp add: \text{starts-of-def composeALMs-def hide-def ALM-ios-def par-def ALM-start-def P2-def})
next
  fix \( s 1, s 2, s 1', s 2' \) act
  assume \( \text{reachable } (\text{composeALMs } id 1 id 2) (s 1, s 2) \) and \( P 2 (s 1, s 2) \) and \( 0 < id 1 \) and \( id 1 < id 2 \) and \( \text{in-trans-comp}(s 1, s 2) \) and \( \text{act} \Longrightarrow \text{composeALMs } id 1 id 2 \Rightarrow (s 1', s 2') \)
  from \( (0 < id 1) \) and \( (id 1 < id 2) \) and \( \text{in-trans-comp} \) show \( P 2 (s 1', s 2') \)
  proof (rule my-rule2)
    assume \( \text{in-trans-cases-fun} id 1 id 2 (s 1, s 2) (s 1', s 2') \)
    thus \( P 2 (s 1', s 2') \) using \( P 2 (s 1, s 2) \) and \( (0 < id 1) \) and \( (id 1 < id 2) \) apply (auto simp add: \text{in-trans-cases-fun-def}) apply (auto simp add: \text{ALM-trans-def P2-def})
  qed
qed

lemma \( P 5\)-invariant: \([|id 1 < id 2; id 1 \neq 0|] \Longrightarrow \text{invariant } (\text{composeALMs } id 1 id 2) P 5\)
proof (rule invariant1, auto)
  fix \( s 1, s 2 \)
  assume \((s 1, s 2) : \text{starts-of } (\text{composeALMs } id 1 id 2) \) and \( 0 < id 1 \)
  thus \( P 5 (s 1, s 2) \) by (simp add: \text{starts-of-def composeALMs-def hide-def ALM-ios-def par-def ALM-start-def P5-def})
next
fix $s_1$ $s_2$ $s_1'$ $s_2'$ act
assume reachable (composeALMs id1 id2) $(s_1, s_2)$ and $P_5 (s_1, s_2)$ and $0 < id1$ and $id1 < id2$ and in-trans-comp:$(s_1, s_2)$ -- act -- composeALMs id1 id2 -> $(s_1', s_2')$
from $0 < id1$ and $(id1 < id2)$ and in-trans-comp show $P_5 (s_1', s_2')$
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 $(s_1, s_2)$ $(s_1', s_2')$
  thus $P_5 (s_1', s_2')$ using $P_5 (s_1, s_2)$ 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

lemma $P_6$-invariant: [|$id1 \neq 0$ ; $id1 < id2$] ==> invariant (composeALMs id1 id2) $P_6$
proof (rule invariantI, rule-tac [2] impI)
fix $s$
assume $s$ : starts-of (composeALMs id1 id2) and $id1 \neq 0$
thus $P_6$ $s$ by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P6-def)
next
fix $s$ $t$
assume $P_6$ $s$
assume $id1 \neq 0$ and $id1 < id2$ and $s$ -- composeALMs id1 id2 -> $t$
thus $P_6$ $t$
proof (rule my-rule)
  assume in-trans-cases-fun id1 id2 $s$ $t$
  thus $P_6$ $t$ using $P_6$ $s$ and $(id1 \neq 0)$ 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) done
qed

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

lemma $P_{10}$-invariant: [$[id_1 < id_2; id_1 = 0] \Longrightarrow$ invariant (composeALMs id1 id2) $P_{10}$
proof (rule invariantI1, auto)
fix $s_1 s_2$
assume ($s_1, s_2$): starts-of (composeALMs id1 id2) and ($0 < id_1$)
thus $P_{10}$ ($s_1, s_2$) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def $P_{10}$-def)
next
fix $s_1 s_2 s_1' s_2'$ act
assume reachable (composeALMs id1 id2) ($s_1, s_2$) and $P_{10}$ ($s_1, s_2$): and ($0 < id_1$ and ($id_1 < id_2$) and in-trans-comp($s_1, s_2$) -- act -- composeALMs id1 id2 -> ($s_1', s_2'$)
from ($0 < id_1$ and ($id_1 < id_2$) and in-trans-comp show $P_{10}$ ($s_1', s_2'$)
proof (rule my-rule2)
assume in-trans-cases-fun id1 id2 ($s_1, s_2$) ($s_1', s_2'$)
thus $P_{10}$ ($s_1', s_2'$) using ($P_{10}$ ($s_1, s_2$): and ($0 < id_1$ and ($id_1 < id_2$) apply(auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_{10}$-def) done
qed
qed

lemma $P_{3}$-invariant: [$[id_1 < id_2; id_1 \neq 0] \Longrightarrow$ invariant (composeALMs id1 id2) $P_3$
proof (rule invariantI1, auto)
fix $s_1 s_2$
assume ($s_1, s_2$): starts-of (composeALMs id1 id2) and ($0 < id_1$)
thus $P_3$ ($s_1, s_2$) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def $P_3$-def)
next
fix $s_1 s_2 s_1' s_2'$ act
assume reachable (composeALMs id1 id2) ($s_1, s_2$) and $P_3$ ($s_1, s_2$): and ($0 < id_1$ and ($id_1 < id_2$) and in-trans-comp($s_1, s_2$) -- act -- composeALMs id1 id2 -> ($s_1', s_2'$)
have $P_{10}$ ($s_1$, $s_2$)
proof -
  from in-trans-comp and reachable (composeALMs id1 id2) ($s_1$, $s_2$); have
  reachable (composeALMs id1 id2) ($s_1'$, $s_2'$) by (auto intro: reachable.reachable-n)
  with reachable (composeALMs id1 id2) ($s_1$, $s_2$) and ($0 < id_1$) and ($id_1 < id_2$) and
  $P_{10}$-invariant show $P_{10}$ ($s_1$, $s_2$) unfolding invariant-def by auto
qed
from ($0 < id_1$) and ($id_1 < id_2$) and in-trans-comp show $P_3$ ($s_1'$, $s_2'$)
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 ($s_1$, $s_2$) ($s_1'$, $s_2'$)
  thus $P_3$ ($s_1'$, $s_2'$) using ($P_3$ ($s_1$, $s_2$)) and ($P_{10}$ ($s_1$, $s_2$)) and ($0 < id_1$) and
  ($id_1 < id_2$) apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_3$-def $P_{10}$-def) done
qed
qed

lemma $P_{7}$-invariant: [[id1 < id2; id1 $\neq$ 0]] $\Longrightarrow$ invariant (composeALMs id1 id2) $P_7$
proof (rule invariantI, auto)
  fix $s_1$ $s_2$
  assume ($s_1$, $s_2$) : starts-of (composeALMs id1 id2) and ($0 < id_1$)
  thus $P_7$ ($s_1$, $s_2$) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def
  par-def ALM-start-def $P_7$-def)
next
  fix $s_1$ $s_2$ $s_1'$ $s_2'$ act
  assume reachable (composeALMs id1 id2) ($s_1$, $s_2$) and $P_7$ ($s_1$, $s_2$) and ($0 < id_1$) and
  ($id_1 < id_2$) and in-trans-comp ($s_1$, $s_2$) -- act -- composeALMs id1 id2 -->
  ($s_1'$, $s_2'$)
  have $P_{6}$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$)
proof -
  from in-trans-comp and reachable (composeALMs id1 id2) ($s_1$, $s_2$); have
  reachable (composeALMs id1 id2) ($s_1'$, $s_2'$) by (auto intro: reachable.reachable-n)
  with reachable (composeALMs id1 id2) ($s_1$, $s_2$) and ($0 < id_1$) and
  ($id_1 < id_2$) and $P_6$-invariant and $P_{10}$-invariant show $P_{6}$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$)
  unfolding invariant-def by auto
qed
from ($0 < id_1$) and ($id_1 < id_2$) and in-trans-comp show $P_7$ ($s_1'$, $s_2'$)
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 ($s_1$, $s_2$) ($s_1'$, $s_2'$)
  thus $P_7$ ($s_1'$, $s_2'$) using ($P_7$ ($s_1$, $s_2$)) and ($P_{6}$ ($s_1$, $s_2$)) and ($0 < id_1$) and
  ($id_1 < id_2$)
  proof (auto simp add: in-trans-cases-fun-def)
  fix $ca$ $ra$
  assume $P_7$ ($s_1$, $s_2$) and $P_6$ ($s_1$, $s_2$) and ($0 < id_1$) and
  ($id_1 < id_2$) and ($s_1$,
  Invoke $ca$ $ra$, $s_1'$) $\in$ ALM-trans $0$ $id_1$ and ($s_2$, $Invoke$ $ca$ $ra$, $s_2'$) $\in$ ALM-trans
  $id_1$ $id_2$
  thus $P_7$ ($s_1'$, $s_2'$) by (auto simp add: ALM-trans-def $P_7$-def)
next
  fix $ca$ $ch$ $ra$

