MessagingSystem

xuwenjing

October 29, 2019

Contents

1 Abstract Syntax of PiCore Language

theory *PiCore-Language*
imports *Main* **begin**

type-synonym $'s\ bexp = 's\ set$

type-synonym $'s\ guard = 's\ set$

type-synonym $('l,'s,'prog)\ event' = 'l \times ('s\ guard \times 'prog)$

definition $guard :: ('l,'s,'prog)\ event' \Rightarrow 's\ guard$ **where**
  $guard\ ev \equiv fst\ (snd\ ev)$

definition $body :: ('l,'s,'prog)\ event' \Rightarrow 'prog$ **where**
  $body\ ev \equiv snd\ (snd\ ev)$

datatype $('l,'k,'s,'prog)\ event =$
      $AnonyEvent\ 'prog$
    $|\ BasicEvent\ ('l,'s,'prog)\ event'$

datatype $('l,'k,'s,'prog)\ esys =$
      $EvtSeq\ ('l,'k,'s,'prog)\ event\ ('l,'k,'s,'prog)\ esys$
    $|\ EvtSys\ ('l,'k,'s,'prog)\ event\ set$

type-synonym $('l,'k,'s,'prog)\ paresys = 'k \Rightarrow ('l,'k,'s,'prog)\ esys$

2 Some Lemmas of Abstract Syntax

primrec $is\text{-}basicevt :: ('l,'k,'s,'prog)\ event \Rightarrow bool$
  where $is\text{-}basicevt\ (AnonyEvent\ \text{-}) = False\ |$
      $is\text{-}basicevt\ (BasicEvent\ \text{-}) = True$

primrec *is-anonyevt* :: ('*l*,'*k*,'*s*,'*prog*) *event* ⇒ *bool*
where *is-anonyevt* (*AnonyEvent* -) = *True* |
is-anonyevt (*BasicEvent* -) = *False*

lemma *basicevt-isnot-anony*: *is-basicevt e* ⟹ ¬ *is-anonyevt e*
by (*metis event.exhaust is-anonyevt.simps*(*2*) *is-basicevt.simps*(*1*))

lemma *anonyevt-isnot-basic*: *is-anonyevt e* ⟹ ¬ *is-basicevt e*
using *basicevt-isnot-anony* **by** *auto*

lemma *evtseq-ne-es*: *EvtSeq e es* ≠ *es*
apply(*induct es*)
apply *auto*[*1*]
by *simp*

end

3 Small-step Operational Semantics of PiCore Language

theory *PiCore-Semantics*
imports *PiCore-Language*
begin

3.1 Datatypes for Semantics

datatype *cmd* = *CMP*

datatype ('*l*,'*k*,'*s*,'*prog*) *act* = *Cmd cmd*
| *EvtEnt* ('*l*,'*k*,'*s*,'*prog*) *event*

record ('*l*,'*k*,'*s*,'*prog*) *actk* = *Act* :: ('*l*,'*k*,'*s*,'*prog*) *act*
K :: '*k*

definition *get-actk* :: ('*l*,'*k*,'*s*,'*prog*) *act* ⇒ '*k* ⇒ ('*l*,'*k*,'*s*,'*prog*) *actk* (-♯- [*91*,*91*] *90*)
where *get-actk a k* ≡ (|*Act*=*a*, *K*=*k*|)

type-synonym ('*l*,'*k*,'*s*,'*prog*) *x* = '*k* ⇒ ('*l*,'*k*,'*s*,'*prog*) *event*

type-synonym ('*s*,'*prog*) *pconf* = '*prog* × '*s*

type-synonym ('*s*,'*prog*) *pconfs* = ('*s*,'*prog*) *pconf list*

definition *getspc-p* :: ('*s*,'*prog*) *pconf* ⇒ '*prog* **where**
getspc-p conf ≡ *fst conf*

definition *gets-p* :: ('*s*,'*prog*) *pconf* ⇒ '*s* **where**

$gets\text{-}p\ conf \equiv snd\ conf$

type-synonym $('l,'k,'s,'prog)\ econf = ((('l,'k,'s,'prog)\ event) \times ('s \times (('l,'k,'s,'prog)\ x))\ )$

type-synonym $('l,'k,'s,'prog)\ econfs = ('l,'k,'s,'prog)\ econf\ list$

definition $getspc\text{-}e :: ('l,'k,'s,'prog)\ econf \Rightarrow ('l,'k,'s,'prog)\ event$ **where**
$getspc\text{-}e\ conf \equiv fst\ conf$

definition $gets\text{-}e :: ('l,'k,'s,'prog)\ econf \Rightarrow 's$ **where**
$gets\text{-}e\ conf \equiv fst\ (snd\ conf)$

definition $getx\text{-}e :: ('l,'k,'s,'prog)\ econf \Rightarrow ('l,'k,'s,'prog)\ x$ **where**
$getx\text{-}e\ conf \equiv snd\ (snd\ conf)$

type-synonym $('l,'k,'s,'prog)\ esconf = ((('l,'k,'s,'prog)\ esys) \times ('s \times (('l,'k,'s,'prog)\ x))\ )$

type-synonym $('l,'k,'s,'prog)\ esconfs = ('l,'k,'s,'prog)\ esconf\ list$

definition $getspc\text{-}es :: ('l,'k,'s,'prog)\ esconf \Rightarrow ('l,'k,'s,'prog)\ esys$ **where**
$getspc\text{-}es\ conf \equiv fst\ conf$

definition $gets\text{-}es :: ('l,'k,'s,'prog)\ esconf \Rightarrow 's$ **where**
$gets\text{-}es\ conf \equiv fst\ (snd\ conf)$

definition $getx\text{-}es :: ('l,'k,'s,'prog)\ esconf \Rightarrow ('l,'k,'s,'prog)\ x$ **where**
$getx\text{-}es\ conf \equiv snd\ (snd\ conf)$

type-synonym $('l,'k,'s,'prog)\ pesconf = ((('l,'k,'s,'prog)\ paresys) \times ('s \times (('l,'k,'s,'prog)\ x))\ )$

type-synonym $('l,'k,'s,'prog)\ pesconfs = ('l,'k,'s,'prog)\ pesconf\ list$

definition $getspc :: ('l,'k,'s,'prog)\ pesconf \Rightarrow ('l,'k,'s,'prog)\ paresys$ **where**
$getspc\ conf \equiv fst\ conf$

definition $gets :: ('l,'k,'s,'prog)\ pesconf \Rightarrow 's$ **where**
$gets\ conf \equiv fst\ (snd\ conf)$

definition $getx :: ('l,'k,'s,'prog)\ pesconf \Rightarrow ('l,'k,'s,'prog)\ x$ **where**
$getx\ conf \equiv snd\ (snd\ conf)$

definition $getact :: ('l,'k,'s,'prog)\ actk \Rightarrow ('l,'k,'s,'prog)\ act$ **where**
$getact\ a \equiv Act\ a$

definition $getk :: ('l,'k,'s,'prog)\ actk \Rightarrow 'k$ **where**

getk $a \equiv K\ a$

locale *event* =
fixes *ptran* :: *'Env* ⇒ ((*'s*,*'prog*) *pconf* × (*'s*,*'prog*) *pconf*) *set*
fixes *petran* :: *'Env* ⇒ (*'s*,*'prog*) *pconf* ⇒ (*'s*,*'prog*) *pconf* ⇒ *bool* (- ⊢ - −*pe*→ - [*81*,*81*,*81*] *80*)
fixes *fin-com* :: *'prog*

assumes *petran-simps*:
  Γ ⊢ (*a*, *b*) −*pe*→ (*c*, *d*) ⟹ *a* = *c*
assumes *none-no-tran'*: ((*fin-com*, *s*),(*P*,*t*)) ∉ *ptran* Γ
assumes *ptran-neq*: ((*P*, *s*),(*P*,*t*)) ∉ *ptran* Γ
begin

definition *ptran'* :: *'Env* ⇒ (*'s*,*'prog*) *pconf* ⇒ (*'s*,*'prog*) *pconf* ⇒ *bool* (- ⊢ - −*c*→ - [*81*,*81*] *80*)
where Γ ⊢ *P* −*c*→ *Q* ≡ (*P*,*Q*) ∈ *ptran* Γ

definition *ptrans* :: *'Env* ⇒ (*'s*,*'prog*) *pconf* ⇒ (*'s*,*'prog*) *pconf* ⇒ *bool* (- ⊢ - −*c*∗→ - [*81*,*81*,*81*] *80*)
where Γ ⊢ *P* −*c*∗→ *Q* ≡ (*P*,*Q*) ∈(*ptran* Γ)ˆ∗

lemma *none-no-tran*: ¬(Γ ⊢ (*fin-com*,*s*) −*c*→ (*P*,*t*))
  using *none-no-tran'* **by**(*simp add*:*ptran'-def*)

lemma *none-no-tran2*: ¬(Γ ⊢ (*fin-com*,*s*) −*c*→ *Q*)
  using *none-no-tran* **by** (*metis prod.collapse*)

lemma *ptran-not-none*: (Γ ⊢ (*Q*,*s*) −*c*→ (*P*,*t*)) ⟹ *Q* ≠ *fin-com*
  using *none-no-tran* **apply**(*simp add*:*ptran'-def*) **using** *not-None-eq* **by** *metis*

3.2 Semantics of Events

inductive *etran* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *econf* ⇒ (*'l*,*'k*,*'s*,*'prog*) *actk* ⇒ (*'l*,*'k*,*'s*,*'prog*) *econf* ⇒ *bool*
(- ⊢ - −*et*−-→ - [*81*,*81*,*81*] *80*)
where
  AnonyEvent: Γ ⊢ (*P*, *s*) −*c*→ (*Q*, *t*) ⟹ Γ ⊢ (*AnonyEvent P*, *s*, *x*) −*et*−(*Cmd CMP*)♯*k*→ (*AnonyEvent Q*, *t*, *x*)
| *EventEntry*: ⟦*P* = *body e*; *P* ≠ *fin-com*; *s* ∈ *guard e*; *x'* = *x*(*k*:= *BasicEvent e*)⟧
  ⟹ Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ ((*AnonyEvent P*), *s*, *x'*)

3.3 Semantics of Event Systems

inductive *estran* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *esconf* ⇒ (*'l*,*'k*,*'s*,*'prog*) *actk* ⇒ (*'l*,*'k*,*'s*,*'prog*) *esconf* ⇒ *bool*
(- ⊢ - −*es*−-→ - [*81*,*81*] *80*)

where
EvtOccur: $[\![evt \in evts;\ \Gamma \vdash (evt,\ (s,\ x))\ -et-(EvtEnt\ evt)\sharp k\rightarrow\ (e,\ (s,\ x'))\ ]\!]$
$\Longrightarrow \Gamma \vdash (EvtSys\ evts,\ (s,\ x))\ -es-(EvtEnt\ evt)\sharp k\rightarrow\ (EvtSeq\ e\ (EvtSys\ evts),\ (s,\ x'))$
| *EvtSeq1*: $[\![\Gamma \vdash (e,\ s,\ x)\ -et-act\sharp k\rightarrow(e',\ s',\ x');\ e' \neq AnonyEvent\ fin\text{-}com]\!]$
$\Longrightarrow \Gamma \vdash (EvtSeq\ e\ es,\ s,\ x)\ -es-act\sharp k\rightarrow\ (EvtSeq\ e'\ es,\ s',\ x')$
| *EvtSeq2*: $[\![\Gamma \vdash (e,\ s,\ x)\ -et-act\sharp k\rightarrow\ (e',\ s',\ x');\ e' = AnonyEvent\ fin\text{-}com]\!]$
$\Longrightarrow \Gamma \vdash (EvtSeq\ e\ es,\ s,\ x)\ -es-act\sharp k\rightarrow\ (es,\ s',\ x')$

3.4 Semantics of Parallel Event Systems

inductive
$pestran :: 'Env \Rightarrow ('l,'k,'s,'prog)\ pesconf \Rightarrow ('l,'k,'s,'prog)\ actk$
$\Rightarrow ('l,'k,'s,'prog)\ pesconf \Rightarrow bool\ \ (\text{-} \vdash \text{-}\ -pes-\text{-}\rightarrow\ \text{-}\ [70,70]\ 60)$
where
ParES: $\Gamma \vdash (pes(k),\ (s,\ x))\ -es-(a\sharp k)\rightarrow\ (es',\ (s',\ x')) \Longrightarrow \Gamma \vdash (pes,\ (s,\ x))\ -pes-(a\sharp k)\rightarrow\ (pes(k{:=}es'),\ (s',\ x'))$

3.5 Lemmas

3.5.1 programs

lemma *list-eq-if* [*rule-format*]:
$\forall\, ys.\ xs{=}ys \longrightarrow (length\ xs\ =\ length\ ys) \longrightarrow (\forall\, i{<}length\ xs.\ xs!i{=}ys!i)$
by (*induct xs*) *auto*

lemma *list-eq*: $(length\ xs\ =\ length\ ys \wedge (\forall\, i{<}length\ xs.\ xs!i{=}ys!i)) = (xs{=}ys)$
apply(*rule iffI*)
apply *clarify*
apply(*erule nth-equalityI*)
apply *simp+*
done

lemma *nth-tl*: $[\![\ ys!0{=}a;\ ys{\neq}[]\ ]\!] \Longrightarrow ys{=}(a\#(tl\ ys))$
by (*cases ys*) *simp-all*

lemma *nth-tl-if* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ ys \longrightarrow P\ (a\#(tl\ ys))$
by (*induct ys*) *simp-all*

lemma *nth-tl-onlyif* [*rule-format*]: $ys{\neq}[] \longrightarrow ys!0{=}a \longrightarrow P\ (a\#(tl\ ys)) \longrightarrow P\ ys$
by (*induct ys*) *simp-all*

lemma *prog-not-eq-in-ctran-aux*:
assumes c: $\Gamma \vdash (P,s)\ -c\rightarrow\ (Q,t)$
shows $P{\neq}Q$ **using** *c*
using *ptran-neq* **apply**(*simp add:ptran'-def*) **apply** *auto*
done

lemma *prog-not-eq-in-ctran* [*simp*]: $\neg\ \Gamma \vdash (P,s)\ -c\rightarrow\ (P,t)$
apply *clarify* **using** *ptran-neq* **apply**(*simp add:ptran'-def*)

done

3.5.2 Events

lemma *ent-spec1*: $\Gamma \vdash (ev,\ s,\ x)\ -et-(EvtEnt\ be)\sharp k\rightarrow\ (e2,\ s1,\ x1) \Longrightarrow ev = be$
  apply(*rule etran.cases*)
  apply(*simp*)
  apply(*simp add*:*get-actk-def*)
  apply(*simp add*:*get-actk-def*)
  done

lemma *ent-spec*: $\Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow\ ec2 \Longrightarrow getspc\text{-}e\ ec1 = BasicEvent\ ev$
  by (*metis ent-spec1 getspc-e-def prod.collapse*)

lemma *ent-spec2'*: $\Gamma \vdash (ev,\ s,\ x)\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (e2,\ s1,\ x1)$
$\Longrightarrow s \in guard\ e \wedge s = s1$
$\wedge\ e2 = AnonyEvent\ ((body\ e)) \wedge x1 = x\ (k := BasicEvent\ e)$
  apply(*rule etran.cases*)
  apply(*simp*)
  apply(*simp add*:*get-actk-def*)+
  done

lemma *ent-spec2*: $\Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow\ ec2$
$\Longrightarrow gets\text{-}e\ ec1 \in guard\ ev \wedge gets\text{-}e\ ec1 = gets\text{-}e\ ec2$
$\wedge\ getspc\text{-}e\ ec2 = AnonyEvent\ ((body\ ev)) \wedge getx\text{-}e\ ec2 = (getx\text{-}e\ ec1)\ (k := BasicEvent\ ev)$
  using *getspc-e-def getx-e-def gets-e-def ent-spec2'* **by** (*metis surjective-pairing*)

lemma *no-tran2basic0*: $\Gamma \vdash (e1,\ s,\ x)\ -et-t\rightarrow\ (e2,\ s1,\ x1) \Longrightarrow \neg(\exists e.\ e2 = BasicEvent\ e)$
  apply(*rule etran.cases*)
  apply(*simp*)+
  done

lemma *no-tran2basic*: $\neg(\exists t\ ec1.\ \Gamma \vdash ec1\ -et-t\rightarrow\ (BasicEvent\ ev,\ s,\ x))$
  using *no-tran2basic0* **by** (*metis prod.collapse*)

lemma *noevtent-notran0*: $\Gamma \vdash (BasicEvent\ e,\ s,\ x)\ -et-(a\sharp k)\rightarrow\ (e2,\ s1,\ x1) \Longrightarrow a = EvtEnt\ (BasicEvent\ e)$
  apply(*rule etran.cases*)
  apply(*simp*)+
  apply(*simp add*:*get-actk-def*)
  done

lemma *noevtent-notran*: $ec1 = (BasicEvent\ e,\ s,\ x) \Longrightarrow \neg\ (\exists k.\ \Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2)$
$\Longrightarrow \neg\ (\Gamma \vdash ec1\ -et-t\rightarrow\ ec2)$

proof −
assume *p0*: *ec1* = (*BasicEvent e*, *s*, *x*) **and**
p1: ¬ (∃ *k*. Γ ⊢ *ec1* −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ *ec2*)
then show ¬ (Γ ⊢ *ec1* −*et*−*t*→ *ec2*)
proof −
{
assume *a0*: Γ ⊢ *ec1* −*et*−*t*→ *ec2*
with *p0* **have** *a1*: *getact t* = *EvtEnt* (*BasicEvent e*) **using** *getact-def noevtent-notran0 get-actk-def*
by (*metis cases prod-cases3 select-convs*(*1*))
from *a0* **obtain** *k* **where** *k* = *getk t* **by** *auto*
with *p1 a0 a1* **have** Γ ⊢ *ec1* −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ *ec2* **using** *get-actk-def getact-def*
by (*metis cases select-convs*(*1*))
with *p1* **have** *False* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *evt-not-eq-in-tran-aux*:Γ ⊢ (*P*,*s*,*x*) −*et*−*et*→ (*Q*,*t*,*y*) ⟹ *P* ≠ *Q*
apply(*erule etran.cases*)
apply (*simp add*: *prog-not-eq-in-ctran-aux*)
by *simp*

lemma *evt-not-eq-in-tran* [*simp*]: ¬ Γ ⊢ (*P*,*s*,*x*) −*et*−*et*→ (*P*,*t*,*y*)
apply *clarify*
apply(*drule evt-not-eq-in-tran-aux*)
apply *simp*
done

lemma *evt-not-eq-in-tran2* [*simp*]: ¬(∃ *et*. Γ ⊢ (*P*,*s*,*x*) −*et*−*et*→ (*P*,*t*,*y*)) **by** *simp*

3.5.3 Event Systems

lemma *esconf-trip*: ⟦*gets-es c* = *s*; *getspc-es c* = *spc*; *getx-es c* = *x*⟧ ⟹ *c* = (*spc*,*s*,*x*)
by (*metis gets-es-def getspc-es-def getx-es-def prod.collapse*)

lemma *evtseq-tran-evtseq*:
⟦Γ ⊢ (*EvtSeq e1 es*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*); *es2* ≠ *es*⟧ ⟹ ∃ *e*. *es2* = *EvtSeq e es*
apply(*rule estran.cases*)
apply(*simp*)+
done

lemma *evtseq-tran-evtseq-anony*:

$[\![\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ {-es-et\rightarrow}\ (es2,\ t1,\ y1);\ es2 \neq es]\!] \Longrightarrow \exists\, e.\ es2 = EvtSeq\ e\ es \wedge is\text{-}anonyevt\ e$
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis event.exhaust is-anonyevt.simps*(*1*) *no-tran2basic0*)
by *simp*

lemma *evtseq-tran-evtsys*:
$[\![\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ {-es-et\rightarrow}\ (es2,\ t1,\ y1);\ \neg(\exists\, e.\ es2 = EvtSeq\ e\ es)]\!] \Longrightarrow es2 = es$
apply(*rule estran.cases*)
apply(*simp*)+
done

lemma *evtseq-tran-exist-etran*:
$\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ {-es-et\rightarrow}\ (EvtSeq\ e2\ es,\ t1,\ y1) \Longrightarrow \exists\, t.\ \Gamma \vdash (e1,\ s1,\ x1)\ {-et-t\rightarrow}\ (e2,\ t1,\ y1)$
apply(*rule estran.cases*)
apply(*simp*)+
apply *blast*
by (*simp add*: *evtseq-ne-es*)

lemma *evtseq-tran-0-exist-etran*:
$\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ {-es-et\rightarrow}\ (es,\ t1,\ y1) \Longrightarrow \exists\, t.\ \Gamma \vdash (e1,\ s1,\ x1)\ {-et-t\rightarrow}\ (AnonyEvent\ fin\text{-}com,\ t1,\ y1)$
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis* (*no-types*, *hide-lams*) *add.commute add-Suc-right esys.size*(*3*) *not-less-eq trans-less-add2*)
by *auto*

lemma *notrans-to-basicevt-insameesys*:
$[\![\Gamma \vdash (es1,\ s1,\ x1)\ {-es-et\rightarrow}\ (es2,\ s2,\ x2);\ \exists\, e.\ es1 = EvtSeq\ e\ esys]\!] \Longrightarrow \neg(\exists\, e.\ es2 = EvtSeq\ (BasicEvent\ e)\ esys)$
apply(*rule estran.cases*)
apply *simp*
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
by (*metis evtseq-tran-exist-etran no-tran2basic*)

lemma *evtseq-tran-sys-or-seq*:
$\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ {-es-et\rightarrow}\ (es2,\ t1,\ y1) \Longrightarrow es2 = es \vee (\exists\, e.\ es2 = EvtSeq\ e\ es)$
by (*meson evtseq-tran-evtseq*)

lemma *evtseq-tran-sys-or-seq-anony*:

$\Gamma \vdash$ (*EvtSeq e1 es*, *s1*, *x1*) $-es-et\rightarrow$ (*es2*, *t1*, *y1*) $\Longrightarrow$ *es2* = *es* $\vee$ ($\exists$ *e*. *es2* = *EvtSeq e es* $\wedge$ *is-anonyevt e*)
by (*meson evtseq-tran-evtseq-anony*)

lemma *evtseq-no-evtent*:
$[\![\Gamma \vdash$ (*EvtSeq e1 es*, *s1*, *x1*) $-es-t\sharp k\rightarrow$ (*es2*, *s2*, *x2*);*is-anonyevt e1*$]\!] \Longrightarrow \neg$($\exists$ *e*. *t* = *EvtEnt e*)
apply(*rule estran.cases*)
apply(*simp*)+
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
done

lemma *evtseq-no-evtent2*:
$[\![\Gamma \vdash$ *esc1* $-es-t\sharp k\rightarrow$ *esc2*; *getspc-es esc1* = *EvtSeq e esys*; *is-anonyevt e*$]\!] \Longrightarrow$ $\neg$($\exists$ *e*. *t* = *EvtEnt e*)
proof −
assume *p0*: $\Gamma \vdash$ *esc1* $-es-t\sharp k\rightarrow$ *esc2*
and *p1*: *getspc-es esc1* = *EvtSeq e esys*
and *p2*: *is-anonyevt e*
then obtain *es1* **and** *s1* **and** *x1* **where** *a1*: *esc1* = (*es1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a2*: *esc2* = (*es2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
from *p1 a1* **have** *es1* = *EvtSeq e esys* **by** (*simp add*:*getspc-es-def*)
with *p0 p2 a1 a2* **show** *?thesis* **using** *evtseq-no-evtent*[*of* Γ *e esys s1 x1 t k es2 s2 x2*]
by *simp*
qed

lemma *esys-not-eseq*: *getspc-es esc* = *EvtSys es* $\Longrightarrow$ $\neg$($\exists$ *e esys*. *getspc-es esc* = *EvtSeq e esys*)
by(*simp add*:*getspc-es-def*)

lemma *eseq-not-esys*: *getspc-es esc* = *EvtSeq e esys* $\Longrightarrow$ $\neg$($\exists$ *es*. *getspc-es esc* = *EvtSys es*)
by(*simp add*:*getspc-es-def*)

lemma *evtent-is-basicevt*: $\Gamma \vdash$ (*es*, *s*, *x*) $-es-$*EvtEnt e*$\sharp k\rightarrow$ (*es'*, *s'*, *x'*) $\Longrightarrow \exists$ *e'*. *e* = *BasicEvent e'*
apply(*rule estran.cases*)
apply(*simp add*:*get-actk-def*)+
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
apply(*rule etran.cases*)
apply *simp*+
apply(*rule etran.cases*)

apply *simp*+
apply *auto*[*1*]
apply (*metis ent-spec1 event.exhaust evtseq-no-evtent get-actk-def is-anonyevt.simps*(*1*))+

done

lemma *evtent-is-basicevt-inevtseq*: $[\![\Gamma \vdash (EvtSeq\ e\ es, s1, x1)\ {-es-}EvtEnt\ e1\sharp k{\rightarrow}\ (esc2, s2, x2)]\!]$
$\Longrightarrow e = e1 \wedge (\exists e'.\ e = BasicEvent\ e')$
apply(*rule estran.cases*)
apply(*simp add*:*get-actk-def*)
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)
apply(*simp add*:*get-actk-def*)
apply *auto*[*1*]
by (*metis Pair-inject ent-spec1 esys.inject*(*1*) *evtent-is-basicevt get-actk-def*)

lemma *evtent-is-basicevt-inevtseq2*: $[\![\Gamma \vdash esc1\ {-es-}EvtEnt\ e1\sharp k{\rightarrow}\ esc2;\ getspc\text{-}es\ esc1 = EvtSeq\ e\ es]\!]$
$\Longrightarrow e = e1 \wedge (\exists e'.\ e = BasicEvent\ e')$
proof −
assume *p0*: $\Gamma \vdash esc1\ {-es-}EvtEnt\ e1\sharp k{\rightarrow}\ esc2$
and *p1*: *getspc-es esc1* = *EvtSeq e es*
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: *esc2* = (*es2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
ultimately show *?thesis*
using *p0 p1 evtent-is-basicevt-inevtseq*[*of* Γ *e es s1 x1 e1 k es2 s2 x2*] *getspc-es-def*[*of esc1*] **by** *auto*
qed

lemma *evtsysent-evtent0*: $\Gamma \vdash (EvtSys\ es,\ s,\ x)\ {-es-}t{\rightarrow}\ (EvtSeq\ ev\ (EvtSys\ es),\ s1, x1) \Longrightarrow$
$s = s1 \wedge (\exists evt\ e.\ evt \in es \wedge evt = BasicEvent\ e \wedge Act\ t = EvtEnt\ (BasicEvent\ e) \wedge$
$\Gamma \vdash (BasicEvent\ e,\ s,\ x)\ {-et-}t{\rightarrow}\ (ev,\ s1,\ x1))$
apply(*rule estran.cases*)
apply(*simp*)
prefer *2*
apply(*simp*)
prefer *2*
apply(*simp*)
apply(*rule etran.cases*)

apply(*simp*)
apply(*simp add*:*get-actk-def*)
apply(*rule conjI*)
apply(*simp*)
using *get-actk-def*
by (*metis Pair-inject esys.inject*(*1*) *esys.inject*(*2*) *select-convs*(*1*))

lemma *evtsysent-evtent*: $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) $\Longrightarrow$
$s = s1 \wedge$ *BasicEvent e* $\in$ *es* $\wedge$ $\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) $-et-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*ev*, *s1*, *x1*)
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis ent-spec1*)
apply(*simp*)+
done

lemma *evtsysent-evtent2*: $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt ev*)$\sharp k\rightarrow$ (*esc2*, *s1*,*x1*) $\Longrightarrow$
$s = s1 \wedge (ev \in es)$
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis ent-spec1*)
apply(*simp*)+
done

lemma *evtsysent-evtent3*: $[\![\Gamma \vdash$ *esc1* $-es-$(*EvtEnt ev*)$\sharp k\rightarrow$ *esc2*; *getspc-es esc1* = *EvtSys es*$]\!] \Longrightarrow$
$(ev \in es)$
proof −
assume *p0*: $\Gamma \vdash$ *esc1* $-es-$(*EvtEnt ev*)$\sharp k\rightarrow$ *esc2*
and *p1*: *getspc-es esc1* = *EvtSys es*
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: *esc2* = (*es2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
from *p1 a0* **have** *es1* = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
with *a0 a1 p0* **show** *?thesis* **using** *evtsysent-evtent2*[*of* Γ *es s1 x1 ev k es2 s2 x2*] **by** *simp*
qed

lemma *evtsys-evtent*: $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-t\rightarrow$ (*es2*, *s1*,*x1*) $\Longrightarrow \exists e.$ *es2* = *EvtSeq e* (*EvtSys es*)
apply(*rule estran.cases*)
apply(*simp*)+
done

lemma *act-in-es-notchgstate*: $[\![\Gamma \vdash (es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!] \Longrightarrow$ $x = x'$
  apply(*rule estran.cases*)
  apply (*simp add*: *get-actk-def*)+
  apply(*rule etran.cases*)
  apply (*simp add*: *get-actk-def*)+
  apply(*rule etran.cases*)
  by (*simp add*: *get-actk-def*)+

