Abortable Linearizable Modules

Rachid Guerraoui  Viktor Kuncak  Giuliano Losa

May 27, 2015

Abstract

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

Contents

1  Introduction .................................................. 2

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

3  The ALM Automata specification ....................................... 3

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

5  Conclusion .................................................. 46
1 Introduction

Linearizability [2] is a key design methodology for reasoning about imple-
mentations 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, espe-
cially 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 de-
signers 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 opti-
mization 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 specu-
lation 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 specula-
tive 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 anal-
ysis of speculative systems: Instead of having to reason about an entangle-
ment of complex protocols, one can devise several modules with the prop-
erty 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 Compo-
sition 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 

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

end

3 The ALM Automata specification

theory ALM
imports 

begin

typedef client

end
— A non-empty set of clients
typedec 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′′ @ hs ∧ h′′ ∈ hs}
definition ALM-asig :: nat ⇒ nat ⇒ ALM-action signature
— The action signature of ALM automata
— Input actions, output actions, and internal actions
  where
  ALM-asig ≡ λ id1 id2 .
    \{act . ∃ c r h .
      act = Invoke c r | act = Switch c id1 h r},
    \{act . ∃ c h r id′ .
      id1 <= id′ ∧ id′ < id2 ∧ act = Commit c id′ h
      | act = Switch c id2 h r},
    \{act . ∃ h .
      act = Abort id1
      | act = Linearize id1 h\}
\[ \text{act} = \text{Initialize id1 h} \}

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

**definition** pendingReqs :: \textit{ALM-state} ⇒ request set
- the set of requests that have been invoked but that are not yet in the hist parameter
  where
  \[ \text{pendingReqs} \equiv \lambda s . \{ r . \exists c . \]
  \[ r \equiv \text{pending s c} \]
  \[ r \notin \text{set (hist s)} \]
  \[ \land \text{phase s c} \in \{ \text{Pending, Aborted} \} \]

**definition** initValidReqs :: \textit{ALM-state} ⇒ request set
- any request that appears in an init hist after the longest common prefix or that is pending
  where
  \[ \text{initValidReqs} \equiv \lambda s . \{ r . \]
  \[ (r \in \text{pendingReqs s} \lor \exists h \in \text{initHists s . r} \in \text{set h}) \]
  \[ \land r \notin \text{set (l-c-p (initHists s))} \]

**definition** ALM-trans :: nat ⇒ nat ⇒ (\textit{ALM-action, ALM-state})transition set
- the transitions of the ALM automaton
  where
  \[ \text{ALM-trans} \equiv \lambda id1 id2 . \{ \text{trans} . \]
  \[ \text{let} s = \text{fst trans}; s' = \text{snd (snd trans)}; a = \text{fst (snd trans)} \text{ in} \]
  \[ \text{case} a \text{ of Invoke c r \Rightarrow} \]
  \[ \text{if} \text{ phase s c} = \text{Ready} \land \text{request-snd r} = c \land r \notin \text{set (hist s)} \]
  \[ \text{then} s' = s[\text{pending} := \text{pending s}(c := r), \]
  \[ \text{phase} := \text{phase s}(c := \text{Pending})] \]
  \[ \text{else} s' = s \]
  \[ | \text{Linearize i h \Rightarrow} \]
  \[ \text{initialized s} \land \neg \text{aborted s} \]
  \[ \land h \in \text{postfix-all (hist s)} (\text{linearizations (pendingReqs s s))} \]
∧ s′ = s[hist := h]

|Initialize i h ⇒
(∃ c . phase s c ≠ Sleep) ∧ ∼ aborted s ∧ ∼ initialized s
∧ h ∈ postfix-all (l-c-p (initHists s)) (linearizations (initValidReqs s))
∧ s′ = s[hist := h, initialized := True]

|Abort i ⇒
∼ aborted s ∧ (∃ c . phase s c ≠ Sleep)
∧ s′ = s[aborted := True]

|Commit c i h ⇒
phase s c = Pending ∧ pending s c ∈ set (hist s)
∧ h = dropWhile (λ r . r ≠ pending s c) (hist s)
∧ s′ = s[phase := (phase s)(c := Ready)]

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

**definition** ALM-start :: nat ⇒ ALM-state set
— the set of start states

**where**
ALM-start ≡ λ id . { s .
∀ c . phase s c = (if id ≠ 0 then Sleep else Ready)
∧ hist s = []
∧ ∼ aborted s
∧ (if id ≠ 0 then ∼ initialized s else initialized s)
∧ initHists s = {}}

**definition** ALM-ioa :: nat ⇒ nat ⇒ (ALM-action, ALM-state)ioa
— The ALM automaton

**where**
ALM-ioa ≡ λ (id1::nat) id2 .
(ALM-asig id1 id2, ALM-start id1,
type-synonym compo-state = ALM-state × ALM-state

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

where

composeALMs ≡ λ id1 id2.
  {act . EX c tr r . act = Switch c id1 tr r}

end

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

theory CompositionCorrectness
imports ALM
begin

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

4.1 A case split useful in the proofs

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

where

in-trans-cases-fun == % id1 id2 s t .
  (EX ca ra. (fst s, Invoke ca ra, fst t) : ALM-trans 0 id1 & (snd s, Invoke ca ra, snd t) : ALM-trans id1 id2)
  | (EX ca h ra. (fst s, Switch ca id1 h ra, fst t) : ALM-trans 0 id1 & (snd s, Switch ca id1 h ra, snd t) : ALM-trans id1 id2)
  | (EX c id′ h. fst t = fst s & (snd s, Commit c id′ h, snd t) : ALM-trans id1 id2 & id1 <= id′ & id′ < id2)
  | (EX c h r. fst t = fst s & (snd s, Switch c id2 h r, snd t) : ALM-trans id1 id2)
  | (EX h . fst t = fst s & (snd s, Linearize id1 h, snd t) : ALM-trans id1 id2)
  | (fst t = fst s & (snd s, Abort id1, snd t) : ALM-trans id1 id2)
  | (EX h. fst t = fst s & (snd s, Initialize id1 h, snd t) : ALM-trans id1 id2)
  | (EX ca ta ra. (fst s, Switch ca 0 ta ra, fst t) : ALM-trans 0 id1 & snd t = snd s)
  | (EX ca id′ h. (fst s, Commit ca id′ h, fst t) : ALM-trans 0 id1 & snd t = snd s & id′ < id1)
  | (EX h . (fst s, Linearize 0 h, fst t) : ALM-trans 0 id1 & snd t = snd s)
  | (EX h. (fst s, Initialize 0 h, fst t) : ALM-trans 0 id1 & snd t = snd s)
lemma \texttt{composeALMsE}:
\begin{itemize}
\item A rule for decomposing proofs
\end{itemize}
\begin{itemize}
\item \textbf{assumes} \( id1 \models 0 \) and \( id1 < id2 \) and \( \text{in-trans-comp:s} - (a::\text{ALM-action}) - - \text{composeALMs} \)
\item \textbf{shows} \( \text{decomp: in-trans-cases-fun \text{id1} \text{id2} \text{s} \text{t}} \)
\end{itemize}
\begin{itemize}
\item \textbf{proof} --
\item from \( \text{in-trans-comp and (id1 \models 0)} \) and \( (id1 < id2) \)
\item have \( a : \{\text{act . EX c r h id'. 0 <= id' & id' < id2 & (}
\begin{itemize}
\item \text{act = Invoke c r}
\item \text{act = \{Switch c 0 h r, Switch c id1 h r, Switch c id2 h r\}}
\item \text{act = \{Linearize 0 h, Linearize id1 h\}}
\item \text{act = \{Initialize 0 h, Initialize id1 h\}}
\item \text{act = \{Abort 0, Abort id1\}}
\item \text{act = \{Commit c id' h\}}
\end{itemize}
\}\} \text{by (auto simp add: composeALMs-def trans-of-def hide-def ALM-iao-def par-def actions-def asig-inputs-def asig-outputs-def asig-externals-def asig-of-def ALM-asig-def)}
\item with this obtain \( c r h id' \) where \( 0 <= id' & id' < id2 \) & \( a : \{\text{act .}
\begin{itemize}
\item \text{act = Invoke c r}
\item \text{act = \{Switch c 0 h r, Switch c id1 h r, Switch c id2 h r\}}
\item \text{act = \{Linearize 0 h, Linearize id1 h\}}
\item \text{act = \{Initialize 0 h, Initialize id1 h\}}
\item \text{act = \{Abort 0, Abort id1\}}
\item \text{act = \{Commit c id' h\}}
\end{itemize}
\}\} \text{by auto}
\item moreover from \( \text{in-trans-comp and (id1 \models 0)} \) and \( (id1 < id2) \)
\item have
\begin{itemize}
\item \( a = \text{Linearize 0 h | a = Abort 0 | a = Initialize 0 h | a = Switch c 0 h r | (a = \text{Commit c id' h & id' < id1}) \models ((fst s, a, fst t) : ALM-trans 0 id1 & snd s = snd t)} \)
\item and \( (a = \text{Linearize id1 h | a = Abort id1 | a = Initialize id1 h | a = Switch c id2 h r | (a = \text{Commit c id' h & id' < id2}) \models ((fst s = fst t & (snd s, a, snd t) : ALM-trans id1 id2)} \)
\item and \( (a = \text{Switch c id1 h r | a = Invoke c r) \models ((fst s, a, fst t) : ALM-trans 0 id1 \& (snd s, a, snd t) : ALM-trans id1 id2)} \)
\item by (auto simp add: composeALMs-def trans-of-def hide-def ALM-iao-def par-def actions-def asig-inputs-def asig-outputs-def asig-externals-def asig-of-def ALM-asig-def)
\end{itemize}
\item ultimately show \( ?\text{thesis unfolding in-trans-cases-fun-def apply simp by (metis linorder-not-less)} \)
\item \textbf{qed}
\end{itemize}

lemma \texttt{my-rule1:}\([ id1 \neq 0; id1 < id2; s - a - - \text{composeALMs id1 id2} \rightarrow t; \left[\left[\text{in-trans-cases-fun id1 id2 s t}\right]\right] \rightarrow P ] \rightarrow P \text{ by (auto intro: composeALMsE[where s=s and t=t and a=a])}\)

lemma \texttt{my-rule2:}\([ 0 < id1; id1 < id2; s - a - - \text{composeALMs id1 id2} \rightarrow t; \left[\left[\text{in-trans-cases-fun id1 id2 s t}\right]\rightarrow P ] \rightarrow P \text{ by (auto intro: composeALMsE[where}
\[ s = s \text{ and } t = t \text{ and } a = a \]
where

\( P7 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in all } c . \text{phase } s1 c = \text{Aborted} \land \neg \text{initialized } s2 \rightarrow (\text{pending } s2 c = \text{pending } s1 c \land \text{phase } s2 c \in \{\text{Pending, Aborted}\}) \)

\textbf{definition} \( P8 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— Init histories of ALM 2 are built from the history of ALM 1 plus pending requests of ALM 1

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

\textbf{definition} \( P9 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— ALM 2 does not abort before ALM 1 aborts

\textbf{where}
\( P9 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in } \text{aborted } s2 \rightarrow \text{aborted } s1 \)

\textbf{definition} \( P10 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— ALM 1 is always initialized and when ALM 2 is not initialized its history is empty

\textbf{where}
\( P10 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in } \text{initialized } s1 \land (\neg \text{initialized } s2 \rightarrow (\text{hist } s2 = []) \text{)} \)

\textbf{definition} \( P11 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, any request found in init histories after their longest common prefix is pending in ALM 1

\textbf{where}
\( P11 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in } (\exists c . \text{phase } s2 c \neq \text{Sleep}) \land \neg \text{initialized } s2 \rightarrow \text{initValidReqs } s2 \subseteq \text{pendingReqs } s1 \)

\textbf{definition} \( P12 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, the longest common prefix of the init histories of ALM 2 is built from appending a set of request pending in ALM 1 to the history of ALM 1