15
assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_1, Switch ca id_1 h ra, s_1') \in ALM\text{-}trans$ \(0\) $id_1$ and $(s_2, Switch ca id_1 h ra, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$

thus $P_7 (s_1', s_2')$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def $P_6$-def)

next

fix $c \ id_1' h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $(s_2, Commit c id_1' h, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$ and $id_1 \leq id' \ and \ id' < id_2$

thus $P_7 (s_1, s_2')$ using $(P_{10} (s_1, s_2))$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def $P_{10}$-def)

next

fix $c \ h r$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_2, Switch c id_2 h r, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$

thus $P_7 (s_1, s_2')$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_2, Linearize \ id_1 h, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$

thus $P_7 (s_1, s_2')$ by (simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_2, Initialize \ id_1 h, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$

thus $P_7 (s_1, s_2')$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $ca \ ta \ ra$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_1, Switch ca 0 ta ra, s_1') \in ALM\text{-}trans$ \(0\) $id_1$

thus $P_7 (s_1', s_2)$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $ca \ id_1' h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $id_1 < id_2$ and $(s_1, Commit ca id_1' h, s_1') \in ALM\text{-}trans$ \(0\) $id_1$ and $id' < id_1$

thus $P_7 (s_1', s_2)$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_1, Linearize \ 0 h, s_1') \in ALM\text{-}trans$ \(0\) $id_1$

thus $P_7 (s_1', s_2)$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

fix $h$

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_1, Initialize \ 0 h, s_1') \in ALM\text{-}trans$ \(0\) $id_1$

thus $P_7 (s_1', s_2)$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)

next

assume $P_7 (s_1, s_2)$ and $P_6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_2, Abort \ id_1, s_2') \in ALM\text{-}trans$ \(id_1\) $id_2$

thus $P_7 (s_1, s_2')$ by (auto simp add: $ALM\text{-}trans$-def $P_7$-def)
next

assume $P^7 (s_1, s_2)$ and $P^6 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $(s_1, Abort 0, s_1') \in \text{ALM-trans } 0 \ id_1$

thus $P^7 (s_1', s_2)$ by (auto simp add: \text{ALM-trans } P^7-def)

qed

qed

lemma $P^4$-invariant: $[[id_1 < id_2; id_1 \neq 0]] \Longrightarrow \text{invariant (composeALMs id1 id2)}$ $P^4$

proof (rule invariantI, auto)

fix $s_1 s_2$

assume $(s_1, s_2) : \text{start-of (composeALMs id1 id2)}$ and $0 < id_1$

thus $P^4 (s_1, s_2)$ by (simp add: \text{start-of-def composeALMs-def hide-def ALM-iaa-def par-def ALM-start-def } P^4-def)

next

fix $s_1 s_2 s_1' s_2' \ act$

assume reachable (composeALMs id1 id2) $(s_1, s_2)$ and $P^4 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp $(s_1, s_2)$ -- act -- composeALMs id1 id2 --> $(s_1', s_2')$

have $P^6 (s_1, s_2)$

proof --

from in-trans-comp and reachable (composeALMs id1 id2) $(s_1, s_2)$ and $P^4 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and $P^6$-invariant show $P^6 (s_1, s_2)$ unfolding invariant-def by auto

qed

from $(0 < id_1)$ and $(id_1 < id_2)$ and in-trans-comp show $P^4 (s_1', s_2')$

proof (rule my-rule2)

assume in-trans-cases-fun id1 id2 $(s_1, s_2)$ $(s_1', s_2')$

thus $P^4 (s_1', s_2')$ using $P^4 (s_1, s_2)$ and $(0 < id_1)$ and $(id_1 < id_2)$ apply(auto simp add: in-trans-cases-fun-def) apply (auto simp add: \text{ALM-trans } P^4-def) done

qed

qed

lemma $P^8$-invariant: $[[id_1 < id_2; id_1 \neq 0]] \Longrightarrow \text{invariant (composeALMs id1 id2)}$ $P^8$

proof (rule invariantI, auto)

fix $s_1 s_2$

assume $(s_1, s_2) : \text{start-of (composeALMs id1 id2)}$ and $0 < id_1$

thus $P^8 (s_1, s_2)$ by (simp add: \text{start-of-def composeALMs-def hide-def ALM-iaa-def par-def ALM-start-def } P^8-def)

next

fix $s_1 s_2 s_1' s_2' \ act$

assume reachable (composeALMs id1 id2) $(s_1, s_2)$ and $P^8 (s_1, s_2)$ and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp $(s_1, s_2)$ -- act -- composeALMs id1 id2 --> $(s_1', s_2')$

have $P^6 (s_1, s_2)$ and $P^10 (s_1, s_2)$ and $P^5 (s_1, s_2)$ and $P^4 (s_1, s_2)$
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-inv1:(s1, Invoke ca ra, s1', s2) ∈ ALM-trans 0 id1 and in-inv2:(s2, Invoke ca ra, s2') ∈ ALM-trans id1 id2

show P8 (s1', s2')

proof (cases s1' = s1)

assume s1' = s1

with in-inv2 and (P8 (s1, s2)) show \?thesis by (auto simp add: ALM-trans-def P8-def)

next

assume s1' ≠ s1

with in-inv1 have pendingReqs s1 ⊆ pendingReqs s1' by (force simp add:pendingReqs-def ALM-trans-def)

moreover from in-inv1 have hist s1' = hist s1 by (auto simp add:ALM-trans-def)

moreover from in-inv2 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-switch1:(s1, Switch ca id1 h ra, s1') ∈ ALM-trans 0 id1 and in-switch2:(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-switch1 and \{0 < id1\} have hist s1' = hist s1 and pendingReqs s1' = pendingReqs s1 by (auto simp add:ALM-trans-def pendingReqs-def)
moreover note \( P_8 (s_1, s_2) \)
ultimately show \( h_1 \in \text{postfix-all} (\text{hist} \, s_1') \) (linearizations (pendingReqs \, s_1')) by (auto simp add: \( P_8\)-def)

next
assume \( h_1 \notin \text{initHists} \, s_2 \)
with \( h_1 \in \text{initHists} \, s_2' \) and in-switch-2 have \( h_1 = h \) by (auto simp add: ALM-trans-def)

with in-switch-1 and \( 0 < \text{id1} \) and \( P_{10} (s_1, s_2) \) have \( h_1 \in \text{postfix-all} (\text{hist} \, s_1) \) (linearizations (pendingReqs \, s_1)) by (auto simp add: ALM-trans-def \( P_{10}\)-def)

moreover from in-switch-1 and \( 0 < \text{id1} \) have \( \text{hist} \, s_1' = \text{hist} \, s_1 \) and pendingReqs \, s_1' = pendingReqs \, s_1 \) by (auto simp add: ALM-trans-def pendingReqs-def)
ultimately show \( \text{thesis} \) by auto
qed

next
fix \( c \, \text{id}' \, h \)
assume \( P_8 (s_1, s_2) \) and \( 0 < \text{id1} \) and \( \text{id1} \, \text{id2} \) and \( \text{id1} \leq \text{id}' \) and \( \text{id}' < \text{id2} \)
thus \( P_8 (s_1, s_2') \) by (auto simp add: ALM-trans-def \( P_8\)-def)

next
fix \( c \, h \, r \)
assume \( P_8 (s_1, s_2) \) and \( 0 < \text{id1} \) and \( \text{id1} < \text{id2} \) and \( \text{id2} \) and \( \text{id1} \in \text{ALM-trans \, id1} \, \text{id2} \)
thus \( P_8 (s_1, s_2') \) by (auto simp add: ALM-trans-def \( P_8\)-def)

next
fix \( c \)
assume \( P_8 (s_1, s_2) \) and \( 0 < \text{id1} \) and \( \text{id1} < \text{id2} \) and \( \text{id2} \) and \( \text{id1} \in \text{ALM-trans \, id1} \, \text{id2} \)
thus \( P_8 (s_1, s_2') \) by (auto simp add: ALM-trans-def \( P_8\)-def)

next
fix \( c \, a \, \text{ta} \, r \)
assume \( P_8 (s_1, s_2) \) and \( 0 < \text{id1} \) and \( \text{id1} < \text{id2} \) and \( \text{id2} \) and \( \text{id1} \in \text{ALM-trans \, id1} \, \text{id2} \)
thus \( P_8 (s_1', s_2) \) using \( P_{5} (s_1, s_2) \) by (auto simp add: ALM-trans-def \( P_{8}\)-def \( P_{5}\)-def)

next
fix \( c \, a \, \text{id}' \)
assume \( P_8 (s_1, s_2) \) and in-commit-1: \( (s_1, \text{Commit} \, c \, \text{id}' \, h, s_1') \) in ALM-trans \( 0 \, \text{id1} \)
from in-commit-1 have pendingReqs \, s_1' = pendingReqs \, s_1 \) and \( \text{hist} \, s_1' = \text{hist} \, s_1 \) by (auto simp add: pendingReqs-def ALM-trans-def)
with \( P_8 (s_1, s_2) \) show \( P_8 (s_1', s_2) \) by (auto simp add: ALM-trans-def \( P_8\)-def pendingReqs-def)
next
  fix h
  assume P8 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Linearize 0 h, s1') ∈ 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') ∈ 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') ∈ ALM-trans 0 id1
  thus P8 (s1', s2) using ⟨P10 (s1, s2)⟩ by (auto simp add: ALM-trans-def P8-def)

next
  assume P8 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Abort 0, s1') ∈ ALM-trans 0 id1
  thus P8 (s1', s2) by (auto simp add: ALM-trans-def P8-def pendingReqs-def)