lemma *cmd-enable-impl-anonyevt*:
  $[\![\Gamma \vdash (es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!]$
    $\Longrightarrow \exists\, e\ e'\ es1.\ es = EvtSeq\ e\ es1 \wedge e = AnonyEvent\ e'$
  apply(*rule estran.cases*)
  apply (*simp add*: *get-actk-def*)+
  apply(*rule etran.cases*)
  apply (*simp add*: *get-actk-def*)+
  apply(*rule etran.cases*)
  apply (*simp add*: *get-actk-def*)+
  done

lemma *cmd-enable-impl-notesys*:
  $[\![\Gamma \vdash (es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!]$
    $\Longrightarrow \neg(\exists\, ess.\ es = EvtSys\ ess)$
  apply(*rule estran.cases*)
  apply (*simp add*: *get-actk-def*)+
  done

lemma *cmd-enable-impl-notesys2*:
  $[\![\Gamma \vdash esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2]\!]$
    $\Longrightarrow \neg(\exists\, ess.\ getspc\text{-}es\ esc1 = EvtSys\ ess)$
  proof −
    assume *p0*: $\Gamma \vdash esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2$
    then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
      using *prod-cases3* **by** *blast*
    moreover
    from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
      using *prod-cases3* **by** *blast*
    ultimately show *?thesis* **using** *p0 cmd-enable-impl-notesys*[*of* Γ *es1 s1 x1 c k es2 s2 x2*] *getspc-es-def*[*of esc1*]
      by *simp*
  qed

lemma *cmd-enable-impl-anonyevt2*:
  $[\![\Gamma \vdash esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2]\!]$
    $\Longrightarrow \exists\, e\ e'\ es1.\ getspc\text{-}es\ esc1 = EvtSeq\ e\ es1 \wedge e = AnonyEvent\ e'$
  proof −
    assume *p0*: $\Gamma \vdash esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2$
    then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
      using *prod-cases3* **by** *blast*

moreover
from $p0$ obtain $es2$ and $s2$ and $x2$ where $a1$: $esc2 = (es2,s2,x2)$
using *prod-cases3* by *blast*
ultimately show *?thesis* using *p0 cmd-enable-impl-anonyevt*[*of* Γ *es1 s1 x1 c k es2 s2 x2*] *getspc-es-def*[*of esc1*]
by *simp*
qed

lemma *entevt-notchgstate*: $[\![\Gamma \vdash (es, s, x) -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow (es', s', x')]\!] \Longrightarrow s = s'$
apply(*rule estran.cases*)
apply(*simp*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*
using *ent-spec2'* *get-actk-def* **by** *metis*

lemma *entevt-ines-notchg-otherx*: $[\![\Gamma \vdash (es, s, x) -es-(EvtEnt\ e)\sharp k\rightarrow (es', s', x')]\!] \Longrightarrow (\forall k'.\ k'\neq k \longrightarrow x\ k' = x'\ k')$
apply(*rule estran.cases*)
apply(*simp*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
done

lemma *entevt-ines-notchg-otherx2*: $[\![\Gamma \vdash esc1 -es-(EvtEnt\ e)\sharp k\rightarrow esc2]\!]$
$\Longrightarrow (\forall k'.\ k'\neq k \longrightarrow (getx\text{-}es\ esc1)\ k' = (getx\text{-}es\ esc2)\ k')$
proof −
assume $p0$: $\Gamma \vdash esc1 -es-(EvtEnt\ e)\sharp k\rightarrow esc2$
then obtain $es1$ **and** $s1$ **and** $x1$ **where** $a0$: $esc1 = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
moreover
from $p0$ **obtain** $es2$ **and** $s2$ **and** $x2$ **where** $a1$: $esc2 = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
ultimately have $\forall k'.\ k'\neq k \longrightarrow x1\ k' = x2\ k'$
using *entevt-ines-notchg-otherx*[*of* Γ *es1 s1 x1 e k es2 s2 x2*] *p0* **by** *simp*
with *a0 a1* **show** *?thesis* **using** *getx-es-def* **by** (*metis snd-conv*)
qed

lemma *cmd-ines-nchg-x*: $[\![\Gamma \vdash (es, s, x) -es-(Cmd\ c)\sharp k\rightarrow (es', s', x')]\!] \Longrightarrow (\forall k.\ x'\ k = x\ k)$
apply(*rule estran.cases*)
apply(*simp*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+

apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
done

lemma *cmd-ines-nchg-x2*: $[\![\Gamma \vdash esc1 \ {-}es{-}(Cmd\ c)\sharp k{\rightarrow}\ esc2]\!] \Longrightarrow (\forall k.\ (getx\text{-}es\ esc2)\ k = (getx\text{-}es\ esc1)\ k)$
proof −
assume *p0*: $\Gamma \vdash esc1 \ {-}es{-}(Cmd\ c)\sharp k{\rightarrow}\ esc2$
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
ultimately have $\forall k.\ x1\ k = x2\ k$ **using** *cmd-ines-nchg-x* [*of* Γ *es1 s1 x1 c k es2 s2 x2*] *p0* **by** *simp*
with *a0 a1* **show** *?thesis* **using** *getx-es-def* **by** (*metis snd-conv*)
qed

lemma *entevt-ines-chg-selfx*: $[\![\Gamma \vdash (es,\ s,\ x) \ {-}es{-}(EvtEnt\ e)\sharp k{\rightarrow}\ (es',\ s',\ x')]\!] \Longrightarrow x'\ k = e$
apply(*rule estran.cases*)
apply(*simp*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
done

lemma *entevt-ines-chg-selfx2*: $[\![\Gamma \vdash esc1 \ {-}es{-}(EvtEnt\ e)\sharp k{\rightarrow}\ esc2]\!] \Longrightarrow (getx\text{-}es\ esc2)\ k = e$
proof −
assume *p0*: $\Gamma \vdash esc1 \ {-}es{-}(EvtEnt\ e)\sharp k{\rightarrow}\ esc2$
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
ultimately have $x2\ k = e$ **using** *entevt-ines-chg-selfx p0* **by** *auto*
with *a1* **show** *?thesis* **using** *getx-es-def* **by** (*metis snd-conv*)
qed

lemma *estran-impl-evtentorcmd*: $[\![\Gamma \vdash (es,\ s,\ x) \ {-}es{-}t{\rightarrow}\ (es',\ s',\ x')]\!]$
$\Longrightarrow (\exists e\ k.\ \Gamma \vdash (es,\ s,\ x) \ {-}es{-}EvtEnt\ e\sharp k{\rightarrow}\ (es',\ s',\ x')) \vee (\exists c\ k.\ \Gamma \vdash (es,\ s,\ x) \ {-}es{-}Cmd\ c\sharp k{\rightarrow}\ (es',\ s',\ x'))$
apply(*rule estran.cases*)

apply (*simp add*: *get-actk-def*)
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*[*1*]
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*[*1*]
apply (*metis get-actk-def*)
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)
apply (*metis get-actk-def*)
apply (*metis get-actk-def*)
done

lemma *estran-impl-evtentorcmd'*: $\llbracket\Gamma \vdash (es,\ s,\ x)\ -es-t\sharp k\rightarrow (es',\ s',\ x')\rrbracket$
$\Longrightarrow (\exists\, e.\ \Gamma \vdash (es,\ s,\ x)\ -es-EvtEnt\ e\sharp k\rightarrow\ (es',\ s',\ x')) \vee (\exists\, c.\ \Gamma \vdash (es,\ s,\ x)$
$-es-Cmd\ c\sharp k\rightarrow\ (es',\ s',\ x'))$
apply(*rule estran.cases*)
apply *simp*
apply (*metis get-actk-def iffs*)
apply(*rule etran.cases*)
apply *simp*
apply (*metis get-actk-def iffs*)
apply (*metis get-actk-def iffs*)
apply(*rule etran.cases*)
apply *simp*
apply (*metis get-actk-def iffs*)
apply (*metis get-actk-def iffs*)
done

lemma *estran-impl-evtentorcmd2*: $\llbracket\Gamma \vdash esc1\ -es-t\rightarrow\ esc2\rrbracket$
$\Longrightarrow (\exists\, e\ k.\ \Gamma \vdash esc1\ -es-EvtEnt\ e\sharp k\rightarrow\ esc2) \vee (\exists\, c\ k.\ \Gamma \vdash esc1\ -es-Cmd$
$c\sharp k\rightarrow\ esc2)$
proof −
assume *p0*: $\Gamma \vdash esc1\ -es-t\rightarrow\ esc2$
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
ultimately show *?thesis* **using** *p0 estran-impl-evtentorcmd*[*of* Γ *es1 s1 x1 t*
es2 s2 x2] **by** *simp*
qed

lemma *estran-impl-evtentorcmd2'*: $\llbracket\Gamma \vdash esc1\ -es-t\sharp k\rightarrow\ esc2\rrbracket$
$\Longrightarrow (\exists\, e.\ \Gamma \vdash esc1\ -es-EvtEnt\ e\sharp k\rightarrow\ esc2) \vee (\exists\, c.\ \Gamma \vdash esc1\ -es-Cmd\ c\sharp k\rightarrow$
$esc2)$
proof −
assume *p0*: $\Gamma \vdash esc1\ -es-t\sharp k\rightarrow\ esc2$

then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: *esc2* = (*es2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
ultimately show *?thesis* **using** *p0 estran-impl-evtentorcmd'*[*of* Γ *es1 s1 x1 t k es2 s2 x2*] **by** *simp*
qed

3.5.4 Parallel Event Systems

lemma *pesconf-trip*: ⟦*gets c* = *s*; *getspc c* = *spc*; *getx c* = *x*⟧ ⟹ *c* = (*spc*,*s*,*x*)
by (*metis gets-def getspc-def getx-def prod.collapse*)

lemma *pestran-estran*: ⟦Γ ⊢ (*pes*, *s*, *x*) −*pes*−(*a*♯*k*)→ (*pes'*, *s'*, *x'*)⟧ ⟹
∃ *es'*. (Γ ⊢ (*pes k*, *s*, *x*) −*es*−(*a*♯*k*)→ (*es'*, *s'*, *x'*)) ∧ *pes'* = *pes*(*k*:=*es'*)
apply(*rule pestran.cases*)
apply(*simp*)
apply(*simp add*:*get-actk-def*)
by *auto*

lemma *act-in-pes-notchgstate*: ⟦Γ ⊢ (*pes*, *s*, *x*) −*pes*−(*Cmd c*)♯*k*→ (*pes'*, *s'*, *x'*)⟧ ⟹ *x* = *x'*
apply(*rule pestran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule estran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
done

lemma *evtent-in-pes-notchgstate*: ⟦Γ ⊢ (*pes*, *s*, *x*) −*pes*−(*EvtEnt e*)♯*k*→ (*pes'*, *s'*, *x'*)⟧ ⟹ *s* = *s'*
apply(*rule pestran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule estran.cases*)
apply (*simp add*: *get-actk-def*)+
apply (*metis entevt-notchgstate evtent-is-basicevt get-actk-def*)
by (*metis entevt-notchgstate evtent-is-basicevt get-actk-def*)

lemma *evtent-in-pes-notchgstate2*: ⟦Γ ⊢ *esc1* −*pes*−(*EvtEnt e*)♯*k*→ *esc2*⟧ ⟹ *gets esc1* = *gets esc2*
using *evtent-in-pes-notchgstate* **by** (*metis pesconf-trip*)

end

end

4 Computations of PiCore Language

theory *PiCore-Computation*
imports *PiCore-Semantics*
begin

4.1 Environment transitions

locale *event-comp* = *event ptran petran fin-com*
for *ptran* :: $'Env \Rightarrow (('s,'prog)\ pconf \times ('s,'prog)\ pconf)\ set$
and *petran* :: $'Env \Rightarrow ('s,'prog)\ pconf \Rightarrow ('s,'prog)\ pconf \Rightarrow bool$ $(- \vdash - -pe\rightarrow -$ $[81,81,81]\ 80)$
and *fin-com* :: $'prog$
+
fixes *cpts-p* :: $'Env \Rightarrow ('s,'prog)\ pconfs\ set$
fixes *cpts-of-p* :: $'Env \Rightarrow 'prog \Rightarrow 's \Rightarrow (('s,'prog)\ pconfs)\ set$

assumes *cpts-p-simps*:
$((\exists P\ s.\ aa = [(P,\ s)]) \vee$
$(\exists P\ t\ xs\ s.\ aa = (P,\ s)\ \#\ (P,\ t)\ \#\ xs \wedge (P,\ t)\ \#\ xs \in cpts\text{-}p\ \Gamma) \vee$
$(\exists P\ s\ Q\ t\ xs.\ aa = (P,\ s)\ \#\ (Q,\ t)\ \#\ xs \wedge \Gamma \vdash (P,\ s) -c\rightarrow (Q,\ t) \wedge (Q,\ t)$ $\#\ xs \in cpts\text{-}p\ \Gamma)) \Longrightarrow (aa \in cpts\text{-}p\ \Gamma)$
assumes *cptn-not-empty* [*simp*]:$[] \notin cpts\text{-}p\ \Gamma$

assumes *cpts-of-p-def*: $l!0 = (P,s) \wedge l \in cpts\text{-}p\ \Gamma \Longrightarrow l \in cpts\text{-}of\text{-}p\ \Gamma\ P\ s$

begin

lemma *CptsPOne*: $[(P,s)] \in cpts\text{-}p\ \Gamma$
using *cpts-p-simps*[*of* $[(P,s)]\ \Gamma$] **by** *auto*

lemma *CptsPEnv*: $(P,\ t)\#xs \in cpts\text{-}p\ \Gamma \Longrightarrow (P,s)\#(P,t)\#xs \in cpts\text{-}p\ \Gamma$
using *cpts-p-simps*[*of* $(P,\ s)\ \#\ (P,\ t)\ \#\ xs\ \Gamma$] **by** *auto*

lemma *CptsPComp*: $[\![\Gamma \vdash (P,s) -c\rightarrow (Q,t);\ (Q,\ t)\#xs \in cpts\text{-}p\ \Gamma]\!] \Longrightarrow (P,s)\#(Q,t)\#xs$ $\in cpts\text{-}p\ \Gamma$
using *cpts-p-simps*[*of* $(P,\ s)\ \#\ (Q,\ t)\ \#\ xs\ \Gamma$] **by** *auto*

4.2 Sequential computations

4.2.1 Sequential computations of programs

inductive
eetran :: $'Env \Rightarrow ('l,'k,'s,'prog)\ econf \Rightarrow ('l,'k,'s,'prog)\ econf \Rightarrow bool$ $(- \vdash -$ $-ee\rightarrow -\ [81,81,81]\ 80)$
for Γ :: $'Env$
where
EnvE: $\Gamma \vdash (P,\ s,\ x) -ee\rightarrow (P,\ t,\ y)$

lemma *eetranE*: $\Gamma \vdash p -ee\rightarrow p' \Longrightarrow (\bigwedge P\ s\ t.\ p = (P,\ s) \Longrightarrow p' = (P,\ t) \Longrightarrow Q)$

$\Longrightarrow Q$
by (*induct p*, *induct p'*, *erule eetran.cases*, *blast*)

inductive
esetran :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *esconf* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *esconf* $\Rightarrow$ *bool* (- $\vdash$ - $-ese\rightarrow$ - [*81*,*81*,*81*] *80*)
where
EnvES: $\Gamma \vdash (P,\ s,\ x)\ -ese\rightarrow (P,\ t,\ y)$

lemma *esetranE*: $\Gamma \vdash p\ -ese\rightarrow p' \Longrightarrow (\bigwedge P\ s\ t.\ p = (P,\ s) \Longrightarrow p' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
by (*induct p*, *induct p'*, *erule esetran.cases*, *blast*)

inductive
pesetran :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *pesconf* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *pesconf* $\Rightarrow$ *bool* (- $\vdash$ - $-pese\rightarrow$ - [*81*,*81*,*81*] *80*)
where
EnvPES: $\Gamma \vdash (P,\ s,\ x)\ -pese\rightarrow (P,\ t,\ y)$

lemma *pesetranE*: $\Gamma \vdash p\ -pese\rightarrow p' \Longrightarrow (\bigwedge P\ s\ t.\ p = (P,\ s) \Longrightarrow p' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
by (*induct p*, *induct p'*, *erule pesetran.cases*, *blast*)

4.2.2 Sequential computations of events

inductive-set *cpts-ev* :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *econfs set*
for Γ :: *'Env*
where
CptsEvOne: $[(e,s,x)] \in$ *cpts-ev* Γ
| *CptsEvEnv*: $(e,\ t,\ x)\#xs \in$ *cpts-ev* $\Gamma \Longrightarrow (e,\ s,\ y)\#(e,\ t,\ x)\#xs \in$ *cpts-ev* Γ
| *CptsEvComp*: $[\![\Gamma \vdash (e1,s,x)\ -et-ct\rightarrow (e2,t,y);\ (e2,t,y)\#xs \in$ *cpts-ev* $\Gamma]\!] \Longrightarrow (e1,s,x)\#(e2,t,y)\#xs \in$ *cpts-ev* Γ

definition *cpts-of-ev* :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *event* $\Rightarrow$ *'s* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *x* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *econfs set* **where**
cpts-of-ev Γ *ev s x* $\equiv \{l.\ l!0=(ev,(s,x)) \wedge l \in$ *cpts-ev* $\Gamma\}$

4.2.3 Sequential computations of event systems

inductive-set *cpts-es* :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *esconfs set*
for Γ :: *'Env*
where
CptsEsOne: $[(es,s,x)] \in$ *cpts-es* Γ
| *CptsEsEnv*: $(es,\ t,\ x)\#xs \in$ *cpts-es* $\Gamma \Longrightarrow (es,\ s,\ y)\#(es,\ t,\ x)\#xs \in$ *cpts-es* Γ
| *CptsEsComp*: $[\![\Gamma \vdash (es1,s,x)\ -es-ct\rightarrow (es2,t,y);\ (es2,t,y)\#xs \in$ *cpts-es* $\Gamma]\!] \Longrightarrow (es1,s,x)\#(es2,t,y)\#xs \in$ *cpts-es* Γ

definition *cpts-of-es* :: *'Env* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *esys* $\Rightarrow$ *'s* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *x* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *esconfs set* **where**
cpts-of-es Γ *es s x* $\equiv \{l.\ l!0=(es,s,x) \wedge l \in$ *cpts-es* $\Gamma\}$

4.2.4 Sequential computations of par event systems

inductive-set *cpts-pes* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *pesconfs set*
for Γ :: *'Env*
where
CptsPesOne: [(*pes*,*s*,*x*)] ∈ *cpts-pes* Γ
| *CptsPesEnv*: (*pes*, *t*, *x*)#*xs* ∈ *cpts-pes* Γ ⟹ (*pes*, *s*, *y*)#(*pes*, *t*, *x*)#*xs* ∈ *cpts-pes* Γ
| *CptsPesComp*: ⟦Γ ⊢ (*pes1*,*s*,*x*) −*pes*−*ct*→ (*pes2*,*t*,*y*); (*pes2*,*t*,*y*)#*xs* ∈ *cpts-pes* Γ⟧ ⟹ (*pes1*,*s*,*x*)#(*pes2*,*t*,*y*)#*xs* ∈ *cpts-pes* Γ

definition *cpts-of-pes* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *paresys* ⇒ *'s* ⇒ (*'l*,*'k*,*'s*,*'prog*) *x* ⇒ (*'l*,*'k*,*'s*,*'prog*) *pesconfs set* **where**
cpts-of-pes Γ *pes s x* ≡ {*l*. *l*!*0*=(*pes*,*s*,*x*) ∧ *l* ∈ *cpts-pes* Γ}

4.3 Lemmas

4.3.1 Events

lemma *cpts-e-not-empty* [*simp*]:[] ∉ *cpts-ev* Γ
apply(*force elim*:*cpts-ev.cases*)
done

lemma *eetran-eqconf*: Γ ⊢ (*e1*, *s1*, *x1*) −*ee*→ (*e2*, *s2*, *x2*) ⟹ *e1* = *e2*
apply(*rule eetran.cases*)
apply(*simp*)+
done

lemma *eetran-eqconf1*: Γ ⊢ *ec1* −*ee*→ *ec2* ⟹ *getspc-e ec1* = *getspc-e ec2*
proof −
assume *a0*: Γ ⊢ *ec1* −*ee*→ *ec2*
then obtain *e1* **and** *s1* **and** *x1* **and** *e2* **and** *s2* **and** *x2* **where** *a1*: *ec1* = (*e1*, *s1*, *x1*) **and** *a2*: *ec2* = (*e2*, *s2*, *x2*)
by (*meson prod-cases3*)
then have *e1* = *e2* **using** *a0 eetran-eqconf* **by** *fastforce*
with *a1* **show** *?thesis* **by** (*simp add*: *a2 getspc-e-def*)
qed

lemma *eqconf-eetran1*: *e1* = *e2* ⟹ Γ ⊢ (*e1*, *s1*, *x1*) −*ee*→ (*e2*, *s2*, *x2*)
by (*simp add*: *eetran.intros*)

lemma *eqconf-eetran*: *getspc-e ec1* = *getspc-e ec2* ⟹ Γ ⊢ *ec1* −*ee*→ *ec2*
proof −
assume *getspc-e ec1* = *getspc-e ec2*
then show *?thesis* **using** *getspc-e-def eetran.EnvE* **by** (*metis eq-fst-iff*)
qed

lemma *cpts-ev-sub0*: ⟦*el* ∈ *cpts-ev* Γ; *Suc 0* < *length el*⟧ ⟹ *drop* (*Suc 0*) *el* ∈ *cpts-ev* Γ

apply(*rule cpts-ev.cases*)
apply(*simp*)+
done

lemma *cpts-ev-subi*: $[\![el \in \mathit{cpts\text{-}ev}\ \Gamma;\ \mathit{Suc}\ i < \mathit{length}\ el]\!] \Longrightarrow \mathit{drop}\ (\mathit{Suc}\ i)\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
proof −
assume *p0*:$el \in \mathit{cpts\text{-}ev}\ \Gamma$ **and** *p1*:$\mathit{Suc}\ i < \mathit{length}\ el$
have $\forall el\ i.\ el \in \mathit{cpts\text{-}ev}\ \Gamma \wedge \mathit{Suc}\ i < \mathit{length}\ el \longrightarrow \mathit{drop}\ (\mathit{Suc}\ i)\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
proof −
{
fix *el i*
have $el \in \mathit{cpts\text{-}ev}\ \Gamma \wedge \mathit{Suc}\ i < \mathit{length}\ el \longrightarrow \mathit{drop}\ (\mathit{Suc}\ i)\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
proof(*induct i*)
case *0* **show** *?case* **by** (*simp add*: *cpts-ev-sub0*)
next
case (*Suc j*)
assume *b0*: $el \in \mathit{cpts\text{-}ev}\ \Gamma \wedge \mathit{Suc}\ j < \mathit{length}\ el \longrightarrow \mathit{drop}\ (\mathit{Suc}\ j)\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
show *?case*
proof
assume *c0*: $el \in \mathit{cpts\text{-}ev}\ \Gamma \wedge \mathit{Suc}\ (\mathit{Suc}\ j) < \mathit{length}\ el$
with *b0* **have** *c1*: $\mathit{drop}\ (\mathit{Suc}\ j)\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
by (*simp add*: *c0 Suc-lessD*)
then show $\mathit{drop}\ (\mathit{Suc}\ (\mathit{Suc}\ j))\ el \in \mathit{cpts\text{-}ev}\ \Gamma$
using *c0 cpts-ev-sub0* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p0 p1* **show** *?thesis* **by** *auto*
qed

lemma *notran-confeq0*: $[\![el \in \mathit{cpts\text{-}ev}\ \Gamma;\ \mathit{Suc}\ 0 < \mathit{length}\ el;\ \neg\ (\exists t.\ \Gamma \vdash el\ !\ 0\ -et-t\rightarrow\ el\ !\ 1)]\!]$
$\Longrightarrow \mathit{getspc\text{-}e}\ (el\ !\ 0) = \mathit{getspc\text{-}e}\ (el\ !\ 1)$
apply(*simp*)
apply(*rule cpts-ev.cases*)
apply(*simp*)+
apply(*simp add*:*getspc-e-def*)+
done

lemma *notran-confeqi*: $[\![el \in \mathit{cpts\text{-}ev}\ \Gamma;\ \mathit{Suc}\ i < \mathit{length}\ el;\ \neg\ (\exists t.\ \Gamma \vdash el\ !\ i\ -et-t\rightarrow\ el\ !\ \mathit{Suc}\ i)]\!]$
$\Longrightarrow \mathit{getspc\text{-}e}\ (el\ !\ i) = \mathit{getspc\text{-}e}\ (el\ !\ (\mathit{Suc}\ i))$
proof −
assume *p0*: $el \in \mathit{cpts\text{-}ev}\ \Gamma$ **and**
p1: $\mathit{Suc}\ i < \mathit{length}\ el$ **and**

p2: $\neg$ ($\exists$ *t*. $\Gamma \vdash$ *el* ! *i* $-et-t\rightarrow$ *el* ! *Suc i*)
have $\forall$ *el i*. *el* $\in$ *cpts-ev* Γ $\wedge$ *Suc i* $<$ *length el* $\wedge$ $\neg$ ($\exists$ *t*. $\Gamma \vdash$ *el* ! *i* $-et-t\rightarrow$ *el* ! *Suc i*)
$\longrightarrow$ *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! (*Suc i*))
proof $-$
{
fix *el i*
assume *a0*: *el* $\in$ *cpts-ev* Γ $\wedge$ *Suc i* $<$ *length el* $\wedge$ $\neg$ ($\exists$ *t*. $\Gamma \vdash$ *el* ! *i* $-et-t\rightarrow$ *el* ! *Suc i*)
then have *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! (*Suc i*))
proof(*induct i*)
case *0* **then show** *?case*
using *notran-confeq0* **by** (*metis One-nat-def*)
next
case (*Suc j*)
let *?subel* = *drop* (*Suc j*) *el*
assume *b0*: *el* $\in$ *cpts-ev* Γ $\wedge$ *Suc* (*Suc j*) $<$ *length el* $\wedge$ $\neg$ ($\exists$ *t*. $\Gamma \vdash$ *el* ! *Suc j* $-et-t\rightarrow$ *el* ! *Suc* (*Suc j*))
then have *b1*: *?subel* $\in$ *cpts-ev* Γ **by** (*simp add*: *Suc-lessD b0 cpts-ev-subi*)

from *b0* **have** *b2*: *Suc 0* $<$ *length ?subel* **by** *auto*
from *b0* **have** *b3*: $\neg$ ($\exists$ *t*. $\Gamma \vdash$ *?subel* ! *0* $-et-t\rightarrow$ *?subel* ! *1*) **by** *auto*
with *b1 b2* **have** *b3*: *getspc-e* (*?subel* ! *0*) = *getspc-e* (*?subel* ! *1*)
using *notran-confeq0* **by** *blast*
then show *?case*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD b0 nth-Cons-0 nth-Cons-Suc*)
qed
}
then show *?thesis* **by** *auto*
qed
with *p0 p1 p2* **show** *?thesis* **by** *auto*
qed

lemma *cpts-ev-onemore*: $[\![$*el* $\in$ *cpts-ev* Γ; *length el* $>$ *0*; $\Gamma \vdash$ *el* ! (*length el* $-$ *1*) $-et-t\rightarrow$ *ec*$]\!] \Longrightarrow$
el @ [*ec*] $\in$ *cpts-ev* Γ
proof $-$
assume *p0*: *el* $\in$ *cpts-ev* Γ
and *p1*: *length el* $>$ *0*
and *p2*: $\Gamma \vdash$ *el* ! (*length el* $-$ *1*) $-et-t\rightarrow$ *ec*

have $\forall$ *el ec t* Γ. *el* $\in$ *cpts-ev* Γ $\wedge$ *length el* $>$ *0* $\wedge$ $\Gamma \vdash$ *el* ! (*length el* $-$ *1*) $-et-t\rightarrow$ *ec* $\longrightarrow$ *el* @ [*ec*] $\in$ *cpts-ev* Γ
proof $-$
{
fix *el ec t* Γ
assume *a0*: *el* $\in$ *cpts-ev* Γ
and *a1*: *length el* $>$ *0*

and *a2*: $\Gamma \vdash$ *el* ! (*length el* − *1*) −*et*−*t*→ *ec*
from *a0 a1 a2* **have** *el* @ [*ec*] ∈ *cpts-ev* Γ
proof(*induct el*)
case (*CptsEvOne e s x*)
assume *b0*: $\Gamma \vdash$ [(*e*, *s*, *x*)] ! (*length* [(*e*, *s*, *x*)] − *1*) −*et*−*t*→ *ec*
then have $\Gamma \vdash$ (*e*, *s*, *x*) −*et*−*t*→ *ec* **by** *simp*
then show *?case* **by** (*metis append-Cons append-Nil cpts-ev.CptsEvComp*