\textbf{where}
\( P12 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in } (\exists c . \text{phase } s2 c \neq \text{Sleep}) \rightarrow (\exists rs . \text{lin-c-p} (\text{initHists } s2) = rs \circ (\text{hist } s1) \land \text{set } rs \subseteq \text{pendingReqs } s1 \land \text{distinct } rs) \)

\textbf{definition} \( P13 :: (\text{ALM-state} \times \text{ALM-state}) \Rightarrow \text{bool} \)
— After ALM 2 has been invoked and before it is initialized, any history that may be chosen at initialization is a valid linearization of the concurrent history of ALM 1

\textbf{where}
\( P13 := \% s . \text{let } s1 = \text{fst } s; s2 = \text{snd } s \text{ in } \)
\[(\exists \ c \ . \ phase \ s2 \ c \neq \text{Sleep}) \land \neg \text{initialized} \ s2 \) \rightarrow \text{postfix-all} \ (l-c-p \ (\text{initHists} \ s2)) \ (\text{linearizations} \ (\text{initValidReqs} \ s2)) \subseteq \text{postfix-all} \ (\text{hist} \ s1) \ (\text{linearizations} \ (\text{pendingReqs} \ s1))\]

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

where

— The history of ALM 1 is a postfix of the history of ALM 2 and requests appearing in ALM 2 after the history of ALM 1 are not in the history of ALM 1

\[P14 = \% s . \ let \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ in\]

\[(\text{hist} \ s2 \neq [] \lor \text{initialized} \ s2) \rightarrow (\exists \ rs . \ \text{hist} \ s2 = \text{rs} \ @ \ (\text{hist} \ s1) \\
\land \ \text{set} \ \text{rs} \ \cap \ \text{set} \ (\text{hist} \ s1) = \{\})\]

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

where

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

\[P15 = \% s . \ let \ s1 = \text{fst} \ s; \ s2 = \text{snd} \ s \ in\]

\[\forall r . \ let \ c = \text{request-snd} \ r \ \text{inphase} \ s2 \ c = \text{Sleep} \land r \ \in \ \text{set} \ (\text{hist} \ s2) \rightarrow (r \ \in \ \text{set} \ (\text{hist} \ s1) \lor r \ \in \ \text{pendingReqs} \ s1)\]

### 4.4 Proofs of invariance

**lemma** invariant-imp: $[\text{invariant} \ ioa \ P; \ \forall s . \ P \ s \rightarrow Q \ s] \Longrightarrow \text{invariant} \ ioa \ Q$

by (simp add:invariant-def)

**declare** phase.split [split]

**declare** phase.split-asn [split]

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

**declare** ALM-action.split-asn [split]

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

— A useful lemma about truncating histories

**proof** (induct \text{xs}, force)

fix \text{a} \ \text{xs}

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

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

**proof** (rule allI, rule implI, cases \text{a} = \text{x})

fix \text{ys}

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

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

from $x \neq \text{xs} = \text{ys} \ @ \ text{zs} \ \text{and} \ (x \notin \text{set} \ \text{ys}) \ \text{have} \ \text{ys} = [] \ \text{by} \ (\text{metis} \ \text{list.sel}(1))$

**hd-append** $\text{hd-in-set}$

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

next
fix ys
assume a ≠ xs = ys @ zs ∧ hd zs = x ∧ zs ≠ [] ∧ x ≠ set ys and a ≠ x
hence a ≠ xs = ys @ zs and hd zs = x and zs ≠ [] and x ≠ set ys by auto
obtain ys' where xs = ys' @ zs and x ≠ set ys'
proof –
from (a ≠ xs = ys @ zs) and (hd zs = x) and (a ≠ x) obtain ys' where ys = a ≠ ys' apply clarify by (metis Cons-eq-append-conv list.sel(1))
moreover with (x ≠ set ys) have x ≠ set ys' by auto
moreover from (ys = a ≠ ys') and (a ≠ xs = ys @ zs) have xs = ys' @ zs by auto
ultimately show (∀ys'. [xs = ys' @ zs; x ≠ set ys'] ==> thesis) ==> thesis
by auto
qed

lemma P2-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P2
proof (rule invariantI, auto)
fix s1 s2
assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
thus P2 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P2-def)
next
fix s1 s2 s1' s2' act
assume reachable (composeALMs id1 id2) (s1, s2) and P2 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp;(s1, s2) − act−−composeALMs id1 id2−− (s1', s2')
from (0 < id1) and (id1 < id2) and in-trans-comp show P2 (s1', s2')
proof (rule my-rule2)
assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
thus P2 (s1', s2') using P2 (s1, s2) and (0 < id1) and (id1 < id2) apply(auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P2-def) done
qed

lemma P5-invariant: [|id1 < id2; id1 ≠ 0|] ==> invariant (composeALMs id1 id2) P5
proof (rule invariantI, auto)
fix s1 s2
assume (s1, s2) : starts-of (composeALMs id1 id2) and 0 < id1
thus P5 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioa-def par-def ALM-start-def P5-def)
next

12
fix \ s1 \ s2 \ s1' \ s2' \ act
assume reachable (composeALMs id1 id2) (s1, s2) and P5 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) — act — composeALMs id1 id2 —> (s1', s2')
from \ 0 < id1, and \ id1 < id2, and \ in-trans-comp show P5 (s1', s2')
proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
  thus P5 (s1', s2') using \P5 (s1, s2) and \ 0 < id1, and \ id1 < id2, apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def P5-def) done
qed

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

lemma \ P9-invariant: [[id1 < id2; id1 \neq 0]] => invariant (composeALMs id1 id2) P9
proof (rule invariantI, auto)
fix \ s1 \ s2
  assume (s1, s2) : starts-of (composeALMs id1 id2)
  thus P9 (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-iao-def par-def ALM-start-def P9-def)
next
fix \ s1 \ s2 \ s1' \ s2' \ act
  assume reachable (composeALMs id1 id2) (s1, s2) and P9 (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) — act — composeALMs id1 id2 —> (s1', s2')
have $P_6$ ($s_1, s_2$)

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: ALM-trans-def $P_9$-def $P_6$-def) done

qed

lemma $P_{10}$-invariant: [$[id_1 < id_2; id_1 < 0]= 0]$ == invariant (composeALMs id1 id2) $P_{10}$

proof (rule invariant1, 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-koa-def par-def ALM-start-def $P_{10}$-def)

next

fix $s_1$ $s_2$ $s_1'$ $s_2'$ act

assume reachable (composeALMs id1 id2) ($s_1, s_2$) and $P_{10}$ ($s_1, s_2$) and $0 < id_1$ and $id_1 < id_2$ and in-trans-comp:($s_1, s_2$) $-$ act $-$ composeALMs id1 id2 $-$ ($s_1', s_2'$)

from $0 < id_1$ and $id_1 < id_2$ and in-trans-comp show $P_{10}$ ($s_1', s_2'$)

proof (rule my-rule2)

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

thus $P_{10}$ ($s_1', s_2'$) using ($P_{10}$ ($s_1, s_2$): and $0 < id_1$ and $id_1 < id_2$): apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P_{10}$-def) done

qed

lemma $P_{3}$-invariant: [$[id_1 < id_2; id_1 \neq 0]= 0]$ == invariant (composeALMs id1 id2) $P_{3}$

proof (rule invariant1, 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-koa-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 $P10$ ($s1$, $s2$)
proof -
  from in-trans-comp and reachable (composeALMs $id1$ $id2$) ($s1$, $s2$): have reachable (composeALMs $id1$ $id2$) ($s1'$, $s2'$) by (auto intro: reachable,reachable-n)
  with reachable (composeALMs $id1$ $id2$) ($s1$, $s2$) and ($0 < id1$) and ($id1' < id2$) and $P10$-invariant show $P10$ ($s1$, $s2$) unfolding invariant-def by auto
next
from ($0 < id1$) and ($id1' < id2$) and in-trans-comp show $P3$ ($s1'$, $s2'$)
proof (rule my-rule2)
  assume in-trans-cases-fun $id1$ $id2$ ($s1$, $s2$) ($s1'$, $s2'$)
thus $P3$ ($s1'$, $s2'$) using ($P3$ ($s1$, $s2$)) and ($P10$ ($s1$, $s2$)) and ($0 < id1$) and ($id1' < id2$) apply (auto simp add: in-trans-cases-fun-def) apply (auto simp add: ALM-trans-def $P3$-def $P10$-def) done
qed
qed

lemma $P7$-invariant: [|$id1' < id2'; id1' ≠ 0$|] ==> invariant (composeALMs $id1$ $id2$) $P7$
proof (rule invariantI, auto)
  fix $s1$ $s2$
  assume ($s1$, $s2$) : starts-of (composeALMs $id1$ $id2$) and ($0 < id1$)
  thus $P7$ ($s1$, $s2$) by (simp add: starts-of-def composeALMs-def hide-def ALM-ioctl-def par-def ALM-start-def $P7$-def)
next
  fix $s1$ $s2$ $s1'$ $s2'$ act
  assume reachable (composeALMs $id1$ $id2$) ($s1$, $s2$) and $P7$ ($s1$, $s2$) and ($0 < id1$) and ($id1 < id2$) and in-trans-comp ($s1$, $s2$) -- act -- composeALMs $id1$ $id2$ --> ($s1'$, $s2'$)
  have $P6$ ($s1$, $s2$) and $P10$ ($s1$, $s2$)
proof -
  from in-trans-comp and reachable (composeALMs $id1$ $id2$) ($s1$, $s2$): have reachable (composeALMs $id1$ $id2$) ($s1'$, $s2'$) by (auto intro: reachable,reachable-n)
  with reachable (composeALMs $id1$ $id2$) ($s1$, $s2$) and ($0 < id1$) and ($id1 < id2$) and $P6$-invariant and $P10$-invariant show $P6$ ($s1$, $s2$) and $P10$ ($s1$, $s2$) unfolding invariant-def by auto
qed

next
from ($0 < id1$) and ($id1 < id2$) and in-trans-comp show $P7$ ($s1'$, $s2'$)
proof (rule my-rule2)
  assume in-trans-cases-fun $id1$ $id2$ ($s1$, $s2$) ($s1'$, $s2'$)
thus $P7$ ($s1'$, $s2'$) using ($P7$ ($s1$, $s2$)) and ($P6$ ($s1$, $s2$)) and ($0 < id1$) and ($id1' < id2$)
proof (auto simp add: in-trans-cases-fun-def)
  fix $ca$ $ra$
  assume $P7$ ($s1$, $s2$) and $P6$ ($s1$, $s2$) and ($0 < id1$) and ($id1 < id2$) and ($s1$, $s1'$) ∈ ALM-trans 0 $id1$ and ($s2$, $s2'$) ∈ ALM-trans $id1$ $id2$
thus $P7$ ($s1'$, $s2'$) by (auto simp add: ALM-trans-def $P7$-def)
next
  fix $ca$ $h$ $ra$
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 id_1 h ra, $s_1'$) $\in$ ALM-trans 0 id_1 and ($s_2$, Switch ca id_1 h ra, $s_2'$) $\in$ ALM-trans id_1 id_2
thus $P_7\ (s_1', s_2')$ by (auto simp add: ALM-trans-def P7-def P6-def)
next
fix $c\ id'\ h$
assume $P_7\ (s_1, s_2)$ and $P_6\ (s_1, s_2)$ and $0 < id_1$ and ($s_2$, Commit c id' h, $s_2'$) $\in$ ALM-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-trans-def P7-def P10-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-trans id_1 id_2
thus $P_7\ (s_1, s_2')$ by (auto simp add: ALM-trans-def P7-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-trans id_1 id_2
thus $P_7\ (s_1, s_2')$ by (simp add: ALM-trans-def P7-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-trans id_1 id_2
thus $P_7\ (s_1, s_2')$ by (auto simp add: ALM-trans-def P7-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-trans 0 id_1
thus $P_7\ (s_1', s_2)$ by (auto simp add: ALM-trans-def P7-def)
next
fix $ca\ id'\ h$
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' h, $s_1'$) $\in$ ALM-trans 0 id_1 and id' $<$ id_1
thus $P_7\ (s_1', s_2)$ by (auto simp add: ALM-trans-def P7-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-trans 0 id_1
thus $P_7\ (s_1', s_2)$ by (auto simp add: ALM-trans-def P7-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-trans 0 id_1
thus $P_7\ (s_1', s_2)$ by (auto simp add: ALM-trans-def P7-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-trans id_1 id_2
thus $P_7\ (s_1, s_2')$ by (auto simp add: ALM-trans-def P7-def)
next
assume $P7 \ (s_1, s_2)$ and $P6 \ (s_1, s_2)$ and $0 < id1$ and $id1 < id2$ and $(s_1, Abort \ 0, s_1') \in ALM-trans \ 0 \ id1$
thus $P7 \ (s_1', s_2)$ by (auto simp add: ALM-trans-def P7-def)
qed
qed