qed

lemma P12-invariant: \[|id1 < id2; id1 \neq 0|\] ==> invariant (composeALMs id1 id2) P12
proof clarify
  assume id1 < id2 and 0 < id1
  with P8-invariant and P4-invariant have invariant (composeALMs id1 id2) (λ (s1, s2) . P8 (s1, s2) ∧ P4 (s1, s2)) by (auto simp add: invariant-def)
moreover have ∀ s . P8 s ∧ P4 s −→ P12 s
proof auto
  fix s1 s2
  assume P8 (s1, s2) and P4 (s1, s2)
  hence initHists-prop: \(\exists h \in \text{initHists} s2 . (\exists h' . h = h' \oplus \text{hist} s1) \land \text{set} h' \subseteq \text{pendingReqs} s1 \land \text{distinct} h'\) by (auto simp add: P8-def postfix-all-def linearizations-def)
  show P12 (s1, s2)
  proof (simp add:P12-def, rule impI)
    assume ∃ c . phase s2 c ≠ Sleep
    with ⟨P4 (s1, s2)⟩ have initHists s2 ≠ {} 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 ⊆ pendingReqs s1
    proof -
      from ⟨initHists s2 ≠ {}⟩ obtain h where h ∈ initHists s2 by auto
      with initHists-prop obtain h' where h = h' @ (hist s1) \land set h' ⊆ pendingReqs s1 by auto
  qed
qed
moreover from \( \text{l-c-p-common-postfix[of initHists s2]} \) and \((\text{h ∈ initHists s2})\) obtain \(h''\) where \(h = h'' @ (\text{l-c-p (initHists s2)})\) by (auto simp add:common-postfix-p-def suffixeq-def)

moreover note \((\text{l-c-p (initHists s2)} = rs @ \text{hist s1})\)

ultimately show \(?thesis\) by auto

qed

moreover have \(\text{distinct rs}\)

proof

from \((\text{initHists s2} ≠ \{\})\) obtain \(h\) where \(h ∈ \text{initHists s2}\) by auto

with \(\text{initHists-prop obtain h' where h = h' @ (\text{hist s1}) and distinct h'}\) by auto

with \(\text{l-c-p-common-postfix[of initHists s2]} \) and \(\text{h ∈ initHists s2 and l-c-p (initHists s2)} = rs @ \text{hist s1}\) obtain \(h''\) where \(h' = h'' @ rs\) by (auto simp add:common-postfix-p-def suffixeq-def)

ultimately show \(?thesis\) by auto

qed

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

qed

lemma \(\text{P11-invariant: } [\text{id1 < id2}; \text{id1} ≠ 0]]\Longrightarrow \text{invariant (composeALMs id1 id2)}\) \(\text{P11}\)

proof clarify

assume \(\text{id1 < id2}\) and \(0 < \text{id1}\)

with \(\text{P8-invariant and P12-invariant and P6-invariant and P7-invariant}\) have

\(\text{invariant (composeALMs id1 id2)}\) \((\lambda (s1, s2). \text{P8 (s1, s2)} \land \text{P12 (s1, s2)} \land \text{P6 (s1, s2)} \land \text{P7 (s1, s2)})\) by (auto simp add:invariant-def)

moreover have \(\forall s . \text{P8 s} \land \text{P12 s} \land \text{P6 s} \land \text{P7 s} \longrightarrow \text{P11 s}\)

proof auto

fix \(s1\) \(s2\)

assume \(\text{P8 (s1, s2)}\) and \(\text{P12 (s1, s2)}\) and \(\text{P6 (s1, s2)}\) and \(\text{P7 (s1, s2)}\)

show \(\text{P11 (s1, s2)}\)

proof (simp add: P11-def initValidReqs-def, auto)

fix \(x c h\)

assume \(\text{phase s2 c} ≠ \text{Sleep}\)

with \(\text{P12 (s1, s2)}\) and \(\text{P8 (s1, s2)}\) have \(\text{initHists-prop: } \forall h ∈ \text{initHists s2}\)

\((\exists h'. h = h' @ (\text{hist s1}) \land \text{set h'} \subseteq \text{pendingReqs s1})\) and \(\text{lcp-prop: } \exists rs . \text{l-c-p (initHists s2)} = rs @ (\text{hist s1})\) by (auto simp add: P12-def P8-def postfix-all-def linearizations-def)

assume \(x \notin \text{set (l-c-p (initHists s2))}\) and \(h ∈ \text{initHists s2 and x ∈ set h}\)

from \(\text{initHists-prop and } h ∈ \text{initHists s2}\) obtain \(h'\) where \(h = h' @ (\text{hist s1})\) and \(\text{set h'} \subseteq \text{pendingReqs s1}\) by auto

moreover from \(\text{lcp-prop obtain rs where l-c-p (initHists s2)} = rs @ (\text{hist s1})\) by auto

moreover note \(x \notin \text{set (l-c-p (initHists s2))}\) and \(\forall x \in \text{set h}\)
ultimately have \( x \in \text{set } h' \) by auto
with \( \{h' \subseteq \text{pendingReqs } s1\} \) show \( x \in \text{pendingReqs } s1 \) by auto
next
fix \( x \in h \)
assume phase \( s2 \not= \text{Sleep} \) and \( \neg \text{initialized } s2 \)
with \( \{P12 \ (s1, s2)\} \) have \( \text{lcp-prop: } \exists rs \cdot l-c-p \ (\text{initHists } s2) = rs @ (\text{hist } s1) \) by \( \text{(auto simp add: } P12\text{-def } P8\text{-def postfix-all-def linearizations-def) } \)
assume \( x \notin \text{set } (l-c-p \ (\text{initHists } s2)) \) and \( x \in \text{pendingReqs } s2 \)
from \( x \notin \text{set } (l-c-p \ (\text{initHists } s2)) \) and \( \text{lcp-prop } \) have \( x \notin \text{set } (\text{hist } s1) \) by auto
moreover obtain \( c' \) where phase \( s1 \) \( c' = \text{Aborted} \) and \( x = \text{pending } s1 \) \( c' \)
proof –
from \( x \in \text{pendingReqs } s2 \) and \( \{P6 \ (s1, s2)\} \) obtain \( c' \) where phase \( s1 \) \( c' = \text{Aborted} \) and \( x = \text{pending } s2 \) \( c' \) by \( \text{(force simp add: pendingReqs-def } P6\text{-def) } \)
moreover with \( \neg \text{initialized } s2 \) and \( \{P7 \ (s1, s2)\} \) have \( x = \text{pending } s1 \) \( c' \) by \( \text{(auto simp add: } P7\text{-def) } \)
ultimately show \( (\bigwedge c'. \ [ \text{phase } s1 \ c' = \text{Aborted}; \ x = \text{pending } s1 \ c'] ) \implies \text{thesis} \) \( \implies \text{thesis} \) by auto
qed
ultimately show \( x \in \text{pendingReqs } s1 \) by \( \text{(auto simp add: pendingReqs-def) } \)
qed
qed
ultimately show \( ?\text{thesis} \) by \( \text{(auto intro: invariant-imp) } \)
qed

lemma \( P1a\text{-invariant: } [\{id1 < id2; \} \not= 0] \implies \text{invariant } (\text{composeALMs } id1 \ id2) \ P1a \)
proof \( \text{(rule invariantI, auto) } \)
fix \( s1 \) \( s2 \)
assume \( (s1, s2) : \text{starts-of } (\text{composeALMs } id1 \ id2) \) and \( 0 < id1 \)
thus \( P1a \ (s1, s2) \) by \( \text{(simp add: starts-of-def composeALMs-def hide-def } \text{ALM}-ioa-def } \text{par-def } \text{ALM-start-def } P1a\text{-def) } \)
next
fix \( s1 \) \( s2 \) \( s1' \) \( s2' \) act
assume reachable \( (\text{composeALMs } id1 \ id2) \ (s1, s2) \) and \( P1a \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( \text{in-trans-comp: } (s1, s2) \) \( \text{act} = \neg \text{composeALMs } id1 \ id2\) \( (s1', s2') \)
have \( P5 \ (s1, s2) \)
proof –
from \( \text{in-trans-comp } \) and \( \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, s2) \) and \( \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1', s2') \) by \( \text{(auto intro: reachable.reachable-n) } \)
with \( \text{reachable } (\text{composeALMs } id1 \ id2) \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( P5\text{-invariant show } P5 \ (s1, s2) \) unfolding \( \text{invariant-def by auto} \)
qed
from \( 0 < id1 \) and \( id1 < id2 \) and \( \text{in-trans-comp } P1a \ (s1', s2') \)
proof \( \text{(rule my-rule2) } \)
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 \( \text{(auto simp add: in-trans-cases-fun-def) } \)
apply \( \text{(auto simp}
add: ALM-trans-def P1a-def P5-def) done

qed

qed

lemma P1b-invariant: [|id1 < id2; id1 ≠ 0]| ==⇒ invariant (composeALMs id1 id2) P1b
proof (rule invariant1, auto)
  fix s1 s2
  assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
  thus P1b (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P1b-def)

next
  fix s1 s2 s1′ s2′ act
  assume reachable (composeALMs id1 id2) (s1, s2) and P1b (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp: (s1, s2) -- act -- composeALMs id1 id2 --> (s1′, s2′)
  have P1a (s1, s2)
  proof –
    from in-trans-comp and reachable (composeALMs id1 id2) (s1, s2) have reachable (composeALMs id1 id2) (s1′, s2′) by (auto intro: reachablereachable-n)
    with reachable (composeALMs id1 id2) (s1, s2) and 0 < id1 and id1 < id2 and P1a-invariant show P1a (s1, s2) 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

lemma P13-invariant: [|id1 < id2; id1 ≠ 0]| ==⇒ invariant (composeALMs id1 id2) P13
proof clarify
  assume id1 < id2 and 0 < id1
  with P11-invariant and P12-invariant have invariant (composeALMs id1 id2) (λ (s1, s2) . P11 (s1, s2) ∧ P12 (s1, s2)) by (auto simp add:invariant-def)
  moreover have ∀ s . P11 s ∧ P12 s → 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 (∃ c . phase s2 c ≠ Sleep) ∧ ¬ initialized s2
    with P12 (s1, s2) and P11 (s1, s2) obtain rs where initValidReqs-prop:initValidReqs s2 ⊆ pendingReqs s1 and l-c-p (initHists s2) = rs ⊎ (hist s1) and set rs ⊆ pendingReqs s1 and distinct rs by (auto simp add:P12-def P11-def postfix-all-def linearizations-def)