cpts-ev.CptsEvOne surj-pair)
next
case (*CptsEvEnv e s1 x xs s2 y*)
assume *b0*: (*e*, *s1*, *x*) # *xs* ∈ *cpts-ev* Γ
and *b1*: *0* < *length* ((*e*, *s1*, *x*) # *xs*) ⟹
$\Gamma \vdash$ ((*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s1*, *x*) # *xs*) − *1*) −*et*−*t*→
ec
⟹ ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-ev* Γ
and *b2*: *0* < *length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*)
and *b3*: $\Gamma \vdash$ ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) − *1*) −*et*−*t*→ *ec*
then show *?case*
proof(*cases xs* = [])
assume *c0*: *xs* = []
with *b3* **have** $\Gamma \vdash$ (*e*, *s1*, *x*)−*et*−*t*→ *ec* **by** *simp*
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-ev* Γ **by** *simp*
then show *?thesis* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
next
assume *c0*: *xs* ≠ []
with *b3* **have** $\Gamma \vdash$ *last xs* −*et*−*t*→ *ec* **by** (*simp add*: *last-conv-nth*)
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-ev* Γ **using** *b3* **by** *auto*
then show *?thesis* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
qed
next
case (*CptsEvComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *b0*: $\Gamma \vdash$ (*e1*, *s1*, *x1*) −*et*−*et*→ (*e2*, *t1*, *y1*)
and *b1*: (*e2*, *t1*, *y1*) # *xs1* ∈ *cpts-ev* Γ
and *b2*: *0* < *length* ((*e2*, *t1*, *y1*) # *xs1*) ⟹
$\Gamma \vdash$ ((*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e2*, *t1*, *y1*) # *xs1*) − *1*) −*et*−*t*→
ec
⟹ ((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] ∈ *cpts-ev* Γ
and *b3*: *0* < *length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*)
and *b4*: $\Gamma \vdash$ ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) − *1*) −*et*−*t*→ *ec*
then show *?case*
proof(*cases xs1* = [])
assume *c0*: *xs1* = []
with *b4* **have** $\Gamma \vdash$ (*e2*, *t1*, *y1*)−*et*−*t*→ *ec* **by** *simp*
with *b2 c0* **have** ((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] ∈ *cpts-ev* Γ **by** *simp*
with *b0* **show** *?thesis* **using** *cpts-ev.CptsEvComp* **by** *fastforce*

next
assume *c0*: *xs1* $\neq$ []
with *b4* **have** $\Gamma \vdash$ *last xs1* $-et-t\rightarrow$ *ec* **by** (*simp add*: *last-conv-nth*)
with *b2 c0* **have** ((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] $\in$ *cpts-ev* Γ **using** *b4* **by** *auto*
then show *?thesis* **using** *b0 cpts-ev.CptsEvComp* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed

then show *el* @ [*ec*] $\in$ *cpts-ev* Γ **using** *p0 p1 p2* **by** *blast*
qed

lemma *cpts-ev-same*: $[\![$*length el* $>$ *0*; $\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*$]\!]$ $\Longrightarrow$ *el* $\in$ *cpts-ev* Γ
proof −
assume *p0*: *length el* $>$ *0*
and *p1*: $\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*
have $\forall$ *el es*. *length el* $>$ *0* $\wedge$ ($\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*) $\longrightarrow$ *el* $\in$ *cpts-ev* Γ
proof −
{
fix *el es*
assume *a0*: *length* (*el* :: ('*l*,'*k*,'*s*,'*prog*) *econfs*) $>$ *0*
and *a1*: $\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*
then have *el* $\in$ *cpts-ev* Γ
proof(*induct el*)
case *Nil* **show** *?case* **using** *Nil.prems*(*1*) **by** *auto*
next
case (*Cons a as*)
assume *b0*: *0* $<$ *length as* $\Longrightarrow$ $\forall$ *i*$<$*length as*. *getspc-e* (*as* ! *i*) = *es* $\Longrightarrow$ *as* $\in$ *cpts-ev* Γ
and *b1*: *0* $<$ *length* (*a* # *as*)
and *b2*: $\forall$ *i*$<$*length* (*a* # *as*). *getspc-e* ((*a* # *as*) ! *i*) = *es*
then show *?case*
proof(*cases as* = [])
assume *c0*: *as* = []
then show *?thesis* **by** (*metis cpts-ev.CptsEvOne old.prod.exhaust*)
next
assume *c0*: $\neg$(*as* = [])
then obtain *b* **and** *bs* **where** *c1*: *as* = *b* # *bs* **by** (*meson neq-Nil-conv*)

from *c0* **have** *0* $<$ *length as* **by** *simp*
with *b0* **have** $\forall$ *i*$<$*length as*. *getspc-e* (*as* ! *i*) = *es* $\Longrightarrow$ *as* $\in$ *cpts-ev* Γ **by** *simp*
with *b2* **have** *as* $\in$ *cpts-ev* Γ **by** *force*
moreover from *b2* **have** *getspc-e a* = *es* **by** *auto*

moreover from *b2 c1* **have** *getspc-e b* = *es* **by** *auto*
ultimately show *?thesis* **using** *c1 getspc-e-def* **by** (*metis cpts-ev.CptsEvEnv fst-conv prod-cases3*)
qed
qed
}
then show *?thesis* **by** *auto*
qed

then show *?thesis* **using** *p0 p1* **by** *auto*
qed

4.3.2 Event systems

lemma *cpts-es-not-empty* [*simp*]:[] ∉ *cpts-es* Γ
apply(*force elim*:*cpts-es.cases*)
done

lemma *esetran-eqconf*: Γ ⊢ (*es1*, *s1*, *x1*) −*ese*→ (*es2*, *s2*, *x2*) ⟹ *es1* = *es2*
apply(*rule esetran.cases*)
apply(*simp*)+
done

lemma *esetran-eqconf1*: Γ ⊢ *esc1* −*ese*→ *esc2* ⟹ *getspc-es esc1* = *getspc-es esc2*
proof −
assume *a0*: Γ ⊢ *esc1* −*ese*→ *esc2*
then obtain *es1* **and** *s1* **and** *x1* **and** *es2* **and** *s2* **and** *x2* **where** *a1*: *esc1* = (*es1*, *s1*, *x1*) **and** *a2*: *esc2* = (*es2*, *s2*, *x2*)
by (*meson prod-cases3*)
then have *es1* = *es2* **using** *a0 esetran-eqconf* **by** *fastforce*
with *a1* **show** *?thesis* **by** (*simp add*: *a2 getspc-es-def*)
qed

lemma *eqconf-esetran1*: *es1* = *es2* ⟹ Γ ⊢ (*es1*, *s1*, *x1*) −*ese*→ (*es2*, *s2*, *x2*)
by (*simp add*: *esetran.intros*)

lemma *eqconf-esetran*: *getspc-es esc1* = *getspc-es esc2* ⟹ Γ ⊢ *esc1* −*ese*→ *esc2*

proof −
assume *a0*: *getspc-es esc1* = *getspc-es esc2*

obtain *es1* **and** *s1* **and** *x1* **where** *a1*: *esc1* = (*es1*, *s1*, *x1*) **using** *prod-cases3* **by** *blast*
obtain *es2* **and** *s2* **and** *x2* **where** *a2*: *esc2* = (*es2*, *s2*, *x2*) **using** *prod-cases3* **by** *blast*
with *a0 a1* **have** *es1* = *es2* **by** (*simp add*:*getspc-es-def*)
with *a1 a2* **have** *a3*: Γ ⊢ (*es1*, *s1*, *x1*) −*ese*→ (*es2*, *s2*, *x2*) **by** (*simp ad-*

d:*eqconf-esetran1*)
  from *a3 a1 a2* **show** *?thesis* **by** *simp*
 qed

lemma *exist-estran*: $[\![(es1, s1, x1) \# (es, s, x) \# esl \in cpts\text{-}es\ \Gamma; es1 \neq es]\!] \Longrightarrow (\exists est.\ \Gamma \vdash (es1, s1, x1) -es-est\rightarrow (es, s, x))$
 apply(*rule cpts-es.cases*)
 apply(*simp*)+
 by *auto*

lemma *cpts-es-drop0*: $[\![el \in cpts\text{-}es\ \Gamma; Suc\ 0 < length\ el]\!] \Longrightarrow drop\ (Suc\ 0)\ el \in cpts\text{-}es\ \Gamma$
 apply(*rule cpts-es.cases*)
 apply(*simp*)+
 done

lemma *cpts-es-dropi*: $[\![el \in cpts\text{-}es\ \Gamma; Suc\ i < length\ el]\!] \Longrightarrow drop\ (Suc\ i)\ el \in cpts\text{-}es\ \Gamma$
 proof −
  assume *p0*:$el \in cpts\text{-}es\ \Gamma$ **and** *p1*:$Suc\ i < length\ el$
  have $\forall el\ i.\ el \in cpts\text{-}es\ \Gamma \wedge Suc\ i < length\ el \longrightarrow drop\ (Suc\ i)\ el \in cpts\text{-}es\ \Gamma$
   proof −
   {
    fix *el i*
    have $el \in cpts\text{-}es\ \Gamma \wedge Suc\ i < length\ el \longrightarrow drop\ (Suc\ i)\ el \in cpts\text{-}es\ \Gamma$
     proof(*induct i*)
      case *0* **show** *?case* **by** (*simp add*: *cpts-es-drop0*)
     next
      case (*Suc j*)
       assume *b0*: $el \in cpts\text{-}es\ \Gamma \wedge Suc\ j < length\ el \longrightarrow drop\ (Suc\ j)\ el \in cpts\text{-}es\ \Gamma$
      show *?case*
       proof
        assume *c0*: $el \in cpts\text{-}es\ \Gamma \wedge Suc\ (Suc\ j) < length\ el$
        with *b0* **have** *c1*: $drop\ (Suc\ j)\ el \in cpts\text{-}es\ \Gamma$
         by (*simp add*: *c0 Suc-lessD*)
        then show $drop\ (Suc\ (Suc\ j))\ el \in cpts\text{-}es\ \Gamma$
         using *c0 cpts-es-drop0* **by** *fastforce*
       qed
     qed
   }
   then show *?thesis* **by** *auto*
   qed
  with *p0 p1* **show** *?thesis* **by** *auto*
 qed

lemma *cpts-es-dropi2*: $[\![el \in cpts\text{-}es\ \Gamma; i < length\ el]\!] \Longrightarrow drop\ i\ el \in cpts\text{-}es\ \Gamma$
 using *cpts-es-dropi* **by** (*metis* (*no-types*, *hide-lams*) *drop-0 lessI less-Suc-eq-0-disj*)

```
lemma cpts-es-take0: ⟦el ∈ cpts-es Γ; i < length el; el1 = take (Suc i) el; j <
length el1⟧
                ⟹ drop (length el1 − Suc j) el1 ∈ cpts-es Γ
 proof −
  assume p0: el ∈ cpts-es Γ
   and  p1: i < length el
   and  p2: el1 = take (Suc i) el
   and  p3: j < length el1
  have ∀ i j. el ∈ cpts-es Γ ∧ i < length el ∧ el1 = take (Suc i) el ∧ j < length
el1
     ⟶ drop (length el1 − Suc j) el1 ∈ cpts-es Γ
   proof −
   {
    fix i j
    assume a0: el ∈ cpts-es Γ
     and  a1: i < length el
     and  a2: el1 = take (Suc i) el
     and  a3: j < length el1
    then have drop (length el1 − Suc j) el1 ∈ cpts-es Γ
     proof(induct j)
      case 0
      have drop (length el1 − Suc 0) el1 = [el ! i]
       by (simp add: a1 a2 take-Suc-conv-app-nth)
      then show ?case by (metis cpts-es.CptsEsOne old.prod.exhaust)
     next
      case (Suc jj)
      assume b0: el ∈ cpts-es Γ ⟹ i < length el ⟹ el1 = take (Suc i) el
              ⟹ jj < length el1 ⟹ drop (length el1 − Suc jj) el1 ∈ cpts-es
Γ
       and  b1: el ∈ cpts-es Γ
       and  b2: i < length el
       and  b3: el1 = take (Suc i) el
       and  b4: Suc jj < length el1
      then have b5: drop (length el1 − Suc jj) el1 ∈ cpts-es Γ
       using Suc-lessD by blast
      let ?el2 = drop (Suc i) el
      from a2 have b6: el1 @ ?el2 = el by simp
      let ?el1sht = drop (length el1 − Suc jj) el1
      let ?el1lng = drop (length el1 − Suc (Suc jj)) el1
      let ?elsht = drop (length el1 − Suc jj) el
      let ?ellng = drop (length el1 − Suc (Suc jj)) el
      from b6 have a7: ?el1sht @ ?el2 = ?elsht
       by (metis diff-is-0-eq diff-le-self drop-0 drop-append)
      from b6 have a8: ?el1lng @ ?el2 = ?ellng
         by (metis (no-types, lifting) a7 append-eq-append-conv diff-is-0-eq'
diff-le-self drop-append)
      have a9: ?ellng = (el ! (length el1 − Suc (Suc jj))) # ?elsht
```

by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc Suc-leI a8*
append-is-Nil-conv b4 diff-diff-cancel drop-all length-drop
list.size(*3*) *not-less old.nat.distinct*(*2*))
from *b1 b4* **have** *a10*: *?elsht* ∈ *cpts-es* Γ
by (*metis a7 append-is-Nil-conv b5 cpts-es-dropi2 drop-all not-less*)
from *b1 b4* **have** *a11*: *?ellng* ∈ *cpts-es* Γ
by (*metis a9 cpts-es-dropi2 drop-all list.simps*(*3*) *not-less*)
have *a12*: *?el1lng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?el1sht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc*
b4 b6 diff-less gr-implies-not0 length-0-conv length-greater-0-conv
nth-append zero-less-Suc)
from *a11* **have** *?el1lng* ∈ *cpts-es* Γ
proof(*induct ?ellng*)
case *CptsEsOne* **show** *?case*
using *CptsEsOne.hyps a7 a9* **by** *auto*
next
case (*CptsEsEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-es* Γ
and *c1*: (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
⟹
drop (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-es* Γ
and *c2*: (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* −
Suc (*Suc jj*)) *el*
from *c0* **have** (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-es* Γ
by (*simp add*: *a11 c2*)
have *c3*: *?el1sht* ! *0* = (*es1*, *t1*, *x1*) **by** (*metis* (*no-types*, *lifting*)
Suc-leI Suc-lessD a7
a9 append-eq-Cons-conv b4 c2 diff-diff-cancel length-drop
list.inject
list.size(*3*) *nth-Cons-0 old.nat.distinct*(*2*))
then have *c4*: ∃ *el1sht′*. *?el1sht* = (*es1*, *t1*, *x1*) # *el1sht′* **by** (*metis*
Cons-nth-drop-Suc b4
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c5*: *?el1lng* = (*es1*, *s1*, *y1*) # *?el1sht* **using** *a12 a9 c2* **by** *auto*

with *b5 c4* **show** *?case* **using** *cpts-es.CptsEsEnv* **by** *fastforce*
next
case (*CptsEsComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *c0*: Γ ⊢ (*es1*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-es* Γ
and *c2*: (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
⟹ *drop* (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-es* Γ
and *c3*: (*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* −
Suc (*Suc jj*)) *el*
have *c4*: *?el1sht* ! *0* = (*es2*, *t1*, *y1*) **by** (*metis* (*no-types*, *lifting*)
Suc-leI Suc-lessD a7
a9 append-eq-Cons-conv b4 c3 diff-diff-cancel length-drop
list.inject
list.size(*3*) *nth-Cons-0 old.nat.distinct*(*2*))

then have *c5*: $\exists$ *el1sht'*. *?el1sht* = (*es2*, *t1*, *y1*) # *el1sht'* **by** (*metis Cons-nth-drop-Suc b4*
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c6*: *?el1lng* = (*es1*, *s1*, *x1*) # *?el1sht* **using** *a12 a9 c3* **by** *auto*
with *b5 c5* **show** *?case* **using** *c0 cpts-es.CptsEsComp* **by** *fastforce*
qed

then show *?case* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed
then show *drop* (*length el1* − *Suc j*) *el1* ∈ *cpts-es* Γ
using *p0 p1 p2 p3* **by** *blast*
qed

lemma *cpts-es-take*: ⟦*el* ∈ *cpts-es* Γ; *i* < *length el*⟧ ⟹ *take* (*Suc i*) *el* ∈ *cpts-es* Γ
using *cpts-es-take0 gr-implies-not0* **by** *fastforce*

lemma *cpts-es-seg*: ⟦*el* ∈ *cpts-es* Γ; *m* ≤ *length el*; *n* ≤ *length el*; *m* < *n*⟧
⟹ *take* (*n* − *m*) (*drop m el*) ∈ *cpts-es* Γ
proof −
assume *p0*: *el* ∈ *cpts-es* Γ
and *p1*: *m* ≤ *length el*
and *p2*: *n* ≤ *length el*
and *p3*: *m* < *n*
then have *drop m el* ∈ *cpts-es* Γ
using *cpts-es-dropi* **by** (*metis* (*no-types*, *lifting*) *drop-0 le-0-eq le-SucE less-le-trans zero-induct*)
then show *?thesis* **using** *cpts-es-take*
by (*metis* (*no-types*, *lifting*) *cpts-es-dropi2 drop-take inc-induct*
leD le-SucE length-take min.absorb2 p0 p1 p2 p3)
qed

lemma *cpts-es-seg2*: ⟦*el* ∈ *cpts-es* Γ; *m* ≤ *length el*; *n* ≤ *length el*; *take* (*n* − *m*) (*drop m el*) ≠ []⟧
⟹ *take* (*n* − *m*) (*drop m el*) ∈ *cpts-es* Γ
proof −
assume *p0*: *el* ∈ *cpts-es* Γ
and *p1*: *m* ≤ *length el*
and *p2*: *n* ≤ *length el*
and *p3*: *take* (*n* − *m*) (*drop m el*) ≠ []
from *p3* **have** *m* < *n* **by** *simp*
then show *?thesis* **using** *cpts-es-seg* **using** *p0 p1 p2* **by** *blast*
qed

lemma *cpts-es-same*: ⟦*length el* > *0*; ∀ *i*. *i* < *length el* ⟶ *getspc-es* (*el*!*i*) = *es*⟧

$\Longrightarrow$ *el* $\in$ *cpts-es* Γ
proof $-$
assume *p0*: *length el* > 0
and *p1*: $\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-es* (*el*!*i*) $=$ *es*
have $\forall$ *el es*. *length el* $> 0 \wedge$ ($\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-es* (*el*!*i*) $=$ *es*) $\longrightarrow$ *el* $\in$ *cpts-es* Γ
proof $-$
{
fix *el es*
assume *a0*: *length* (*el* :: ('l,'k,'s,'prog) *esconf list*) > 0
and *a1*: $\forall$ *i*. *i* $<$ *length el* $\longrightarrow$ *getspc-es* (*el*!*i*) $=$ *es*
then have *el* $\in$ *cpts-es* Γ
proof(*induct el*)
case *Nil* **show** *?case* **using** *Nil.prems*(*1*) **by** *auto*
next
case (*Cons a as*)
assume *b0*: $0 <$ *length as* $\Longrightarrow \forall$ *i*<*length as*. *getspc-es* (*as* ! *i*) $=$ *es* $\Longrightarrow$ *as* $\in$ *cpts-es* Γ
and *b1*: $0 <$ *length* (*a* # *as*)
and *b2*: $\forall$ *i*<*length* (*a* # *as*). *getspc-es* ((*a* # *as*) ! *i*) $=$ *es*
then show *?case*
proof(*cases as* $=$ [])
assume *c0*: *as* $=$ []
then show *?thesis* **by** (*metis cpts-es.CptsEsOne old.prod.exhaust*)
next
assume *c0*: $\neg$(*as* $=$ [])
then obtain *b* **and** *bs* **where** *c1*: *as* $=$ *b* # *bs* **by** (*meson neq-Nil-conv*)

from *c0* **have** $0 <$ *length as* **by** *simp*
with *b0* **have** $\forall$ *i*<*length as*. *getspc-es* (*as* ! *i*) $=$ *es* $\Longrightarrow$ *as* $\in$ *cpts-es* Γ **by** *simp*
with *b2* **have** *as* $\in$ *cpts-es* Γ **by** *force*
moreover from *b2* **have** *getspc-es a* $=$ *es* **by** *auto*
moreover from *b2 c1* **have** *getspc-es b* $=$ *es* **by** *auto*
ultimately show *?thesis* **using** *c1 getspc-es-def* **by** (*metis cpts-es.CptsEsEnv fst-conv prod-cases3*)
qed
qed
}
then show *?thesis* **by** *auto*
qed

then show *?thesis* **using** *p0 p1* **by** *auto*
qed

lemma *noevtent-inmid-eq*:
($\neg$ ($\exists$ *j*. *j* $> 0 \wedge$ *Suc j* $<$ *length esl* $\wedge$ *getspc-es* (*esl* ! *j*) $=$ *EvtSys es* $\wedge$ *getspc-es* (*esl* ! *Suc j*) $\neq$ *EvtSys es*))

$= (\forall j.\ j > 0 \wedge Suc\ j < length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es \longrightarrow getspc\text{-}es\ (esl\ !\ Suc\ j) = EvtSys\ es)$
by *blast*

lemma *evtseq-next-in-cpts*:
$esl \in cpts\text{-}es\ \Gamma \Longrightarrow \forall i.\ Suc\ i < length\ esl \wedge getspc\text{-}es\ (esl!i) = EvtSeq\ e\ esys \longrightarrow getspc\text{-}es\ (esl!Suc\ i) = esys \vee (\exists e.\ getspc\text{-}es\ (esl!Suc\ i) = EvtSeq\ e\ esys)$
proof −
assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
then show *?thesis*
proof −
{
fix *i*
assume *a0*: $Suc\ i < length\ esl$
and *a1*: $getspc\text{-}es\ (esl!i) = EvtSeq\ e\ esys$
let $?esl1 = drop\ i\ esl$
from *p0 a0* **have** *a2*: $?esl1 \in cpts\text{-}es\ \Gamma$ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv less-or-eq-imp-le list.size*(*3*))
from *a0 a1* **have** $getspc\text{-}es\ (?esl1!0) = EvtSeq\ e\ esys$ **by** *auto*
then obtain *s1* **and** *x1* **where** *a3*: $?esl1!0 = (EvtSeq\ e\ esys, s1, x1)$
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** $getspc\text{-}es\ (?esl1!1) = esys \vee (\exists e.\ getspc\text{-}es\ (?esl1!1) = EvtSeq\ e\ esys)$
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0 le-add-diff-inverse2 length-Cons length-drop less-imp-le list.size*(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: $(es', s', y') \# (es', t', x') \# xs' = drop\ i\ esl$
and *b1*: $getspc\text{-}es\ (esl\ !\ i) = EvtSeq\ e\ esys$
then have $es' = EvtSeq\ e\ esys$ **using** *getspc-es-def* **by** (*metis a3 fst-conv nth-Cons-0*)
with *b0* **have** $getspc\text{-}es\ (drop\ i\ esl\ !\ 1) = EvtSeq\ e\ esys$ **using** *getspc-es-def* **by** (*metis One-nat-def fst-conv nth-Cons-0 nth-Cons-Suc*)
then show *?case* **by** *auto*
next
case (*CptsEsComp es1′ s′ x′ et′ es2′ t′ y′ xs′*)
assume *b0*: $\Gamma \vdash (es1', s', x') -es-et'\rightarrow (es2', t', y')$
and *b1*: $(es1', s', x') \# (es2', t', y') \# xs' = drop\ i\ esl$
and *b2*: $getspc\text{-}es\ (esl\ !\ i) = EvtSeq\ e\ esys$
then have *b3*: $es1' = EvtSeq\ e\ esys$
by (*metis Pair-inject a3 nth-Cons-0*)
from *b0 b3* **have** $es2' = esys \vee (\exists e.\ es2' = EvtSeq\ e\ esys)$
using *evtseq-tran-sys-or-seq* **by** *simp*

with *b1* **show** *?case* **using** *getspc-es-def*
by (*metis One-nat-def fst-conv nth-Cons-0 nth-Cons-Suc*)

qed

then have *getspc-es* (*esl*!*Suc i*) = *esys* ∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e esys*)
using *a0* **by** *fastforce*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *evtseq-next-in-cpts-anony*:
esl∈*cpts-es* Γ ⟹ ∀ *i*. *Suc i* < *length esl* ∧ *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
⟶ *getspc-es* (*esl*!*Suc i*) = *esys*
∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e esys* ∧ *is-anonyevt e*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *Suc i* < *length esl*
and *a1*: *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*∈*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **have** *getspc-es* (*?esl1*!*0*) = *EvtSeq e esys* **by** *auto*
then obtain *s1* **and** *x1* **where** *a3*: *?esl1*!*0* = (*EvtSeq e esys*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** *getspc-es* (*?esl1*!*1*) = *esys*
∨ (∃ *e*. *getspc-es* (*?esl1*!*1*) = *EvtSeq e esys* ∧ *is-anonyevt e*)
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0*
le-add-diff-inverse2 length-Cons length-drop less-imp-le
list.size(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *drop i esl*
and *b1*: *getspc-es* (*esl* ! *i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
then have *es′* = *EvtSeq e esys* **using** *getspc-es-def* **by** (*metis a3 fst-conv nth-Cons-0*)
with *b0* **have** *getspc-es* (*drop i esl* ! *1*) = *EvtSeq e esys* ∧ *is-anonyevt e*
using *getspc-es-def* **by** (*metis One-nat-def b1 fst-conv nth-Cons-0*

nth-Cons-Suc)
then show *?case* **by** *auto*
next
case (*CptsEsComp es1' s' x' et' es2' t' y' xs'*)
assume *b0*: $\Gamma \vdash$ (*es1'*, *s'*, *x'*) $-es-et' \rightarrow$ (*es2'*, *t'*, *y'*)
and *b1*: (*es1'*, *s'*, *x'*) # (*es2'*, *t'*, *y'*) # *xs'* = *drop i esl*
and *b2*: *getspc-es* (*esl* ! *i*) = *EvtSeq e esys* $\wedge$ *is-anonyevt e*
then have *b3*: *es1'* = *EvtSeq e esys*
by (*metis Pair-inject a3 nth-Cons-0*)
from *b0 b3* **have** *es2'* = *esys* $\vee$ ($\exists$ *e*. *es2'* = *EvtSeq e esys* $\wedge$ *is-anonyevt e*)
using *evtseq-tran-sys-or-seq-anony*
by *simp*
with *b1* **show** *?case* **using** *getspc-es-def*
by (*metis One-nat-def fst-conv nth-Cons-0 nth-Cons-Suc*)

qed

then have *getspc-es* (*esl*!*Suc i*) = *esys*
$\vee$ ($\exists$ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e esys* $\wedge$ *is-anonyevt e*)
using *a0* **by** *fastforce*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *evtsys-next-in-cpts*:
esl$\in$*cpts-es* $\Gamma \Longrightarrow \forall$ *i*. *Suc i* < *length esl* $\wedge$ *getspc-es* (*esl*!*i*) = *EvtSys es*
$\longrightarrow$ *getspc-es* (*esl*!*Suc i*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*))
proof −
assume *p0*: *esl*$\in$*cpts-es* Γ
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *Suc i* < *length esl*
and *a1*: *getspc-es* (*esl*!*i*) = *EvtSys es*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*$\in$*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **have** *getspc-es* (*?esl1*!*0*) = *EvtSys es* **by** *auto*
then obtain *s1* **and** *x1* **where** *a3*: *?esl1*!*0* = (*EvtSys es*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** *getspc-es* (*?esl1*!*1*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*?esl1*!*1*) = *EvtSeq e* (*EvtSys es*))
proof(*induct ?esl1*)

case (*CptsEsOne es' s' x'*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0 le-add-diff-inverse2 length-Cons length-drop less-imp-le list.size*(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es' t' x' xs' s' y'*)
assume *b0*: (*es'*, *s'*, *y'*) # (*es'*, *t'*, *x'*) # *xs'* = *drop i esl*
and *b1*: *getspc-es* (*esl* ! *i*) = *EvtSys es*
then have *es'* = *EvtSys es* **using** *getspc-es-def* **by** (*metis a3 fst-conv nth-Cons-0*)
with *b0* **have** *getspc-es* (*drop i esl* ! *1*) = *EvtSys es* **using** *getspc-es-def* **by** (*metis One-nat-def fst-conv nth-Cons-0 nth-Cons-Suc*)
then show *?case* **by** *simp*
next
case (*CptsEsComp es1' s' x' et' es2' t' y' xs'*)
assume *b0*: $\Gamma \vdash$ (*es1'*, *s'*, *x'*) −*es*−*et'*→ (*es2'*, *t'*, *y'*)
and *b1*: (*es1'*, *s'*, *x'*) # (*es2'*, *t'*, *y'*) # *xs'* = *drop i esl*
and *b2*: *getspc-es* (*esl* ! *i*) = *EvtSys es*
then have *b3*: *es1'* = *EvtSys es*
by (*metis Pair-inject a3 nth-Cons-0*)
from *b0 b3* **have** ∃ *e*. *es2'* = *EvtSeq e* (*EvtSys es*) **using** *evtsys-evtent* **by** *simp*
then obtain *e* **where** *es2'* = *EvtSeq e* (*EvtSys es*) **by** *auto*
with *b1* **have** ∃ *e*. *getspc-es* (*drop i esl* ! *1*) = *EvtSeq e* (*EvtSys es*)
using *getspc-es-def* **by** (*metis One-nat-def eq-fst-iff nth-Cons-0 nth-Cons-Suc*)
then show *?case* **by** *simp*
qed

then have *getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*))
using *a0* **by** *fastforce*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *evtsys-next-in-cpts-anony*:
esl∈*cpts-es* Γ ⟹ ∀ *i*. *Suc i* < *length esl* ∧ *getspc-es* (*esl*!*i*) = *EvtSys es*
⟶ *getspc-es* (*esl*!*Suc i*) = *EvtSys es*
∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *Suc i* < *length esl*