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

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

from in-trans-comp and reachable (composeALMs id1 id2) (s1, s2) have reachable (composeALMs id1 id2) (s1', s2') by (auto intro: reachable.reachable-n)
with reachable (composeALMs id1 id2) (s1, s2) and (0 < id1) and (id1 < id2) and P6-invariant and P10-invariant and P5-invariant and P4-invariant show P6 (s1, s2) and P10 (s1, s2) and P5 (s1, s2) and P4 (s1, s2) unfolding invariant-def by auto

qed

from (0 < id1) and (id1 < id2) and in-trans-comp show P8 (s1', s2')
proof (rule my-rule2)
assume in-trans-cases-fun id1 id2 (s1, s2) (s1', s2')
thus P8 (s1', s2') using P8 (s1, s2); and (0 < id1) and (id1 < id2)
proof (auto simp add: in-trans-cases-fun-def)
fix ca ra
assume P8 (s1, s2) and 0 < id1 and id1 < id2 and in-inv-one:(s1, Invoke ca ra, s1') \in ALM-trans 0 id1 and in-inv-two:(s2, Invoke ca ra, s2') \in ALM-trans id1 id2
show P8 (s1', s2')
proof (cases s1' = s1)
assume s1' = s1
with in-inv-two and (P8 (s1, s2)) show ?thesis by (auto simp add: ALM-trans-def P8-def)

next
assume s1' \neq s1
with in-inv-one have pendingReqs s1 \subseteq pendingReqs s1' by (force simp add: pendingReqs-def ALM-trans-def)
moreover from in-inv-one have hist s1' = hist s1 by (auto simp add: ALM-trans-def)
moreover from in-inv-two have initHists s2' = initHists s2 by (auto simp add: ALM-trans-def)
moreover note (P8 (s1, s2))
ultimately show ?thesis by (auto simp add: ALM-trans-def P8-def linearizations-def postfix-all-def)

qed

next
fix ca h ra
assume P8 (s1, s2) and 0 < id1 and id1 < id2 and in-switch-one:(s1, Switch ca id1 h ra, s1') \in ALM-trans 0 id1 and in-switch-two:(s2, Switch ca id1 h ra, s2') \in ALM-trans id1 id2
show P8 (s1', s2')
proof (auto simp add: P8-def)
fix h1
assume h1 \in initHists s2'
show h1 \in postfix-all (hist s1') (linearizations (pendingReqs s1'))
proof (cases h1 \in initHists s2)
assume h1 \in initHists s2
moreover from in-switch-one and (0 < id1) have hist s1' = hist s1 and pendingReqs s1' = pendingReqs s1 by (auto simp add: ALM-trans-def pendingReqs-def)
moreover note \( P8 \left( s_1, s_2 \right) \)
ultimately show \( h_1 \in \text{postfix-all} \left( \text{hist} \ s_1' \right) \) (linearizations (pendingReqs s1')) by (auto simp add:P8-def)
next
assume \( h_1 \notin \text{initHists} \ s_2 \)
with \( h_1 \in \text{initHists} \ s_2' \) and \( \text{in-switch-2} \) have \( h_1 = h \) by (auto simp add:ALM-trans-def)
with \( \text{in-switch-1} \) and \( 0 < \text{id}_1 \) and \( P10 \left( s_1, s_2 \right) \) have \( h_1 \in \text{postfix-all} \left( \text{hist} \ s_1 \right) \) (linearizations (pendingReqs s1)) by (auto simp add:ALM-trans-def P10-def)
moreover from \( \text{in-switch-1} \) and \( 0 < \text{id}_1 \) have \( \text{hist} \ s_1 = \text{hist} \ s_1' \) and \( \text{pendingReqs} \ s_1' = \text{pendingReqs} \ s_1 \) by (auto simp add:ALM-trans-def pendingReqs-def)
ultimately show \( \text{thesis} \) by auto
qed
qed
next
fix \( c \ id' \ h \)
assume \( P8 \left( s_1, s_2 \right) \) and \( 0 < \text{id}_1 \) and \( \left( s_2, \text{Commit} \ c \ id' \ h, s_2' \right) \in \text{ALM-trans} \)
\( \text{id}_1 \ \text{id}_2 \) and \( \text{id}_1 < \text{id}_2' \) and \( \text{id}_1 < \text{id}_2 \)
thus \( P8 \left( s_1, s_2' \right) \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( c \ h \ r \)
assume \( P8 \left( s_1, s_2 \right) \) and \( 0 < \text{id}_1 \) and \( \text{id}_1 < \text{id}_2 \) and \( \left( s_2, \text{Switch} \ c \ \text{id}_2 \ h \ r, s_2' \right) \in \text{ALM-trans} \text{id}_1 \\text{id}_2 \)
thus \( P8 \left( s_1, s_2' \right) \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( h \)
assume \( P8 \left( s_1, s_2 \right) \) and \( 0 < \text{id}_1 \) and \( \text{id}_1 < \text{id}_2 \) and \( \left( s_2, \text{Linearize} \ \text{id}_1 \ h, s_2' \right) \in \text{ALM-trans} \text{id}_1 \\text{id}_2 \)
thus \( P8 \left( s_1, s_2' \right) \) by (auto simp add: ALM-trans-def P8-def)
next
fix \( c \ a \ t \ \text{ra} \)
assume \( P8 \left( s_1, s_2 \right) \) and \( 0 < \text{id}_1 \) and \( \text{id}_1 < \text{id}_2 \) and \( \left( s_1, \text{Switch} \ c \ a \ 0 \ \text{ta} \ \text{ra}, s_1' \right) \in \text{ALM-trans} \ 0 \text{id}_1 \)
thus \( P8 \left( s_1', s_2 \right) \) using \( \left( P5 \left( s_1, s_2 \right) \right) \) by (auto simp add: ALM-trans-def P8-def P5-def)
next
fix \( c \ a \ id' \ h \)
assume \( P8 \left( s_1, s_2 \right) \) and \( \text{in-commit-1} \left( s_1, \text{Commit} \ c \ id' \ h, s_1' \right) \in \text{ALM-trans} \ 0 \text{id}_1 \)
from \( \text{in-commit-1} \) have \( \text{pendingReqs} \ s_1' = \text{pendingReqs} \ s_1 \) and \( \text{hist} \ s_1' = \text{hist} \ s_1 \) by (auto simp add: pendingReqs-def ALM-trans-def)
with \( \left( P8 \left( s_1, s_2 \right) \right) \) show \( P8 \left( s_1', s_2 \right) \) by (auto simp add: ALM-trans-def P8-def pendingReqs-def)
19
next
  fix h
  assume \( P_8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Linearize } 0 h, s_1') \in \text{ALM-trans } 0 \ id1 \)
  thus \( P_8 (s_1', s_2) \) using \( P_6 (s_1, s_2) \) and \( P_4 (s_1, s_2) \) by (auto simp add: ALM-trans-def P8-def P6-def P4-def)
next
  assume \( P_8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_2, \text{Abort } id1, s_2') \in \text{ALM-trans } id1 \ id2 \)
  thus \( P_8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def)
next
  fix h
  assume \( P_8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Initialize } 0 h, s_1') \in \text{ALM-trans } 0 \ id1 \)
  thus \( P_8 (s_1', s_2) \) using \( \langle P_10 (s_1, s_2) \rangle \) by (auto simp add: ALM-trans-def P8-def)
next
  assume \( P_8 (s_1, s_2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s_1, \text{Abort } 0, s_1') \in \text{ALM-trans } 0 \ id1 \)
  thus \( P_8 (s_1, s_2') \) by (auto simp add: ALM-trans-def P8-def pendingReqs-def)
qed

lemma \( P_{12}\)-invariant: \( \| id1 < id2; id1 \neq 0 \| \implies \) invariant (composeALMs id1 id2) \( P_{12} \)
proof clarify
  assume \( id1 < id2 \) and \( 0 < id1 \)
  with \( P_8\)-invariant and \( P_4\)-invariant have invariant (composeALMs id1 id2) \( \lambda (s_1, s_2). P_8 (s_1, s_2) \land P_4 (s_1, s_2) \) by (auto simp add: invariant-def)
  moreover have \( \forall s . P_8 s \land P_4 s \rightleftharpoons P_{12} s \)
proof auto
  fix s1 s2
  assume \( P_8 (s_1, s_2) \) and \( P_4 (s_1, s_2) \)
  hence initHists-prop: \( \exists h \in \text{initHists } s_2 . \ (\exists h'. h = h' \circ (\text{hist } s_1) \land \text{set } h' \subseteq \text{pendingReqs } s_1 \land \text{distinct } h') \) by (auto simp add: P8-def postfix-all-def linearizations-def)
  show \( P_{12} (s_1, s_2) \)
proof (simp add:P12-def, rule impI)
  assume \( \exists c . \text{phase } s_2 c \neq \text{Sleep} \)
  with \( P_4 (s_1, s_2) \) have initHists s2 \( \neq \) \{\} by (auto simp add:P4-def)
  with l-c-p-lemma[of initHists s2 hist s1] and initHists-prop obtain rs where l-c-p (initHists s2) = rs \& hist s1 by (auto simp add: suffixeq-def)
  moreover have set rs \subseteq pendingReqs s1
proof -
  from \( \text{initHists } s_2 \neq \{\} \) obtain \( h \) where \( h \in \text{initHists } s_2 \) by auto
  with initHists-prop obtain \( h' \) where \( h = h' \circ (\text{hist } s_1) \land \text{set } h' \subseteq \text{pendingReqs } s_1 \) by auto
moreover from l-c-p-common-postfix[of initHists s2] and \( h \in \text{initHists s2} \) obtain \( h'' \) where \( h = h'' @ (l-c-p (\text{initHists s2})) \) by (auto simp add: common-postfix-p-def suffixeq-def)

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

lemma \( P11\text{-invariant: } [\mid id1 < id2; id1 \neq 0 \mid] \Rightarrow \text{invariant } (\text{composeALMs id1 id2}) P11 \)

proof clarify
    assume \( id1 < id2 \) and \( 0 < id1 \)
with \( P8\text{-invariant and } P12\text{-invariant and } P6\text{-invariant and } P7\text{-invariant} \) have
invariant \( (\text{composeALMs id1 id2}) (\lambda (s1, s2). P8 (s1, s2) \land P12 (s1, s2) \land P6 (s1, s2) \land P7 (s1, s2)) \) by (auto simp add: invariant-def)
moreover have \( \forall s . P8 s \land P12 s \land P6 s \land P7 s \rightarrow P11 s \)

proof auto
    fix \( s1 s2 \)
    assume \( P8 (s1, s2) \) and \( P12 (s1, s2) \) and \( P6 (s1, s2) \) and \( P7 (s1, s2) \)
    show \( P11 (s1, s2) \)
    proof (simp add: P11-def initValidReqs-def, auto)
    fix \( x c h \)
    assume phase \( s2 c \neq \text{Sleep} \)
with \( (P12 (s1, s2) \land P8 (s1, s2)) \) have \( \text{initHists-prop} \forall h \in \text{initHists s2} \).
(\exists h'. h = h' @ (\text{hist s1}) \land \text{set } h' \subseteq \text{pendingReqs s1}) \) and \( lcp-prop: \exists rs . l-c-p (\text{initHists s2}) = rs @ (\text{hist s1}) \) by (auto simp add: P12-def P8-def postfix-all-def linearizations-def)
    assume \( x \notin \text{set } (l-c-p (\text{initHists s2})) \) and \( h \in \text{initHists s2} \) and \( x \in \text{set } h \)
from \( \text{initHists-prop and } h \in \text{initHists s2} \) obtain \( h' \) where \( h = h' @ (\text{hist s1}) \) and \( \text{set } h' \subseteq \text{pendingReqs s1} \) by auto
moreover from \( lcp-prop \) obtain \( rs \) where \( l-c-p (\text{initHists s2}) = rs @ (\text{hist s1}) \) by auto
moreover note \( x \notin \text{set } (l-c-p (\text{initHists s2})) \) and \( x \in \text{set } h \)
ultimately have \( x \in \text{set } h' \) by \texttt{auto}