  done
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: \([\{id1 < id2; id1 \neq 0\}] \Rightarrow \text{invariant (composeALMs id1 id2)}\)
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-ios-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 \langle id1 < id2; \rangle
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') \in ALM-trans 0 id1 and (s2, Invoke ca ra, s2') \in 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') \in ALM-trans 0 id1 and (s2, Switch ca id1 h ra, s2') \in ALM-trans id1 id2
thus P14 (s1', s2') by (auto simp add: ALM-trans-def P14-def)
next
fix c id' h
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and ($s_2$, Commit $c$ id' $h$, $s_2'$) $\in$ ALM-trans
id1 id2 and id1 $\leq$ id' and id' $< id_2$
thus $P_{14}$ ($s_1$, $s_2'$) by (auto simp add: ALM-trans-def $P_{14}$-def)
next
fix $c$ $h$ $r$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_2$, Switch $c$ id2 $h$
$r$, $s_2'$) $\in$ ALM-trans id1 id2
thus $P_{14}$ ($s_1$, $s_2'$) by (auto simp add: ALM-trans-def $P_{14}$-def)
next
fix $h$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_2$, Linearize id1 $h$, $s_2'$) $\in$ ALM-trans id1 id2
thus $P_{14}$ ($s_1$, $s_2'$) by (auto simp add: ALM-trans-def $P_{14}$-def linearizations-def
postfix-all-def pendingReqs-def)
next
fix $h$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_2$, Initialize id1 $h$, $s_2'$) $\in$ ALM-trans id1 id2
thus $P_{14}$ ($s_1$, $s_2'$) using ($P_{13}$ ($s_1$, $s_2$)) apply (auto simp add: ALM-trans-def
$P_{14}$-def $P_{13}$-def linearizations-def postfix-all-def pendingReqs-def) prefer 2 apply
force apply blast done
next
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_2$, Abort id1, $s_2'$)
$\in$ ALM-trans id1 id2
thus $P_{14}$ ($s_1$, $s_2'$) by (auto simp add: ALM-trans-def $P_{14}$-def)
next
fix $c$ $a$ $t$ $a$ $r$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_1$, Switch $ca$ $0$ $ta$
$r$, $s_1'$) $\in$ ALM-trans 0 id1
thus $P_{14}$ ($s_1'$, $s_2$) by (auto simp add: ALM-trans-def $P_{14}$-def)
next
fix $c$ $a$ $d'$ $h$
assume $P_{14}$ ($s_1$, $s_2$) and id1 $< id_2$ and ($s_1$, Commit $ca$ id' $h$, $s_1'$) $\in$
ALM-trans 0 id1 and id' $< id_1$
thus $P_{14}$ ($s_1'$, $s_2$) by (auto simp add: ALM-trans-def $P_{14}$-def)
next
fix $h$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and in-lin:($s_1$, Linearize
0 $h$, $s_1'$) $\in$ ALM-trans 0 id1
from in-lin have "initialized $s_2$ and hist $s_2$ = [] using $P_6$ ($s_1$, $s_2$); $P_2$
($s_1$, $s_2$); and ($P_10$ ($s_1$, $s_2$); and $P_2$ ($s_1$, $s_2$)) by (auto simp add: ALM-trans-def
$P_{14}$-def $P_6$-def $P_{10}$-def $P_2$-def $P_2$-def)
thus $P_{14}$ ($s_1'$, $s_2$) by (auto simp add: $P_{14}$-def)
next
fix $h$
assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and id1 $< id_2$ and ($s_1$, Initialize 0 $h$, $s_1'$)
$\in$ ALM-trans 0 id1
thus $P_{14}$ ($s_1'$, $s_2$) using ($P_{10}$ ($s_1$, $s_2$)) by (auto simp add: ALM-trans-def
$P_{14}$-def $P_{10}$-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

lemma P15-invariant: |id1 < id2; id1 ≠ 0| ==> invariant (composeALMs id1 id2) P15
proof (rule invariant1, 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 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 \(s_1' \neq s_1\) and \((P15 \ (s_1, s_2))\) 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 \(P15 \ (s_1, s_2)\) and \(0 < id1\) and \(id1 < id2\) and \((s_1, \text{Switch ca id1 h ra}, s_1') \in \text{ALM-trans}\ 0\ id1\) and \((s_2, \text{Switch ca id1 h ra}, s_2') \in \text{ALM-trans}\ id1\ id2\)

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

next

fix c id h

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

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

next

fix c h r

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

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

next

fix h

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

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

proof

\((\text{auto simp add: P15-def})\)

fix r

assume phase \(s_2'\) \((\text{request-snd r}) = \text{Sleep}\) and \(r \in \text{set}\ \text{hist}\ s_2'\) and \(r \notin\)

pendingReqs \(s_1\)

show \(r \in \text{set}\ \text{hist}\ s_1\)

proof

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

\((\text{with \ 'P1b (s1, s2)' have r \notin pendingReqs s2 by (auto simp add:pendingReqs-def P1b-def)})\)

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

\((\text{with \ 'phase s_2 \ (request-snd r) = Sleep' and \ 'P15 (s1, s2)' and \ (r \notin pendingReqs s1) show \ (?\)thesis by (auto simp add:P15-def)})\)

qed

qed

next

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

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

next

fix h

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

show \(P15 \ (s_1, s_2')\)
proof (auto simp add:P15-def)
  fix r
  assume phase s2' (request-snd r) = Sleep and r ∈ set (hist s2') and r ∉ pendingReqs s1
  show r ∈ set (hist s1)
  proof
    from in-init and P13 (s1, s2)
    have hist s2' ∈ postfix-all (hist s1) (linearizations (pendingReqs s1)) by (auto simp add:ALM-trans-def P13-def)
    with ⟨r ∈ set (hist s2')⟩ have r ∈ set (hist s1) ∨ r ∈ pendingReqs s1 by (auto simp add:postfix-all-def linearizations-def)
    with ⟨r ∉ pendingReqs s1⟩ show ?thesis by auto
  qed
qed

next
fix catara
assume (s1, Switch ca 0 ta ra, s1') ∈ ALM-trans 0 id1
  hence s1' = s1 using P5 (s1, s2) by (auto simp add: ALM-trans-def P5-def)
thus P15 (s1', s2) using P15 (s1, s2) by auto
next
fix ca id'h
  assume P15 (s1, s2) and id1 < id2 and (s1, Commit ca id' h, s1') ∈ ALM-trans 0 id1 and id' < id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
next
fix h
  assume P15 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Linearize 0 h, s1') ∈ ALM-trans 0 id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def postfix-all-def)
next
fix h
  assume (s1, Initialize 0 h, s1') ∈ ALM-trans 0 id1
  hence s1' = s1 using P10 (s1, s2) by (auto simp add: ALM-trans-def P10-def)
thus P15 (s1', s2) using P15 (s1, s2) by auto
next
  assume P15 (s1, s2) and 0 < id1 and id1 < id2 and (s1, Abort 0, s1') ∈ ALM-trans 0 id1
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
qed
qed

4.5 The refinement proof

definition ref-mapping :: (ALM-state * ALM-state) => ALM-state
  — The refinement mapping between the composition of two ALMs and a single
where
ref-mapping \equiv \lambda (s1, s2).

\begin{align*}
\text{pending} &= \lambda c. (\text{if phase } s1 c \neq \text{Aborted} \text{ then pending } s1 c \text{ else pending } s2 c), \\
\text{initHist} &= \{\}, \\
\text{phase} &= \lambda c. (\text{if phase } s1 c \neq \text{Aborted} \text{ then phase } s1 c \text{ else phase } s2 c), \\
\text{hist} &= (\text{if hist } s2 = [] \text{ then hist } s1 \text{ else hist } s2), \\
\text{aborted} &= \text{aborted } s2, \\
\text{initialized} &= \text{True}
\end{align*}

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

\text{— The composition theorem}

\textbf{proof —}

\text{assume } id1 \neq 0 \text{ and } id1 < id2

\text{show } \text{composeALMs } id1 \ id2 = <| \text{ALM-ioa } 0 \ id2

\text{proof (simp add: ioa-implements-def, rule conjI, rule-tac[2] conjI)}

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

\text{— First we show that both automata have the same input and output signature}

\text{using } (id1 \neq 0 \text{ and } id1 < id2) \text{ 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)}

\text{from } id1 \neq 0 \text{ and } id1 < id2;

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

\text{— Then we show that output signatures match}

\text{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)}

\text{show traces (composeALMs } id1 \ id2) <= traces (ALM-ioa } 0 \ id2)

\text{— Finally we show trace inclusion}

\text{proof (rule trace-inclusion[where } f=\text{ref-mapping}]\}

\text{— We use the mapping ref-mapping, defined before}

\text{from same-input-sig and same-output-sig show ext (composeALMs } id1 \ id2)

\text{= ext (ALM-ioa } 0 \ id2)

\text{— First we show that they have the same external signature}

\text{by (simp add: externals-def)}

\textbf{next}

\text{show is-ref-map ref-mapping (composeALMs } id1 \ id2) (ALM-ioa } 0 \ id2)