and *a1*: *getspc-es* (*esl*!*i*) = *EvtSys es*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*∈*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **have** *getspc-es* (*?esl1*!*0*) = *EvtSys es* **by** *auto*
then obtain *s1* **and** *x1* **where** *a3*: *?esl1*!*0* = (*EvtSys es*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** *getspc-es* (*?esl1*!*1*) = *EvtSys es*
∨ (∃ *e*. *getspc-es* (*?esl1*!*1*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0*
le-add-diff-inverse2 length-Cons length-drop less-imp-le
list.size(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *drop i esl*
and *b1*: *getspc-es* (*esl* ! *i*) = *EvtSys es*
then have *es′* = *EvtSys es* **using** *getspc-es-def* **by** (*metis a3 fst-conv nth-Cons-0*)
with *b0* **have** *getspc-es* (*drop i esl* ! *1*) = *EvtSys es* **using** *getspc-es-def*
by (*metis One-nat-def fst-conv nth-Cons-0 nth-Cons-Suc*)
then show *?case* **by** *simp*
next
case (*CptsEsComp es1′ s′ x′ et′ es2′ t′ y′ xs′*)
assume *b0*: Γ ⊢ (*es1′*, *s′*, *x′*) −*es*−*et′*→ (*es2′*, *t′*, *y′*)
and *b1*: (*es1′*, *s′*, *x′*) # (*es2′*, *t′*, *y′*) # *xs′* = *drop i esl*
and *b2*: *getspc-es* (*esl* ! *i*) = *EvtSys es*
then have *b3*: *es1′* = *EvtSys es*
by (*metis Pair-inject a3 nth-Cons-0*)
from *b0 b3* **have** ∃ *e*. *es2′* = *EvtSeq e* (*EvtSys es*) **using** *evtsys-evtent* **by** *simp*
then obtain *e* **where** *es2′* = *EvtSeq e* (*EvtSys es*) **by** *auto*
with *b0 b1 b3* **have** ∃ *e*. *getspc-es* (*drop i esl* ! *1*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*
using *getspc-es-def* **by** (*metis One-nat-def ent-spec2′ evtsysent-evtent0*

fst-conv is-anonyevt.simps(*1*) *noevtent-notran nth-Cons-0 nth-Cons-Suc*)

then show *?case* **by** *simp*
qed

then have *getspc-es* (*esl*!*Suc i*) = *EvtSys es*
∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
using *a0* **by** *fastforce*
}

then show *?thesis* **by** *auto*
qed
qed

lemma *evtsys-all-es-in-cpts*:
$[\![$*esl*$\in$*cpts-es* Γ; *length esl* $>$ *0*; *getspc-es* (*esl*!*0*) = *EvtSys es* $]\!] \Longrightarrow$
$\forall$ *i*. *i* $<$ *length esl* $\longrightarrow$ *getspc-es* (*esl*!*i*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*))
proof −
assume *p0*: *esl*$\in$*cpts-es* Γ
and *p1*: *length esl* $>$ *0*
and *p2*: *getspc-es* (*esl*!*0*) = *EvtSys es*
show *?thesis*
proof −
{
fix *i*
assume *a0*: *i* $<$ *length esl*
then have *getspc-es* (*esl*!*i*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*))
proof(*induct i*)
case *0* **from** *p2* **show** *?case* **by** *simp*
next
case (*Suc j*)
assume *b0*: *j* $<$ *length esl* $\Longrightarrow$
getspc-es (*esl* ! *j*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*))
and *b1*: *Suc j* $<$ *length esl*
then have *getspc-es* (*esl* ! *j*) = *EvtSys es* $\vee$ ($\exists$ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*))
by *simp*
then show *?case*
proof
assume *c0*: *getspc-es* (*esl* ! *j*) = *EvtSys es*
with *p0 b1* **show** *?thesis* **using** *evtsys-next-in-cpts* **by** *auto*
next
assume *c0*: $\exists$ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*)
with *p0 b1* **show** *?thesis* **using** *evtseq-next-in-cpts* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *evtsys-all-es-in-cpts-anony*:
$[\![$*esl*$\in$*cpts-es* Γ; *length esl* $>$ *0*; *getspc-es* (*esl*!*0*) = *EvtSys es* $]\!] \Longrightarrow$
$\forall$ *i*. *i* $<$ *length esl* $\longrightarrow$ *getspc-es* (*esl*!*i*) = *EvtSys es*
$\vee$ ($\exists$ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*) $\wedge$ *is-anonyevt e*)
proof −

assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *length esl* > *0*
and *p2*: *getspc-es* (*esl*!*0*) = *EvtSys es*
show *?thesis*
proof −
{
fix *i*
assume *a0*: *i* < *length esl*
then have *getspc-es* (*esl*!*i*) = *EvtSys es* ∨ (∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
proof(*induct i*)
case *0* **from** *p2* **show** *?case* **by** *simp*
next
case (*Suc j*)
assume *b0*: *j* < *length esl* ⟹
getspc-es (*esl* ! *j*) = *EvtSys es*
∨ (∃ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
and *b1*: *Suc j* < *length esl*
then have *getspc-es* (*esl* ! *j*) = *EvtSys es*
∨ (∃ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
by *simp*
then show *?case*
proof
assume *c0*: *getspc-es* (*esl* ! *j*) = *EvtSys es*
with *p0 b1* **show** *?thesis* **using** *evtsys-next-in-cpts-anony* **by** *auto*
next
assume *c0*: ∃ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*
with *p0 b1* **show** *?thesis* **using** *evtseq-next-in-cpts-anony* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *not-anonyevt-none-in-evtseq*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSeq e es*,*s1*,*x1*)#(*es*,*s2*,*x2*)#*xs* ⟧ ⟹ *e* ≠ *AnonyEvent fin-com*
apply(*rule cpts-es.cases*)
apply(*simp*)+
apply (*metis Suc-eq-plus1 add.commute add.right-neutral esys.size*(*3*) *le-add1 lessI not-le*)
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis Suc-eq-plus1 add.commute add.right-neutral esys.size*(*3*) *le-add1 lessI not-le*)
apply(*rule etran.cases*)

apply(*simp*)+
prefer *2*
apply(*simp*) **using** *ptran-not-none* **apply** *auto*[*1*]
done

lemma *not-anonyevt-none-in-evtseq1*:
$\llbracket$*esl*$\in$*cpts-es* Γ; *length esl* $>$ *1*; *getspc-es* (*esl*!*0*) = *EvtSeq e es*;
getspc-es (*esl*!*1*) = *es* $\rrbracket \Longrightarrow$ *e* $\neq$ *AnonyEvent fin-com*
using *getspc-es-def not-anonyevt-none-in-evtseq*
by (*metis* (*no-types*, *hide-lams*) *Cons-nth-drop-Suc drop-0 eq-fst-iff less-Suc-eq less-Suc-eq-0-disj less-one*)

lemma *fst-esys-snd-eseq-exist-evtent*:
$\llbracket$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*$\rrbracket$
$\Longrightarrow$
$\exists$ *t*. $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-t\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
apply(*rule cpts-es.cases*)
apply(*simp*)+
apply *blast*
by *blast*

lemma *fst-esys-snd-eseq-exist-evtent2*:
$\llbracket$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*$\rrbracket$
$\Longrightarrow$
$\exists$ *e k*. $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
apply(*rule cpts-es.cases*)
apply(*simp*)+
apply *blast*
by (*metis* (*no-types*, *hide-lams*) *cmd-enable-impl-notesys2 estran-impl-evtentorcmd*

evtent-is-basicevt fst-conv getspc-es-def nth-Cons-0 nth-Cons-Suc)

lemma *fst-esys-snd-eseq-exist*:
$\llbracket$*esl*$\in$*cpts-es* Γ; *length esl* $\geq$ *2* $\wedge$ *getspc-es* (*esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*1*) $\neq$ *EvtSys es*$\rrbracket$
$\Longrightarrow \exists$ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
proof $-$
assume *a0*: *length esl* $\geq$ *2* $\wedge$ *getspc-es* (*esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*1*) $\neq$ *EvtSys es*
and *c1*: *esl*$\in$*cpts-es* Γ
from *a0* **have** *b0*: *getspc-es* (*esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*1*) $\neq$ *EvtSys es*
by (*metis* (*no-types*, *lifting*))

from *a0* **have** *b1*: *2* $\leq$ *length esl* **by** *fastforce*
moreover from *b0 b1* **have** $\exists$ *s x*. *esl*!*0* = (*EvtSys es*, *s*, *x*) **using** *getspc-es-def*

by (*metis eq-fst-iff*)
moreover have $\exists$ *ev s1 x1. esl!1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **using** *getspc-es-def*
proof −
from *c1 a0 b0* **have** $\exists$ *ev. getspc-es* (*esl!1*) = *EvtSeq ev* (*EvtSys es*)
by (*metis One-nat-def Suc-1 Suc-le-lessD evtsys-next-in-cpts*)
then show *?thesis* **using** *getspc-es-def* **by** (*metis fst-conv surj-pair*)
qed
ultimately show *?thesis* **by** (*metis* (*no-types*, *hide-lams*) *One-nat-def Suc-1 Suc-n-not-le-n diff-is-0-eq hd-Cons-tl hd-conv-nth length-tl list.size*(*3*) *not-numeral-le-zero nth-Cons-Suc order-trans*)
qed

lemma *notevtent-cptses-isenvorcmd*:
⟦*esl*∈*cpts-es* Γ; *length esl* ≥ *2*; ¬ (∃ *e k*. Γ ⊢ *esl* ! *0* −*es*−*EvtEnt e*♯*k*→ *esl* ! *1*)⟧
⟹ Γ ⊢ *esl* ! *0* −*ese*→ *esl* ! *1* ∨ (∃ *c k*. Γ ⊢ *esl* ! *0* −*es*−*Cmd c*♯*k*→ *esl* ! *1*)
apply(*rule cpts-es.cases*)
apply *simp+*
apply (*simp add*: *esetran.intros*)
using *estran-impl-evtentorcmd2*
by (*metis One-nat-def nth-Cons-0 nth-Cons-Suc*)

lemma *only-envtran-to-basicevt*:
esl∈*cpts-es* Γ ⟹ ∀ *i*. *Suc i* < *length esl* ∧ (∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys*)
∧ *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*BasicEvent e*) *esys*
⟶ *getspc-es* (*esl*!*i*) = *EvtSeq* (*BasicEvent e*) *esys*
proof −
assume *p0*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *Suc i* < *length esl*
and *a1*: *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*BasicEvent e*) *esys*
and *a00*: ∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*∈*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv less-or-eq-imp-le list.size*(*3*))
from *a0 a1* **have** *getspc-es* (*?esl1*!*1*) = *EvtSeq* (*BasicEvent e*) *esys* **by** *auto*

then obtain *s1* **and** *x1* **where** *a3*: *?esl1*!*1* = (*EvtSeq* (*BasicEvent e*) *esys*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** *getspc-es* (*?esl1*!*0*) = *EvtSeq* (*BasicEvent e*) *esys*
proof(*induct ?esl1*)

case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0 le-add-diff-inverse2 length-Cons length-drop less-imp-le list.size*(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *drop i esl*
and *b1*: *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*BasicEvent e*) *esys*
then have *es′* = *EvtSeq* (*BasicEvent e*) *esys*
by (*metis One-nat-def a3 nth-Cons-0 nth-Cons-Suc prod.inject*)
with *b0* **show** *?case* **using** *getspc-es-def* **by** (*metis fst-conv nth-Cons-0*)

next
case (*CptsEsComp es1′ s′ x′ et′ es2′ t′ y′ xs′*)
assume *b0*: $\Gamma \vdash$ (*es1′*, *s′*, *x′*) −*es*−*et′*→ (*es2′*, *t′*, *y′*)
and *b1*: (*es1′*, *s′*, *x′*) # (*es2′*, *t′*, *y′*) # *xs′* = *drop i esl*
and *b2*: *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*BasicEvent e*) *esys*
then have *b3*: *es2′* = *EvtSeq* (*BasicEvent e*) *esys*
by (*metis One-nat-def Pair-inject a3 nth-Cons-0 nth-Cons-Suc*)
from *a00* **obtain** *e′* **where** *b4*: *getspc-es* (*esl* ! *i*) = *EvtSeq e′ esys* **by** *auto*
then have *es1′* = *EvtSeq e′ esys*
by (*metis* (*no-types*, *lifting*) *CptsEsComp.hyps*(*4*) *fst-conv getspc-es-def nth-via-drop*)
with *b0 b3* **have** ¬ (∃ *e*. *es2′* = *EvtSeq* (*BasicEvent e*) *esys*)
using *notrans-to-basicevt-insameesys*[*of* Γ *es1′ s′ x′ et′ es2′ t′ y′ esys*] **by** *auto*
with *b3* **show** *?case* **by** *blast*
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *incpts-es-impl-evnorcomptran*:
esl∈*cpts-es* Γ ⟹ ∀ *i*. *Suc i* < *length esl* ⟶ Γ ⊢ *esl* ! *i* −*ese*→ *esl* ! *Suc i* ∨ (∃ *et*. Γ ⊢ *esl* ! *i* −*es*−*et*→ *esl* ! *Suc i*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
{
fix *i*
assume *a0*: *Suc i* < *length esl*
let *?esl1* = *take 2* (*drop i esl*)
from *a0 p0* **have** *take* (*Suc* (*Suc i*) − *i*) (*drop i esl*) ∈ *cpts-es* Γ
using *cpts-es-seg*[*of esl* Γ *i Suc* (*Suc i*)] **by** *simp*
then have *?esl1* ∈ *cpts-es* Γ **by** *auto*
moreover
from *a0* **obtain** *esc1* **and** *s1* **and** *x1* **where** *a1*: *esl* ! *i* = (*esc1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*

moreover
from *a0* **obtain** *esc2* **and** *s2* **and** *x2* **where** *a2*: *esl* ! *Suc i* = (*esc2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
moreover
from *a0* **have** *esl* ! *i* = *?esl1* ! *0* **by** (*simp add*: *Cons-nth-drop-Suc Suc-lessD*)

moreover
from *a0* **have** *esl* ! *Suc i* = *?esl1* ! *1* **by** (*simp add*: *Cons-nth-drop-Suc Suc-lessD*)
ultimately have (*esc1*, *s1*, *x1*)#[(*esc2*, *s2*, *x2*)] ∈ *cpts-es* Γ
by (*metis Cons-nth-drop-Suc Suc-lessD a0 numeral-2-eq-2 take-0 take-Suc-Cons*)
then have Γ ⊢ (*esc1*, *s1*, *x1*) −*ese*→ (*esc2*, *s2*, *x2*) ∨ (∃ *et*. Γ ⊢ (*esc1*, *s1*, *x1*) −*es*−*et*→ (*esc2*, *s2*, *x2*))
apply(*rule cpts-es.cases*)
apply *simp+*
apply (*simp add*: *esetran.intros*)
by *auto*
with *a1 a2* **have** Γ ⊢ *esl* ! *i* −*ese*→ *esl* ! *Suc i* ∨ (∃ *et*. Γ ⊢ *esl* ! *i* −*es*−*et*→ *esl* ! *Suc i*) **by** *simp*
}
then show *?thesis* **by** *auto*
qed

lemma *incpts-es-eseq-not-evtent*:
⟦*esl*∈*cpts-es* Γ; *Suc i* < *length esl*; ∃ *e esys*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*⟧
⟹ ¬(∃ *e k*. *t* = *EvtEnt e* ∧ Γ ⊢ *esl*!*i* −*es*−*t*♯*k*→ *esl*!*Suc i*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *a0*: *Suc i* < *length esl*
and *a1*: ∃ *e esys*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*∈*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **obtain** *e* **and** *esys* **where** *a3*: *getspc-es* (*?esl1*!*0*) = *EvtSeq e esys* **by** *auto*
then obtain *s1* **and** *x1* **where** *a4*: *?esl1*!*0* = (*EvtSeq e esys*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a3* **have** ¬(∃ *e k*. *t* = *EvtEnt e* ∧ Γ ⊢ *?esl1*!*0* −*es*−*t*♯*k*→ *?esl1*!*1*)
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0*
le-add-diff-inverse2 length-Cons length-drop less-imp-le
list.size(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *?esl1*

and *b1*: *getspc-es* (*?esl1* ! *0*) = *EvtSeq e esys*
then have *es′* = *EvtSeq e esys*
by (*metis Pair-inject a4 nth-Cons-0*)
with *b0* **show** *?case* **using** *getspc-es-def*
by (*metis* (*mono-tags*, *lifting*) *a1 evtseq-no-evtent2 nth-Cons-0 nth-via-drop*)

next
case (*CptsEsComp es1′ s′ x′ et′ es2′ t′ y′ xs′*)
assume *b0*: $\Gamma \vdash$ (*es1′*, *s′*, *x′*) −*es*−*et′*→ (*es2′*, *t′*, *y′*)
and *b1*: (*es1′*, *s′*, *x′*) # (*es2′*, *t′*, *y′*) # *xs′* = *drop i esl*
and *b2*: *getspc-es* (*?esl1* ! *0*) = *EvtSeq e esys*
then have *b3*: *es1′* = *EvtSeq e esys*
by (*metis Pair-inject a4 nth-Cons-0*)
with *b0 b1* **show** *?case* **using** *getspc-es-def*
by (*metis* (*no-types*, *lifting*) *a1 evtseq-no-evtent2 nth-Cons-0 nth-via-drop*)

qed

with *a0* **show** *?thesis* **by** (*simp add*: *Cons-nth-drop-Suc Suc-lessD*)
qed

lemma *evtsys-not-eq-in-tran-aux*:$\Gamma \vdash (P,s,x)$ −*es*−*est*→ $(Q,t,y) \Longrightarrow P \neq Q$
apply(*erule estran.cases*)
apply (*simp add*: *evt-not-eq-in-tran-aux*)
apply (*simp add*: *evt-not-eq-in-tran-aux*)
by (*simp add*: *evtseq-ne-es*)

lemma *evtsys-not-eq-in-tran-aux1*:$\Gamma \vdash$ *esc1* −*es*−*est*→ *esc2* $\Longrightarrow$ *getspc-es esc1* $\neq$ *getspc-es esc2*
proof −
assume *p0*: $\Gamma \vdash$ *esc1* −*es*−*est*→ *esc2*
obtain *es1* **and** *s1* **and** *x1* **and** *es2* **and** *s2* **and** *x2* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*) $\wedge$ *esc2* = (*es2*,*s2*,*x2*)
by (*metis prod.collapse*)
with *p0* **have** *es1* $\neq$ *es2* **using** *evtsys-not-eq-in-tran-aux* **by** *simp*
with *a0* **show** *?thesis* **by** (*simp add*:*getspc-es-def*)
qed

lemma *evtsys-not-eq-in-tran* [*simp*]: $\neg\, \Gamma \vdash (P,s,x)$ −*es*−*est*→ (P,t,y)
apply *clarify*
apply(*drule evtsys-not-eq-in-tran-aux*)
apply *simp*
done

lemma *evtsys-not-eq-in-tran2* [*simp*]: $\neg(\exists$ *est*. $\Gamma \vdash (P,s,x)$ −*es*−*est*→ $(P,t,y))$ **by** *simp*

lemma *es-tran-not-etran2*: $\Gamma \vdash (P,s,x)$ −*es*−*pt*→ $(Q,t,y) \Longrightarrow \neg(\Gamma \vdash (P,s,x)$ −*ese*→$(Q,t,y))$

by (*metis esetran.cases evtsys-not-eq-in-tran-aux*)

lemma *es-tran-not-etran1*: $\Gamma \vdash esc1\ -es-pt\rightarrow\ esc2 \Longrightarrow \neg(\Gamma \vdash esc1\ -ese\rightarrow esc2)$
using *esetran-eqconf1 evtsys-not-eq-in-tran-aux1* **by** *blast*

4.3.3 Parallel event systems

lemma *cpts-pes-not-empty* [*simp*]:$[] \notin$ *cpts-pes* Γ
apply(*force elim*:*cpts-pes.cases*)
done

lemma *pesetran-eqconf*: $\Gamma \vdash (es1,\ s1,\ x1)\ -pese\rightarrow\ (es2,\ s2,\ x2) \Longrightarrow es1 = es2$
apply(*rule pesetran.cases*)
apply(*simp*)+
done

lemma *pesetran-eqconf1*: $\Gamma \vdash esc1\ -pese\rightarrow\ esc2 \Longrightarrow getspc\ esc1 = getspc\ esc2$
proof −
assume *a0*: $\Gamma \vdash esc1\ -pese\rightarrow\ esc2$
then obtain *es1* **and** *s1* **and** *x1* **and** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc1 = (es1,\ s1,\ x1)$ **and** *a2*: $esc2 = (es2,\ s2,\ x2)$
by (*meson prod-cases3*)
then have $es1 = es2$ **using** *a0 pesetran-eqconf* **by** *fastforce*
with *a1* **show** *?thesis* **by** (*simp add*: *a2 getspc-def*)
qed

lemma *eqconf-pesetran1*: $es1 = es2 \Longrightarrow \Gamma \vdash (es1,\ s1,\ x1)\ -pese\rightarrow\ (es2,\ s2,\ x2)$
by (*simp add*: *pesetran.intros*)

lemma *eqconf-pesetran*: $getspc\ esc1 = getspc\ esc2 \Longrightarrow \Gamma \vdash esc1\ -pese\rightarrow\ esc2$
proof −
assume *a0*: $getspc\ esc1 = getspc\ esc2$
obtain *es1* **and** *s1* **and** *x1* **where** *a1*: $esc1 = (es1,\ s1,\ x1)$ **using** *prod-cases3* **by** *blast*
obtain *es2* **and** *s2* **and** *x2* **where** *a2*: $esc2 = (es2,\ s2,\ x2)$ **using** *prod-cases3* **by** *blast*
with *a0 a1* **have** $es1 = es2$ **by** (*simp add*:*getspc-def*)
with *a1 a2* **have** *a3*: $\Gamma \vdash (es1,\ s1,\ x1)\ -pese\rightarrow\ (es2,\ s2,\ x2)$ **by** (*simp add*:*eqconf-pesetran1*)
from *a3 a1 a2* **show** *?thesis* **by** *simp*
qed

lemma *pestran-cpts-pes*: $[\![\Gamma \vdash C1\ -pes-ct\rightarrow\ C2;\ C2\#xs \in cpts\text{-}pes\ \Gamma]\!] \Longrightarrow C1\#C2\#xs \in cpts\text{-}pes\ \Gamma$
proof −
assume *p0*: $\Gamma \vdash C1\ -pes-ct\rightarrow\ C2$
and *p1*: $C2\#xs \in cpts\text{-}pes\ \Gamma$
moreover

obtain *pes1* **and** *s1* **and** *x1* **where** $C1 = (pes1, s1, x1)$
using *prod-cases3* **by** *blast*
moreover
obtain *pes2* **and** *s2* **and** *x2* **where** $C2 = (pes2, s2, x2)$
using *prod-cases3* **by** *blast*
ultimately show *?thesis* **by** (*simp add*: *cpts-pes.CptsPesComp*)
qed

lemma *cpts-pes-onemore*: $[\![el \in cpts\text{-}pes\ \Gamma;\ (\Gamma \vdash el\ !\ (length\ el - 1)\ -pes-t\rightarrow ec) \lor (\Gamma \vdash el\ !\ (length\ el - 1)\ -pese\rightarrow ec)]\!] \Longrightarrow$
$el\ @\ [ec] \in cpts\text{-}pes\ \Gamma$
proof −
assume *p0*: $el \in cpts\text{-}pes\ \Gamma$
and *p2*: $(\Gamma \vdash el\ !\ (length\ el - 1)\ -pes-t\rightarrow ec) \lor (\Gamma \vdash el\ !\ (length\ el - 1)\ -pese\rightarrow ec)$
from *p0* **have** *p1*: $el \neq []$
using *cpts-pes.simps* **by** *blast*
have $\forall el\ ec\ t.\ el \in cpts\text{-}pes\ \Gamma \land ((\Gamma \vdash el\ !\ (length\ el - 1)\ -pes-t\rightarrow ec) \lor (\Gamma \vdash el\ !\ (length\ el - 1)\ -pese\rightarrow ec))$
$\longrightarrow el\ @\ [ec] \in cpts\text{-}pes\ \Gamma$
proof −
{
fix *el ec t*
assume *a0*: $el \in cpts\text{-}pes\ \Gamma$
and *a2*: $(\Gamma \vdash el\ !\ (length\ el - 1)\ -pes-t\rightarrow ec) \lor (\Gamma \vdash el\ !\ (length\ el - 1)\ -pese\rightarrow ec)$
then have *a1*: $length\ el > 0$
using *cpts-pes.simps* **by** *blast*
from *a0 a1 a2* **have** $el\ @\ [ec] \in cpts\text{-}pes\ \Gamma$
proof(*induct el*)
case (*CptsPesOne e s x*)
assume *b0*: $(\Gamma \vdash [(e, s, x)]\ !\ (length\ [(e, s, x)] - 1)\ -pes-t\rightarrow ec)$
$\lor \Gamma \vdash [(e, s, x)]\ !\ (length\ [(e, s, x)] - 1)\ -pese\rightarrow ec$
then have $(\Gamma \vdash (e, s, x)\ -pes-t\rightarrow ec) \lor (\Gamma \vdash (e, s, x)\ -pese\rightarrow ec)$ **by** *simp*
then show *?case*
proof
assume $\Gamma \vdash (e, s, x)\ -pes-t\rightarrow ec$
then show *?thesis* **by** (*metis append-Cons append-Nil*
cpts-pes.CptsPesComp cpts-pes.CptsPesOne surj-pair)
next
assume $\Gamma \vdash (e, s, x)\ -pese\rightarrow ec$
then show *?thesis*
by (*metis append-Cons append-Nil cpts-pes.CptsPesEnv*
cpts-pes.CptsPesOne pesetranE surj-pair)
qed
next
case (*CptsPesEnv e s1 x xs s2 y*)
assume *b0*: $(e, s1, x)\ \#\ xs \in cpts\text{-}pes\ \Gamma$

and *b1*: *0* < *length* ((*e*, *s1*, *x*) # *xs*) ⟹
(Γ ⊢ ((*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s1*, *x*) # *xs*) − *1*) −*pes*−*t*→ *ec*) ∨
(Γ ⊢ ((*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s1*, *x*) # *xs*) − *1*) −*pese*→ *ec*) ⟹
((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ
and *b2*: *0* < *length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*)
and *b3*: (Γ ⊢ ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) − *1*) −*pes*−*t*→ *ec*) ∨
(Γ ⊢ ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) − *1*) −*pese*→ *ec*)
then show *?case*
proof(*cases xs* = [])
assume *c0*: *xs* = []
with *b3* **have** (Γ ⊢ (*e*, *s1*, *x*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ (*e*, *s1*, *x*) −*pese*→ *ec*) **by** *simp*
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ **by** *simp*
then show *?thesis* **by** (*simp add*: *cpts-pes.CptsPesEnv*)
next
assume *c0*: *xs* ≠ []
with *b3* **have** (Γ ⊢ *last xs* −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *last xs* −*pese*→ *ec*) **by** (*simp add*: *last-conv-nth*)
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ **using** *b3* **by** *auto*
then show *?thesis* **by** (*simp add*: *cpts-pes.CptsPesEnv*)
qed
next
case (*CptsPesComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *b0*: Γ ⊢ (*e1*, *s1*, *x1*) −*pes*−*et*→ (*e2*, *t1*, *y1*)
and *b1*: (*e2*, *t1*, *y1*) # *xs1* ∈ *cpts-pes* Γ
and *b2*: *0* < *length* ((*e2*, *t1*, *y1*) # *xs1*) ⟹
(Γ ⊢ ((*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e2*, *t1*, *y1*) # *xs1*) − *1*) −*pes*−*t*→ *ec*) ∨
(Γ ⊢ ((*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e2*, *t1*, *y1*) # *xs1*) − *1*) −*pese*→ *ec*) ⟹
((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] ∈ *cpts-pes* Γ
and *b3*: *0* < *length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*)
and *b4*: (Γ ⊢ ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) − *1*) −*pes*−*t*→ *ec*) ∨
Γ ⊢ ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! (*length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) − *1*) −*pese*→ *ec*
then show *?case*
proof(*cases xs1* = [])
assume *c0*: *xs1* = []
with *b4* **have** (Γ ⊢ (*e2*, *t1*, *y1*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ (*e2*, *t1*, *y1*) −*pese*→ *ec*) **by** *simp*
with *b2 c0* **have** ((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] ∈ *cpts-pes* Γ **by** *simp*
with *b0* **show** *?thesis* **using** *cpts-pes.CptsPesComp* **by** *fastforce*
next

assume *c0*: *xs1* $\neq$ []
with *b4* **have** ($\Gamma \vdash$ *last xs1* $-pes-t\rightarrow$ *ec*) $\vee$ ($\Gamma \vdash$ *last xs1* $-pese\rightarrow$ *ec*) **by** (*simp add*: *last-conv-nth*)
with *b2 c0* **have** ((*e2*, *t1*, *y1*) # *xs1*) @ [*ec*] $\in$ *cpts-pes* Γ **using** *b4* **by** *auto*
then show *?thesis* **using** *b0 cpts-pes.CptsPesComp* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *blast*
qed

then show *el* @ [*ec*] $\in$ *cpts-pes* Γ **using** *p0 p1 p2* **by** *blast*
qed

lemma *pes-not-eq-in-tran-aux*:$\Gamma \vdash (P,s,x)$ $-pes-est\rightarrow$ $(Q,t,y) \Longrightarrow P \neq Q$
apply(*erule pestran.cases*)
by (*metis Pair-inject evtsys-not-eq-in-tran fun-upd-same*)

lemma *pes-not-eq-in-tran* [*simp*]: $\neg$ $\Gamma \vdash (P,s,x)$ $-pes-est\rightarrow$ (P,t,y)
apply *clarify*
apply(*drule pes-not-eq-in-tran-aux*)
apply *simp*
done

lemma *pes-tran-not-etran1*: $\Gamma \vdash$ *pes1* $-pes-t\rightarrow$ *pes2* $\Longrightarrow \neg(\Gamma \vdash$ *pes1* $-pese\rightarrow$*pes2*)
by (*metis pes-not-eq-in-tran pesetranE surj-pair*)

lemma *pes-tran-not-etran2*: $\Gamma \vdash (P,s,x)$ $-pes-pt\rightarrow$ $(Q,t,y) \Longrightarrow \neg(\Gamma \vdash (P,s,x)$ $-pese\rightarrow(Q,t,y))$
by (*simp add*: *pes-tran-not-etran1*)

lemma *incpts-pes-impl-evnorcomptran*:
esl$\in$*cpts-pes* $\Gamma \Longrightarrow \forall i.$ *Suc i* < *length esl* $\longrightarrow \Gamma \vdash$ *esl* ! *i* $-pese\rightarrow$ *esl* ! *Suc i* $\vee$ ($\exists$ *et*. $\Gamma \vdash$ *esl* ! *i* $-pes-et\rightarrow$ *esl* ! *Suc i*)
proof –
assume *p0*: *esl*$\in$*cpts-pes* Γ
then show *?thesis*
proof(*induct esl*)
case (*CptsPesOne*) **show** *?case* **by** *simp*
next
case (*CptsPesEnv pes t x xs s y*)
assume *a0*: (*pes*, *t*, *x*) # *xs* $\in$ *cpts-pes* Γ
and *a1*: $\forall i.$ *Suc i* < *length* ((*pes*, *t*, *x*) # *xs*) $\longrightarrow$
$\Gamma \vdash$ ((*pes*, *t*, *x*) # *xs*) ! *i* $-pese\rightarrow$ ((*pes*, *t*, *x*) # *xs*) ! *Suc i* $\vee$
($\exists$ *et*. $\Gamma \vdash$ ((*pes*, *t*, *x*) # *xs*) ! *i* $-pes-et\rightarrow$ ((*pes*, *t*, *x*) # *xs*) ! *Suc i*)
then show *?case*
proof –