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

next

fix \( x \in h \)

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

with \( P12 \ (s1, s2) \) have \( \text{lcp-prop} : \exists rs . l-c-p (\text{initHists } s2) = rs @ (\text{hist } s1) \) by \texttt{(auto simp add:P12-def P8-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 \texttt{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 \texttt{(force simp add:pendingReqs-def P6-def)}

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

ultimately show \( \bigwedge c' . \ [\text{phase } s1 c' = \text{Aborted}; x = \text{pending } s1 c'] \implies \text{thesis} \) \( \implies \text{thesis by } \text{auto} \)

qed

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

qed

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

qed

lemma \( P1a\text{-invariant} : \lbrack \lvert id1 < id2 ; id1 \neq 0 \rvert \rbrack \implies \text{invariant } (\text{composeALMs } id1 id2) P1a \)

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

fix \( s1 \) \( s2 \)

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

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

next

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

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

have \( P5 \ (s1, s2) \)

proof

from \( \text{in-trans-comp} \) and \( \text{reachable } (\text{composeALMs } id1 id2) (s1, s2) \) have reachable \( (\text{composeALMs } id1 id2) (s1', s2') \) by \texttt{(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 \( P5 \ (s1, s2) \) \texttt{unfolding invariant-def by auto} qed

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

proof (rule \texttt{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 \texttt{(auto simp add: in-trans-cases-fun-def)} apply \texttt{(auto simp ...
add: \texttt{ALM\-trans-def P1a-def P5-def}) \texttt{done}

\texttt{qed}

\texttt{qed}

\textbf{lemma} \texttt{P1b-invariant:} \texttt{||id1 < id2; id1 \neq 0||} \implies \texttt{invariant (composeALMs id1 id2) P1b}

\texttt{proof (rule invariantI, auto)}

\texttt{fix s1 s2}

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

\texttt{thus P1b (s1, s2) by (simp add: starts-of-def composeALMs-def hide-def ALM-\textit{ioa-def par-def ALM-start-def P1b-def})}

\texttt{next}

\texttt{fix s1 s2 s1' s2' act}

\texttt{assume reachable (composeALMs id1 id2) (s1, s2) and P1b (s1, s2) and 0 < id1 and id1 < id2 and in-trans-comp:(s1, s2) \rightarrow \rightarrow composeALMs id1 id2-> (s1', s2')}

\texttt{have P1a (s1, s2)

\texttt{proof (rule my-rule2)}

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

\texttt{with \texttt{reachable (composeALMs id1 id2) (s1, s2) and 0 < id1 and id1 < id2, and P1a-invariant show P1a (s1, s2) unfolding invariant-def by auto

\texttt{qed

from (0 < id1) and (id1 < id2) and in-trans-comp show P1b (s1', s2')

\texttt{proof (rule my-rule2)}

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

\texttt{thus P1b (s1', s2') using \texttt{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

\texttt{qed

\texttt{qed

\textbf{lemma} \texttt{P13-invariant:} \texttt{||id1 < id2; id1 \neq 0||} \implies \texttt{invariant (composeALMs id1 id2) P13}

\texttt{proof clarify}

\texttt{assume id1 < id2 and 0 < id1}

\texttt{with P11-invariant and P12-invariant have invariant (composeALMs id1 id2) (\lambda (s1, s2). \texttt{P11 (s1, s2) \land P12 (s1, s2)}) by (auto simp add:invariant-def)

\texttt{moreover have \forall s. P11 s \land P12 s \rightarrow P13 s

\texttt{proof auto

\texttt{fix s1 s2}

\texttt{assume P11 (s1, s2) and P12 (s1, s2)

\texttt{show P13 (s1, s2)

\texttt{proof (simp add:P13-def, rule impl)

\texttt{assume (\exists c. phase s2 c \neq Sleep) \land \neg initialized s2

\texttt{with \texttt{P12 (s1, s2) and P11 (s1, s2) obtain rs where initValidReqs-prop:initValidReqs s2 \subseteq pendingReqs s1 and l-c-p (initHists s2) = rs @ hist s1 and set rs \subseteq pendingReqs s1 and distinct rs by (auto simp add:P12-def P11-def postfix-all-def linearizations-def)
moreover from \( (\text{l-c-p} \ (\text{initHists} \ s2)) = \text{rs} \otimes (\text{hist} \ s1) \) have \( \text{initValidReqs} \ s2 \cap \text{set} \ \text{rs} = \{\}\) by (auto simp add: initValidReqs-def)

ultimately show \( \text{postfix-all} \ (\text{l-c-p} \ (\text{initHists} \ s2)) \ (\text{linearizations} \ (\text{initValidReqs} \ s2)) \subseteq \text{postfix-all} \ (\text{hist} \ s1) \ (\text{linearizations} \ (\text{pendingReqs} \ s1)) \) by (force simp add: postfix-all-def linearizations-def)

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

qed

\begin{verbatim}
lemma \( P_{14}\)-invariant: \([|id1 < id2; id1 \neq 0|] \Rightarrow \text{invariant} \ (\text{composeALMs} \ id1 \ id2) \ P_{14}\)
proof (rule invariantI, auto)
  fix \( s1 \ s2 \)
  assume \( (s1, s2) : \text{starts-of} \ (\text{composeALMs} \ id1 \ id2) \) and \( 0 < id1 \)
  thus \( P_{14} \ (s1, s2) \) by (simp add: starts-of-def composeALMs-def hide-def ALM-iaa-def
par-def ALM-start-def P_{14}-def)
next
  fix \( s1 \ s2 \ s1' \ s2' \)
  assume reachable \( (\text{composeALMs} \ id1 \ id2) \ (s1, s2) \) and \( P_{14} \ (s1, s2) \) and \( 0 < id1 \)
and \( id1 < id2 \) and \text{in-trans-comp}(s1, s2) -- \text{act} -- \text{composeALMs} id1 id2 -> (s1', s2')
  have \( P_6 \ (s1, s2) \) and \( P_{13} \ (s1, s2) \) and \( P_{10} \ (s1, s2) \) and \( P_2 \ (s1, s2) \) and \( P_4 \ (s1, s2) \)
  proof --
    from \text{in-trans-comp} and \text{reachable} \( (\text{composeALMs} \ id1 \ id2) \ (s1, s2) \) have
    reachable \( (\text{composeALMs} \ id1 \ id2) \ (s1', s2') \) by (auto intro: reachable.reachable-n)
    with \text{reachable} \( (\text{composeALMs} \ id1 \ id2) \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( P_6\)-invariant and \( P_{13}\)-invariant and \( P_{10}\)-invariant and \( P_4\)-invariant
and \( P_2\)-invariant show \( P_6 \ (s1, s2) \) and \( P_{13} \ (s1, s2) \) and \( P_{10} \ (s1, s2) \) and \( P_2 \ (s1, s2) \) and \( P_4 \ (s1, s2) \) unfolding invariant-def by auto
  qed
from \( 0 < id1 \) and \( id1 < id2 \) and \text{in-trans-comp} show \( P_{14} \ (s1', s2') \)
proof (rule my-rule2)
  assume \text{in-trans-cases-fun} \( id1 \ id2 \ (s1, s2) (s1', s2') \)
  thus \( P_{14} \ (s1', s2') \) using \( P_{14} \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \)
  proof (auto simp add: in-trans-cases-fun-def)
    fix \( ca \ ra \)
    assume \( P_{14} \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s1, \text{Invoke} \ ca \ ra, s1') \in \text{ALM-trans} \ 0 \ id1 \) and \( (s2, \text{Invoke} \ ca \ ra, s2') \in \text{ALM-trans} \ id1 \ id2 \)
    thus \( P_{14} \ (s1', s2') \) by (auto simp add: ALM-trans-def P_{14}-def)
next
  fix \( ca \ h \ ra \)
  assume \( P_{14} \ (s1, s2) \) and \( 0 < id1 \) and \( id1 < id2 \) and \( (s1, \text{Switch} \ ca \ id1 \ h \ ra, s1') \in \text{ALM-trans} \ 0 \ id1 \) and \( (s2, \text{Switch} \ ca \ id1 \ h \ ra, s2') \in \text{ALM-trans} \ id1 \ id2 \)
    thus \( P_{14} \ (s1', s2') \) by (auto simp add: ALM-trans-def P_{14}-def)
next
  fix \( c \ id' \ h \)
\end{verbatim}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } (s_2, \ Commit \ c \ id' h, s_2') \in \text{ALM-trans}
\begin{itemize}
  \item id1 id2 and id1 \leq id' \textbf{and } id' < id2
\end{itemize}
\textbf{thus }P_{14} \ (s_1, s_2') \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ c \ h \ r
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_2, \ Switch \ c \ id2 h \ r, \ s_2') \in \text{ALM-trans id1 id2}
\textbf{thus }P_{14} \ (s_1, s_2') \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_2, \ Linearize \ id1 h, \ s_2') \in \text{ALM-trans id1 id2}
\textbf{thus }P_{14} \ (s_1, s_2') \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def linearizations-def postfix-all-def pendingReqs-def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_2, \ Commit \ id1 h, \ s_2') \in \text{ALM-trans id1 id2}
\textbf{thus }P_{14} \ (s_1, s_2') \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ c \ a \ t \ a \ r a
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_1, \ Switch \ c \ 0 \ ta \ r a, \ s_1') \in \text{ALM-trans 0 id1 id1}
\textbf{thus }P_{14} \ (s_1', s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ c \ a \ id' \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } id1 < id2 \textbf{ and } (s_1, \ Commit \ c \ id1 h, \ s_1') \in \text{ALM-trans 0 id1 and id' < id1}
\textbf{thus }P_{14} \ (s_1', s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_1, \ Linearize \ 0 \ h, \ s_1') \in \text{ALM-trans 0 id1 id1}
\textbf{from }\text{in-lin have } \text{initialized } s_2 \textbf{ and } \text{hist } s_2 = [] \textbf{ using } P_6 \ (s_1, s_2) ; \textbf{ and } P_2 \ (s_1, s_2); \textbf{ and } P_10 \ (s_1, s_2); \textbf{ and } P_2 \ (s_1, s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def P6-def P10-def P2-def})
\textbf{thus }P_{14} \ (s_1', s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def P10-def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_1, \ Initialize \ 0 \ h, \ s_1') \in \text{ALM-trans 0 id1 id1}
\textbf{thus }P_{14} \ (s_1', s_2) \textbf{ using } P_10 \ (s_1, s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def P10-def})
\begin{itemize}
  \item next
\end{itemize}
\begin{itemize}
  \item fix \ h
\end{itemize}
\textbf{assume }P_{14} \ (s_1, s_2) \textbf{ and } 0 < id1 \textbf{ and } id1 < id2 \textbf{ and } (s_1, \ Initialize \ 0 \ h, \ s_1') \in \text{ALM-trans 0 id1 id1}
\textbf{thus }P_{14} \ (s_1', s_2) \textbf{ using } P_{10} \ (s_1, s_2) \textbf{ by } (\text{auto simp add: ALM-trans-def }P_{14}-\text{def P10-def})
next
  assume $P_{14}$ ($s_1$, $s_2$) and $0 < id_1$ and $id_1 < id_2$ and ($s_1$, Abort 0, $s'_1$) ∈ $\text{ALM-trans}$ 0 $id_1$
  thus $P_{14}$ ($s'_1$, $s_2$) by (auto simp add: $\text{ALM-trans-def}$ $P_{14}$-def)
qed
qed