\text{— Then we show that ref-mapping-comp is a refinement mapping}

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

\textbf{proof —}

\text{— First we show that start states correspond}

\text{fix s1 s2}

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

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

\textbf{next}
— Then we show the main property of a refinement mapping

\[
\text{fix } s_1 s_2 s_1' s_2' \text{ act} \\
\text{assume reachable:reachable (composeALMs id1 id2) } (s_1, s_2) \text{ and in-trans-comp:(s_1, s_2) } \Rightarrow \quad \text{composeALMs id1 id2-- } (s_1', s_2')
\]

We make the invariants available for later use

\[
\text{have } P_6 (s_1, s_2) \text{ and } P_6 (s_1', s_2') \text{ and } P_9 (s_1, s_2) \text{ and } P_7 (s_1, s_2) \text{ and } P_{10} (s_1, s_2) \text{ and } P_4 (s_1, s_2) \text{ and } P_5 (s_1, s_2) \text{ and } P_{13} (s_1, s_2) \text{ and } P_{1a} (s_1, s_2) \text{ and } P_{14} (s_1, s_2) \text{ and } P_{14} (s_1', s_2') \text{ and } P_{15} (s_1, s_2) \text{ and } P_2 (s_1, s_2) \text{ and } P_3 (s_1, s_2)
\]

\text{proof —}

\text{from reachable and in-trans-comp have reachable (composeALMs id1 id2) } (s_1', s_2') \text{ by } (\text{rule reachable.reachable-n)}

\text{with } P_6\text{-invariant and } P_9\text{-invariant and } P_2\text{-invariant and } P_7\text{-invariant and } P_{10}\text{-invariant and } P_4\text{-invariant and } P_5\text{-invariant and } P_{13}\text{-invariant and } P_{1a}\text{-invariant and } P_{14}\text{-invariant and } P_{15}\text{-invariant and } P_3\text{-invariant (id1 } \neq 0) \text{ and (id1 < id2) and reachable}

\text{show } P_6 (s_1, s_2) \text{ and } P_6 (s_1', s_2') \text{ and } P_9 (s_1, s_2) \text{ and } P_7 (s_1, s_2) \text{ and } P_{10} (s_1, s_2) \text{ and } P_4 (s_1, s_2) \text{ and } P_5 (s_1, s_2) \text{ and } P_{13} (s_1, s_2) \text{ and } P_{1a} (s_1, s_2) \text{ and } P_{14} (s_1, s_2) \text{ and } P_{14} (s_1', s_2') \text{ and } P_{15} (s_1, s_2) \text{ and } P_2 (s_1, s_2) \text{ and } P_3 (s_1, s_2) \text{ by } (\text{auto simp add: invariant-def)}

\text{qed}

\text{let } ?t = \text{ref-mapping } (s_1, s_2)
\text{let } ?t' = \text{ref-mapping } (s_1', s_2')
\text{show } \text{EX ex. move (ALM-iao 0 id2) ex } ?t \text{ act } ?t'
— the main part of the proof

\text{proof } (\text{simp add: move-def, auto)}
\text{assume act : ext (ALM-iao 0 id2)}
\text{hence act : } \{\text{act . } \text{EX c r . act = Invoke c r } | (\text{EX t . act = Switch c 0 t r})\} \text{ Un } \{\text{act . } \text{EX c tr . (EX id' . 0 <= id' & id' < id2 & act = Commit c id' tr) | (EX t . act = Switch c id2 tr r)}\} \text{ by } (\text{auto simp add: ALM-iao-def ALM-asig-def externals-def asig-outputs-def asig-of-def)}

\text{with in-trans-comp show } \text{EX ex. is-exec-frag (ALM-iao 0 id2) } (?t, ex) \text{ & Finite ex & laststate } (?t, ex) = ?t' \text{ & mk-trace (ALM-iao 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"

\text{apply auto)}

\text{proof —}
\text{fix } c r
\text{assume in-invoke:(s_1, s_2) --Invoke c r -- composeALMs id1 id2--> (s_1', s_2')}
— If the current action is Invoke
\text{show EX ex. is-exec-frag (ALM-iao 0 id2) } (?t, ex) \text{ & Finite ex & laststate } (?t, ex) = ?t' \text{ & mk-trace (ALM-iao 0 id2)$ex = [Invoke c r]}$
\text{proof —}
\text{let } ?ex = [(\text{Invoke c r, } ?t')!]

\text{have Finite ?ex by auto)
moreover have laststate } (?t, ?ex) = ?t' \text{ by } (\text{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-involve and (id1 ≠ 0) and (id1 < id2) and :P6 (s1′, s2′) have


moreover

{assume s1′ = s1 and s2′ = s2

with in-involve have pre-s1:~(phase s1 c = Ready & request-snd r = c & r \notin set (hist s1)) and pre-s2:~(phase s2 c = Ready & request-snd r = c & r \notin 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-lac!! 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 \notin 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-involve have pre-s1:phase s1 c = Ready & request-snd r = c & r \notin 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 \notin 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 \notin set (hist ?t)

proof (cases hist s2 = [])
assume $\operatorname{hist} s_2 = []$
with pre-$s_1$ show $\theta$thesis by (auto simp add:ref-mapping-def)
next
assume $\operatorname{hist} s_2 \neq []$
show $r \notin \operatorname{set}(\operatorname{hist} ?t)$
proof auto
assume $r \in \operatorname{set}(\operatorname{hist} ?t)$
with $\langle \operatorname{hist} s_2 \neq [] \rangle$ have $r \in \operatorname{set}(\operatorname{hist} s_2)$ by (auto simp add:ref-mapping-def)
moreover from pre-$s_1$ and $(P6 \ (s_1, s_2))$ have phase $s_2$
(request-snd $r) = \operatorname{Sleep}$ by (force simp add:P6-def)
moreover note $(P15 \ (s_1, s_2))$
ultimately have $r \in \operatorname{set}(\operatorname{hist} ?t) \lor r \in \operatorname{pendingReqs} s_1$
by (auto simp add:P15-def)
with pre-$s_1$ have $r \in \operatorname{pendingReqs} s_1$ by auto
with $(P1a \ (s_1, s_2))$ and pre-$s_1$ show False by (auto simp add:pendingReqs-def P1a-def)
qed
ultimately show $\theta$thesis by auto
qed
moreover from pre-$s_1$ and trans-$s_1$ and $(s_2' = s_2)$ have trans-$t$: $?t'$
= $?t$$\langle \operatorname{pending} := (\operatorname{pending} ?t)(c := r), \operatorname{phase} := (\operatorname{phase} ?t)(c := \operatorname{Pending}) \rangle$ by
(auto simp add:ref-mapping-def fun-eq-iff)
ultimately have $\theta$thesis apply (simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp add:ALM-trans-def) done
}
moreover
{
assume $s_1' = s_1$ and $s_2' \neq s_2$
with in-invoke and $(\langle id1 \neq 0 \rangle)$ have pre-$s_2$: phase $s_2$ c =
Ready & request-send $r = c \land r \notin \operatorname{set}(\operatorname{hist} s_2)$ and trans-$s_2$: $s_2' = s_2$$\langle \operatorname{pending} := (\operatorname{pending} s_2)(c := r), \operatorname{phase} := (\operatorname{phase} s_2)(c := \operatorname{Pending}) \rangle$ 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-$s_2$ and $(P6 \ (s_1, s_2))$ have aborted-$s_1$-c: phase $s_1$ c =
Aborted by (auto simp add: P6-def)
with pre-$s_2$ and $(P3 \ (s_1, s_2))$ and $(P14 \ (s_1, s_2))$ have pre-$t$:phase $?t$ c = Ready & request-send $r = c \land r \notin \operatorname{set}(\operatorname{hist} ?t)$ apply (auto simp add: fun-eq-iff ref-mapping-def P3-def P14-def) done
moreover have trans-$t$: $?t' = ?t$$\langle \operatorname{pending} := (\operatorname{pending} ?t)(c := r), \operatorname{phase} := (\operatorname{phase} ?t)(c := \operatorname{Pending}) \rangle$ using aborted-$s_1$-c and $(s_1' = s_1)$ and trans-$s_2$ apply(force simp add: fun-eq-iff ref-mapping-def) done
ultimately have $\theta$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 \(c\ h\ r\)
assume in-switch:(\(s_1, s_2\)) \(\rightarrow\) \(\text{Switch} \ c 0 \ h \ r\) \(\rightarrow\) composeALMs \(id_1\ id_2\) \(\rightarrow\) \((s_1', s_2')\)
— If we get a switch 0 input (nothing happens)
show EX \(ex\). is-exec-frag \((\text{ALM-\(ioa\) 0 \(id_2\))} \ (?t, \(ex\)) \& Finite \(ex\) \& laststate \((?t, \(ex\)) = ?t' \& \text{mk-trace} \((\text{ALM-\(ioa\) 0 \(id_2\))} \$(ex) = [\text{Switch} \ c 0 \ h \ r!]
proof –
let \(?ex = \{(\text{Switch} \ c 0 \ h \ r, \ ?t')\}]
have Finite \(?ex\) by auto
moreover have laststate \((?t, \(?ex\)) = ?t' by (simp add: laststate-def)
moreover have \text{mk-trace} \((\text{ALM-\(ioa\) 0 \(id_2\))} \$(?ex) = [\text{Switch} \ c 0 \ h \ r!]
by (simp add: \text{mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-\(ioa\)-def ALM-asig-def})

moreover have is-exec-frag \((\text{ALM-\(ioa\) 0 \(id_2\))} \ (?t, \(?ex\))
proof –
from in-switch and \(\langle \text{id_1} \neq 0 \rangle\) and \(\langle \text{id_1} < \text{id_2} \rangle\) and \(\langle P5 \ (s_1, s_2) \rangle\) have \(s_1' = s_1\) and \(s_2' = s_2\) and \(\langle c . \text{phase} \ s_1 \ c \neq \text{Sleep} \rangle\) apply (simp-all add: composeALMs-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 \(?t = ?t' \& \langle c . \text{phase} \ ?t \ c \neq \text{Sleep} \rangle\) using \(\langle P6 \ (s_1, s_2) \rangle\)
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:(\(s_1, s_2\)) \(\rightarrow\) \(\text{Switch} \ c \text{id_2} \ h \ r\) \(\rightarrow\) composeALMs \(id_1\ id_2\) \(\rightarrow\) \((s_1', s_2')\)
— The case when the system switches to a third, new, instance
show EX \(ex\). is-exec-frag \((\text{ALM-\(ioa\) 0 \(id_2\))} \ (?t, \(ex\)) \& Finite \(ex\) \& laststate \((?t, \(ex\)) = ?t' \& \text{mk-trace} \((\text{ALM-\(ioa\) 0 \(id_2\))} \$(ex) = [\text{Switch} \ c \text{id_2} \ h \ r!]
proof –
let \(?ex = \{(\text{Switch} \ c \text{id_2} \ h \ r, \ ?t')\}]
have Finite ?ex by auto
moreover have laststate \((?t, \(?ex\)) = ?t' by (simp add: laststate-def)