{
  fix *i*
  assume *b0*: *Suc i* < *length* ((*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*)
  have Γ ⊢ ((*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*) ! *i* −*pese*→ ((*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*) ! *Suc i* ∨
      (∃ *et*. Γ ⊢ ((*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*) ! *i* −*pes*−*et*→ ((*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*) ! *Suc i*)
    proof(*cases i* = *0*)
      assume *c0*: *i* = *0*
      then show *?thesis* **by** (*simp add*: *eqconf-pesetran1 nth-Cons′*)
    next
      assume *c0*: *i* ≠ *0*
      then have *i* > *0* **by** *auto*
      with *a1 b0* **show** *?thesis* **by** (*simp add*: *length-Cons*)
    qed
}
then show *?thesis* **by** *auto*
qed
next
  case (*CptsPesComp pes1 s x ct pes2 t y xs*)
  assume *a0*: Γ ⊢ (*pes1*, *s*, *x*) −*pes*−*ct*→ (*pes2*, *t*, *y*)
    and *a1*: (*pes2*, *t*, *y*) # *xs* ∈ *cpts-pes* Γ
    and *a2*: ∀ *i*. *Suc i* < *length* ((*pes2*, *t*, *y*) # *xs*) ⟶
        Γ ⊢ ((*pes2*, *t*, *y*) # *xs*) ! *i* −*pese*→ ((*pes2*, *t*, *y*) # *xs*) ! *Suc i* ∨
        (∃ *et*. Γ ⊢ ((*pes2*, *t*, *y*) # *xs*) ! *i* −*pes*−*et*→ ((*pes2*, *t*, *y*) # *xs*) ! *Suc i*)
  then show *?case*
    proof −
    {
      fix *i*
      assume *b0*: *Suc i* < *length* ((*pes1*, *s*, *x*) # (*pes2*, *t*, *y*) # *xs*)
      have Γ ⊢ ((*pes1*, *s*, *x*) # (*pes2*, *t*, *y*) # *xs*) ! *i* −*pese*→ ((*pes1*, *s*, *x*) # (*pes2*, *t*, *y*) # *xs*) ! *Suc i* ∨
        (∃ *et*. Γ ⊢ ((*pes1*, *s*, *x*) # (*pes2*, *t*, *y*) # *xs*) ! *i* −*pes*−*et*→ ((*pes1*, *s*, *x*) # (*pes2*, *t*, *y*) # *xs*) ! *Suc i*)
      proof(*cases i* = *0*)
        assume *c0*: *i* = *0*
        with *a0* **show** *?thesis* **using** *nth-Cons-0 nth-Cons-Suc* **by** *auto*
      next
        assume *c0*: *i* ≠ *0*
        then have *i* > *0* **by** *auto*
        with *a2 b0* **show** *?thesis* **using** *Suc-inject Suc-less-eq2 Suc-pred length-Cons nth-Cons-Suc* **by** *auto*
      qed
    }
    then show *?thesis* **by** *auto*
    qed
  qed
qed

```
lemma cpts-pes-drop0: ⟦el ∈ cpts-pes Γ; Suc 0 < length el⟧ ⟹ drop (Suc 0) el
∈ cpts-pes Γ
  apply(rule cpts-pes.cases)
  apply(simp)+
  done

lemma cpts-pes-dropi: ⟦el ∈ cpts-pes Γ; Suc i < length el⟧ ⟹ drop (Suc i) el ∈
cpts-pes Γ
  proof −
    assume p0:el ∈ cpts-pes Γ and p1:Suc i < length el
    have ∀ el i. el ∈ cpts-pes Γ ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-pes
Γ
      proof −
      {
        fix el i
        have el ∈ cpts-pes Γ ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-pes Γ
          proof(induct i)
            case 0 show ?case by (simp add: cpts-pes-drop0)
          next
            case (Suc j)
             assume b0: el ∈ cpts-pes Γ ∧ Suc j < length el ⟶ drop (Suc j) el ∈
cpts-pes Γ
            show ?case
              proof
                assume c0: el ∈ cpts-pes Γ ∧ Suc (Suc j) < length el
                with b0 have c1: drop (Suc j) el ∈ cpts-pes Γ
                  by (simp add: c0 Suc-lessD)
                then show drop (Suc (Suc j)) el ∈ cpts-pes Γ
                  using c0 cpts-pes-drop0 by fastforce
              qed
          qed
      }
      then show ?thesis by auto
      qed
    with p0 p1 show ?thesis by auto
  qed

lemma cpts-pes-take0: ⟦el ∈ cpts-pes Γ; i < length el; el1 = take (Suc i) el; j <
length el1⟧
                        ⟹ drop (length el1 − Suc j) el1 ∈ cpts-pes Γ
  proof −
    assume p0: el ∈ cpts-pes Γ
      and  p1: i < length el
      and  p2: el1 = take (Suc i) el
      and  p3: j < length el1
   have ∀ i j. el ∈ cpts-pes Γ ∧ i < length el ∧ el1 = take (Suc i) el ∧ j < length
el1
          ⟶ drop (length el1 − Suc j) el1 ∈ cpts-pes Γ
```

proof –
{
fix *i j*
assume *a0*: *el* ∈ *cpts-pes* Γ
and *a1*: *i* < *length el*
and *a2*: *el1* = *take* (*Suc i*) *el*
and *a3*: *j* < *length el1*
then have *drop* (*length el1* − *Suc j*) *el1* ∈ *cpts-pes* Γ
proof(*induct j*)
case *0*
have *drop* (*length el1* − *Suc 0*) *el1* = [*el* ! *i*]
by (*simp add*: *a1 a2 take-Suc-conv-app-nth*)
then show *?case* **by** (*metis cpts-pes.CptsPesOne old.prod.exhaust*)
next
case (*Suc jj*)
assume *b0*: *el* ∈ *cpts-pes* Γ ⟹ *i* < *length el* ⟹ *el1* = *take* (*Suc i*) *el*
⟹ *jj* < *length el1* ⟹ *drop* (*length el1* − *Suc jj*) *el1* ∈ *cpts-pes*
Γ
and *b1*: *el* ∈ *cpts-pes* Γ
and *b2*: *i* < *length el*
and *b3*: *el1* = *take* (*Suc i*) *el*
and *b4*: *Suc jj* < *length el1*
then have *b5*: *drop* (*length el1* − *Suc jj*) *el1* ∈ *cpts-pes* Γ
using *Suc-lessD* **by** *blast*
let *?el2* = *drop* (*Suc i*) *el*
from *a2* **have** *b6*: *el1* @ *?el2* = *el* **by** *simp*
let *?el1sht* = *drop* (*length el1* − *Suc jj*) *el1*
let *?el1lng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el1*
let *?elsht* = *drop* (*length el1* − *Suc jj*) *el*
let *?ellng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
from *b6* **have** *a7*: *?el1sht* @ *?el2* = *?elsht*
by (*metis diff-is-0-eq diff-le-self drop-0 drop-append*)
from *b6* **have** *a8*: *?el1lng* @ *?el2* = *?ellng*
by (*metis* (*no-types*, *lifting*) *a7 append-eq-append-conv diff-is-0-eq′*
diff-le-self drop-append)
have *a9*: *?ellng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?elsht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc Suc-leI a8*
append-is-Nil-conv b4 diff-diff-cancel drop-all length-drop
list.size(*3*) *not-less old.nat.distinct*(*2*))
from *b1 b4* **have** *a10*: *?elsht* ∈ *cpts-pes* Γ
by (*metis Suc-diff-Suc a7 append-is-Nil-conv b5 cpts-pes-dropi drop-all*
not-less)
from *b1 b4* **have** *a11*: *?ellng* ∈ *cpts-pes* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc a9 cpts-pes-dropi diff-is-0-eq*
drop-0 drop-all leI list.simps(*3*))
have *a12*: *?el1lng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?el1sht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc b4 b6*
diff-less
gr-implies-not0 length-0-conv length-greater-0-conv nth-append

zero-less-Suc)
from *a11* **have** *?el1lng* $\in$ *cpts-pes* Γ
proof(*induct ?ellng*)
case *CptsPesOne* **show** *?case*
using *CptsPesOne.hyps a7 a9* **by** *auto*
next
case (*CptsPesEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* $\in$ *cpts-pes* Γ
and *c1*: (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* $-$ *Suc* (*Suc jj*)) *el*
$\Longrightarrow$
drop (*length el1* $-$ *Suc* (*Suc jj*)) *el1* $\in$ *cpts-pes* Γ
and *c2*: (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* $-$
Suc (*Suc jj*)) *el*
from *c0* **have** (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* $\in$ *cpts-pes* Γ
by (*simp add*: *a11 c2*)
have *c3*: *?el1sht* ! *0* = (*es1*, *t1*, *x1*) **by** (*metis* (*no-types*, *lifting*)
Suc-leI Suc-lessD a7
a9 append-eq-Cons-conv b4 c2 diff-diff-cancel length-drop
list.inject
list.size(*3*) *nth-Cons-0 old.nat.distinct*(*2*))
then have *c4*: $\exists$ *el1sht'*. *?el1sht* = (*es1*, *t1*, *x1*) # *el1sht'* **by** (*metis*
Cons-nth-drop-Suc b4
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c5*: *?el1lng* = (*es1*, *s1*, *y1*) # *?el1sht* **using** *a12 a9 c2* **by** *auto*

with *b5 c4* **show** *?case* **using** *cpts-pes.CptsPesEnv* **by** *fastforce*
next
case (*CptsPesComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *c0*: $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-$*pes*$-$*et*$\rightarrow$ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* $\in$ *cpts-pes* Γ
and *c2*: (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* $-$ *Suc* (*Suc jj*)) *el*
$\Longrightarrow$ *drop* (*length el1* $-$ *Suc* (*Suc jj*)) *el1* $\in$ *cpts-pes* Γ
and *c3*: (*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* $-$
Suc (*Suc jj*)) *el*
have *c4*: *?el1sht* ! *0* = (*es2*, *t1*, *y1*) **by** (*metis* (*no-types*, *lifting*)
Suc-leI Suc-lessD a7
a9 append-eq-Cons-conv b4 c3 diff-diff-cancel length-drop
list.inject
list.size(*3*) *nth-Cons-0 old.nat.distinct*(*2*))
then have *c5*: $\exists$ *el1sht'*. *?el1sht* = (*es2*, *t1*, *y1*) # *el1sht'* **by** (*metis*
Cons-nth-drop-Suc b4
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c6*: *?el1lng* = (*es1*, *s1*, *x1*) # *?el1sht* **using** *a12 a9 c3* **by** *auto*
with *b5 c5* **show** *?case* **using** *c0 cpts-pes.CptsPesComp* **by** *fastforce*

qed

then show *?case* **by** *simp*
qed

}
then show *?thesis* by *auto*
qed
then show *drop* (*length el1* $-$ *Suc j*) *el1* $\in$ *cpts-pes* Γ
using *p0 p1 p2 p3* by *blast*
qed

lemma *cpts-pes-take*: $[\![$*el* $\in$ *cpts-pes* Γ; *i* $<$ *length el*$]\!] \Longrightarrow$ *take* (*Suc i*) *el* $\in$ *cpts-pes* Γ
using *cpts-pes-take0 gr-implies-not0* **by** *fastforce*

lemma *cpts-pes-seg*: $[\![$*el* $\in$ *cpts-pes* Γ; *m* $\leq$ *length el*; *n* $\leq$ *length el*; *m* $<$ *n*$]\!]$
$\Longrightarrow$ *take* (*n* $-$ *m*) (*drop m el*) $\in$ *cpts-pes* Γ
proof $-$
assume *p0*: *el* $\in$ *cpts-pes* Γ
and *p1*: *m* $\leq$ *length el*
and *p2*: *n* $\leq$ *length el*
and *p3*: *m* $<$ *n*
then have *drop m el* $\in$ *cpts-pes* Γ
using *cpts-pes-dropi* **by** (*metis* (*no-types*, *lifting*) *drop-0 le-0-eq le-SucE less-le-trans zero-induct*)
then show *?thesis* **using** *cpts-pes-take*
by (*smt Suc-diff-Suc diff-diff-cancel diff-less-Suc diff-right-commute length-drop less-le-trans p2 p3*)
qed

lemma *cpts-pes-seg2*: $[\![$*el* $\in$ *cpts-pes* Γ; *m* $\leq$ *length el*; *n* $\leq$ *length el*; *take* (*n* $-$ *m*) (*drop m el*) $\neq$ []$]\!]$
$\Longrightarrow$ *take* (*n* $-$ *m*) (*drop m el*) $\in$ *cpts-pes* Γ
proof $-$
assume *p0*: *el* $\in$ *cpts-pes* Γ
and *p1*: *m* $\leq$ *length el*
and *p2*: *n* $\leq$ *length el*
and *p3*: *take* (*n* $-$ *m*) (*drop m el*) $\neq$ []
from *p3* **have** *m* $<$ *n* **by** *simp*
then show *?thesis* **using** *cpts-pes-seg* **using** *p0 p1 p2* **by** *blast*
qed

4.4 Compositionality of the Semantics

4.4.1 Definition of the conjoin operator

definition *same-length* :: ('*l*,'*k*,'*s*,'*prog*) *pesconfs* $\Rightarrow$ ('*k* $\Rightarrow$ ('*l*,'*k*,'*s*,'*prog*) *esconfs*) $\Rightarrow$ *bool* **where**
same-length c cs $\equiv \forall$ *k*. *length* (*cs k*) $=$ *length c*

definition *same-state* :: ('*l*,'*k*,'*s*,'*prog*) *pesconfs* $\Rightarrow$ ('*k* $\Rightarrow$ ('*l*,'*k*,'*s*,'*prog*) *esconfs*) $\Rightarrow$ *bool* **where**
same-state c cs $\equiv \forall$ *k j*. *j* $<$ *length c* $\longrightarrow$ *gets* (*c*!*j*) $=$ *gets-es* ((*cs k*)!*j*) $\wedge$ *getx* (*c*!*j*) $=$ *getx-es* ((*cs k*)!*j*)

definition *same-spec* :: $('l,'k,'s,'prog)$ *pesconfs* $\Rightarrow$ $('k \Rightarrow ('l,'k,'s,'prog)$ *esconfs*$)$ $\Rightarrow$ *bool* **where**
same-spec c cs $\equiv \forall k\, j.\ j <$ *length c* $\longrightarrow$ $($*getspc* $(c!j))\ k =$ *getspc-es* $((cs\ k)\ !\ j)$

definition *compat-tran* :: *'Env* $\Rightarrow$ $('l,'k,'s,'prog)$ *pesconfs* $\Rightarrow$ $('k \Rightarrow ('l,'k,'s,'prog)$ *esconfs*$)$ $\Rightarrow$ *bool* **where**
compat-tran Γ *c cs* $\equiv \forall j.$ *Suc j* $<$ *length c* $\longrightarrow$
$((\exists t\ k.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow c!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow c!Suc\ j) \longrightarrow (\Gamma \vdash cs\ k!j$ $-es-(t\sharp k)\rightarrow cs\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ -ese\rightarrow cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ -pese\rightarrow (c!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!j)$ $-ese\rightarrow ((cs\ k)!\ Suc\ j))))$

definition *conjoin* :: *'Env* $\Rightarrow$ $('l,'k,'s,'prog)$ *pesconfs* $\Rightarrow$ $('k \Rightarrow ('l,'k,'s,'prog)$ *esconfs*$)$ $\Rightarrow$ *bool* (- - $\propto$ - [65,65] 64) **where**
Γ *c* $\propto$ *cs* $\equiv$ (*same-length c cs*) $\wedge$ (*same-state c cs*) $\wedge$ (*same-spec c cs*) $\wedge$ (*compat-tran* Γ *c cs*)

4.4.2 Lemmas of conjoin

lemma *acts-in-conjoin-cpts*: $\Gamma\ c \propto cs \Longrightarrow \forall i.\ Suc\ i < length\ (cs\ k) \longrightarrow \Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i)$
$\vee (\exists e.\ \Gamma \vdash ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow ((cs\ k)!\ Suc\ i))$
$\vee (\exists c.\ \Gamma \vdash ((cs\ k)!i)\ -es-(Cmd\ c\sharp k)\rightarrow ((cs\ k)!\ Suc\ i))$
proof −
assume *p0*: $\Gamma\ c \propto cs$
{
fix *i*
assume *a0*: *Suc i* $<$ *length* (*cs k*)
from *p0* **have** *a1*: *length c* $=$ *length* (*cs k*) **by** (*simp add:conjoin-def same-length-def*)
from *p0* **have** *compat-tran* Γ *c cs* **by** (*simp add:conjoin-def*)
with *a0 a1* **have** $(\exists t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$ $-es-(t\sharp k)\rightarrow cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow cs\ k'!\ Suc\ i))))$
$\vee$
$((\Gamma \vdash (c!i)\ -pese\rightarrow (c!Suc\ i)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow$ $((cs\ k)!\ Suc\ i))))$
by (*simp add: compat-tran-def*)
then have $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i)$
$\vee (\exists e.\ \Gamma \vdash ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow ((cs\ k)!\ Suc\ i))$
$\vee (\exists c.\ \Gamma \vdash ((cs\ k)!i)\ -es-(Cmd\ c\sharp k)\rightarrow ((cs\ k)!\ Suc\ i))$
proof
assume *b0*: $\exists t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$

$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc i*) $\wedge$

($\forall k'$. $k' \neq k \longrightarrow$ ($\Gamma \vdash$ *cs k'*!*i* $-ese\rightarrow$ *cs k'*! *Suc i*)))

then obtain *t* **and** *k1* **where** *b1*: ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k1)\rightarrow$ *c*!*Suc i*) $\wedge$

($\forall k$ *t*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*i*
$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc i*) $\wedge$

($\forall k'$. $k' \neq k \longrightarrow$ ($\Gamma \vdash$ *cs k'*!*i* $-ese\rightarrow$ *cs k'*! *Suc i*))) **by**
auto

then show *?thesis*

proof(*cases k = k1*)

assume *c0*: *k* = *k1*

with *b1* **show** *?thesis* **by** (*meson estran-impl-evtentorcmd2'*)

next

assume *c0*: $k \neq$ *k1*

with *b1* **show** *?thesis* **by** *auto*

qed

next

assume *b0*: ($\Gamma \vdash$ (*c*!*i*) $-pese\rightarrow$ (*c*!*Suc i*)) $\wedge$ ($\forall k$. ($\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$
((*cs k*)! *Suc i*)))

then show *?thesis* **by** *simp*

qed

}

then show *?thesis* **by** *simp*

qed

lemma *entevt-in-conjoin-cpts*:

$[\![$$\Gamma$ *c* $\propto$ *cs*; *Suc i* < *length* (*cs k*); *getspc-es* ((*cs k*)!*i*) = *EvtSys es*;

getspc-es ((*cs k*)!*Suc i*) $\neq$ *EvtSys es* $]\!]$

$\Longrightarrow$ ($\exists e$. $\Gamma \vdash$ ((*cs k*)!*i*) $-es-(EvtEnt\ e\sharp k)\rightarrow$ ((*cs k*)! *Suc i*))

proof −

assume *p0*: Γ *c* $\propto$ *cs*

and *p1*: *Suc i* < *length* (*cs k*)

and *p2*: *getspc-es* ((*cs k*)!*i*) = *EvtSys es*

and *p3*: *getspc-es* ((*cs k*)!*Suc i*) $\neq$ *EvtSys es*

then have $\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*)

$\vee$ ($\exists e$. $\Gamma \vdash$ ((*cs k*)!*i*) $-es-(EvtEnt\ e\sharp k)\rightarrow$ ((*cs k*)! *Suc i*))

$\vee$ ($\exists c$. $\Gamma \vdash$ ((*cs k*)!*i*) $-es-(Cmd\ c\sharp k)\rightarrow$ ((*cs k*)! *Suc i*))

using *acts-in-conjoin-cpts* **by** *fastforce*

then show *?thesis*

proof

assume $\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*)

with *p2 p3* **show** *?thesis* **by** (*simp add*: *esetran-eqconf1*)

next

assume ($\exists e$. $\Gamma \vdash$ *cs k* ! *i* $-es-EvtEnt\ e\sharp k\rightarrow$ *cs k* ! *Suc i*)

$\vee$ ($\exists c$. $\Gamma \vdash$ *cs k* ! *i* $-es-Cmd\ c\sharp k\rightarrow$ *cs k* ! *Suc i*)

then show *?thesis*

proof

assume $\exists e$. $\Gamma \vdash$ *cs k* ! *i* $-es-EvtEnt\ e\sharp k\rightarrow$ *cs k* ! *Suc i*

then show *?thesis* **by** *simp*

next

assume $\exists c.\ \Gamma \vdash$ *cs k* ! *i* $-es-$*Cmd c*$\sharp$*k*$\rightarrow$ *cs k* ! *Suc i*
with *p2 p3* **show** *?thesis*
by (*meson cmd-enable-impl-anonyevt2 esys-not-eseq*)
qed
qed
qed

lemma *notentevt-in-conjoin-cpts*:
$[\![\Gamma\ c \propto cs;\ Suc\ i < length\ (cs\ k);\ \neg(getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es \wedge getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es);$
$\forall i < length\ (cs\ k).\ getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSys\ es$
$\vee (\exists e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))]\!]$
$\Longrightarrow \neg(\exists e.\ \Gamma \vdash ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow ((cs\ k)!\ Suc\ i))$
proof –
assume *p0*: $\Gamma\ c \propto cs$
and *p1*: *Suc i* $<$ *length* (*cs k*)
and *p2*: $\neg(getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es \wedge getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es)$
and *p3*: $\forall i < length\ (cs\ k).\ getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSys\ es$
$\vee (\exists e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))$
from *p2* **have** $getspc\text{-}es\ ((cs\ k)!i) \neq EvtSys\ es \vee getspc\text{-}es\ ((cs\ k)!Suc\ i) = EvtSys\ es$ **by** *simp*
with *p3* **have** $(\exists e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))$
$\vee\ getspc\text{-}es\ ((cs\ k)!Suc\ i) = EvtSys\ es$
using *Suc-lessD p1* **by** *blast*
then show *?thesis*
proof
assume $\exists e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es)$
then obtain *e1* **where** $is\text{-}anonyevt\ e1 \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e1\ (EvtSys\ es)$ **by** *auto*
then show *?thesis* **using** *evtent-is-basicevt-inevtseq2* **by** *fastforce*
next
assume $getspc\text{-}es\ ((cs\ k)!Suc\ i) = EvtSys\ es$
then show *?thesis* **by** (*metis Suc-lessD evtseq-no-evtent2 evtsys-not-eq-in-tran-aux1 p1 p3*)
qed
qed

lemma *take-n-conjoin*: $[\![\Gamma\ c \propto cs;\ n \le length\ c;\ c1 = take\ n\ c;\ cs1 = (\lambda k.\ take\ n\ (cs\ k))]\!]$
$\Longrightarrow \Gamma\ c1 \propto cs1$
proof –
assume *p0*: $\Gamma\ c \propto cs$
and *p1*: $n \le length\ c$
and *p2*: $c1 = take\ n\ c$
and *p3*: $cs1 = (\lambda k.\ take\ n\ (cs\ k))$

have *a0*: *same-length c1 cs1* **by** (*metis conjoin-def length-take p0 p2 p3 same-length-def*)
then have *a1*: $\forall$ *k*. *length* (*cs1 k*) = *length c1* **by** (*simp add:same-length-def*)

have *same-state c1 cs1*
proof −
{
fix *k j*
assume *b0*: *j* < *length c1*
from *p1 p3 a1* **have** *b1*: *cs1 k* = *take n* (*cs k*) **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall$ *k j*. *j* < *length c*
$\longrightarrow$ *gets* (*c*!*j*) = *gets-es* ((*cs k*)!*j*) $\wedge$ *getx* (*c*!*j*) = *getx-es* ((*cs k*)!*j*)
by (*simp add:conjoin-def same-state-def*)
from *p2 b1 b0* **have** *gets* (*c* ! *j*) = *gets* (*c1* ! *j*) $\wedge$ *gets-es* ((*cs k*)!*j*) = *gets-es* ((*cs1 k*)!*j*)
$\wedge$ *getx* (*c*!*j*) = *getx* (*c1*!*j*)
by (*simp add*: *nth-append*)
with *p1 p2 b1 b2*[*of j k*] *b0* **have** *gets* (*c1*!*j*) = *gets-es* ((*cs1 k*)!*j*) $\wedge$ *getx* (*c1*!*j*) = *getx-es* ((*cs1 k*)!*j*)
by *simp*
}
then show *?thesis* **by** (*simp add:same-state-def*)
qed
moreover
have *same-spec c1 cs1*
proof −
{
fix *k j*
assume *b0*: *j* < *length c1*
from *p1 p3 a1* **have** *b1*: *cs1 k* = *take n* (*cs k*) **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall$ *k j*. *j* < *length c*
$\longrightarrow$ (*getspc* (*c*!*j*)) *k* = *getspc-es* ((*cs k*) ! *j*)
by (*simp add:conjoin-def same-spec-def*)
from *p2 b1 b0* **have** *getspc* (*c1*!*j*) = *getspc* (*c*!*j*)
$\wedge$ *getspc-es* ((*cs k*) ! *j*) = *getspc-es* ((*cs1 k*) ! *j*)
by (*simp add*: *nth-append*)
then have (*getspc* (*c1*!*j*)) *k* = *getspc-es* ((*cs1 k*) ! *j*)
using *b0 b2 p2* **by** *auto*
}
then show *?thesis* **by** (*simp add:same-spec-def*)
qed
moreover
have *compat-tran* Γ *c1 cs1*
proof −
{
fix *j*
assume *b0*: *Suc j* < *length c1*
with *p0 p2* **have** (($\exists$ *t k*. ($\Gamma \vdash$ *c*!*j* −*pes*−(*t*$\sharp$*k*)$\rightarrow$ *c*!*Suc j*)) $\wedge$
($\forall$ *k t*. ($\Gamma \vdash$ *c*!*j* −*pes*−(*t*$\sharp$*k*)$\rightarrow$ *c*!*Suc j*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*j*