lemma $P_{15}$-invariant: $[[id_1 < id_2; id_1 \neq 0]] \implies \text{invariant (composeALMs id1 id2)}$ $P_{15}$
proof (rule invariantI, auto)
  fix $s_1$ $s_2$
  assume $(s_1$, $s_2$) : $\text{starts-of (composeALMs id1 id2)}$ and $0 < id_1$
  thus $P_{15}$ ($s_1$, $s_2$) by (simp add: $\text{starts-of-def composeALMs-def hide-def}$ $\text{ALM-ioa-def}$ $\text{par-def}$ $\text{ALM-start-def}$ $P_{15}$-def)
next
  fix $s_1$ $s_2$ $s'_1$ $s'_2$ act
  assume reachable (composeALMs id1 id2) ($s_1$, $s_2$) and $P_{15}$ ($s_1$, $s_2$) and $0 < id_1$ and $id_1 < id_2$ and $\text{in-trans-comp}$:($s_1$, $s_2$) − $\text{act}$ $\text{−}$ composeALMs id1 id2 $\implies$ ($s'_1$, $s'_2$)
  have $P_{13}$ ($s_1$, $s_2$) and $P_{1b}$ ($s_1$, $s_2$) and $P_{6}$ ($s_1$, $s_2$) and $P_{1a}$ ($s_1$, $s_2$) and $P_{5}$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$)
proof −
  from $\text{in-trans-comp}$ 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 $P_{13}$-invariant and $P_{1b}$-invariant and $P_{1a}$-invariant and $P_{6}$-invariant and $P_{5}$-invariant and $P_{10}$-invariant
  show $P_{13}$ ($s_1$, $s_2$) and $P_{1b}$ ($s_1$, $s_2$) and $P_{6}$ ($s_1$, $s_2$) and $P_{1a}$ ($s_1$, $s_2$) and $P_{5}$ ($s_1$, $s_2$) and $P_{10}$ ($s_1$, $s_2$)
  unfolding invariant-def by auto
qed
from ($0 < id_1$: and $id_1 < id_2$: and $\text{in-trans-comp}$ show $P_{15}$ ($s'_1$, $s'_2$) proof (rule my-rule2)
  assume in-trans-cases-fun id1 id2 ($s_1$, $s_2$) ($s'_1$, $s'_2$)
  thus $P_{15}$ ($s'_1$, $s'_2$) using $\langle P_{15}$ ($s_1$, $s_2$): and $0 < id_1$: and $id_1 < id_2$: proof (auto simp add: $\text{in-trans-cases-fun-def}$)
  fix $ca$ $ra$
  assume $P_{15}$ ($s_1$, $s_2$) and $\text{in-invokes}$ ($s_1$, $s_2$: $\text{Invoke}$ $ca$ $ra$, $s'_1$) ∈ $\text{ALM-trans}$ 0 $id_1$
  and $\text{in-invokes}$ ($s_2$, $s_2$: $\text{Invoke}$ $ca$ $ra$, $s'_2$) ∈ $\text{ALM-trans}$ id1 id2
  show $P_{15}$ ($s'_1$, $s'_2$)
proof −
  { assume $s'_1 = s_1$
  with $\langle P_{15}$ ($s_1$, $s_2$): and $\text{in-invokes}$ and $\text{in-invokes}$ and $0 < id_1$: and $id_1 < id_2$: have $?\text{thesis}$ by (auto simp add:$\text{ALM-trans-def P15-def}$)
  } note case1 = this
  { assume $s'_1 \neq s_1$
  with $\text{in-invokes}$ and $\text{in-invokes}$ and $\langle P_{6}$ ($s_1$, $s_2$): have $s'_2 = s_2$ apply (auto simp add:$\text{ALM-trans-def P6-def}$) by (metis phase.simps(12) phase.simps(4)))
  }
\[\langle s_1' \neq s_1 \rangle \text{ and } \langle P15 (s_1, s_2) \rangle \text{ and in-invokes1 have } ?\text{thesis by (force simp add: P15-def ALM-trans-def pendingReqs-def)}\]

\}

\text{note case2 = this from case1 and case2 show } ?\text{thesis by auto qed}

next

fix ca h ru

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

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 h

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

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

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

proof

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

with \( \langle P1b (s_1, s_2) \rangle \) have \( r \notin \text{pendingReqs } s_2 \) by (auto simp add: pendingReqs-def P1b-def)

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

with \( \langle \text{phase } s_2 \) (request-snd \( r \)) = \text{Sleep and } \langle P15 (s_1, s_2) \rangle \) and \r \notin \text{pendingReqs } s_1 \) show ?\text{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 (auto simp add:P15-def)
    from in-init and ⟨P13 (s1, s2)⟩ have hist s2' ∈ postfix-all (hist s1) (linearizations (pendingReqs s1)) by (auto simp add:ALM-trans-def P13-def)
    with ⟨r ∈ set (hist s2')⟩ have r ∈ set (hist s1) ∨ r ∈ pendingReqs s1 by (auto simp add:postfix-all-def linearizations-def)
    with ⟨r ∉ pendingReqs s1⟩ show thesis by auto
  qed
next
  fix catara
  assume ⟨s1, Switch catara⟩ ∈ ALM-trans 0 id1
  hence s1' = s1 using ⟨P5 (s1, s2)⟩ by (auto simp add: ALM-trans-def P5-def)
  thus P15 (s1', s2) by (auto simp add: ALM-trans-def P15-def pendingReqs-def)
next
  fix h
  assume P15 (s1, s2) and id1 < id2 and (s1, Commit catara, 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 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 ≡ λ (s1, s2).

∥pending = λc. (if phase s1 c ≠ Aborted then pending s1 c else pending s2 c),
initHists = { },
phase = λc. (if phase s1 c ≠ Aborted then phase s1 c else phase s2 c),
hist = (if hist s2 = [] then hist s1 else hist s2),
aborted = aborted s2,
initialized = True)\)
We make the invariants available for later use

\[ P6 \ (s_1, s_2) \text{ and } P6 \ (s_1', s_2') \text{ and } P9 \ (s_1, s_2) \text{ and } P7 \ (s_1, s_2) \]
\[ \text{ and } P10 \ (s_1, s_2) \text{ and } P4 \ (s_1, s_2) \text{ and } P5 \ (s_1, s_2) \text{ and } P13 \ (s_1, s_2) \text{ and } P1a \ (s_1, s_2) \text{ and } P14 \ (s_1, s_2) \]
\[ \text{ and } P14 \ (s_1', s_2') \text{ and } P15 \ (s_1, s_2) \text{ and } P2 \ (s_1, s_2) \]
\[ \text{ and } P3 \ (s_1, s_2) \]

proof

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

with \(P6\)-invariant and \(P9\)-invariant and \(P2\)-invariant and \(P7\)-invariant and \(P10\)-invariant and \(P4\)-invariant and \(P5\)-invariant and \(P13\)-invariant and \(P1a\)-invariant and \(P14\)-invariant and \(P15\)-invariant and \(P3\)-invariant \((id1 \neq 0)\)

and \((id1 < id2)\) and reachable

show \(P6 \ (s_1, s_2)\) and \(P6 \ (s_1', s_2')\) and \(P9 \ (s_1, s_2)\) and \(P7 \ (s_1, s_2)\) and \(P10 \ (s_1, s_2)\) and \(P4 \ (s_1, s_2)\) and \(P5 \ (s_1, s_2)\) and \(P13 \ (s_1, s_2)\) and \(P1a \ (s_1, s_2)\) and \(P14 \ (s_1, s_2)\) and \(P14 \ (s_1', s_2')\) and \(P15 \ (s_1, s_2)\) and \(P2 \ (s_1, s_2)\) and \(P3 \ (s_1, s_2)\) by \((\text{auto simp add: invaraint-def})\)

qed

- let \(\mathtt{act} = \text{ref-mapping} \ (s_1, s_2)\)
- let \(\mathtt{act}' = \text{ref-mapping} \ (s_1', s_2')\)
- show \(\text{EX } \mathtt{ex} \ . \text{move} \ (\text{ALM-iao} \ 0 \ id2) \ \mathtt{ex} \ \mathtt{act} \ \mathtt{ex}'\)
  - the main part of the proof

proof \((\text{simp add: move-def, auto})\)

assume \(\mathtt{act} : \text{ext} \ (\text{ALM-iao} \ 0 \ id2)\)

hence \(\mathtt{act} : \{ \text{act} \ . \text{EX } c \ . \text{act} = \text{Invoke } c \ r \ | \ (\text{EX } t \ . \text{act} = \text{Switch } c \ 0 \ t \ r)\} \cup \{ \text{act} \ . \text{EX } c \ . \text{ex} \ | \ (\text{EX } c \ . \text{act} = \text{Switch } c \ id2 \ tr \ r)\} \) by \((\text{auto simp add: ALM-iao-def ALM-asig-def externals-def asig-inputs-def asig-outputs-def asig-def})\)

with \(\text{in-trans-comp} \ \text{show} \ \text{EX } \mathtt{ex} \ . \text{is-exec-frag} \ (\text{ALM-iao} \ 0 \ id2) \ (\mathtt{ex}, \mathtt{ex}) \)

Finite \(\mathtt{ex} \text{ and laststate} \ (\mathtt{ex}, \mathtt{ex}) = \mathtt{ex}' \text{ and mk-trace} \ (\text{ALM-iao} \ 0 \ id2) \mathtt{ex} = [\mathtt{act}]\)

- If act is an external action of the composition, then there must be an execution of the spec with matching states and forming trace "act"

apply auto

proof

fix \(c \ r\)

assume \(\text{in-inv}:(s_1, s_2) \text{--Invoke } c \ r \text{--composeALMs id1 id2} \rightarrow (s_1', s_2')\)

- If the current action is Invoke

show \(\text{EX } \mathtt{ex} \ . \text{is-exec-frag} \ (\text{ALM-iao} \ 0 \ id2) \ (\mathtt{ex}, \mathtt{ex}) \) \& \(\text{Finite } \mathtt{ex} \text{ and laststate} \ (\mathtt{ex}, \mathtt{ex}) = \mathtt{ex}' \text{ and mk-trace} \ (\text{ALM-iao} \ 0 \ id2) \mathtt{ex} = [\text{Invoke } c \ r!]\)

proof

let \(\mathtt{ex} = ([\text{Invoke } c \ r, \mathtt{ex}'])!\)

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

moreover have \(\text{laststate} \ (\mathtt{ex}, \mathtt{ex}) = \mathtt{ex}'\) by \((\text{simp add: laststate-def})\)
moreover have \(mk\text{-}trace\ (ALM\text{-}ioa\ 0\ id2)(?\text{ex}) = \{\text{Invoke } c\ r!\}\)

by (simp add: \(mk\text{-}trace\text{-}def\ externals\text{-}def\ asig\text{-}inputs\text{-}def\ asig\text{-}outputs\text{-}def\ asig\text{-}of\text{-}def\ ALM\text{-}ioa\text{-}def\ ALM\text{-}asig\text{-}def\))