33
moreover have \( 	ext{mk-trace} (\text{ALM-ioa} \ 0 \ id2) \) \((\text{?ex}) = [\text{Switch} \ c \ id2 \ h \ r!] \)
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 \( \text{is-exec-frag} (\text{ALM-ioa} \ 0 \ id2) \) \((?t, \ ?ex) \)
proof
  from \( \text{in-switch} \) and \((\text{id1} < \text{id2}) \) 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)
from \( (\text{id1} \neq 0) \) and \((\text{id1} < \text{id2}) \) in-switch have \( \text{pre-s2}: \text{aborted} \ s2 \) & phase \( \ ?t \ c = \text{Pending} \) & \( r = \text{pending} \ s2 \) & (if initialized \( s2 \) then \( h \in \text{postfix-all} \ (\text{hist} \ s2) \) (linearizations (pendingReqs \( s2 \))) else \( h : \text{postfix-all} \ (l-c-p \ (\text{initHists} \ s2)) \) (linearizations (initValidReqs \( s2 \)))) and trans-s2: \( s2' = s2 \)\((\text{phase} = (\text{phase} \ s2) (\text{c} := \text{Aborted})) \) apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def-actions-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def)
apply(auto simp add: P6-def)
done
from \( \text{pre-s2} \) have \( \text{aborted} \) \((?s1, ?s2) \)
apply(auto simp add: P6-def)
done
have \( \text{pre-t}: \text{aborted} \ ?t \) & phase \( \ ?t \ c = \text{Pending} \) & initialized \( ?t \) & \( h : \text{postfix-all} \ (\text{hist} \ ?t) \) (linearizations (pendingReqs \( ?t))) & \( r = \text{pending} \ ?t \ c \)
proof
  from \( \text{pre-s2} \) aborted \( ?t \) & pending \( ?t \ c = r \)
and phase \( \ ?t \ c = \text{Pending} \) and initialized \( ?t \) by (auto simp add: ref-mapping-def fun-eq-iff)
moreover have \( h : \text{postfix-all} \ (\text{hist} \ ?t) \) (linearizations (pendingReqs \( ?t)))
proof
  from \( \text{pre-s2} \) have \( (\text{if initialized} \ s2 \) then \( h : \text{postfix-all} \ (\text{hist} \ s2) \) (linearizations (pendingReqs \( s2))) \) else \( h : \text{postfix-all} \ (l-c-p \ (\text{initHists} \ s2)) \) (linearizations (initValidReqs \( s2))) \) by auto
  thus \( ?t\)thesis
proof auto
assume case1-1: initialized \( s2 \) and case1-2: \( h : \text{postfix-all} \ (\text{hist} \ s2) \) (linearizations (pendingReqs \( s2)))
  hence suffixeq (\( \text{hist} \ s1 \)) (\( \text{hist} \ s2 \)) using P14 (\( s1, s2 \)) by (auto simp add: P14-def suffixeq-def)
  show \( h \in \text{postfix-all} \ (\text{hist} \ ?t) \) (linearizations (pendingReqs \( ?t)))
proof
  have \( \text{hist} \ \?t = \text{hist} \ s2 \)
  proof (cases \( \text{hist} \ s2 = [] \))
    assume \( \text{hist} \ s2 = [] \)
    show \( \text{hist} \ \?t = \text{hist} \ s2 \)
  proof
    from \( \text{hist} \ s2 = [] \) and \( \text{suffixeq} \ (\text{hist} \ s1) \) (\( \text{hist} \ s2) \) have \( \text{hist} \ s1 = [] \) by (auto simp add: suffixeq-def)
    with \( \text{hist} \ s2 = [] \) show \( \text{hist} \ \?t = \text{hist} \ s2 \) by (auto simp add: ref-mapping-def)
    qed
  next

  qed

next

34
assume \( \text{hist } s2 \neq [] \)
thus \( \text{hist } ?t = \text{hist } s2 \) by (simp add:ref-mapping-def)
qed

moreover have \( \text{pendingReqs } s2 \leq \text{pendingReqs } ?t \)
proof (simp add: pendingReqs-def, clarify)
fix \( c \)

assume \( \text{pending } s2 \ c \notin \text{set } (\text{hist } s2) \) and \( \text{phase } s2 \ c = \text{Pending} \lor \text{phase } s2 \ c = \text{Aborted} \)
moreover with \( \langle P6 \ (s1, s2) \rangle \) have \( \text{phase } s1 \ c = \text{Aborted} \)
by (auto simp add: P6-def)

moreover note \( \langle \text{suffixeq } (\text{hist } s1) \ (\text{hist } s2) \rangle \)
ultimately show \( \exists ca. \text{pending } s2 \ c = \text{pending } ?t \ ca \land \text{pending } s2 \ c \notin \text{set } (?t ) \land (\text{phase } ?t \ ca = \text{Pending} \lor \text{phase } ?t \ ca = \text{Aborted}) \)
apply (simp add:ref-mapping-def suffixeq-def) by (metis prefixeq-Nil prefixeq-def self-append-conv2)
qed

moreover note \( \text{case1-2} \)
ultimately show \( ?\text{thesis} \) by (auto simp add: linearizations-def postfix-all-def)
qed

next
assume \( \text{case2-1}: \neg \text{initialized } s2 \) and \( \text{case2-2}: \text{h : postfix-all } (l-c-p \ (\text{initHists } s2)) \) (linearizations \( (\text{initValidReqs } s2) \))
from \( \text{case2-1} \) and \( \langle P10 \ (s1, s2) \rangle \) have \( \text{hist } s2 = [] \) by (auto simp add:P10-def)

have \( h : \text{postfix-all } (\text{hist } s1) \) (linearizations \( (\text{pendingReqs } s1) \))
proof –
  from \( \text{pre-s2} \) have \( \text{phase } s2 \ c \neq \text{Sleep} \) by auto
moreover note \( \langle P13 \ (s1, s2) \rangle \) and \( \text{case2-1} \) and \( \text{case2-2} \)
ultimately show \( ?\text{thesis} \) by (auto simp add:P13-def)
qed

moreover from \( \langle \text{hist } s2 = [] \rangle \) have \( \text{hist } ?t = \text{hist } s1 \) by (auto simp add:P10-def ref-mapping-def)
moreover have \( \text{pendingReqs } ?t = \text{pendingReqs } s1 \)
proof auto
fix \( r \)
assume \( r \in \text{pendingReqs } ?t \)
with this obtain \( c' \) where \( r = \text{pending } ?t \ c' \) and \( r \notin \text{set } (\text{hist } ?t) \) and \( \text{phase } ?t \ c' \in \{ \text{Pending}, \text{Aborted} \} \) by (auto simp add:pendingReqs-def)
show \( r \in \text{pendingReqs } s1 \)
proof (cases phase \( s1 \ c' = \text{Aborted} \))
  assume \( \text{phase } s1 \ c' = \text{Aborted} \)
  with \( \langle \text{phase } ?t \ c' \in \{ \text{Pending}, \text{Aborted} \} \rangle \) and \( \langle r = \text{pending } ?t \ c' \rangle \) have \( \text{phase } s2 \ c' \in \{ \text{Pending}, \text{Aborted} \} \) and \( r = \text{pending } s2 \ c' \) by (auto simp add:ref-mapping-def)
  with \( \langle P6 \ (s1, s2) \rangle \) and \( \text{case2-1} \) and \( \langle P7 \ (s1, s2) \rangle \) and \( \langle \text{hist } ?t = \text{hist } s1 \rangle \) and \( \langle r \notin \text{set } (\text{hist } ?t) \rangle \) have \( \text{phase } s1 \ c' = \text{Aborted} \) and \( r = \text{pending } s1 \ c' \) and \( r \notin \text{set } (\text{hist } s1) \) apply (auto simp add: P6-def P7-def) apply force apply force done