$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc j*) $\wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ -ese\rightarrow\ cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ -pese\rightarrow\ (c!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!j)\ -ese\rightarrow ((cs\ k)!\ Suc\ j))))$
by (*simp add:conjoin-def compat-tran-def*)
moreover
from *p2 b0* **have** $c!j = c1!j$ **by** *simp*
moreover
from *p2 b0* **have** $c!Suc\ j = c1!Suc\ j$ **by** *simp*
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!j = cs\ k!j$
by (*simp add: Suc-lessD*)
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!Suc\ j = cs\ k!Suc\ j$
by (*simp add: Suc-lessD*)
ultimately
have $((\exists t\ k.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j) \longrightarrow (\Gamma \vdash cs1\ k!j\ -es-(t\sharp k)\rightarrow\ cs1\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs1\ k'!j\ -ese\rightarrow\ cs1\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c1!j)\ -pese\rightarrow\ (c1!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs1\ k)!j)\ -ese\rightarrow ((cs1\ k)!\ Suc\ j))))$ **by** *simp*
}
then show *?thesis* **by** (*simp add:compat-tran-def*)
qed
ultimately show *?thesis* **by** (*simp add:conjoin-def a0*)
qed

lemma *drop-n-conjoin*: $[\![\Gamma\ c \propto cs;\ n \leq length\ c;\ c1 = drop\ n\ c;\ cs1 = (\lambda k.\ drop\ n\ (cs\ k))]\!]$
$\Longrightarrow \Gamma\ c1 \propto cs1$
proof −
assume *p0*: $\Gamma\ c \propto cs$
and *p1*: $n \leq length\ c$
and *p2*: $c1 = drop\ n\ c$
and *p3*: $cs1 = (\lambda k.\ drop\ n\ (cs\ k))$
have *a0*: *same-length c1 cs1* **by** (*metis conjoin-def length-drop p0 p2 p3 same-length-def*)
then have *a1*: $\forall k.\ length\ (cs1\ k) = length\ c1$ **by** (*simp add:same-length-def*)

have *same-state c1 cs1*
proof −
{
fix *k j*
assume *b0*: $j < length\ c1$
from *p1 p3 a1* **have** *b1*: $cs1\ k = drop\ n\ (cs\ k)$ **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < length\ c$

```
          ⟶ gets (c!j) = gets-es ((cs k)!j) ∧ getx (c!j) = getx-es ((cs k)!j)
        by (simp add:conjoin-def same-state-def)
      from p2 b1 b0 have gets (c ! (n + j)) = gets (c1 ! j) ∧ gets-es ((cs k)!(n
+ j)) = gets-es ((cs1 k)!j)
        ∧ getx (c!(n + j)) = getx (c1!j)
        proof −
          have f1: n + j ≤ length c
            using b0 p2 by auto
          then have n + j ≤ length (cs k)
            by (metis (no-types) conjoin-def p0 same-length-def)
          then show ?thesis
            using f1 by (simp add: b1 p2)
        qed

       with p1 p2 b1 b2[of n + j k] b0 have gets (c1!j) = gets-es ((cs1 k)!j) ∧
getx (c1!j) = getx-es ((cs1 k)!j)
          by (smt a1 add-diff-cancel-left′ drop-eq-Nil length-drop less-diff-conv2
list.size(3)
        nat-le-linear nat-neq-iff not-less-zero nth-drop order.asym semiring-normalization-rules(24))

    }
    then show ?thesis by (simp add:same-state-def)
    qed
  moreover
  have same-spec c1 cs1
    proof −
    {
      fix k j
      assume b0: j < length c1
      from p1 p3 a1 have b1: cs1 k = drop n (cs k) by simp
      from p0 have b2[rule-format]: ∀ k j. j < length c
          ⟶ (getspc (c!j)) k = getspc-es ((cs k) ! j)
        by (simp add:conjoin-def same-spec-def)
      from p2 b1 b0 have getspc (c1!j) = getspc (c!(n+j))
        ∧ getspc-es ((cs k) ! (n+j)) = getspc-es ((cs1 k) ! j)
        proof −
          have f1: n + j ≤ length c
            using b0 p2 by auto
          then have n + j ≤ length (cs k)
            by (metis (no-types) conjoin-def p0 same-length-def)
          then show ?thesis
            using f1 by (simp add: b1 p2)
        qed
      then have (getspc (c1!j)) k = getspc-es ((cs1 k) ! j)
        using b0 b2 p2 by auto
    }
    then show ?thesis by (simp add:same-spec-def)
    qed
  moreover
```

have *compat-tran* Γ *c1 cs1*
proof −
{
fix *j*
assume *b0*: *Suc j* < *length c1*
with *p0 p2* **have** $((\exists\, t\ k.\ (\Gamma \vdash c!(n+j)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (n+j))) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(n+j)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (n+j)) \longrightarrow (\Gamma \vdash cs\ k!(n+j)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (n+j)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(n+j)\ -ese\rightarrow\ cs\ k'!\ Suc\ (n+j)))))$
$\vee$
$((\Gamma \vdash (c!(n+j))\ -pese\rightarrow\ (c!Suc\ (n+j))) \wedge (\forall\, k.\ (\Gamma \vdash ((cs\ k)!(n+j))\ -ese\rightarrow\ ((cs\ k)!\ Suc\ (n+j)))))$
by (*simp add*:*conjoin-def compat-tran-def*)
moreover
from *p2 b0* **have** $c!(n+j) = c1!j$ **by** *simp*
moreover
from *p2 b0* **have** $c!Suc\ (n+j) = c1!Suc\ j$ **by** *simp*
moreover
from *p1 p2 p3 a1 b0* **have** $\forall\, k.\ cs1\ k!j = cs\ k!(n+j)$
by (*metis drop-eq-Nil length-greater-0-conv less-imp-Suc-add linear nth-drop zero-less-Suc*)
moreover
from *p1 p2 p3 a1 b0* **have** $\forall\, k.\ cs1\ k!Suc\ j = cs\ k!Suc\ (n+j)$
by (*smt Suc-lessE add-Suc-right drop-eq-Nil length-greater-0-conv linear nth-drop zero-less-Suc*)
ultimately
have $((\exists\, t\ k.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j) \longrightarrow (\Gamma \vdash cs1\ k!j\ -es-(t\sharp k)\rightarrow\ cs1\ k!\ Suc\ j) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs1\ k'!j\ -ese\rightarrow\ cs1\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c1!j)\ -pese\rightarrow\ (c1!Suc\ j)) \wedge (\forall\, k.\ (\Gamma \vdash ((cs1\ k)!j)\ -ese\rightarrow\ ((cs1\ k)!\ Suc\ j))))$ **by** *simp*
}
then show *?thesis* **by** (*simp add*:*compat-tran-def*)
qed
ultimately show *?thesis* **by** (*simp add*:*conjoin-def a0*)
qed

lemma *conjoin-imp-cptses-k-help*: $[\![c \in cpts\text{-}pes\ \Gamma]\!] \Longrightarrow$
$\forall\, cs\ k.\ \Gamma\ c \propto cs \longrightarrow (cs\ k \in cpts\text{-}es\ \Gamma)$
proof −
assume *p0*: $c \in cpts\text{-}pes\ \Gamma$
{
fix *k*
from *p0* **have** $\forall\, cs.\ c \in cpts\text{-}pes\ \Gamma \wedge \Gamma\ c \propto cs \longrightarrow (cs\ k \in cpts\text{-}es\ \Gamma)$
proof(*induct c*)
case (*CptsPesOne pes s x*)

{
fix *cs*
assume *a0*: Γ [(*pes*, *s*, *x*)] $\propto$ *cs*
then have *p3*:*length* (*cs k*) = *1* **by** (*simp add*:*conjoin-def same-length-def*)
from *a0* **have** *p5*: *same-spec* [(*pes*, *s*, *x*)] *cs* $\wedge$ *same-state* [(*pes*, *s*, *x*)] *cs* **by** (*simp add*:*conjoin-def*)
with *a0 p3* **have** *cs k* ! *0* = (*pes k*, *s*, *x*)
using *esconf-trip pesconf-trip same-spec-def same-state-def*
by (*metis One-nat-def length-Cons list.size*(*3*) *nth-Cons-0 prod.sel*(*1*) *prod.sel*(*2*) *zero-less-one*)
with *p3* **have** *cs k* $\in$ *cpts-es* Γ
by (*metis* (*no-types*, *hide-lams*) *One-nat-def Suc-less-eq cpts-es.CptsEsOne length-0-conv length-Cons neq0-conv neq-Nil-conv prod-cases3*)
}
then show *?case* **by** *auto*
next
case (*CptsPesEnv pes t x xs s y*)
assume *a0*: (*pes*, *t*, *x*) # *xs* $\in$ *cpts-pes* Γ
and *a1*[*rule-format*]: $\forall$ *cs*. (*pes*, *t*, *x*) # *xs* $\in$ *cpts-pes* Γ $\wedge$ Γ (*pes*, *t*, *x*) # *xs* $\propto$ *cs* $\longrightarrow$ *cs k* $\in$ *cpts-es* Γ
{
fix *cs*
assume *b0*: (*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs* $\in$ *cpts-pes* Γ
and *b1*: Γ (*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs* $\propto$ *cs*
let *?esl* = (*pes*, *t*, *x*) # *xs*
let *?esllon* = (*pes*, *s*, *y*) # (*pes*, *t*, *x*) # *xs*
let *?cs* = (λ*k*. *drop 1* (*cs k*))
from *b1* **have** Γ *?esl* $\propto$ *?cs* **using** *drop-n-conjoin*[*of* Γ *?esllon cs 1 ?esl ?cs*] **by** *auto*
with *a0 a1*[*of ?cs*] **have** *b2*: *?cs k* $\in$ *cpts-es* Γ **by** *simp*
from *b1* **have** *b3*: *cs k* ! *0* = (*pes k*, *s*, *y*)
using *conjoin-def*[*of* Γ *?esllon cs*] *same-state-def*[*of ?esllon cs*] *same-spec-def*[*of ?esllon cs*]
by (*metis esconf-trip gets-def getspc-def getx-def length-greater-0-conv list.simps*(*3*) *nth-Cons-0 prod.sel*(*1*) *prod.sel*(*2*))

from *b1* **have** *getspc-es* (*cs k* ! *1*) = (*getspc* (*?esllon* ! *1*)) *k*
using *conjoin-def*[*of* Γ *?esllon cs*] *same-spec-def*[*of ?esllon cs*]
by (*metis diff-Suc-1 length-Cons zero-less-Suc zero-less-diff*)
moreover
from *b1* **have** *gets* (*?esllon* ! *1*) = *gets-es* ((*cs k*)!*1*) $\wedge$ *getx* (*?esllon* ! *1*) = *getx-es* ((*cs k*)!*1*)
using *conjoin-def*[*of* Γ *?esllon cs*] *same-state-def*[*of ?esllon cs*] *diff-Suc-1 length-Cons zero-less-Suc zero-less-diff* **by** *fastforce*
ultimately have *cs k* ! *1* = (*pes k*, *t*, *x*)
using *b0 getspc-def gets-def getx-def*
by (*metis One-nat-def esconf-trip fst-conv nth-Cons-0 nth-Cons-Suc snd-conv*)

with *b2 b3* **have** *cs k* ∈ *cpts-es* Γ **using** *CptsEsEnv*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD cpts-es-not-empty drop-0 drop-eq-Nil not-le*)
}
then show *?case* **by** *auto*
next
case (*CptsPesComp pes1 s y ct pes2 t x xs*)
assume *a0*: Γ ⊢ (*pes1*, *s*, *y*) −*pes*−*ct*→ (*pes2*, *t*, *x*)
and *a1*: (*pes2*, *t*, *x*) # *xs* ∈ *cpts-pes* Γ
and *a2*[*rule-format*]: ∀ *cs*. (*pes2*, *t*, *x*) # *xs* ∈ *cpts-pes* Γ ∧ Γ (*pes2*, *t*, *x*) # *xs* ∝ *cs* ⟶ *cs k* ∈ *cpts-es* Γ
{
fix *cs*
assume *b0*: (*pes1*, *s*, *y*) # (*pes2*, *t*, *x*) # *xs* ∈ *cpts-pes* Γ
and *b1*: Γ (*pes1*, *s*, *y*) # (*pes2*, *t*, *x*) # *xs* ∝ *cs*
let *?esl* = (*pes2*, *t*, *x*) # *xs*
let *?esllon* = (*pes1*, *s*, *y*) # (*pes2*, *t*, *x*) # *xs*
let *?cs* = (λ*k*. *drop 1* (*cs k*))
from *b1* **have** Γ *?esl* ∝ *?cs* **using** *drop-n-conjoin*[*of* Γ *?esllon cs 1 ?esl ?cs*] **by** *auto*
with *a1 a2*[*of ?cs*] **have** *b2*: *?cs k* ∈ *cpts-es* Γ **by** *simp*
from *b1* **have** *b3*: *cs k* ! *0* = (*pes1 k*, *s*, *y*)
using *conjoin-def*[*of* Γ *?esllon cs*] *same-state-def*[*of ?esllon cs*] *same-spec-def*[*of ?esllon cs*]
by (*metis esconf-trip gets-def getspc-def getx-def length-greater-0-conv list.simps*(*3*) *nth-Cons-0 prod.sel*(*1*) *prod.sel*(*2*))

from *b1* **have** *getspc-es* (*cs k* ! *1*) = (*getspc* (*?esllon* ! *1*)) *k*
using *conjoin-def*[*of* Γ *?esllon cs*] *same-spec-def*[*of ?esllon cs*]
by (*metis diff-Suc-1 length-Cons zero-less-Suc zero-less-diff*)
moreover
from *b1* **have** *gets* (*?esllon* ! *1*) = *gets-es* ((*cs k*)!*1*) ∧ *getx* (*?esllon* ! *1*) = *getx-es* ((*cs k*)!*1*)
using *conjoin-def*[*of* Γ *?esllon cs*] *same-state-def*[*of ?esllon cs*]
diff-Suc-1 length-Cons zero-less-Suc zero-less-diff **by** *fastforce*
ultimately have *b4*: *cs k* ! *1* = (*pes2 k*, *t*, *x*)
using *b0 getspc-def gets-def getx-def*
by (*metis One-nat-def esconf-trip fst-conv nth-Cons-0 nth-Cons-Suc snd-conv*)

from *b1* **have** *compat-tran* Γ *?esllon cs* **by** (*simp add*:*conjoin-def*)
then have ((∃ *t k*. (Γ ⊢ *?esllon*!*0* −*pes*−(*t*♯*k*)→ *?esllon*!*Suc 0*)) ∧
(∀ *k t*. (Γ ⊢ *?esllon*!*0* −*pes*−(*t*♯*k*)→ *?esllon*!*Suc 0*) ⟶ (Γ ⊢ *cs k*!*0* −*es*−(*t*♯*k*)→ *cs k*! *Suc 0*) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (Γ ⊢ *cs k*′!*0* −*ese*→ *cs k*′! *Suc 0*))))
∨
((Γ ⊢ (*?esllon*!*0*) −*pese*→ (*?esllon*!*Suc 0*)) ∧ (∀ *k*. (Γ ⊢

$((cs\ k)!0)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ 0))))$
using *compat-tran-def*[*of* Γ *?esllon cs*] **by** *fastforce*
then have $cs\ k \in cpts\text{-}es\ \Gamma$
proof
assume *c0*: $(\exists\, t\ k.\ (\Gamma \vdash ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0))\ \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash\ ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0)\longrightarrow$
$(\Gamma \vdash cs\ k!0\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ 0)\ \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!0\ -ese\rightarrow\ cs\ k'!\ Suc\ 0)))$
then obtain *t1* **and** *k1* **where** *c1*: $(\Gamma \vdash\ ?esllon!0\ -pes-(t1\sharp k1)\rightarrow$
$?esllon!Suc\ 0)$ **by** *auto*
with *c0* **have** *c2*: $(\Gamma \vdash cs\ k1!0\ -es-(t1\sharp k1)\rightarrow\ cs\ k1!\ Suc\ 0)\ \wedge$
$(\forall\, k'.\ k' \neq k1\ \longrightarrow (\Gamma \vdash cs\ k'!0\ -ese\rightarrow\ cs\ k'!\ Suc\ 0))$
by *auto*
show *?thesis*
proof(*cases* $k = k1$)
assume *d0*: $k = k1$
with *c2* **have** $(\Gamma \vdash cs\ k!0\ -es-(t1\sharp k)\rightarrow\ cs\ k!\ Suc\ 0)$ **by** *auto*
with *b2 b3 b4* **show** *?thesis* **using** *CptsEsComp*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD cpts-es-not-empty drop-0 drop-eq-Nil not-le*)
next
assume *d0*: $k \neq k1$
with *c2* **have** $\Gamma \vdash cs\ k!0\ -ese\rightarrow\ cs\ k!\ Suc\ 0$ **by** *auto*
with *b2 b3 b4* **show** *?thesis* **using** *CptsEsEnv*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD cpts-es-not-empty*

drop-0 drop-eq-Nil esetran-eqconf not-le)
qed
next
assume *c0*: $(\Gamma \vdash (?esllon!0)\ -pese\rightarrow\ (?esllon!Suc\ 0))\ \wedge\ (\forall\, k.\ (\Gamma \vdash$
$((cs\ k)!0)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ 0)))$
then have $\Gamma \vdash ((cs\ k)!0)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ 0)$ **by** *simp*
with *b2 b3 b4* **show** *?thesis* **using** *CptsEsEnv a0 c0 pes-tran-not-etran1*
by *fastforce*
qed
}
then show *?case* **by** *auto*
qed
}
with *p0* **show** *?thesis* **by** *simp*
qed

lemma *conjoin-imp-cptses-k*:
$[\![c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ \Gamma\ c \propto cs]\!]$
$\Longrightarrow cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
proof −
assume *p0*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p1*: $\Gamma\ c \propto cs$
from *p0* **have** *a1*: $c{\in}cpts\text{-}pes\ \Gamma \wedge c!0 = (pes,s,x)$ **by** (*simp add*:*cpts-of-pes-def*)

from *a1 p1* **have** $cs\ k \in cpts\text{-}es\ \Gamma$ **using** *conjoin-imp-cptses-k-help* **by** *auto*
moreover
from *p0 p1* **have** $cs\ k\ !\ 0 = (pes\ k,s,x)$
by (*metis a1 conjoin-def cpts-pes-not-empty esconf-trip fst-conv gets-def getspc-def getx-def length-greater-0-conv same-spec-def same-state-def snd-conv*)

ultimately show *?thesis* **by** (*simp add*:*cpts-of-es-def*)
qed

4.4.3 Semantics is Compositional

lemma *conjoin-cs-imp-cpt*: $[\![\exists k\ p.\ pes\ k = p; (\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs)]\!]$
$\Longrightarrow c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
proof −
assume *p0*: $\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$
and *p1*: $\exists k\ p.\ pes\ k = p$
then obtain *cs* **where** $(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$ **by** *auto*
then have *a0*: $(\forall k.\ (cs\ k)!0=(pes\ k,s,x) \wedge (cs\ k) \in cpts\text{-}es\ \Gamma) \wedge \Gamma\ c \propto cs$ **by** (*simp add*:*cpts-of-es-def*)
from *p1* **obtain** *p* **and** *k* **where** *a1*: $pes\ k = p$ **by** *auto*

from *p1* **obtain** *k* **and** *p* **where** $pes\ k = p$ **by** *auto*
with *a0* **have** *a2*: $(cs\ k)!0=(pes\ k,s,x) \wedge (cs\ k) \in cpts\text{-}es\ \Gamma$ **by** *auto*
then have $(cs\ k) \neq []$ **by** *auto*
moreover
from *a0* **have** *same-length c cs* **by** (*simp add*:*conjoin-def*)
ultimately have *a3*: $c \neq []$ **using** *same-length-def* **by** *force*

have *g0*: $c!0 = (pes,s,x)$
proof −
from *a3 a0* **have** *same-spec c cs* **by** (*simp add*:*conjoin-def*)
with *a3* **have** *b2*: $\forall k.\ (getspc\ (c!0))\ k = getspc\text{-}es\ ((cs\ k)\ !\ 0)$ **by** (*simp add*:*same-spec-def*)
with *a0* **have** $\forall k.\ (getspc\ (c!0))\ k = pes\ k$ **by** (*simp add*:*getspc-es-def*)
then have *b3*: $getspc\ (c!0) = pes$ **by** *auto*

from *a0* **have** *same-state c cs* **by** (*simp add*:*conjoin-def*)
with *a3* **have** $gets\ (c!0) = gets\text{-}es\ ((cs\ k)!0) \wedge getx\ (c!0) = getx\text{-}es\ ((cs\ k)!0)$
by (*simp add*:*same-state-def*)
with *a2* **have** $gets\ (c!0) = s \wedge getx\ (c!0) = x$
by (*simp add*:*gets-def getx-def gets-es-def getx-es-def*)
with *b3* **show** *?thesis* **using** *gets-def getx-def getspc-def* **by** (*metis prod.collapse*)
qed
have $\forall i.\ i > 0 \wedge i \leq length\ c \longrightarrow take\ i\ c \in cpts\text{-}pes\ \Gamma$
proof −
{

fix *i*
assume *b0*: $i > 0 \wedge i \leq$ *length c*
then have *take i c* $\in$ *cpts-pes* Γ
proof(*induct i*)
case *0* **show** *?case* **using** *0.prems* **by** *auto*
next
case (*Suc j*)
assume *c0*: $0 < j \wedge j \leq$ *length c* $\Longrightarrow$ *take j c* $\in$ *cpts-pes* Γ
and *c1*: $0 <$ *Suc j* $\wedge$ *Suc j* $\leq$ *length c*
show *?case*
proof(*cases j = 0*)
assume *d0*: $j = 0$
with *c0* **show** *?case* **by** (*simp add*: *a3 cpts-pes.CptsPesOne g0 hd-conv-nth take-Suc*)
next
assume *d0*: $j \neq 0$
from *a0* **have** *d1*: *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
then have *d2*: $\forall j$. *Suc j* $<$ *length c* $\longrightarrow$
$(\exists\, t\ k.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow\ c!Suc\ j) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow\ c!Suc\ j) \longrightarrow (\Gamma \vdash cs\ k!j\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ j) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ -ese\rightarrow\ cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ -pese\rightarrow\ (c!Suc\ j)) \wedge (\forall\, k.\ (\Gamma \vdash ((cs\ k)!j)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ j))))$
by (*simp add*:*compat-tran-def*)

from *d0* **have** *d3*: $j - 1 \geq 0$ **by** *simp*
from *c1* **have** *d6*: *Suc* $(j - 1) <$ *length c* **using** *d0* **by** *auto*

with *d3* **have** *d4*: $(\exists\, t\ k.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \longrightarrow (\Gamma \vdash cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (j-1)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(j-1)\ -ese\rightarrow\ cs\ k'!\ Suc\ (j-1)))))$
$\vee$
$((\Gamma \vdash (c!(j-1))\ -pese\rightarrow\ (c!Suc\ (j-1))) \wedge (\forall\, k.\ (\Gamma \vdash ((cs\ k)!(j-1))\ -ese\rightarrow\ ((cs\ k)!Suc\ (j-1)))))$
using *d2* **by** *auto*
from *c0 c1 d0* **have** *d5*: *take j c* $\in$ *cpts-pes* Γ **by** *auto*
from *d4* **show** *?case*
proof
assume $(\exists\, t\ k.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \longrightarrow (\Gamma \vdash cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (j-1)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(j-1)\ -ese\rightarrow\ cs\ k'!\ Suc\ (j-1)))))$
then obtain *t* **and** *k* **where** *e0*: $(\Gamma \vdash (c!(j-1))\ -pes-(t\sharp k)\rightarrow\ (c!Suc\ (j-1)))$ **by** *auto*

then have $\Gamma \vdash ((take\ j\ c) ! (length\ (take\ j\ c) - 1))\ -pes-(t\sharp k)\rightarrow (c!Suc\ (j-1))$
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leD Suc-lessD d6 butlast-take c1 d0 length-butlast neq0-conv nth-append-length take-Suc-conv-app-nth*)
with *d5* have $(take\ j\ c)\ @\ [c!Suc\ (j-1)] \in cpts\text{-}pes\ \Gamma$ using *cpts-pes-onemore* by *blast*
then show *?thesis* using *d0 d6 take-Suc-conv-app-nth* by *fastforce*
next
assume $(\Gamma \vdash (c!(j-1))\ -pese\rightarrow (c!Suc\ (j-1))) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!(j-1))\ -ese\rightarrow ((cs\ k)!Suc\ (j-1))))$
then have $\Gamma \vdash ((take\ j\ c) ! (length\ (take\ j\ c) - 1))\ -pese\rightarrow (c!Suc\ (j-1))$
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leD Suc-lessD d6 butlast-take c1 d0 length-butlast neq0-conv nth-append-length take-Suc-conv-app-nth*)
with *d5* have $(take\ j\ c)\ @\ [c!Suc\ (j-1)] \in cpts\text{-}pes\ \Gamma$ using *cpts-pes-onemore* by *blast*
then show *?thesis* using *d0 d6 take-Suc-conv-app-nth* by *fastforce*
qed

qed
qed
}
then show *?thesis* by *auto*
qed

with *a3* have *g1*: $c \in cpts\text{-}pes\ \Gamma$ by *auto*
from *g0 g1* show *?thesis* by (*simp add:cpts-of-pes-def*)
qed

lemma *comp-tran-env*: $[\![(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ t1\ x1);\ c = (pes,\ t1,\ x1)\ \#\ xs;\ c \in cpts\text{-}pes\ \Gamma;$
$\Gamma\ c \propto cs;\ c' = (pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs]\!] \Longrightarrow$
compat-tran $\Gamma\ c'\ (\lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k)$
proof −
let $?cs' = \lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k$
assume *p0*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ t1\ x1$
and *p1*: $c \in cpts\text{-}pes\ \Gamma$
and *p2*: $\Gamma\ c \propto cs$
and *p3*: $c' = (pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs$
and *p4*: $c = (pes,\ t1,\ x1)\ \#\ xs$
from *p0* **have** *b3*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma \wedge (cs\ k)!0 = (pes\ k,t1,x1)$ **by** (*simp add:cpts-of-es-def*)
show *compat-tran* $\Gamma\ c'\ ?cs'$
proof −
{
fix *j*
assume *dd0*: $Suc\ j < length\ c'$

have $(\exists\, t\ k.\ (\Gamma \vdash (c'!j)\ -pes-(t\sharp k)\rightarrow (c'!Suc\ j)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c'!j\ -pes-(t\sharp k)\rightarrow c'!Suc\ j) \longrightarrow (\Gamma \vdash ?cs'\ k!j$
$-es-(t\sharp k)\rightarrow ?cs'\ k!\ Suc\ j) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash ?cs'\ k'!j\ -ese\rightarrow ?cs'\ k'!\ Suc$
$j))))$
$\vee$
$((\Gamma \vdash (c'!j)\ -pese\rightarrow (c'!Suc\ j)) \wedge (\forall\, k.\ (\Gamma \vdash ((?cs'\ k)!j)\ -ese\rightarrow$
$((?cs'\ k)!\ Suc\ j))))$
proof(*cases* $j = 0$)
assume *d0*: $j = 0$
from *p3* **have** $(\Gamma \vdash (c'!0)\ -pese\rightarrow (c'!1))$
by (*simp add*: *pesetran.intros*)
moreover
have $\forall\, k.\ (\Gamma \vdash ((?cs'\ k)!0)\ -ese\rightarrow ((?cs'\ k)!1))$
by (*simp add*: *b3 esetran.intros*)
ultimately show *?thesis* **using** *d0* **by** *simp*
next
assume *d0*: $j \neq 0$
then have *d0-1*: $j > 0$ **by** *simp*
from *p2* **have** *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
then have *d1*: $\forall\, j.\ Suc\ j < length\ c \longrightarrow$
$(\exists\, t\ k.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow c!Suc\ j) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow c!Suc\ j) \longrightarrow (\Gamma \vdash cs\ k!j$
$-es-(t\sharp k)\rightarrow cs\ k!\ Suc\ j) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ -ese\rightarrow cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ -pese\rightarrow (c!Suc\ j)) \wedge (\forall\, k.\ (\Gamma \vdash ((cs\ k)!j)$
$-ese\rightarrow ((cs\ k)!\ Suc\ j))))$
by (*simp add*:*compat-tran-def*)
from *p3 p4 dd0 d0* **have** *d2*: $Suc\ (j-1) < length\ c$ **by** *auto*
let $?j1 = j - 1$
from *d1 d2* **have** *d3*: $(\exists\, t\ k.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow c!Suc\ (j-1)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow c!Suc\ (j-1)) \longrightarrow (\Gamma \vdash$
$cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow cs\ k!\ Suc\ (j-1)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(j-1)\ -ese\rightarrow cs\ k'!$
$Suc\ (j-1)))))$
$\vee$
$((\Gamma \vdash (c!(j-1))\ -pese\rightarrow (c!Suc\ (j-1))) \wedge (\forall\, k.\ (\Gamma \vdash ((cs$
$k)!(j-1))\ -ese\rightarrow ((cs\ k)!Suc\ (j-1)))))$
by *auto*
from *p3 p4 d0 dd0* **have** *d4*: $c'!j = c!(j-1) \wedge c'!Suc\ j = c!Suc\ (j-1)$
by *simp*
have *d5*: $(\forall\, k.\ (?cs'\ k)\ !\ j= (cs\ k)!\ (j-1)) \wedge (\forall\, k.\ (?cs'\ k)\ !\ Suc\ j= (cs$
$k)!\ Suc\ (j-1))$
by (*simp add*: *d0-1*)
with *d3 d4* **show** *?thesis* **by** *auto*
qed