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

proof —
{ 
  assume \(s1' \neq s1\) \& \(s2' \neq s2\)
  — contradiction
  with \(\text{in\text{-}invoke}\ \text{and} \ (id1 \neq 0)\ \text{and} \ (id1 < id2)\ \text{and} \ P6\ (s1', s2')\) have
  \(\text{thesis}\ \text{apply}\ \text{auto\ simp\ add:}\ \text{is\text{-}exec\text{-}frag\text{-}def}\ \text{composeALMs\text{-}def}\ \text{trans\text{-}of\text{-}def}\ \text{hide\text{-}def}\ \text{ALM\text{-}ioa\text{-}def}\ \text{ALM\text{-}asig\text{-}def}\ \text{par\text{-}def}\ \text{actions\text{-}def}\ \text{asig\text{-}outputs\text{-}def}\ \text{asig\text{-}inputs\text{-}def}\ \text{asig\text{-}internals\text{-}def}\ \text{asig\text{-}of\text{-}def}\) \text{apply}(\text{auto\ simp\ add:}\ ALM\text{-}trans\text{-}def\ P6\text{-}def)\ \text{done}
}

moreover
{ 
  assume \(s1' = s1\) \& \(s2' = s2\)
  with \(\text{in\text{-}invoke}\ \text{have}\ \text{pre\text{-}s1:}\ ~ (\text{phase } s1\ c = \text{Ready } k\ \text{request\text{-}snd } r\ = c\ \&\ r \notin \text{set}\ (\text{hist } s1))\ \text{and}\ \text{pre\text{-}s2:}\ ~ (\text{phase } s2\ c = \text{Ready } k\ \text{request\text{-}snd } r\ = c\ \&\ r \notin \text{set}\ (\text{hist } s2))\ \text{using}\ \text{[hypsubst\text{-}thin]}\ \text{apply}\ \text{(auto\ simp\ add:}\ \text{is\text{-}exec\text{-}frag\text{-}def}\ \text{composeALMs\text{-}def}\ \text{trans\text{-}of\text{-}def}\ \text{hide\text{-}def}\ \text{ALM\text{-}ioa\text{-}def}\ \text{ALM\text{-}asig\text{-}def}\ \text{par\text{-}def}\ \text{actions\text{-}def}\ \text{asig\text{-}outputs\text{-}def}\ \text{asig\text{-}inputs\text{-}def}\ \text{asig\text{-}internals\text{-}def}\ \text{asig\text{-}of\text{-}def})\ \text{apply}(\text{simp\add:}\ ALM\text{-}trans\text{-}def)\ \text{apply}\ \text{(drule\text{-}tac[!] arg\text{-}cong[where } f = \text{phase}])\ \text{apply}\ \text{simp\text{-}all}\ \text{apply}\ \text{(metis\ phase\text{-}simps(8) fun\text{-}upd\text{-}idem\text{-}iff)}\ \text{apply}\ \text{(metis\ phase\text{-}simps(8) fun\text{-}upd\text{-}idem\text{-}iff)}\ \text{apply}\ \text{(metis\ phase\text{-}simps(8) fun\text{-}upd\text{-}idem\text{-}iff)}\ \text{done}
  \}

moreover
{ 
  assume \(s1' \neq s1\) \& \(s2' = s2\)
  with \(\text{in\text{-}invoke}\ \text{have}\ \text{pre\text{-}s1:}\ \text{phase } s1\ c = \text{Ready } k\ \text{request\text{-}snd } r\ = c\ \&\ r \notin \text{set}\ (\text{hist } s1)\ \text{and}\ \text{trans\text{-}s1:}\ \text{phase } s1\ c = \text{Pending}\) \text{apply}\ \text{(simp\add:}\ \text{is\text{-}exec\text{-}frag\text{-}def}\ \text{composeALMs\text{-}def}\ \text{trans\text{-}of\text{-}def}\ \text{hide\text{-}def}\ \text{ALM\text{-}ioa\text{-}def}\ \text{ALM\text{-}asig\text{-}def}\ \text{par\text{-}def}\ \text{actions\text{-}def}\ \text{asig\text{-}outputs\text{-}def}\ \text{asig\text{-}inputs\text{-}def}\ \text{asig\text{-}internals\text{-}def}\ \text{asig\text{-}of\text{-}def})\ \text{apply}(\text{simp\add:}\ ALM\text{-}trans\text{-}def)\ \text{done}
  \}

moreover
{ 
  assume \(s1' \neq s1\) \& \(s2' = s2\)
  with \(\text{in\text{-}invoke}\ \text{have}\ \text{pre\text{-}s1:}\ \text{phase } s1\ c = \text{Ready } k\ \text{request\text{-}snd } r\ = c\ \&\ r \notin \text{set}\ (\text{hist } s1)\ \text{and}\ \text{trans\text{-}s1:}\ \text{phase } s1\ c = \text{Pending}\) \text{apply}\ \text{(auto\ simp\add:}\ \text{ref\text{-}mapping\text{-}def})\ \text{apply}\ \text{(auto\ simp\add:}\ \text{ref\text{-}mapping\text{-}def})\ \text{done}

  \}

moreover have \(\mathit{r} \notin \text{set}\ (\text{hist } ?t)\)

proof —

from \text{pre\text{-}s1}\ have\ \text{phase } ?t\ c = \text{Ready } k\ \text{request\text{-}snd } r\ = c\ \text{by}
\text{(auto\ simp\add:}\ \text{ref\text{-}mapping\text{-}def})

moreover have \(\mathit{r} \notin \text{set}\ (\text{hist } ?t)\)

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

31
assume \( \text{hist } s_2 = [] \)
with \( \text{pre-s1} \) show \( \text{?thesis} \) by (auto simp add:ref-mapping-def)
next
assume \( \text{hist } s_2 \neq [] \)
show \( r \notin \text{set } (\text{hist } ?t) \)
proof auto
assume \( r \in \text{set } (\text{hist } ?t) \)
with \( \langle \text{hist } s_2 \neq [] \rangle \) have \( r \in \text{set } (\text{hist } s_2) \) by (auto simp add:ref-mapping-def)
moreover from \( \text{pre-s1} \) and \( \langle \text{P6 } (s_1, s_2) \rangle \) have \( \text{phase } s_2 \)
\( (\text{request-snd } r) = \text{Sleep} \) by (force simp add:P6-def)
moreover note \( \langle \text{P15 } (s_1, s_2) \rangle \)
ultimately have \( r \in \text{set } (\text{hist } s_1) \lor r \in \text{pendingReqs } s_1 \)
by (auto simp add:P15-def)
with \( \text{pre-s1} \) have \( r \in \text{pendingReqs } s_1 \) by auto
with \( \langle \text{P1a } (s_1, s_2) \rangle \) and \( \text{pre-s1} \) show \( \text{False} \) by (auto simp add:pendingReqs-def P1a-def)
qed
qed
moreover from \( \text{pre-s1} \) and \( \text{trans-s1} \) and \( \langle s_2' = s_2 \rangle \) have \( \text{trans-t}?:?t' \)
\( = ?t\langle \text{pending} \rangle = (\text{pending } ?t)(c := r) \), \( \text{phase} := (\text{phase } ?t)(c := \text{Pending}) \rangle \)
by (auto simp add:ref-mapping-def fun-eq-iff)
ultimately have \( \text{?thesis} \) 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 \( \text{in-invoked } \) and \( \langle \text{id1} \neq 0 \rangle \) have \( \text{pre-s2} \): \( \text{phase } s_2 c = \text{Ready } \) & \( \text{request-snd } r = c \) & \( r \notin \text{set } (\text{hist } s_2) \) and \( \text{trans-s2} \): \( s_2' = s_2\langle \text{pending} \rangle = (\text{pending } s_2)(c := r) \), \( \text{phase} := (\text{phase } ?t)(c := \text{Pending}) \rangle \)
apply(simp-all add:ALM-trans-def ref-mapping-def) done
from \( \text{pre-s2} \) and \( \langle \text{P6 } (s_1, s_2) \rangle \) have \( \text{aborted-s1-c}:\text{phase } s_1 c = \text{Aborted} \) by (auto simp add: P6-def)
with \( \text{pre-s2} \) and \( \langle \text{P3 } (s_1, s_2) \rangle \) and \( \langle \text{P14 } (s_1, s_2) \rangle \) have \( \text{pre-t}?:\text{phase} \)
\( ?t c = \text{Ready } \) & \( \text{request-snd } r = c \) & \( r \notin \text{set } (\text{hist } ?t) \) apply (auto simp add: fun-eq-iff ref-mapping-def P3-def P14-def) done
moreover have \( \text{trans-t}?:?t' = ?t\langle \text{pending} \rangle = (\text{pending } ?t)(c := r) \), \( \text{phase} := (\text{phase } ?t)(c := \text{Pending}) \rangle \) using aborted-s1-c and \( s_1' = s_1 \) and \( \text{trans-s2} \)
apply(force simp add: fun-eq-iff ref-mapping-def) done
ultimately have \( \text{?thesis} \) apply (simp add: is-exec-frag-def
composeALMs-def trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def
asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply(simp add:ALM-trans-def)
done

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

next
fix c r h
assume in-switch:(s1, s2) − Switch c 0 h r -- composeALMs id1 id2 ->
(s1', s2')
— If we get a switch 0 input (nothing happens)
show EX ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) & Finite ex & laststate
(?t, ex) = ?t' & mk-trace (ALM-ioa 0 id2)$ex = [Switch c 0 h r!]
proof −
let ?ex = [(Switch c 0 h r, ?t')!]

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

moreover have is-exec-frag (ALM-ioa 0 id2) (?t, ?ex)
proof −
from in-switch and ⟨id1 ≠ 0⟩ and ⟨id1 < id2⟩ and ⟨P5 (s1, s2)⟩ have s1' = s1 and s2' = s2 and \( \land \) c . phase s1 c ≠ Sleep apply (simp-all add: composeALMs-def trans-of-def hide-def \_par-def actions-def asig-outputs-def
asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(simp-all add: ALM-trans-def P5-def) done

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

next
fix c r h
assume in-switch:(s1, s2) − Switch c id2 h r -- composeALMs id1 id2 ->
(s1', s2')
— The case when the system switches to a third, new, instance
show EX ex. is-exec-frag (ALM-ioa 0 id2) (?t, ex) &
Finite ex & laststate (?t, ex) = ?t' & mk-trace (ALM-ioa 0 id2)$ex =
[Switch c id2 h r!]
proof −
let ?ex = [(Switch c id2 h r, ?t')!]
have Finite ?ex by auto
moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
moreover have \text{mk-trace} (\text{ALM-ioa} \ 0 \ \text{id}2) (\text{?ex}) = [\text{Switch} \ c \ \text{id}2 \ h \ r!] \\
by (\text{simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def})

moreover have \text{is-exec-frag} (\text{ALM-ioa} \ 0 \ \text{id}2) (\text{?t}, \ ?\text{ex})

proof -
from \text{in-switch} \ and \ (\text{id}1 < \text{id}2) \ have \ s1' = s1 \ apply (\text{simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def}) \\
done

from (\text{id}1 \neq 0) \ and \ (\text{id}1 < \text{id}2) \ in-switch have \ \text{pre-s2:aborted} \ s2 \ & \ phase \ s2 \ c = \text{Pending} \ & \ r = \text{pending} \ s2 \ c \ & (\text{if} \ \text{initialized} \ s2 \ then \ (h \in \text{postfix-all} \ (\text{hist} \ s2) \ (\text{linearizations} \ (\text{pendingReqs} \ s2))) \ else \ (h : \text{postfix-all} \ (l-c-p \ (\text{initHists} \ s2)) \ (\text{linearizations} \ (\text{initValidReqs} \ s2))))) \ and \ \text{trans-s2}: \ s2' = s2\ [(\text{phase} := (\text{phase} \ s2)(c := \text{Aborted}))] \ apply (\text{simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def}) applied (\text{auto simp add: P6-def}) \\
done