35
thus \( \text{thesis} \) by (auto simp add: pendingReqs-def)

next
  assume phase \( s1 \) \( c' \neq \) Aborted
  with \( r = \text{pending} \ ?t \ c' \) and \( r \notin \text{set} \ (\text{hist} \ ?t) \) and \( \text{phase} \ ?t \ c' \in \{\text{Pending}, \text{Aborted}\} \) and \( \text{hist} \ ?t = \text{hist} \ s1 \) show \( \text{thesis} \) by (auto simp add: ref-mapping-def pendingReqs-def)
  qed
next
  fix \( r \)
  assume \( r \in \text{pendingReqs} \ s1 \)
  with this obtain \( c \) where \( r = \text{pending} \ s1 \ c \) and \( \text{phase} \ s1 \ c \in \{\text{Pending}, \text{Aborted}\} \) and \( r \notin \text{set} \ (\text{hist} \ s1) \) by (auto simp add: pendingReqs-def)
  with \( \text{hist} \ s2 = [] \) and \( \neg \text{initialized} \ s2 \) and \( \text{P7} (s1, s2) \) show \( r \in \text{pendingReqs} \ ?t \) by (auto simp add: ref-mapping-def pendingReqs-def P7-def)
  qed
ultimately show \( \text{thesis} \) by (auto simp add: postfix-all-def linearizations-def)
  qed
ultimately show \( \text{thesis} \) by auto
  qed
moreover have trans-\( t; ?t' = ?t (\text{phase} := (\text{phase} \ ?t)(c := \text{Aborted})) \)
  using \( s1\)-aborted and \( s1' = s1 \) and trans-s2 by (auto simp add: ref-mapping-def fun-eq-iff)
ultimately show \( \text{thesis} \) using \( \text{id1 < id2} \) 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 \( \text{thesis} \) by (auto intro: exI [where \( x = ?ex \)])
  qed
next
  fix \( c \ h \ id' \)
  assume in-commit: \( (s1, s2) \rightarrow \text{Commit} \ c \ id' \ h \rightarrow \text{composeALMs} \ id1 \ id2 \rightarrow (s1', s2') \) and \( id' < id2 \)
  -- Case when the composition commits a request
  show \( \exists \ ex. \ \text{is-exec-frag} (\text{ALM-ioa} \ 0 \ id2) \ (\text{\( ?t, ex \))} \land \text{Finite} \ ex \land \text{laststate} \ (\text{\( ?t, ex \))} = \text{\( ?t' \land mk\text{-}trace} (\text{ALM-ioa} \ 0 \ id2)\cdot ex = [\text{Commit} \ c \ id' \ h] \)
  proof
    let \( ?ex = [(\text{Commit} \ c \ id' \ h, \text{\( ?t'\))] \)
    have Finite \( ?ex \) by auto
    moreover have laststate \( (\text{\( ?t, ?ex \))} = \text{\( ?t' \) by (simp add: laststate-def) \)
      moreover have mk-trace \( (\text{ALM-ioa} \ 0 \ id2)\cdot \text{\( ?ex \) = \text{[Commit} \ c \ id' \ h]} \) using \( \text{id'} < id2 \) 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 \( (\text{ALM-ioa} \ 0 \ id2) \ (\text{\( ?t, ?ex \))} \)
      proof

\[
\begin{proof}
\begin{align*}
\{ & \text{assume } id' \prec id \\
\text{with } \text{in-commit} \ & \text{have } s2' = s2 \text{ and } pre-s1: \text{phase } s1 c = C = \text{Pending} \\
\text{and } \text{pending } s1 c \in \{ \text{hist } s1 \} \land h = \text{dropWhile} (\lambda r . r \neq \text{pending } s1 c) (\text{hist } s1) \} \quad \text{and } trans-s1:s1' = s1 \quad \text{apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply (auto simp add: ALM-trans-def) done}
\end{align*}
\end{proof}

\begin{proof}
\begin{align*}
\text{from } pre-s1 & \text{ have } s1\text{-not-aborted-c:phase } s1 c \neq \text{Aborted} \text{ by auto} \\
\text{have } pre-t: \text{phase } ?t c = \text{Pending} & \text{ pending } ?t c \in \{ \text{hist } ?t \} \land h = \text{dropWhile} (\lambda r . r \neq \text{pending } ?t c) (\text{hist } ?t) \\
\text{proof} & \langle \text{cases hist } s2 = [] \rangle \\
\text{assume } & \text{hist } s2 = [] \text{ with pre-s1 and } \langle \text{phase } s1 c \neq \text{Aborted} \rangle \text{ show } ?thesis \text{ by (auto simp add: ref-mapping-def) next} \\
\text{assume } & \text{hist } s2 \neq [] \\
\text{hence } initialized & \text{ s2 using } \langle \text{P10 } s1, s2 \rangle \text{ by (auto simp add: P10-def) from pre-s1 and } \langle \text{phase } s1 c \neq \text{Aborted} \rangle \text{ have phase } ?t c = \text{Pending} \\
\text{and } pending & \langle ?t c = \text{pending } s1 c \text{ and pending } s1 c \in \{ \text{hist } s1 \} \langle \text{auto simp add: ref-mapping-def} \rangle \\
\text{moreover have } pending & \langle ?t c \in \{ \text{hist } ?t \} \rangle \\
\text{proof} \quad & \langle \text{from } \langle \text{initialized } s2 \rangle \text{ and } \langle \text{P14 } s1, s2 \rangle \text{ obtain } rs3 \text{ where hist s2 = rs3 @ } \langle \text{hist } s1 \rangle \text{ by (auto simp add: P14-def) with } \langle \text{pending } s1 c \in \{ \text{hist } s1 \} \rangle \text{ and } \langle \text{hist } s2 = \text{rs3 @ hist s1} \rangle \text{ and } \langle \text{pending } ?t c = \text{pending } s1 c \rangle \text{ show } ?thesis \langle ?t \rangle \text{ by (auto simp add: ref-mapping-def suffizeq-def) qed} \\
\text{moreover have } h = & \text{dropWhile } (\lambda r . r \neq \text{pending } ?t c) (\text{hist } ?t) \\
\text{proof} \quad & \langle \text{from } \langle \text{pending } s1 c \in \{ \text{hist } s1 \} \rangle \text{ obtain } rs1 \text{ rs2 where hist s1 = rs2 @ rs1 and } \text{hd } rs1 = \text{pending } s1 c \text{ and } rs1 \neq [] \text{ and pending } s1 c \notin \{ \text{set rs2 by (metis list.sel(1) in-set-decomp-first list.sel(3))} \rangle \\
\text{with } \langle \text{pending } ?t c = \text{pending } s1 c \rangle \text{ and } \langle \text{dropWhile-lemma}[of hist s1 rs1 pending s1 c] \rangle \text{ and pre-s1 have } h = \text{rs1 by auto} \\
\text{moreover have } & \text{dropWhile } (\lambda r . r \neq \text{pending } ?t c) (\text{hist } ?t) = \text{rs1} \\
\text{proof} \quad & \langle \text{from } \langle \text{initialized s2} \rangle \text{ and } \langle \text{P14 } s1, s2 \rangle \text{ obtain } rs3 \text{ where hist s2 = rs3 @ hist s1} \text{ and set rs3 } \cap \{ \text{hist } s1 \} = [] \text{ by (auto simp add: P14-def) with } \langle \text{pending } s1 c \in \{ \text{hist } s1 \} \rangle \text{ and } \langle \text{hist } s1 = rs2 @ rs1 \rangle \\
\text{have } & \text{hist s2 = rs3 @ rs2 @ rs1 and pending s1 c } \notin \{ \text{set rs3 by auto} \\
\text{with } \langle \text{pending } s1 c \notin \{ \text{set rs3} \} \rangle \text{ obtain } rs4 \text{ where } \text{hist s2 = rs4} \at \text{rs1 } \text{and pending } s1 c \notin \{ \text{set rs4 by auto} \\
\text{with } \langle \text{hd } rs1 = \text{pending } s1 c \rangle \text{ and } \langle rs1 \neq [] \rangle \text{ and } \langle \text{dropWhile-lemma}[of hist s2 rs1 pending s1 c] \rangle \text{ have } \text{dropWhile } (\lambda r . r \neq \text{pending } s1 c) (\text{hist } s2) = \text{rs1} \text{ by auto} \\
\text{thus } ?thesis \text{ using } \langle \text{hist s2 } \neq [] \rangle \text{ and } \langle \text{pending } ?t c = \text{pending }}
\end{proof}
\]

37
ultimately show \( \text{thesis} \) by auto

qed

ultimately show \( \text{thesis} \) by auto

qed

moreover from \( s_2' = s_2 \) and \( s_1\text{-not-aborted-c} \) and \( \text{trans-s1} \)

have \( \text{trans-t'} \): \( ?t' = ?t \ (\{\text{phase} := (\text{phase } ?t)(c := \text{Ready})\}) \) by (simp add: fun-eq-iff ref-mapping-def)

ultimately have \( \text{thesis} \) using \( \langle id_1 < id_2 \rangle \) as 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 \( id_1 \leq id' \)

with in-commit have \( s_1' = s_1 \) and \( \text{pre-s2:phase s2 c = Pending \wedge pending s2 c \in set (hist s2)} \) and \( \text{h = dropWhile } (\lambda r . r \neq \text{pending s2 c}) (\text{hist s2}) \) and \( \text{trans-s2: s2' = s2 \ (\{\text{phase} := (\text{phase s2})(c := \text{Ready})\})} \)

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) apply (auto simp add: ref-mapping-def)

moreover from \( s_1' = s_1 \) and facts and \( \text{trans-s2} \)

have \( \text{trans-t'}: ?t' = ?t \ (\{\text{phase} := (\text{phase } ?t)(c := \text{Ready})\}) \) by (auto simp add: fun-eq-iff ref-mapping-def)

ultimately have \( \text{thesis} \) using \( \langle id_1 < id_2 \rangle \) as 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 \( \text{thesis} \) using \( \langle id' < id_2 \rangle \) by force

qed

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

qed

qed

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

next

assume \( \text{act} \notin \text{ext} \) (ALM-ioa 0 id2)