}

then show *?thesis* **by** (*simp add*:*compat-tran-def*)
qed
qed

lemma *comp-tran-pestran*: $[\![(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes2\ k)\ t1\ x1);\ c = (pes2,\ t1,\ x1)\ \#\ xs;\ c{\in}cpts\text{-}pes\ \Gamma;$
$\Gamma\ c \propto cs;\ c' = (pes1,\ s1,\ y1)\ \#\ (pes2,\ t1,\ x1)\ \#\ xs;\ \Gamma \vdash (pes1,\ s1,\ y1)\ {-pes-ct}{\rightarrow}\ (pes2,\ t1,\ x1)]\!]$
$\Longrightarrow compat\text{-}tran\ \Gamma\ c'\ (\lambda k.\ (pes1\ k,\ s1,\ y1)\ \#\ cs\ k)$
proof −
let $?cs' = \lambda k.\ (pes1\ k,\ s1,\ y1)\ \#\ cs\ k$
assume $p0$: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes2\ k)\ t1\ x1$
and $p1$: $c \in cpts\text{-}pes\ \Gamma$
and $p2$: $\Gamma\ c \propto cs$
and $p3$: $c' = (pes1,\ s1,\ y1)\ \#\ (pes2,\ t1,\ x1)\ \#\ xs$
and $p4$: $c = (pes2,\ t1,\ x1)\ \#\ xs$
and $p5$: $\Gamma \vdash (pes1,\ s1,\ y1)\ {-pes-ct}{\rightarrow}\ (pes2,\ t1,\ x1)$
from *p0* **have** $b3$: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma \wedge (cs\ k)!0 = (pes2\ k,t1,x1)$ **by** (*simp add*:*cpts-of-es-def*)
show *compat-tran* $\Gamma\ c'\ ?cs'$
proof −
{
fix *j*
assume $dd0$: $Suc\ j < length\ c'$
have $(\exists t\ k.\ (\Gamma \vdash (c'!j)\ {-pes-(t\sharp k)}{\rightarrow}\ (c'!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c'!j\ {-pes-(t\sharp k)}{\rightarrow}\ c'!Suc\ j) \longrightarrow (\Gamma \vdash ?cs'\ k!j\ {-es-(t\sharp k)}{\rightarrow}\ ?cs'\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash ?cs'\ k'!j\ {-ese}{\rightarrow}\ ?cs'\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c'!j)\ {-pese}{\rightarrow}\ (c'!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((?cs'\ k)!j)\ {-ese}{\rightarrow}\ ((?cs'\ k)!\ Suc\ j))))$
proof(*cases* $j = 0$)
assume $d0$: $j = 0$
from *p5* **obtain** *k* **and** *aa* **where** $c0$: $ct = (aa\sharp k)$ **using** *get-actk-def* **by** (*metis cases*)
with *p5* **have** $\exists es'.\ (\Gamma \vdash (pes1\ k,\ s1,\ y1)\ {-es-(aa\sharp k)}{\rightarrow}\ (es',\ t1,\ x1)) \wedge pes2 = pes1(k{:=}es')$
using *pestran-estran* **by** *auto*
then obtain *es'* **where** $c1$: $(\Gamma \vdash (pes1\ k,\ s1,\ y1)\ {-es-(aa\sharp k)}{\rightarrow}\ (es',\ t1,\ x1)) \wedge pes2 = pes1(k{:=}es')$
by *auto*
from *b3* **have** $c2$: $cs\ k \in cpts\text{-}es\ \Gamma \wedge (cs\ k)!0 = (pes2\ k,t1,x1)$ **by** *auto*
then obtain *xs1* **where** $c4$: $(cs\ k) = (pes2\ k,t1,x1)\#xs1$
by (*metis cpts-es-not-empty neq-Nil-conv nth-Cons-0*)
then have $c3$: $?cs'\ k = (pes1\ k,\ s1,\ y1)\ \#\ (pes2\ k,t1,x1)\#xs1$ **by** *simp*

from *p3 p5 c0* **have** $g0$: $\Gamma \vdash (c'!0)\ {-pes-(aa\sharp k)}{\rightarrow}\ (c'!Suc\ 0)$ **by** *auto*
moreover

have $\forall$ *k1 t1*. ($\Gamma \vdash c'!0$ $-pes-(t1\sharp k1)\rightarrow$ $c'!Suc\ 0$) $\longrightarrow$ ($\Gamma \vdash$ *?cs' k1*!*0* $-es-(t1\sharp k1)\rightarrow$ *?cs' k1*! *Suc 0*) $\wedge$
($\forall k'$. $k' \neq k1 \longrightarrow$ ($\Gamma \vdash$ *?cs' k'*!*0* $-ese\rightarrow$ *?cs' k'*! *Suc 0*))
proof –
{
fix *k1 t1*
assume *d0*: $\Gamma \vdash c'!0$ $-pes-(t1\sharp k1)\rightarrow$ $c'!Suc\ 0$
with *p3* **have** $\Gamma \vdash$ *?cs' k1*!*0* $-es-(t1\sharp k1)\rightarrow$ *?cs' k1*! *Suc 0*
using *b3 fun-upd-apply nth-Cons-0 nth-Cons-Suc pestran-estran* **by** *fastforce*
moreover
from *d0* **have** $\forall k'$. $k' \neq k1 \longrightarrow$ ($\Gamma \vdash$ *?cs' k'*!*0* $-ese\rightarrow$ *?cs' k'*! *Suc 0*)
using *b3 esetran.intros fun-upd-apply nth-Cons-0 nth-Cons-Suc p3 pestran-estran* **by** *fastforce*
ultimately have ($\Gamma \vdash c'!0$ $-pes-(t1\sharp k1)\rightarrow$ $c'!Suc\ 0$) $\longrightarrow$ ($\Gamma \vdash$ *?cs' k1*!*0* $-es-(t1\sharp k1)\rightarrow$ *?cs' k1*! *Suc 0*) $\wedge$
($\forall k'$. $k' \neq k1 \longrightarrow$ ($\Gamma \vdash$ *?cs' k'*!*0* $-ese\rightarrow$ *?cs' k'*! *Suc 0*)) **by** *simp*
}
then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **using** *d0* **by** *auto*
next
assume *d0*: $j \neq 0$
then have *d0-1*: $j > 0$ **by** *simp*
from *p2* **have** *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
then have *d1*: $\forall j$. *Suc j* $<$ *length c* $\longrightarrow$
($\exists t\ k$. ($\Gamma \vdash c!j$ $-pes-(t\sharp k)\rightarrow$ $c!Suc\ j$) $\wedge$
($\forall k\ t$. ($\Gamma \vdash c!j$ $-pes-(t\sharp k)\rightarrow$ $c!Suc\ j$) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*j* $-es-(t\sharp k)\rightarrow$ *cs k*! *Suc j*) $\wedge$
($\forall k'$. $k' \neq k \longrightarrow$ ($\Gamma \vdash$ *cs k'*!*j* $-ese\rightarrow$ *cs k'*! *Suc j*))))
$\vee$
(($\Gamma \vdash$ ($c!j$) $-pese\rightarrow$ ($c!Suc\ j$)) $\wedge$ ($\forall k$. ($\Gamma \vdash$ ((*cs k*)!*j*) $-ese\rightarrow$ ((*cs k*)! *Suc j*))))
by (*simp add*:*compat-tran-def*)
from *p3 p4 dd0 d0* **have** *d2*: *Suc* ($j-1$) $<$ *length c* **by** *auto*
with *d0 d0-1 d1* **have** *d3*: ($\exists t\ k$. ($\Gamma \vdash c!(j-1)$ $-pes-(t\sharp k)\rightarrow$ $c!Suc\ (j-1)$) $\wedge$
($\forall k\ t$. ($\Gamma \vdash c!(j-1)$ $-pes-(t\sharp k)\rightarrow$ $c!Suc\ (j-1)$) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!($j-1$) $-es-(t\sharp k)\rightarrow$ *cs k*! *Suc* ($j-1$)) $\wedge$
($\forall k'$. $k' \neq k \longrightarrow$ ($\Gamma \vdash$ *cs k'*!($j-1$) $-ese\rightarrow$ *cs k'*! *Suc* ($j-1$)))))
$\vee$
(($\Gamma \vdash$ ($c!(j-1)$) $-pese\rightarrow$ ($c!Suc\ (j-1)$)) $\wedge$ ($\forall k$. ($\Gamma \vdash$ ((*cs k*)!($j-1$)) $-ese\rightarrow$ ((*cs k*)!*Suc* ($j-1$)))))
by *blast*
from *p3 p4 d0 dd0* **have** *d4*: $c'!j = c!(j-1) \wedge c'!Suc\ j = c!Suc\ (j-1)$

by *simp*
have *d5*: ($\forall$ *k*. (*?cs′ k*) ! *j*= (*cs k*)! (*j*−*1*)) $\wedge$ ($\forall$ *k*. (*?cs′ k*) ! *Suc j*= (*cs k*)! *Suc* (*j*−*1*))
by (*simp add*: *d0-1*)
with *d3 d4* **show** *?thesis* **by** *auto*
qed

}
then show *?thesis* **by** (*simp add*:*compat-tran-def*)
qed
qed

lemma *cpt-imp-exist-conjoin-cs0*:
$\forall$ *c*. *c* $\in$ *cpts-pes* Γ $\longrightarrow$
($\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ ((*getspc* (*c*!*0*)) *k*) (*gets* (*c*!*0*)) (*getx* (*c*!*0*))) $\wedge$ Γ *c* $\propto$ *cs*)
proof −
{
fix *c*
assume *p0*: *c* $\in$ *cpts-pes* Γ
then have $\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ ((*getspc* (*c*!*0*)) *k*) (*gets* (*c*!*0*)) (*getx* (*c*!*0*))) $\wedge$ Γ *c* $\propto$ *cs*
proof(*induct c*)
case (*CptsPesOne pes1 s1 x1*)
let *?cs* = λ*k*. [(*pes1 k*, *s1*,*x1*)]
let *?c* = [(*pes1*, *s1*, *x1*)]
have $\forall$ *k*. *?cs k* $\in$ *cpts-of-es* Γ (*getspc* (*?c* ! *0*) *k*) (*gets* (*?c* ! *0*)) (*getx* (*?c* ! *0*))
proof −
{
fix *k*
have *?cs k* = [(*pes1 k*,*s1*,*x1*)] **by** *simp*
moreover
have *?cs k* $\in$ *cpts-es* Γ **by** (*simp add*: *cpts-es*.*CptsEsOne*)
ultimately have *?cs k* $\in$ *cpts-of-es* Γ (*pes1 k*) *s1 x1* **by** (*simp add*: *cpts-of-es-def*)
}
then show *?thesis* **by** (*simp add*: *gets-def getspc-def getx-def*)
qed
moreover
have Γ *?c* $\propto$ *?cs*
proof −
have *same-length ?c ?cs* **by** (*simp add*: *same-length-def*)
moreover
have *same-state ?c ?cs* **using** *same-state-def gets-def gets-es-def getx-def getx-es-def*
by (*smt length-Cons less-Suc0 list.size*(*3*) *nth-Cons-0 snd-conv*)
moreover
have *same-spec ?c ?cs* **using** *same-spec-def getspc-def getspc-es-def*

by (*metis* (*mono-tags*, *lifting*) *fst-conv length-Cons less-Suc0 list.size*(*3*) *nth-Cons-0*)
moreover
have *compat-tran* Γ *?c ?cs* **by** (*simp add*: *compat-tran-def*)
ultimately show *?thesis* **by** (*simp add*:*conjoin-def*)
qed
ultimately show *?case* **by** *auto*
next
case (*CptsPesEnv pes1 t1 x1 xs s1 y1*)
let $?c = (pes1,\ t1,\ x1)\ \#\ xs$
assume *b0*: $?c \in$ *cpts-pes* Γ
and *b1*: $\exists\, cs.\ (\forall\, k.\ cs\ k \in$ *cpts-of-es* Γ (*getspc* ($?c$! *0*) *k*) (*gets* ($?c$! *0*)) (*getx* ($?c$! *0*))) $\wedge$ Γ $?c \propto cs$
then obtain *cs* **where** *b2*: $(\forall\, k.\ cs\ k \in$ *cpts-of-es* Γ (*pes1 k*) *t1 x1*) $\wedge$ Γ $?c \propto cs$
using *getspc-def gets-def getx-def* **by** (*metis fst-conv nth-Cons-0 snd-conv*)

then have *b3*: $\forall\, k.\ cs\ k \in$ *cpts-es* Γ $\wedge$ (*cs k*)!*0* = (*pes1 k*,*t1*,*x1*) **by** (*simp add*:*cpts-of-es-def*)
let $?c' = (pes1,\ s1,\ y1)\ \#\ (pes1,\ t1,\ x1)\ \#\ xs$
let $?cs' = \lambda k.\ (pes1\ k, s1, y1)\#(cs\ k)$
have *g0*: $\forall\, k.\ ?cs'\ k \in$ *cpts-of-es* Γ (*getspc* ($?c'$! *0*) *k*) (*gets* ($?c'$! *0*)) (*getx* ($?c'$! *0*))
proof −
{
fix *k*
from *b3* **have** *c0*: *cs k* ∈*cpts-es* Γ $\wedge$ (*cs k*)!*0* = (*pes1 k*,*t1*,*x1*) **by** *auto*
then obtain *xs1* **where** (*cs k*) = (*pes1 k*,*t1*,*x1*)#*xs1*
by (*metis cpts-es-not-empty neq-Nil-conv nth-Cons-0*)
with *c0* **have** *c1*: $?cs'\ k \in$ *cpts-es* Γ **by** (*simp add*: *cpts-es.CptsEsEnv*)
then have $?cs'\ k \in$ *cpts-of-es* Γ (*getspc* ($?c'$! *0*) *k*) (*gets* ($?c'$! *0*)) (*getx* ($?c'$! *0*))
by (*simp add*: *cpts-of-es-def gets-def getspc-def getx-def*)
}
then show *?thesis* **by** *auto*
qed
from *b2* **have** *b4*: Γ $?c \propto cs$ **by** *simp*
from *b1* **have** *g1*: Γ $?c' \propto ?cs'$
proof −
from *b4* **have** *same-length* $?c'$ $?cs'$
by (*simp add*: *conjoin-def same-length-def*)
moreover
have *same-state* $?c'$ $?cs'$
proof −
{
fix k' *j*
assume *c0*: $j < length\ ?c'$
have *gets* ($?c'$!*j*) = *gets-es* (($?cs'\ k'$)!*j*) $\wedge$ *getx* ($?c'$!*j*) = *getx-es* (($?cs'\ k'$)!*j*)

```
            proof(cases j = 0)
              assume d0: j = 0
                then show ?thesis by (simp add:gets-def gets-es-def getx-def
getx-es-def)
            next
              assume d0: j ≠ 0
               with b4 show ?thesis using same-state-def gets-def gets-es-def
getx-def getx-es-def
                using c0 conjoin-def length-Cons less-Suc-eq-0-disj nth-Cons-Suc
by fastforce
            qed
          }
          then show ?thesis by (simp add: same-state-def)
          qed

        moreover
        have same-spec ?c' ?cs'
          proof -
          {
            fix k' j
            assume c0: j < length ?c'
            have (getspc (?c'!j)) k' = getspc-es ((?cs' k') ! j)
              proof(cases j = 0)
                assume d0: j = 0
                then show ?thesis by (simp add:getspc-def getspc-es-def)
              next
                assume d0: j ≠ 0
               with b4 show ?thesis using same-spec-def getspc-def getspc-es-def
                   by (metis (no-types, lifting) Nat.le-diff-conv2 One-nat-def c0
conjoin-def
                   less-Suc0 list.size(4) not-less nth-Cons')
              qed
          }
          then show ?thesis by (simp add: same-spec-def)
          qed
        moreover
        from b0 b2 b4 have compat-tran Γ ?c' ?cs'
          using comp-tran-env [of cs Γ pes1 t1 x1 ?c xs ?c' s1 y1] by simp
        ultimately show ?thesis by (simp add:conjoin-def)
      qed
    from g0 g1 show ?case by auto
  next
    case (CptsPesComp pes1 s1 y1 ct pes2 t1 x1 xs)
    let ?c = (pes2, t1, x1) # xs
    assume b0: ?c∈ cpts-pes Γ
      and b1: ∃ cs. (∀ k. cs k ∈ cpts-of-es Γ (getspc (?c ! 0) k) (gets (?c ! 0))
                (getx (?c ! 0))) ∧ Γ ?c ∝ cs
      and b00: Γ ⊢ (pes1, s1, y1) −pes−ct→ (pes2, t1, x1)
    then obtain cs where b2: (∀ k. cs k ∈ cpts-of-es Γ (pes2 k) t1 x1) ∧ Γ ?c
```

$\propto$ *cs*
using *getspc-def gets-def getx-def* **by** (*metis fst-conv nth-Cons-0 snd-conv*)

then have *b3*: $\forall k.$ *cs k* $\in$ *cpts-es* $\Gamma \wedge$ (*cs k*)!*0* = (*pes2 k*,*t1*,*x1*) **by** (*simp add*:*cpts-of-es-def*)
let *?c'* = (*pes1*, *s1*, *y1*) # (*pes2*, *t1*, *x1*) # *xs*
let *?cs'* = $\lambda k.$ (*pes1 k*,*s1*,*y1*)#(*cs k*)
have *g0*: $\forall k.$ *?cs' k* $\in$ *cpts-of-es* Γ (*getspc* (*?c'* ! *0*) *k*) (*gets* (*?c'* ! *0*)) (*getx* (*?c'* ! *0*))
proof −
{
fix *k*
obtain *ka* **and** *aa* **where** *c0*: *ct* = (*aa*$\sharp$*ka*) **using** *get-actk-def* **by** (*metis cases*)
with *b00* **have** $\exists$ *es'*. ($\Gamma \vdash$ (*pes1 ka*, *s1*, *y1*) −*es*−(*aa*$\sharp$*ka*)$\rightarrow$ (*es'*, *t1*, *x1*)) $\wedge$ *pes2* = *pes1*(*ka*:=*es'*)
using *pestran-estran* **by** *auto*
then obtain *es'* **where** *c1*: ($\Gamma \vdash$ (*pes1 ka*, *s1*, *y1*) −*es*−(*aa*$\sharp$*ka*)$\rightarrow$ (*es'*, *t1*, *x1*)) $\wedge$ *pes2* = *pes1*(*ka*:=*es'*)
by *auto*
from *b3* **have** *c2*: *cs k* $\in$*cpts-es* $\Gamma \wedge$ (*cs k*)!*0* = (*pes2 k*,*t1*,*x1*) **by** *auto*
then obtain *xs1* **where** *c4*: (*cs k*) = (*pes2 k*,*t1*,*x1*)#*xs1*
by (*metis cpts-es-not-empty neq-Nil-conv nth-Cons-0*)
then have *c3*: *?cs' k* = (*pes1 k*, *s1*, *y1*) # (*pes2 k*,*t1*,*x1*)#*xs1* **by** *simp*
have *?cs' k* $\in$ *cpts-of-es* Γ (*getspc* (*?c'* ! *0*) *k*) (*gets* (*?c'* ! *0*)) (*getx* (*?c'* ! *0*))
proof(*cases k* = *ka*)
assume *d0*: *k* = *ka*
with *c1* **have** $\Gamma \vdash$ (*pes1 k*, *s1*, *y1*) −*es*−(*aa*$\sharp$*k*)$\rightarrow$ (*pes2 k*, *t1*, *x1*) **by** *auto*
with *c2 c3 d0* **have** *?cs' k* $\in$ *cpts-es* Γ
using *cpts-es.CptsEsComp* **by** *fastforce*
then show *?thesis* **by** (*simp add*: *cpts-of-es-def gets-def getspc-def getx-def*)
next
assume *d0*: *k* $\neq$ *ka*
with *c1* **have** *pes1 k* = *pes2 k* **by** *simp*
with *c2 c3* **have** *d1*: *?cs' k* $\in$ *cpts-es* Γ
by (*simp add*: *cpts-es.CptsEsEnv*)
then show *?thesis* **by** (*simp add*: *cpts-of-es-def gets-def getspc-def getx-def*)
qed
}
then show *?thesis* **by** *auto*
qed
from *b2* **have** *b4*: Γ *?c* $\propto$ *cs* **by** *simp*
from *b1* **have** *g1*: Γ *?c'* $\propto$ *?cs'*
proof −
from *b4* **have** *same-length ?c' ?cs'*

by (*simp add*: *conjoin-def same-length-def*)
moreover
have *same-state ?c′ ?cs′*
proof −
{
fix *k′ j*
assume *c0*: *j* < *length ?c′*
have *gets* (*?c′*!*j*) = *gets-es* ((*?cs′ k′*)!*j*) ∧ *getx* (*?c′*!*j*) = *getx-es* ((*?cs′ k′*)!*j*)
proof(*cases j* = *0*)
assume *d0*: *j* = *0*
then show *?thesis* **by** (*simp add*:*gets-def gets-es-def getx-def getx-es-def*)
next
assume *d0*: *j* ≠ *0*
with *b4* **show** *?thesis* **using** *same-state-def gets-def gets-es-def getx-def getx-es-def*
using *c0 conjoin-def length-Cons less-Suc-eq-0-disj nth-Cons-Suc* **by** *fastforce*
qed
}
then show *?thesis* **by** (*simp add*: *same-state-def*)
qed

moreover
have *same-spec ?c′ ?cs′*
proof −
{
fix *k′ j*
assume *c0*: *j* < *length ?c′*
have (*getspc* (*?c′*!*j*)) *k′* = *getspc-es* ((*?cs′ k′*) ! *j*)
proof(*cases j* = *0*)
assume *d0*: *j* = *0*
then show *?thesis* **by** (*simp add*:*getspc-def getspc-es-def*)
next
assume *d0*: *j* ≠ *0*
with *b4* **show** *?thesis* **using** *same-spec-def getspc-def getspc-es-def*
by (*metis* (*no-types*, *lifting*) *Nat.le-diff-conv2 One-nat-def Suc-leI c0 conjoin-def*
list.size(*4*) *neq0-conv not-less nth-Cons′*)
qed
}
then show *?thesis* **by** (*simp add*: *same-spec-def*)
qed
moreover
from *b0 b00 b2 b4* **have** *compat-tran* Γ *?c′ ?cs′*
using *comp-tran-pestran* [*of cs* Γ *pes2 t1 x1 ?c xs ?c′ pes1 s1 y1 ct*] **by** *simp*

ultimately show *?thesis* **by** (*simp add*:*conjoin-def*)
qed
from *g0 g1* **show** *?case* **by** *auto*
qed

}
then show *?thesis* **by** (*metis* (*mono-tags*, *lifting*))
qed

lemma *cpt-imp-exist-conjoin-cs*: $c \in$ *cpts-of-pes* Γ *pes s x*
$\Longrightarrow \exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*
proof −
assume *p0*: $c \in$ *cpts-of-pes* Γ *pes s x*
then have *c*!*0*=(*pes*,*s*,*x*) $\wedge$ $c \in$ *cpts-pes* Γ **by** (*simp add*:*cpts-of-pes-def*)
then show *?thesis*
using *cpt-imp-exist-conjoin-cs0 getspc-def gets-def getx-def*
by (*metis fst-conv snd-conv*)
qed

theorem *par-evtsys-semantics-comp*:
cpts-of-pes Γ *pes s x* = {*c*. $\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*}
proof −
have $\forall$ *c*. *c*$\in$*cpts-of-pes* Γ *pes s x* $\longrightarrow$ ($\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*)
proof −
{
fix *c*
assume *a0*: *c*$\in$*cpts-of-pes* Γ *pes s x*
then have $\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*
using *cpt-imp-exist-conjoin-cs cpts-of-pes-def getx-def mem-Collect-eq prod.sel*(*2*) **by** *fastforce*
}
then show *?thesis* **by** *auto*
qed
moreover
have $\forall$ *c*. ($\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*) $\longrightarrow$ *c*$\in$*cpts-of-pes* Γ *pes s x*
proof −
{
fix *c*
assume *a0*: $\exists$ *cs*. ($\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* Γ (*pes k*) *s x*) $\wedge$ Γ *c* $\propto$ *cs*
then have *c*$\in$*cpts-of-pes* Γ *pes s x*
using *conjoin-cs-imp-cpt* **by** *fastforce*
}
then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*

qed

end

end

5 Rely-guarnatee Validity of Picore Computations

theory *PiCore-Validity*
imports *PiCore-Computation*
begin

5.1 Definitions Correctness Formulas

locale *event-validity* = *event-comp ptran petran fin-com cpts-p cpts-of-p*
for *ptran* :: *'Env* ⇒ ((*'prog* × *'s*) × *'prog* × *'s*) *set*
and *petran* :: *'Env* ⇒ (*'s*,*'prog*) *pconf* ⇒ (*'s*,*'prog*) *pconf* ⇒ *bool* (- ⊢ - −*pe*→ - [*81*,*81*,*81*] *80*)
and *fin-com* :: *'prog*
and *cpts-p* :: *'Env* ⇒ (*'s*,*'prog*) *pconfs set*
and *cpts-of-p* :: *'Env* ⇒ *'prog* ⇒ *'s* ⇒ ((*'s*,*'prog*) *pconfs*) *set*
+
fixes *prog-validity* :: *'Env* ⇒ *'prog* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*
(- ⊨ - sat_p [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)
fixes *assume-p* :: *'Env* ⇒ (*'s set* × (*'s* × *'s*) *set*) ⇒ ((*'s*,*'prog*) *pconfs*) *set*
fixes *commit-p* :: *'Env* ⇒ ((*'s* × *'s*) *set* × *'s set*) ⇒ ((*'s*,*'prog*) *pconfs*) *set*
assumes *prog-validity-def*: Γ ⊨ *P* sat_p [*pre*, *rely*, *guar*, *post*] ⟹
∀ *s*. *cpts-of-p* Γ *P s* ∩ *assume-p* Γ (*pre*, *rely*) ⊆ *commit-p* Γ (*guar*, *post*)

assumes *assume-p-def*: *gets-p* (*c*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*pe*→ *c*!(*Suc i*) ⟶ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) ∈ *rely*)
⟹ *c* ∈ *assume-p* Γ (*pre*, *rely*)

assumes *commit-p-def*: *c* ∈ *commit-p* Γ (*guar*, *post*) ⟹ (∀ *i*. *Suc i*<*length c* ⟶

Γ ⊢ *c*!*i* −*c*→ *c*!(*Suc i*) ⟶ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) ∈ *guar*) ∧
(*getspc-p* (*last c*) = *fin-com* ⟶ *gets-p* (*last c*) ∈ *post*)
begin

definition *assume-e* :: *'Env* ⇒ (*'s set* × (*'s* × *'s*) *set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *econfs*) *set* **where**
assume-e Γ ≡ λ(*pre*, *rely*). {*c*. *gets-e* (*c*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*ee*→ *c*!(*Suc i*) ⟶ (*gets-e* (*c*!*i*), *gets-e* (*c*!*Suc i*)) ∈ *rely*)}

definition *commit-e* :: *'Env* ⇒ ((*'s* × *'s*) *set* × *'s set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *econfs*) *set* **where**
commit-e Γ ≡ λ(*guar*, *post*). {*c*. (∀ *i*. *Suc i*<*length c* ⟶
(∃ *t*. Γ ⊢ *c*!*i* −*et*−*t*→ *c*!(*Suc i*)) ⟶ (*gets-e* (*c*!*i*), *gets-e* (*c*!*Suc i*)) ∈

guar) ∧
(*getspc-e* (*last c*) = *AnonyEvent fin-com* ⟶ *gets-e* (*last c*) ∈ *post*)}

definition *evt-validity* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *event* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*
(- ⊨ - sat_e [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*) **where**
Γ ⊨ *Evt* sat_e [*pre*, *rely*, *guar*, *post*] ≡
∀ *s x*. (*cpts-of-ev* Γ *Evt s x*) ∩ *assume-e* Γ (*pre*, *rely*) ⊆ *commit-e* Γ (*guar*, *post*)

definition *assume-es* :: *'Env* ⇒ (*'s set* × (*'s* × *'s*) *set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *esconfs*) *set* **where**
assume-es Γ ≡ λ(*pre*, *rely*). {*c*. *gets-es* (*c*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*ese*→ *c*!(*Suc i*) ⟶ (*gets-es* (*c*!*i*), *gets-es* (*c*!*Suc i*)) ∈ *rely*)}

definition *commit-es* :: *'Env* ⇒ ((*'s* × *'s*) *set* × *'s set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *esconfs*) *set* **where**
commit-es Γ ≡ λ(*guar*, *post*). {*c*. (∀ *i*. *Suc i*<*length c* ⟶
(∃ *t*. Γ ⊢ *c*!*i* −*es*−*t*→ *c*!(*Suc i*)) ⟶ (*gets-es* (*c*!*i*), *gets-es* (*c*!*Suc i*)) ∈ *guar*) }

definition *es-validity* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *esys* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*
(- ⊨ - sat_s [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*) **where**
Γ ⊨ *es* sat_s [*pre*, *rely*, *guar*, *post*] ≡
∀ *s x*. (*cpts-of-es* Γ *es s x*) ∩ *assume-es* Γ (*pre*, *rely*) ⊆ *commit-es* Γ (*guar*, *post*)

definition *assume-pes* :: *'Env* ⇒ (*'s set* × (*'s* × *'s*) *set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *pescon-fs*) *set* **where**
assume-pes Γ ≡ λ(*pre*, *rely*). {*c*. *gets* (*c*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*pese*→ *c*!(*Suc i*) ⟶ (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely*)}

definition *commit-pes* :: *'Env* ⇒ ((*'s* × *'s*) *set* × *'s set*) ⇒ ((*'l*,*'k*,*'s*,*'prog*) *pescon-fs*) *set* **where**
commit-pes Γ ≡ λ(*guar*, *post*). {*c*. (∀ *i*. *Suc i*<*length c* ⟶
(∃ *t*. Γ ⊢ *c*!*i* −*pes*−*t*→ *c*!(*Suc i*)) ⟶ (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *guar*)}

definition *pes-validity* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *paresys* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*
(- ⊨ - *SAT* [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*) **where**
Γ ⊨ *pes SAT* [*pre*, *rely*, *guar*, *post*] ≡
∀ *s x*. (*cpts-of-pes* Γ *pes s x*) ∩ *assume-pes* Γ (*pre*, *rely*) ⊆ *commit-pes* Γ (*guar*, *post*)