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

have \ \text{pre-t:aborted} \ ?\text{t} \ & \ \text{phase} \ ?\text{t} \ c = \text{Pending} \ & \ \text{initialized} \ ?\text{t} \ & \ h : \text{postfix-all} \ (\text{hist} \ ?\text{t}) \ (\text{linearizations} \ (\text{pendingReqs} \ ?\text{t})) \ & \ r = \text{pending} \ ?\text{t} \ c \\
proof -
from \text{s1:aborted} \ and \ \text{pre-s2} \ have \ \text{aborted} \ ?\text{t} \ & \ \text{pending} \ ?\text{t} \ c = r \ and \ \text{phase} \ ?\text{t} \ c = \text{Pending} \ and \ \text{initialized} \ ?\text{t} \ by (\text{auto simp add: ref-mapping-def fun-eq-iff})

moreover have \ h : \text{postfix-all} \ (\text{hist} \ ?\text{t}) \ (\text{linearizations} \ (\text{pendingReqs} \ ?\text{t})) \\
proof -
from \text{pre-s2} \ have \ (\text{if} \ \text{initialized} \ s2 \ then \ (h : \text{postfix-all} \ (\text{hist} \ s2) \ (\text{linearizations} \ (\text{pendingReqs} \ s2))) \ else \ (h : \text{postfix-all} \ (l-c-p \ (\text{initHists} \ s2)) \ (\text{linearizations} \ (\text{initValidReqs} \ s2)))) \ by \text{auto} \\
thus \text{?thesis} \\
proof \text{auto} \\
assume \text{case1-1:initialized} \ s2 \ and \ \text{case1-2:h : postfix-all} \ (\text{hist} \ s2) \ (\text{linearizations} \ (\text{pendingReqs} \ s2)) \\
\text{hence} \ \text{suffixeq} \ (\text{hist} \ s1) \ (\text{hist} \ s2) \ using (P14 \ (s1, s2)) \ by (\text{auto simp add: P14-def suffixeq-def}) \\
\show h \in \text{postfix-all} \ (\text{hist} \ ?\text{t}) \ (\text{linearizations} \ (\text{pendingReqs} \ ?\text{t})) \\
proof - \\
\have \text{hist} \ ?\text{t} \ = \text{hist} \ s2 \\
\proof \text{(cases hist} \ s2 = []) \\
\assume \text{hist} \ s2 = [] \\
\show \text{hist} \ ?\text{t} \ = \text{hist} \ s2 \\
\proof - \\
\from \text{hist} \ s2 = [] \ and \ \text{suffixeq} \ (\text{hist} \ s1) \ (\text{hist} \ s2) \ \have \text{hist} \ s1 = [] \ \text{by (auto simp add: suffixeq-def)} \\
\with \text{hist} \ s2 = [] \ \text{show hist} \ ?\text{t} = \text{hist} \ s2 \ \text{by (auto simp add: ref-mapping-def)} \\
\qed \\
\text{next}

34
assume hist s2 ≠ []
thus hist ?t = hist s2 by (simp add:ref-mapping-def)
qed
moreover have pendingReqs s2 ≤ pendingReqs ?t
proof (simp add: pendingReqs-def, clarify)
fix c
  assume pending s2 c /∈ set (hist s2) and phase s2 c = Pending ∨ phase s2 c = Aborted
moreover with ⟨P6 (s1, s2): have phase s1 c = Aborted by (auto simp add: P6-def) ⟩
moreover note (suffixeq (hist s1) (hist s2))
ultimately show ?thesis by (auto simp add: suffixeq-def prefixeq-Nil prefixeq-def self-append-conv2)
qed
moreover note case1-2
ultimately show ?thesis by (auto simp add: linearizations-def postfix-all-def)
qed
next
assume case2-1:¬ initialized s2 and case2-2: h : postfix-all (l-c-p (initHists s2)) (linearizations (initValidReqs s2))
from case2-1 and ⟨P10 (s1, s2): have hist s2 = [] by (auto simp add: P10-def) ⟩
have h : postfix-all (hist s1) (linearizations (pendingReqs s1))
proof –
  from pre-s2 have phase s2 c ≠ Sleep by auto
moreover note (P13 (s1, s2)) and case2-1 and case2-2
ultimately show ?thesis by (auto simp add: P13-def)
qed
moreover from ⟨hist s2 = []: have hist ?t = hist s1 by (auto simp add:ref-mapping-def) ⟩
moreover have pendingReqs ?t = pendingReqs s1
proof auto
fix r
  assume r ∈ pendingReqs ?t
  with this obtain c' where r = pending ?t c' and r /∈ set (hist ?t) and phase ?t c' ∈ {Pending, Aborted} by (auto simp add:pendingReqs-def)
  show r ∈ pendingReqs s1
  proof (cases phase s1 c' = Aborted)
    assume phase s1 c' = Aborted
    with ⟨P6 (s1, s2): and case2-1 and ⟨P7 (s1, s2): and ⟨hist ?t = hist s1: and r /∈ set (hist ?t): have phase s1 c' = Aborted and r = pending s1 c' and r /∈ set (hist s1) apply (auto simp add: P6-def P7-def) ⟩
    apply force
done
thus \$thesis\$ by (auto simp add: pendingReqs-def)

next
  assume phase s1 c' \neq Aborted
  with \$r = pending \langle t, c' \rangle\$ and \$r \notin \text{set} (\langle \text{hist} \ t \rangle)\$ and \$\langle \\text{phase} \ t' \in \{\text{Pending}, \text{Aborted}\}\rangle\$ and \$\langle \text{hist} \ t = \text{hist} \ s1 \rangle\$ show \$thesis\$ by (auto simp add:ref-mapping-def pendingReqs-def)

  qed

next
  fix r
  assume r \in pendingReqs s1
  with this obtain c where \$r = \text{pending} \ s1 \ c\$ and \$\langle \\text{phase} \ s1 \ c' \in \{\text{Pending}, \text{Aborted}\}\rangle\$ and \$\langle \text{hist} \ s2 = []\rangle\$ and \$\langle \neg \text{initialized} \ s2 \rangle\$ and \$\langle P7 (s1, s2) \rangle\$

  show r \in pendingReqs ?t by (auto simp add:ref-mapping-def pendingReqs-def P7-def)

  qed

ultimately show \$thesis\$ by auto

qed

next
  fix c h id id'
  assume in-commit:(s1, s2) -Commit c id' h -composeALMs id1 id2 \rightarrow (s1', s2') and id' < id2

  — Case when the composition commits a request
  show \exists ex. is-exec-frag (ALM-ioa 0 id2) (\langle t, ex \rangle) and Finite ex and laststate (\langle t, ex \rangle) = t' \& mk-trace (ALM-ioa 0 id2)-ex = [Commit c id' h]

  proof
    let \$ex = [\langle \text{Commit} \ c \ id' \ h, t' \rangle]\$

    have Finite \$ex\$ by auto
    moreover have laststate (\$t, ex\$) = t' by (simp add: laststate-def)
    moreover have mk-trace (ALM-ioa 0 id2)$\$ (\$ex\$) = [Commit c id' h] using \$id' < id2\$ by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-internals-def asig-of-def)

    moreover have is-exec-frag (ALM-ioa 0 id2) (\$t, ex\$)
    proof
\{ 
  assume id' < id1 
  with in-commit have s2' = s2 and pre-s1:phase s1 c = Pending 
  \land pending s1 c \in \set{\text{hist } s1} \land h = \text{dropWhile } (\lambda \ r . \ r \neq \text{pending } s1 c) \text{ (hist } s1) \text { and trans-s1:s1' = s1 \langle \text{phase} := \langle \text{phase } s1 \rangle(c := \text{Ready})\rangle) \text{ apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def def asig-internals-def def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done} 
  from pre-s1 have s1-not-aborted-c:phase s1 c \neq \text{Aborted} by auto 
  have pre-t:phase ?t c = Pending \& pending ?t c \in \set{\text{hist } ?t} \land h = \text{dropWhile } (\lambda \ r . \ r \neq \text{pending } ?t c) \text{ (hist } ?t) 
  proof (cases hist s2 = [[]]) 
  assume hist s2 = [[]] 
  with pre-s1 and (phase s1 c \neq \text{Aborted}) show \(\text{thesis}\) by (auto simp add:ref-mapping-def) 
  moreover have pending ?t c \in \set{\text{hist } ?t} 
  proof - 
  from (initialized s2) and \(\text{P10 } (s1, s2)\) obtain rs3 where hist s2 = rs3 @ (hist s1) by (auto simp add:P10-def) 
  withPending s1 c \in \set{\text{hist } s1} and hist s2 = rs3 @ (hist s1) and Pending pending ?t c = pending s1 c and pending s1 c \in \set{\text{hist } s1} by (auto simp add:ref-mapping-def suffizeq-def) 
  qed 
  moreover have h = dropWhile (\lambda \ r . \ r \neq pending ?t c) (hist ?t) 
  proof - 
  from (pending s1 c \in \set{\text{hist } s1}) obtain rs1 rs2 where hist s1 = rs2 @ rs1 and hd rs1 = pending s1 c and rs1 \neq [] and pending s1 c \notin \set{\text{rs2}} by (metis list.sel(1) in-set-comp-decomp-first list.simps(3)) 
  with (pending ?t c = pending s1 c and dropWhile-lemma[of hist s1 rs1 pending s1 c]) and pre-s1 have h = rs1 by auto 
  moreover have dropWhile (\lambda \ r . \ r \neq pending ?t c) (hist ?t) = rs1 
  proof - 
  from (initialized s2) and \(\text{P14 } (s1, s2)\) obtain rs3 where hist s2 = rs3 @ (hist s1) and set rs3 \cap \set{\text{hist } s1} = {} by (auto simp add:P14-def) 
  with (pending s1 c \in \set{\text{hist } s1}) and hist s1 = rs2 @ rs1) 
  have hist s2 = rs3 @ rs2 @ rs1 and pending s1 c \notin \set{\text{rs3}} by auto 
  with (pending s1 c \notin \set{\text{rs2}}) obtain rs4 where hist s2 = rs4 @ rs1 and pending s1 c \notin \set{\text{rs4}} by auto 
  with (hd rs1 = pending s1 c and rs1 \neq []) and dropWhile-lemma[of hist s2 rs1 pending s1 c] have dropWhile (\lambda \ r . \ r \neq pending s1 c) (hist s2) = rs1 by auto 
  thus \(\text{thesis}\) using (hist s2 \neq []) and (pending ?t c = pending

37

s1 c \vdash \text{by (auto simp add:ref-mapping-def)}

qed
ultimately show \(?thesis by auto

qed
ultimately show \(?thesis by auto

qed
moreover from \(s2' = s2\) and \(s1\)-not-aborted-c and \(\text{trans-}\_\text{s1}

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

ultimately have \(?thesis using (id1 < id2) \text{apply (simp add:}

is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-\text{ioa-def ALM-asig-def}

\text{par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def)}\text{apply(simp add:ALM-trans-def)} \text{done}\}

moreover
{
  \text{assume id1 \leq id'}

  with \text{in-commit have s1' = s1 and pre-s2-phase s2 c = Pending}

  & \text{pending s2 c \in set (hist s2) \& h = dropWhile (\lambda r . r \neq pending s2 c) (hist s2) and trans-s2:s2' = s2 \{(phase := (phase s2)(c := \text{Ready}))\} \text{apply (simp-all add:composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-\text{ioa-def ALM-asig-def}}\text{apply(auto simp add:ALM-trans-def)} \text{done}\}

  \text{from pre-s2 and (P6 (s1, s2)): have facts:aborted s1 & phase s1 c = Aborted \& hist s2 \neq [] by (force simp add:P6-def)}

  \text{with pre-s2 have pre-t-phase ?t c = Pending \& pending ?t c \in set (hist ?t) \& h = dropWhile (\lambda r . r \neq pending ?t c) (hist ?t) by (auto simp add:ref-mapping-def)}

  moreover from \(s1' = s1\) and facts and trans-s2 have \(\text{trans-t}:?t' = ?t\ \{(phase := (phase ?t)(c := \text{Ready}))\} \text{by (auto simp add:fun-eq-iff ref-mapping-def)}\)

  ultimately have \(?thesis using (id1 < id2) \text{apply (simp add:}

  is-exec-frag-def composeALMs-def trans-of-def hide-def ALM-\text{ioa-def ALM-asig-def}

  \text{par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def)}\text{apply(simp add:ALM-trans-def)} \text{done}\}