— Now the case when the composition takes an action that is not in the external signature of the spec

with \( \text{in-trans-comp} \) and \( \langle id_1 < id_2 \rangle \) and \( \langle id_1 \neq 0 \rangle \) have \( \text{act} : \{ \text{act . act = Abort 0 | act = Abort id1 | (EX c r h . act = Linearize 0 h | act =} \)

38
Linearize id1 h \mid act = \text{Switch } c \text{ id1 } h r \mid act = \text{Initialize } 0 h \mid act = \text{Initialize id1 } h) \text{ by } (\text{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 \text{in-trans-comp show } \exists \text{ex. is-exec-frag } (\text{ALM-ioa 0 id2}) (\text{?t, ex}) \land \text{Finite ex } \land \text{laststate } (\text{?t, ex}) = \text{?t'} \land \text{mk-trace } (\text{ALM-ioa 0 id2}).\text{ex} = \text{nil}

proof auto

assume \text{in-abort:}(s1, s2) -\text{Abort 0 - composeALMs id1 id2} \rightarrow (s1', s2')

— The case where the first Abstract aborts

moreover with (id1 \neq 0) \land (id1 < id2) \land (P6 (s1, s2)) \land (P2 (s1, s2)) have \forall c . phase s1 c \neq \text{Aborted} \land \text{hist s2} = \[] \land \forall c . phase s2 c = \text{Sleep}


moreover note (id1 \neq 0)

ultimately have \text{?t'} = \text{?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 ALM-ioa-def ALM-asig-def) apply (auto simp add: fun-eq-iff ALM-trans-def ref-mapping-def) done

thus \text{?thesis}

proof simp

let \text{?ex} = \text{nil}

have \text{Finite ?ex} by auto

moreover have \text{laststate } (\text{?t, ?ex}) = \text{?t} by (simp add: laststate-def)

moreover have \text{mk-trace } (\text{ALM-ioa 0 id2}).\text{?ex} = \text{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 \text{is-exec-frag } (\text{ALM-ioa 0 id2}) (\text{?t, ?ex}) by (auto simp add: is-exec-frag-def)

ultimately show \exists \text{ex. is-exec-frag } (\text{ALM-ioa 0 id2}) (\text{?t, ex}) \land \text{Finite ex } \land \text{laststate } (\text{?t, ex}) = \text{?t} \land \text{mk-trace } (\text{ALM-ioa 0 id2}).\text{ex} = \text{nil} by (auto intro: exf[where \text{x=?ex}])

qed

next

assume \text{in-abort:}(s1, s2) -\text{Abort id1 - composeALMs id1 id2} \rightarrow (s1', s2')

— The case where the second ALM aborts

show \text{?thesis}

proof

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

have \text{Finite ?ex} by auto

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

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

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

proof

from \text{in-abort and } (id1 \neq 0) \text{ have } s1' = s1 \text{ and pre-s2:~ aborted s2 } \& \text{ (}\exists c . phase s2 c \neq \text{Sleep}) \text{ and trans-s2:s2' = s2[aborted:= True]} apply (simp-all

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)


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

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 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

40
assume \( x \in \text{pendingReqs s1} \)

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

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

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

qed

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

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

qed

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

qed

next

fix \( h \)

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

— If the composition executes Linearize id1

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

have Finite ?ex by auto

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

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

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

proof –

from in-lin and \( \langle \text{id1 \neq 0} \rangle \) have \( s1' = s1 \) and \( \text{pre-s2: initialized s2} \)
\( \land \neg \text{aborted } s_2 \land h \in \text{postfix-all} \ (\text{hist } s_2) \) (linearizations (pendingReqs \( s_2 \))) \land 
\text{trans-s}\_2:\ s_2' = s_2[\text{hist} := h] \ \text{apply} \ \text{(simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def \( \text{ALM-iaa-def} \ \text{ALM-asig-def} \) \text{apply})} \ (\text{auto simp add:ALM-trans-def}) \ \text{done}

\text{have} \ \text{pre-s}_2:\ \text{initialized } ?t \land \neg \text{aborted } ?t \land h \in \text{postfix-all} \ (\text{hist } ?t) \ (\text{linearizations (pendingReqs } ?t))

\text{proof} –
\text{have} \ \neg \text{aborted } ?t \ \text{and} \ \text{initialized } ?t \ \text{using} \ \text{pre-s}_2 \ \text{by} \ \text{(auto simp add:ref-mapping-def)}

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

\text{proof} –
\text{from} \ \text{pre-s}_2 \ \text{have} \ \text{initialized } s_2 \ \text{by} \ \text{auto}
\text{hence} \ \text{suffixeq} \ (\text{hist } s_1) \ (\text{hist } s_2) \ \text{using} \ \langle \text{P14 } (s_1, s_2) \rangle \ \text{by} \ \text{(auto simp add:P14-def suffixeq-def)}
\text{hence} \ \text{hist } ?t = \text{hist } s_2 \ \text{by} \ \text{(auto simp add:ref-mapping-def)}
\text{moreover have} \ \text{pendingReqs } s_2 \subseteq \text{pendingReqs } ?t
\text{proof} \ \text{auto}
\text{fix } x
\text{assume} \ x \in \text{pendingReqs } s_2
\text{from} \ \text{this} \ \text{obtain} \ c \ \text{where} \ x = \text{pending } s_2 \ c \ \text{and} \ \text{phase } s_2 \ c \in \{\text{Pending, Aborted}\} \ \text{and} \ \text{pending } s_2 \ c \notin \ \text{set} \ (\text{hist } s_2) \ \text{by} \ \text{(auto simp add:pendingReqs-def)}
\text{with} \ \langle \text{P6 } (s_1, s_2) \rangle \ \text{and} \ \text{hist } ?t = \text{hist } s_2 \ \text{by} \ \text{(force simp add:ref-mapping-def P6-def pendingReqs-def)}
\text{qed}
\text{moreover from} \ \text{pre-s}_2 \ \text{have} \ h \in \text{postfix-all} \ (\text{hist } s_2) \ (\text{linearizations (pendingReqs } s_2)) \ \text{by} \ \text{auto}
\text{ultimately show} \ ?\text{thesis} \ \text{by} \ \text{(auto simp add:postfix-all-def linearizations-def)}}
\text{qed}
\text{ultimately show} \ ?\text{thesis by auto}
\text{qed}
\text{moreover have} \ \text{trans-s}_2: \ ?t' = ?t[\text{hist} := h]
\text{proof –}
\text{from} \ \text{pre-s}_2 \ \text{and} \ \text{trans-s}_2 \ \text{have} \ \text{initialized } s_2' \ \text{by} \ \text{auto}
\text{hence} \ \text{suffixeq} \ (\text{hist } s_1') \ (\text{hist } s_2') \ \text{using} \ \langle \text{P14 } (s_1', s_2') \rangle \ \text{by} \ \text{(auto simp add:P14-def suffixeq-def)}
\text{hence} \ \text{hist } ?t' = \text{hist } s_2' \ \text{by} \ \text{(auto simp add:ref-mapping-def)}
\text{with} \ \text{trans-s}_2 \ \text{and} \ (s_1' = s_1) \ \text{show} \ ?\text{thesis by} \ \text{(auto simp add:ref-mapping-def fun-eq-iff)}}
\text{qed}
\text{ultimately show} \ ?\text{thesis apply} \ \text{(simp add: is-exec-frag-def composeALMs-def trans-of-def hide-def \( \text{ALM-iaa-def} \ \text{ALM-asig-def} \) \text{par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def})} \ \text{apply}) \ \text{(auto simp add:ALM-trans-def)} \ \text{done}
\text{qed}

\text{ultimately show} \ ?\text{thesis by} \ \text{(auto intro: exI[where } x=\text{?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)
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 (i-c-p (initHists s1)) (linearizations (initValidReqs s1))))) and trans-s1: s1' = s1 (phase := (phase s1) (c := Abort)) 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' & 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

43
assume \$\text{in-initialize:} (s_1, s_2) \rightarrow \text{Initialize } 0 \cdot \text{composeALMs } id1 \ id2 \rightarrow (s_1', s_2') \\
\text{hence False using } \langle P10 (s_1, s_2) \rangle \text{ apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add: ALM-trans-def P10-def) done}

thus \$?\text{thesis by auto\$}

next

fix \$h\$

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

— If the second ALM of the composition initializes

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

have \$\text{Finite } ?\text{ex by auto}\$

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

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

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

proof –

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

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

proof –

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

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

proof –

from \$\text{pre-s2 have } h \in \text{postfix-all } (l-c-p (\text{initHists } s_2)) (\text{linearizations } (\text{initValidReqs } s_2)) \ \text{and } \neg \text{initialized } s_2 \ \text{and} \ (\exists \ c . \ \text{phase } s_2 \ c \neq \text{Sleep} \text{ by auto with } (P13 (s_1, s_2)) \text{ have } h \in \text{postfix-all } (\text{hist } s_1) (\text{linearizations } (\text{pendingReqs } s_1)) \text{ by (auto simp add:P13-def)}\$

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

moreover have \$\text{pendingReqs } s_1 \subseteq \text{pendingReqs } ?t\$

proof auto

fix \$x\$

assume \$x \in \text{pendingReqs } s_1\$

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

show \$x \in \text{pendingReqs } ?t\$

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

44
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
done
qed
ultimately show ?thesis by (auto intro: exI[where x=?ex])
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