} ultimately show \(?thesis using (id' < id2) \text{by force}

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

qed

— We finished the case when the composition takes an action that is in

the external signature of the spec

next
\text{assume act \notin ext (ALM-\text{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 (id1 < id2) and (id1 \neq 0): have act : {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 \mid id1 h \mid act = \text{Initialize} 0 h \mid act = \text{Initialize} id1 h \}) by (auto simp add: composeALMs-def hide-def hide-asig-def ALM-ioa-def ALM-asig-def externals-def asig-inputs-def asig-outputs-def asig-internals-def asig-of-def trans-of-def par-def actions-def)

with in-trans-comp show \exists ex. \text{is-exec-frag} (ALM-ioa 0 id2) (?) \land Finite ex \land laststate (\exists t, ex) = \exists t' \land mk-trace (ALM-ioa 0 id2)\cdot ex = \text{nil}

proof auto

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

— The case where the first Abstract aborts

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


apply (auto simp add: fun-eq-iff ALM-trans-def ref-mapping-def P6-def P2-def) done

thus \exists t

proof simp

let ?ex = \text{nil}

have Finite ?ex by auto

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

moreover have mk-trace (ALM-ioa 0 id2)\cdot ?ex = \text{nil} using \exists id1 < id2 by (simp add: mk-trace-def externals-def asig-inputs-def asig-outputs-def asig-of-def ALM-ioa-def ALM-asig-def)

moreover have is-exec-frag (ALM-ioa 0 id2) (?) ?ex by (auto simp add: is-exec-frag-def)

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

qed

next

assume in-abort:(s1, s2) - Abort id1 - composeALMs id1 id2 -> (s1', s2')

— The case where the second ALM aborts

show \exists t

proof

let ?ex = [(Abort 0, \exists t')] have Finite ?ex by auto

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

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

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

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

moreover from trans-s2 and ⟨s1' = s1⟩ have trans-t: ?t' =
?t(abort := True) by (auto simp add: fun-eq-iff ref-mapping-def)

ultimately show ?thesis apply (simp add:ALM-trans-def) done

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))
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
\begin{verbatim}
assume x \in \text{pendingReqs} \ s1
moreover note (\neg \text{aborted} \ s1) and (P6 \ (s1, s2));
ultimately obtain c where x = pending \ s1 \ c and phase \ s1 \ c \notin \text{set} (\text{hist} \ s1) by (auto simp add: pendingReqs-def P6-def)

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

moreover from \text{pre-s1} have h \in \text{postfix-all} (\text{hist} \ s1) (\text{linearizations} (\text{pendingReqs} \ s1)) by auto
ultimately show \text{thesis} by (auto simp add: postfix-all-def linearizations-def)
qed

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

moreover have \text{trans-t:} \ ?t' = ?t (| hist := h, initialized := True|)
proof for
have (hist ?t' = hist s1')
proof for
from \text{pre-s1} have (\neg \text{aborted} \ s1) by auto
with (P6 \ (s1, s2)) and (P2 \ (s1, s2)) have \text{hist} \ s2' = [] by (auto simp add: P6-def P2-def)
with (s2' = s2) show \text{thesis} by (auto simp add: ref-mapping-def)
qed
with \text{trans-s1} have \text{hist} ?t' = h by auto
thus \text{thesis} using (s2' = s2) and \text{trans-s1} by (auto simp add: ref-mapping-def fun-eq-iff)
qed
ultimately show \text{thesis} apply (simp add: \text{is-exec-frag-def}
composeALMs-def \text{trans-of-def} hide-def \text{ALM-ioa-def} \text{ALM-asig-def} par-def actions-def
asig-outputs-def asig-inputs-def asig-internals-def asig-of-def) apply (auto simp add: \text{ALM-trans-def})
done

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

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

– If the composition executes Linearize \text{id1}
let \text{?ex} = [(\text{Linearize} \text{id1} h, \ ?t')]

have \text{Finite} \ ?ex by auto
moreover have laststate (?t, ?ex) = ?t' by (simp add: laststate-def)
moreover have \text{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 \text{is-exec-frag} (\text{ALM-ioa 0 id2}) (?t, ?ex)
proof for
from in-lin and \text{id1} \neq 0 have s1' = s1 and \text{pre-s2: initialized} s2
\end{verbatim}
∧ ¬ aborted s2 ∧ h ∈ postfix-all (hist s2) (linearizations (pendingReqs s2)) and
trans-s2: s2′ = s2[hist := h] apply (simp-all add: composeALMs-def trans-of-def
hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def
ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done
have pre-t: initialized ?t ∧ ¬ aborted ?t ∧ h ∈ postfix-all (hist ?t)
(linearizations (pendingReqs ?t))
proof –
have ¬ aborted ?t and initialized ?t using pre-s2 by (auto simp
add:ref-mapping-def)
moreover have h ∈ postfix-all (hist ?t) (linearizations (pendingReqs
?t))
proof –
from pre-s2 have initialized s2 by auto
hence suffixeq (hist s1) (hist s2) using (P14 (s1, s2)) by (auto
simp add:P14-def suffixeq-def)
hence hist ?t = hist s2 by (auto simp add:ref-mapping-def)
moreover have pendingReqs s2 ⊆ pendingReqs ?t
proof auto
fix x
assume x ∈ pendingReqs s2
from this obtain c where x = pending s2 c and phase
s2 c ∈ {Pending, Aborted} and pending s2 c ∉ set (hist s2) by (auto simp
add:pendingReqs-def)
with (P6 (s1, s2)) and hist ?t = hist s2; show x ∈ pendingReqs
?t by (force simp add:ref-mapping-def P6-def pendingReqs-def)
qed
moreover from pre-s2 have h ∈ postfix-all (hist s2) (linearizations
(pendingReqs s2)) by auto
ultimately show ?thesis by (auto simp add:postfix-all-def
linearizations-def)
qed
ultimately show ?thesis by auto
qed
moreover have trans-t: ?t′ = ?t[hist := h]
proof –
from pre-s2 and trans-s2 have initialized s2′ by auto
hence suffixeq (hist s1′) (hist s2′) using (P14 (s1′, s2′)) by (auto
simp add:P14-def suffixeq-def)
hence hist ?t′ = hist s2′ by (auto simp add:ref-mapping-def)
with trans-s2 and (s1′ = s1) show ?thesis by (auto simp
add:ref-mapping-def fun-eq-iff)
qed
ultimately show ?thesis apply (simp add: is-exec-frag-def composeALMs-def
trans-of-def hide-def ALM-ioa-def ALM-asig-def par-def actions-def asig-outputs-def
asig-inputs-def asig-internals-def asig-of-def) apply(auto simp add:ALM-trans-def)
done
qed
ultimately show ?thesis by (auto intro: exI[where x=?ex])

42
next

fix c r h

assume in-switch:(s1, s2) \rightarrow Switch c id1 h \rightarrow composeALMs id1 id2 \rightarrow (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 \neq 0) have pre-s1: aborted s1 \wedge phase s1 c = Pending \wedge r = pending s1 c \wedge (if initialized s1 then (h \in postfix-all (hist s1) (linearizations (pendingReqs s1)))) else (h : postfix-all (l-c-p (initHists s1) (linearizations (initValidReqs s1)))) and trans-s1: s1' = s1 (phase := (phase s1) (c := Aborted)) apply (simp-all add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add:ALM-trans-def) done

have pre-s2: phase s2 c = Sleep and trans-s2: s2' = s2 (initHists := \{h\} \cup (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 \neq 0) and (id1 < id2) show phase s2 c = Sleep and s2' = s2 (initHists := \{h\} \cup (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' \wedge 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 in-initialize: (s1, s2) −Initialize 0 h −composeALMs id1 id2 → (s1′, s2′)
hence False using :P10 (s1, s2) apply (simp add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add: ALM-trans-def P10-def) done

thus ?thesis by auto

next

fix h

assume in-initialize: (s1, s2) −Initialize id1 h −composeALMs id1 id2 → (s1′, s2′)
— If the second ALM of the composition initializes
let ?ex = [(Linearize id1 h, ?t′)!]

have Finite ?ex by auto

moreover have laststate (?t, ?ex) = ?t′ by (simp add: laststate-def)

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

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

proof —

from in-initialize and id1 ≠ 0 have s1′ = s1 and pre-s2: (∃ c. phase s2 c ≠ Sleep) ∧ ¬ aborted s2 ∧ ¬ initialized s2 ∧ h ∈ postfix-all (l-c-p (initHists s2)) (linearizations (initValidReqs s2)) and trans-s2:s2′ = s2{hist := h, initialized := True} apply (simp add: composeALMs-def trans-of-def hide-def par-def actions-def asig-outputs-def asig-inputs-def asig-internals-def asig-of-def ALM-ioa-def ALM-asig-def) apply(auto simp add: ALM-trans-def) done

have pre-t: initialized ?t ∧ ¬ aborted ?t ∧ h ∈ postfix-all (hist ?t) (linearizations (pendingReqs ?t))

proof —

from pre-s2 have initialized ?t ∧ ¬ aborted ?t by (auto simp add: ref-mapping-def)

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

proof —

from pre-s2 have h ∈ postfix-all (l-c-p (initHists s2)) (linearizations (initValidReqs s2)) and ¬ initialized s2 and ∃ c. phase s2 c ≠ Sleep by auto with ⟨P13 (s1, s2)⟩ have h ∈ postfix-all (hist s1) (linearizations (pendingReqs s1)) by (auto simp add: P13-def)

moreover from ¬ initialized s2 and :P10 (s1, s2) have hist ?t = hist s1 by (auto simp add: ref-mapping-def P10-def)

moreover have pendingReqs s1 ⊆ pendingReqs ?t

proof auto

fix x

assume x ∈ pendingReqs s1

from this obtain c where x = pending s1 c and phase s1 c ∈ {Pending, Aborted} and pending s1 c ∈ set (hist s1) by (auto simp add: pendingReqs-def)

show x ∈ pendingReqs ?t

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

with \( \langle x = \text{pending } s_1 c \rangle \) and \( \langle \text{pending } s_1 c \notin \text{set} \text{ (hist } s_1) \rangle \) and \( \langle \text{hist } \hat{t} = \text{hist } s_1 \rangle \) show \( \?\thesis \) by (force simp add:ref-mapping-def pendingReqs-def)

next

assume \( \text{phase } s_1 c \neq \text{Pending} \)

with \( \langle \text{phase } s_1 c \in \{\text{Pending, Aborted}\} \rangle \) have \( \text{phase } s_1 c = \text{Aborted} \) by auto

with \( \langle \neg \text{initialized } s_2 \rangle \) and \( \langle P_6 \ (s_1, s_2) \rangle \) and \( \langle P_7 \ (s_1, s_2) \rangle \) have

\( \langle \text{pending } s_2 c = \text{pending } s_1 c \rangle \) and \( \langle \text{phase } s_2 c \in \{\text{Pending, Aborted}\} \rangle \) by (auto simp add:P6-def P7-def)

with \( \langle x = \text{pending } s_1 c \rangle \) and \( \langle \text{pending } s_1 c \notin \text{set} \text{ (hist } s_1) \rangle \) and \( \langle \text{hist } \hat{t} = \text{hist } s_1 \rangle \) and \( \langle P_6 \ (s_1, s_2) \rangle \) show \( \?\thesis \) by (auto simp add:ref-mapping-def pendingReqs-def P6-def)

qed

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

qed

ultimately show \( \?\thesis \) by auto

qed

moreover have \( \text{trans-} : \hat{t} = \hat{t}[\text{hist} := h] \)
proof —
from pre-s2 have \( \exists \ c . \text{phase } s_2 c \neq \text{Sleep} \) by auto

with \( \text{trans-} : \hat{t} = \hat{t}[\text{hist} := h] \) hence suffixeq (hist \( s_1' \)) (hist \( s_2' \)) using \( \langle P_{14} \ (s_1', s_2') \rangle \) by (auto simp add:P14-def suffixeq-def)

hence \( \text{hist } \hat{t}' = \text{hist } s_2' \) by (auto simp add:ref-mapping-def)

with \( \text{trans-} : \hat{t} = \hat{t}[\text{hist} := h] \) and \( \langle s_1' = s_1 \rangle \) show \( \?\thesis \) by (auto simp add:ref-mapping-def fun-eq-iff)

qed

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

qed

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

qed

qed

qed

qed

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