5.2 Lemmas of Correctness Formulas

lemma *assume-es-one-more*:
⟦*esl*∈*cpts-es* Γ; *m* > *0*; *m* < *length esl*; *take m esl*∈*assume-es* Γ (*pre*, *rely*); ¬(Γ ⊢ *esl*!(*m*−*1*) −*ese*→*esl*!*m*)⟧

$\Longrightarrow$ *take* (*Suc m*) *esl* $\in$ *assume-es* Γ (*pre*, *rely*)
proof –
assume *p0*: *esl*$\in$*cpts-es* Γ
and *p1*: *m* > *0*
and *p2*: *m* < *length esl*
and *p3*: *take m esl*$\in$*assume-es* Γ (*pre*, *rely*)
and *p4*: $\neg$(Γ $\vdash$ *esl*!(*m*–*1*) –*ese*$\rightarrow$*esl*!*m*)
let *?esl1* = *take* (*Suc m*) *esl*
let *?esl* = *take m esl*
have *gets-es* (*?esl1*!*0*) $\in$ *pre* $\wedge$ ($\forall$ *i*. *Suc i*<*length ?esl1* $\longrightarrow$
Γ $\vdash$ *?esl1*!*i* –*ese*$\rightarrow$ *?esl1*!(*Suc i*) $\longrightarrow$ (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) $\in$ *rely*)
proof
from *p1 p2 p3* **show** *gets-es* (*?esl1*!*0*) $\in$ *pre* **by** (*simp add*:*assume-es-def*)
next
show $\forall$ *i*. *Suc i*<*length ?esl1* $\longrightarrow$
Γ $\vdash$ *?esl1*!*i* –*ese*$\rightarrow$ *?esl1*!(*Suc i*) $\longrightarrow$ (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) $\in$ *rely*
proof –
{
fix *i*
assume *a0*: *Suc i*<*length ?esl1*
and *a1*: Γ $\vdash$ *?esl1*!*i* –*ese*$\rightarrow$ *?esl1*!(*Suc i*)
have (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) $\in$ *rely*
proof(*cases i* < *m* – *1*)
assume *b0*: *i* < *m* – *1*
with *p1* **have** *b1*: *gets-es* (*?esl1*!*i*) = *gets-es* (*?esl*!*i*) **by** *simp*
from *b0 p1* **have** *b2*: *gets-es* (*?esl1*!*Suc i*) = *gets-es* (*?esl*!*Suc i*) **by** *simp*
from *p3* **have** $\forall$ *i*. *Suc i*<*length ?esl* $\longrightarrow$
Γ $\vdash$ *?esl*!*i* –*ese*$\rightarrow$ *?esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *rely*
by (*simp add*:*assume-es-def*)
with *b0* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *rely*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-mono Suc-pred a1*
length-take less-SucI less-imp-le-nat min.absorb2 nth-take p1 p2)
with *b1 b2* **show** *?thesis* **by** *simp*
next
assume $\neg$(*i* < *m* – *1*)
with *a0* **have** *b0*: *i* = *m* – *1* **by** (*simp add*: *less-antisym p1*)
with *p1 p4 a1* **show** *?thesis* **by** *simp*
qed
} **then show** *?thesis* **by** *auto* **qed**
qed
then show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

lemma *assume-es-take-n*:

⟦$m > 0$; $m \leq$ *length esl*; *esl*∈*assume-es* Γ (*pre*, *rely*)⟧
⟹ *take m esl* ∈ *assume-es* Γ (*pre*, *rely*)
proof −
assume *p1*: $m > 0$
and *p2*: $m \leq$ *length esl*
and *p3*: *esl*∈*assume-es* Γ (*pre*, *rely*)
let *?esl1* = *take m esl*
from *p3* **have** *gets-es* (*esl*!*0*)∈*pre* **by** (*simp add*:*assume-es-def*)
with *p1 p2 p3* **have** *gets-es* (*?esl1*!*0*) ∈ *pre* **by** *simp*
moreover
have ∀ *i*. *Suc i*<*length ?esl1* ⟶
Γ ⊢ *?esl1*!*i* −*ese*→ *?esl1*!(*Suc i*) ⟶ (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *rely*
proof −
{
fix *i*
assume *a0*: *Suc i*<*length ?esl1*
and *a1*: Γ ⊢ *?esl1*!*i* −*ese*→ *?esl1*!(*Suc i*)
with *p3* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
with *p1 p2 a0* **have** (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *rely*
using *Suc-lessD length-take min.absorb2 nth-take* **by** *auto*
}
then show *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

lemma *assume-es-drop-n*:
⟦$m <$ *length esl*; *esl*∈*assume-es* Γ (*pre*, *rely*); *gets-es* (*esl*!*m*) ∈ *pre1*⟧
⟹ *drop m esl* ∈ *assume-es* Γ (*pre1*, *rely*)
proof −
assume *p1*: $m <$ *length esl*
and *p3*: *esl*∈*assume-es* Γ (*pre*, *rely*)
and *p2*: *gets-es* (*esl*!*m*) ∈ *pre1*
let *?esl1* = *drop m esl*
from *p1 p2 p3* **have** *gets-es* (*?esl1*!*0*) ∈ *pre1*
by (*simp add*: *hd-conv-nth hd-drop-conv-nth not-less*)
moreover
have ∀ *i*. *Suc i*<*length ?esl1* ⟶
Γ ⊢ *?esl1*!*i* −*ese*→ *?esl1*!(*Suc i*) ⟶ (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *rely*
proof −
{
fix *i*
assume *a0*: *Suc i*<*length ?esl1*
and *a1*: Γ ⊢ *?esl1*!*i* −*ese*→ *?esl1*!(*Suc i*)
with *p1 p3* **have** (*gets-es* (*esl*!(*m*+*i*)), *gets-es* (*esl*!*Suc* (*m*+*i*))) ∈ *rely* **by** (*simp add*: *assume-es-def*)
with *p1 p2 a0* **have** (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *rely*

using *Suc-lessD length-take min.absorb2 nth-take* **by** *auto*
}
then show *?thesis* **by** *auto* **qed**
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

lemma *commit-es-take-n*:
$[\![m > 0;\ m \leq length\ esl;\ esl{\in}commit\text{-}es\ \Gamma\ (guar,\ post)]\!]$
$\Longrightarrow take\ m\ esl \in commit\text{-}es\ \Gamma\ (guar,\ post)$
proof −
assume *p1*: $m > 0$
and *p2*: $m \leq length\ esl$
and *p3*: $esl{\in}commit\text{-}es\ \Gamma\ (guar,\ post)$
let $?esl1 = take\ m\ esl$
have $\forall i.\ Suc\ i{<}length\ ?esl1 \longrightarrow$
$(\exists t.\ \Gamma \vdash ?esl1!i\ {-es-t\rightarrow}\ ?esl1!(Suc\ i)) \longrightarrow (gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in guar$
proof −
{
fix *i*
assume *a0*: $Suc\ i{<}length\ ?esl1$
and *a1*: $(\exists t.\ \Gamma \vdash ?esl1!i\ {-es-t\rightarrow}\ ?esl1!(Suc\ i))$
with *p3* **have** $(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$ **by** (*simp add*:*commit-es-def*)
with *p1 p2 a0* **have** $(gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in guar$
using *Suc-lessD length-take min.absorb2 nth-take* **by** *auto*
}
then show *?thesis* **by** *auto* **qed**
then show *?thesis* **by** (*simp add*:*commit-es-def*)
qed

lemma *commit-es-drop-n*:
$[\![m < length\ esl;\ esl{\in}commit\text{-}es\ \Gamma\ (guar,\ post)]\!]$
$\Longrightarrow drop\ m\ esl \in commit\text{-}es\ \Gamma\ (guar,\ post)$
proof −
assume *p1*: $m < length\ esl$
and *p3*: $esl{\in}commit\text{-}es\ \Gamma\ (guar,\ post)$
let $?esl1 = drop\ m\ esl$
have $\forall i.\ Suc\ i{<}length\ ?esl1 \longrightarrow$
$(\exists t.\ \Gamma \vdash ?esl1!i\ {-es-t\rightarrow}\ ?esl1!(Suc\ i)) \longrightarrow (gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in guar$
proof −
{
fix *i*
assume *a0*: $Suc\ i{<}length\ ?esl1$
and *a1*: $(\exists t.\ \Gamma \vdash ?esl1!i\ {-es-t\rightarrow}\ ?esl1!(Suc\ i))$
with *p3* **have** $(gets\text{-}es\ (esl!(m{+}i)),\ gets\text{-}es\ (esl!Suc\ (m{+}i))) \in guar$ **by** (*simp add*:*commit-es-def*)
with *p1 a0* **have** $(gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in guar$

using *Suc-lessD length-take min.absorb2 nth-take* **by** *auto*
}
then show *?thesis* **by** *auto* **qed**
then show *?thesis* **by** (*simp add*:*commit-es-def*)
qed

lemma *assume-es-imp*: $[\![$*pre1*$\subseteq$*pre*; *rely1*$\subseteq$*rely*; *c*$\in$*assume-es* Γ (*pre1*,*rely1*)$]\!] \Longrightarrow$ *c*$\in$*assume-es* Γ (*pre*,*rely*)
proof −
assume *p0*: *pre1*$\subseteq$*pre*
and *p1*: *rely1*$\subseteq$*rely*
and *p3*: *c*$\in$*assume-es* Γ (*pre1*,*rely1*)
then have *a0*: *gets-es* (*c*!*0*) $\in$ *pre1* $\wedge$ ($\forall$ *i*. *Suc i*<*length c* $\longrightarrow$
$\Gamma \vdash$ *c*!*i* −*ese*$\rightarrow$ *c*!(*Suc i*) $\longrightarrow$ (*gets-es* (*c*!*i*), *gets-es* (*c*!*Suc i*)) $\in$ *rely1*)
by (*simp add*:*assume-es-def*)
show *?thesis*
proof(*simp add*:*assume-es-def*,*rule conjI*)
from *p0 a0* **show** *gets-es* (*c* ! *0*) $\in$ *pre* **by** *auto*
next
from *p1 a0* **show** $\forall$ *i*. *Suc i* < *length c* $\longrightarrow \Gamma \vdash$ *c* ! *i* −*ese*$\rightarrow$ *c* ! *Suc i*
$\longrightarrow$ (*gets-es* (*c* ! *i*), *gets-es* (*c* ! *Suc i*)) $\in$ *rely*
by *auto*
qed
qed

lemma *commit-es-imp*: $[\![$*guar1*$\subseteq$*guar*; *post1*$\subseteq$*post*; *c*$\in$*commit-es* Γ (*guar1*,*post1*)$]\!]$ $\Longrightarrow$ *c*$\in$*commit-es* Γ (*guar*,*post*)
proof −
assume *p0*: *guar1*$\subseteq$*guar*
and *p1*: *post1*$\subseteq$*post*
and *p3*: *c*$\in$*commit-es* Γ (*guar1*,*post1*)
then have *a0*: $\forall$ *i*. *Suc i*<*length c* $\longrightarrow$
($\exists$ *t*. $\Gamma \vdash$ *c*!*i* −*es*−*t*$\rightarrow$ *c*!(*Suc i*)) $\longrightarrow$ (*gets-es* (*c*!*i*), *gets-es* (*c*!*Suc i*)) $\in$ *guar1*
by (*simp add*:*commit-es-def*)
show *?thesis*
proof(*simp add*:*commit-es-def*)
from *p0 a0* **show** $\forall$ *i*. *Suc i* < *length c* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *c* ! *i* −*es*−*t*$\rightarrow$ *c* ! *Suc i*)
$\longrightarrow$ (*gets-es* (*c* ! *i*), *gets-es* (*c* ! *Suc i*)) $\in$ *guar*
by *auto*
qed
qed

lemma *assume-pes-imp*: $[\![$*pre1*$\subseteq$*pre*; *rely1*$\subseteq$*rely*; *c*$\in$*assume-pes* Γ (*pre1*,*rely1*)$]\!] \Longrightarrow$ *c*$\in$*assume-pes* Γ (*pre*,*rely*)
proof −
assume *p0*: *pre1*$\subseteq$*pre*
and *p1*: *rely1*$\subseteq$*rely*

and *p3*: $c \in assume\text{-}pes\ \Gamma\ (pre1, rely1)$
then have *a0*: $gets\ (c!0) \in pre1 \wedge (\forall i.\ Suc\ i < length\ c \longrightarrow$
$\Gamma \vdash c!i\ -pese\rightarrow\ c!(Suc\ i) \longrightarrow (gets\ (c!i),\ gets\ (c!Suc\ i)) \in rely1)$
by (*simp add*:*assume-pes-def*)
show *?thesis*
proof(*simp add*:*assume-pes-def*,*rule conjI*)
from *p0 a0* **show** $gets\ (c\ !\ 0) \in pre$ **by** *auto*
next
from *p1 a0* **show** $\forall i.\ Suc\ i < length\ c \longrightarrow \Gamma \vdash c\ !\ i\ -pese\rightarrow\ c\ !\ Suc\ i$
$\longrightarrow (gets\ (c\ !\ i),\ gets\ (c\ !\ Suc\ i)) \in rely$
by *auto*
qed
qed

lemma *commit-pes-imp*: $[\![guar1 \subseteq guar;\ post1 \subseteq post;\ c \in commit\text{-}pes\ \Gamma\ (guar1, post1)]\!]$
$\Longrightarrow c \in commit\text{-}pes\ \Gamma\ (guar, post)$
proof −
assume *p0*: $guar1 \subseteq guar$
and *p1*: $post1 \subseteq post$
and *p3*: $c \in commit\text{-}pes\ \Gamma\ (guar1, post1)$
then have *a0*: $\forall i.\ Suc\ i < length\ c \longrightarrow$
$(\exists t.\ \Gamma \vdash c!i\ -pes-t\rightarrow\ c!(Suc\ i)) \longrightarrow (gets\ (c!i),\ gets\ (c!Suc\ i)) \in$
$guar1$
by (*simp add*:*commit-pes-def*)
show *?thesis*
proof(*simp add*:*commit-pes-def*)
from *p0 a0* **show** $\forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ \Gamma \vdash c\ !\ i\ -pes-t\rightarrow\ c\ !$
$Suc\ i)$
$\longrightarrow (gets\ (c\ !\ i),\ gets\ (c\ !\ Suc\ i)) \in guar$
by *auto*
qed
qed

lemma *assume-pes-take-n*:
$[\![m > 0;\ m \leq length\ esl;\ esl \in assume\text{-}pes\ \Gamma\ (pre,\ rely)]\!]$
$\Longrightarrow take\ m\ esl \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
proof −
assume *p1*: $m > 0$
and *p2*: $m \leq length\ esl$
and *p3*: $esl \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
let $?esl1 = take\ m\ esl$
from *p3* **have** $gets\ (esl!0) \in pre$ **by** (*simp add*:*assume-pes-def*)
with *p1 p2 p3* **have** $gets\ (?esl1!0) \in pre$ **by** *simp*
moreover
have $\forall i.\ Suc\ i < length\ ?esl1 \longrightarrow$
$\Gamma \vdash ?esl1!i\ -pese\rightarrow\ ?esl1!(Suc\ i) \longrightarrow (gets\ (?esl1!i),\ gets\ (?esl1!Suc\ i))$
$\in rely$
proof −
{

```
      fix i
      assume a0: Suc i<length ?esl1
        and  a1: Γ ⊢ ?esl1!i −pese→ ?esl1!(Suc i)
     with p3 have (gets (esl!i), gets (esl!Suc i)) ∈ rely by (simp add:assume-pes-def)
      with p1 p2 a0 have (gets (?esl1!i), gets (?esl1!Suc i)) ∈ rely
        using Suc-lessD length-take min.absorb2 nth-take by auto
     }
     then show ?thesis by auto qed
   ultimately show ?thesis by (simp add:assume-pes-def)
 qed

lemma assume-pes-drop-n:
 ⟦m < length esl; esl∈assume-pes Γ (pre, rely); gets (esl!m) ∈ pre1⟧
      ⟹ drop m esl ∈ assume-pes Γ (pre1, rely)
 proof −
   assume p1: m < length esl
     and  p3: esl∈assume-pes Γ (pre, rely)
     and  p2: gets (esl!m) ∈ pre1
   let ?esl1 = drop m esl
   from p1 p2 p3 have gets (?esl1!0) ∈ pre1
     by (simp add: hd-conv-nth hd-drop-conv-nth not-less)
   moreover
   have ∀ i. Suc i<length ?esl1 ⟶
         Γ ⊢ ?esl1!i −pese→ ?esl1!(Suc i) ⟶ (gets (?esl1!i), gets (?esl1!Suc i))
∈ rely
     proof −
     {
      fix i
      assume a0: Suc i<length ?esl1
        and  a1: Γ ⊢ ?esl1!i −pese→ ?esl1!(Suc i)
      with p1 p3 have (gets (esl!(m+i)), gets (esl!Suc (m+i))) ∈ rely by (simp
add: assume-pes-def)
      with p1 p2 a0 have (gets (?esl1!i), gets (?esl1!Suc i)) ∈ rely
        using Suc-lessD length-take min.absorb2 nth-take by auto
     }
     then show ?thesis by auto qed
   ultimately show ?thesis by (simp add:assume-pes-def)
 qed

end

end
```

6 The Rely-guarantee Proof System and its Soundness of PiCore

```
theory PiCore-Hoare
imports PiCore-Validity
```

begin

declare [[*smt-timeout* = *300*]]

6.1 Proof System for Programs

declare *Un-subset-iff* [*simp del*] *sup.bounded-iff* [*simp del*]

definition *stable* :: $'a$ *set* $\Rightarrow$ $('a \times 'a)$ *set* $\Rightarrow$ *bool* **where**
stable $\equiv \lambda f\ g.\ (\forall\, x\ y.\ x \in f \longrightarrow (x,\ y) \in g \longrightarrow y \in f)$

lemma $Id = \{(s,\ t).\ s = t\}$
by *auto*

lemma *stable-id*: *stable P Id*
unfolding *stable-def Id-def* **by** *auto*

lemma *stable-id2*: *stable P* $\{(s,t).\ s = t\}$
unfolding *stable-def* **by** *auto*

lemma *stable-int2*: *stable s r* $\Longrightarrow$ *stable t r* $\Longrightarrow$ *stable* $(s \cap t)$ *r*
by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-def*)

lemma *stable-int3*: *stable k r* $\Longrightarrow$ *stable s r* $\Longrightarrow$ *stable t r* $\Longrightarrow$ *stable* $(k \cap s \cap t)$
r
by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-def*)

lemma *stable-un2*: *stable s r* $\Longrightarrow$ *stable t r* $\Longrightarrow$ *stable* $(s \cup t)$ *r*
by (*simp add*: *stable-def*)

6.2 Rely-guarantee Condition

record $'s$ *rgformula* =
pre-rgf :: $'s$ *set*
rely-rgf :: $('s \times 's)$ *set*
guar-rgf :: $('s \times 's)$ *set*
post-rgf :: $'s$ *set*

definition *getrgformula* ::
$'s$ *set* $\Rightarrow$ $('s \times 's)$ *set* $\Rightarrow$ $('s \times 's)$ *set* $\Rightarrow$ $'s$ *set* $\Rightarrow$ $'s$ *rgformula* (*RG*[-,-,-,-]
[*91*,*91*,*91*,*91*] *90*)
where *getrgformula pre r g pst* $\equiv$ (|*pre-rgf* = *pre*, *rely-rgf* = *r*, *guar-rgf* = *g*,
post-rgf = *pst*|)

definition Pre_f :: $'s$ *rgformula* $\Rightarrow$ $'s$ *set*
where Pre_f *rg* = *pre-rgf rg*

definition $Rely_f$ :: $'s$ *rgformula* $\Rightarrow$ $('s \times 's)$ *set*

where $Rely_f\ rg = rely\text{-}rgf\ rg$

definition $Guar_f :: 's\ rgformula \Rightarrow ('s \times 's)\ set$
where $Guar_f\ rg = guar\text{-}rgf\ rg$

definition $Post_f :: 's\ rgformula \Rightarrow 's\ set$
where $Post_f\ rg = post\text{-}rgf\ rg$

type-synonym $('l,'k,'s,'prog)\ rgformula\text{-}e = ('l,'k,'s,'prog)\ event \times 's\ rgformula$

definition $get\text{-}int\text{-}pre :: ('l,'k,'s,'prog)\ rgformula\text{-}e\ set \Rightarrow 's\ set$
where $get\text{-}int\text{-}pre\ S \equiv \{s.\ \forall f{\in}S.\ s \in Pre_f\ (snd\ f)\}$

definition $get\text{-}int\text{-}rely :: ('l,'k,'s,'prog)\ rgformula\text{-}e\ set \Rightarrow ('s \times 's)\ set$
where $get\text{-}int\text{-}rely\ S \equiv \{s.\ \forall f{\in}S.\ s \in Rely_f\ (snd\ f)\}$

definition $get\text{-}un\text{-}guar :: ('l,'k,'s,'prog)\ rgformula\text{-}e\ set \Rightarrow ('s \times 's)\ set$
where $get\text{-}un\text{-}guar\ S \equiv \{s.\ \exists f{\in}S.\ s \in Guar_f\ (snd\ f)\}$

definition $get\text{-}un\text{-}post :: ('l,'k,'s,'prog)\ rgformula\text{-}e\ set \Rightarrow 's\ set$
where $get\text{-}un\text{-}post\ S \equiv \{s.\ \exists f{\in}S.\ s \in Post_f\ (snd\ f)\}$

datatype $('l,'k,'s,'prog)\ rgformula\text{-}ess =$
$rgf\text{-}EvtSeq\ ('l,'k,'s,'prog)\ rgformula\text{-}e\ ('l,'k,'s,'prog)\ rgformula\text{-}ess \times 's\ rgformula$
$\mid rgf\text{-}EvtSys\ ('l,'k,'s,'prog)\ rgformula\text{-}e\ set$

type-synonym $('l,'k,'s,'prog)\ rgformula\text{-}es =$
$('l,'k,'s,'prog)\ rgformula\text{-}ess \times 's\ rgformula$

type-synonym $('l,'k,'s,'prog)\ rgformula\text{-}par =$
$'k \Rightarrow ('l,'k,'s,'prog)\ rgformula\text{-}es$

definition $E_e :: ('l,'k,'s,'prog)\ rgformula\text{-}e \Rightarrow ('l,'k,'s,'prog)\ event$
where $E_e\ rg = fst\ rg$

definition $Pre_e :: ('l,'k,'s,'prog)\ rgformula\text{-}e \Rightarrow 's\ set$
where $Pre_e\ rg = pre\text{-}rgf\ (snd\ rg)$

definition $Rely_e :: ('l,'k,'s,'prog)\ rgformula\text{-}e \Rightarrow ('s \times 's)\ set$
where $Rely_e\ rg = rely\text{-}rgf\ (snd\ rg)$

definition $Guar_e :: ('l,'k,'s,'prog)\ rgformula\text{-}e \Rightarrow ('s \times 's)\ set$
where $Guar_e\ rg = guar\text{-}rgf\ (snd\ rg)$

definition $Post_e$ :: $('l,'k,'s,'prog)$ *rgformula-e* $\Rightarrow$ *'s set*
where $Post_e$ $rg = post\text{-}rgf\ (snd\ rg)$

definition Pre_{es} :: $('l,'k,'s,'prog)$ *rgformula-es* $\Rightarrow$ *'s set*
where Pre_{es} $rg = pre\text{-}rgf\ (snd\ rg)$

definition $Rely_{es}$ :: $('l,'k,'s,'prog)$ *rgformula-es* $\Rightarrow$ $('s \times 's)$ *set*
where $Rely_{es}$ $rg = rely\text{-}rgf\ (snd\ rg)$

definition $Guar_{es}$ :: $('l,'k,'s,'prog)$ *rgformula-es* $\Rightarrow$ $('s \times 's)$ *set*
where $Guar_{es}$ $rg = guar\text{-}rgf\ (snd\ rg)$

definition $Post_{es}$ :: $('l,'k,'s,'prog)$ *rgformula-es* $\Rightarrow$ *'s set*
where $Post_{es}$ $rg = post\text{-}rgf\ (snd\ rg)$

fun *evtsys-spec* :: $('l,'k,'s,'prog)$ *rgformula-ess* $\Rightarrow$ $('l,'k,'s,'prog)$ *esys* **where**
evtsys-spec-evtseq: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *EvtSeq* (E_e *ef*) (*evtsys-spec* (*fst esf*)) |
evtsys-spec-evtsys: *evtsys-spec* (*rgf-EvtSys esf*) = *EvtSys* (*Domain esf*)

definition *paresys-spec* :: $('l,'k,'s,'prog)$ *rgformula-par* $\Rightarrow$ $('l,'k,'s,'prog)$ *paresys*
where *paresys-spec pesf* $\equiv \lambda k.$ *evtsys-spec* (*fst* (*pesf k*))

locale *event-hoare* = *event-validity ptran petran fin-com cpts-p cpts-of-p prog-validity assume-p commit-p*
for *ptran* :: *'Env* $\Rightarrow$ $(('prog \times 's) \times 'prog \times 's)$ *set*
and *petran* :: *'Env* $\Rightarrow$ $('s,'prog)$ *pconf* $\Rightarrow$ $('s,'prog)$ *pconf* $\Rightarrow$ *bool* (- $\vdash$ - $-pe\rightarrow$ - [*81*,*81*,*81*] *80*)
and *fin-com* :: *'prog*
and *cpts-p* :: *'Env* $\Rightarrow$ $('s,'prog)$ *pconfs set*
and *cpts-of-p* :: *'Env* $\Rightarrow$ *'prog* $\Rightarrow$ *'s* $\Rightarrow$ $(('s,'prog)\ pconfs)$ *set*
and *prog-validity* :: *'Env* $\Rightarrow$ *'prog* $\Rightarrow$ *'s set* $\Rightarrow$ $('s \times 's)$ *set* $\Rightarrow$ $('s \times 's)$ *set* $\Rightarrow$ *'s set* $\Rightarrow$ *bool*
(- $\models$ - sat_p [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)
and *assume-p* :: *'Env* $\Rightarrow$ $('s\ set \times ('s \times 's)\ set) \Rightarrow (('s,'prog)\ pconfs)$ *set*
and *commit-p* :: *'Env* $\Rightarrow$ $(('s \times 's)\ set \times 's\ set) \Rightarrow (('s,'prog)\ pconfs)$ *set*
+
fixes *rghoare-p* :: *'Env* $\Rightarrow$ [*'prog*, *'s set*, $('s \times 's)$ *set*, $('s \times 's)$ *set*, *'s set*] $\Rightarrow$ *bool*
(- $\vdash$ - sat_p [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)
assumes *rgsound-p*: $\Gamma \vdash P\ sat_p\ [pre, rely, guar, post] \longrightarrow \Gamma \models P\ sat_p\ [pre, rely, guar, post]$
begin

6.3 Proof System for Events

inductive *rghoare-e* :: *'Env* $\Rightarrow$ [$('l,'k,'s,'prog)$ *event*, *'s set*, $('s \times 's)$ *set*, $('s \times 's)$ *set*, *'s set*] $\Rightarrow$ *bool*
(- $\vdash$ - sat_e [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)

where
AnonyEvt: $\Gamma \vdash P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow \Gamma \vdash AnonyEvent\ P\ sat_e\ [pre,\ rely,\ guar,\ post]$

| *BasicEvt*: $[\![\ \Gamma \vdash body\ ev\ sat_p\ [pre \cap (guard\ ev),\ rely,\ guar,\ post];$
$stable\ pre\ rely;\ \forall s.\ (s,\ s) \in guar]\!] \Longrightarrow \Gamma \vdash BasicEvent\ ev\ sat_e\ [pre,\ rely,\ guar,\ post]$

| *Evt-conseq*: $[\![\ pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\Gamma \vdash ev\ sat_e\ [pre',\ rely',\ guar',\ post']\ ]\!]$
$\Longrightarrow \Gamma \vdash ev\ sat_e\ [pre,\ rely,\ guar,\ post]$

definition *Evt-sat-RG*:: $'Env \Rightarrow ('l,'k,'s,'prog)\ event \Rightarrow 's\ rgformula \Rightarrow bool$ ((- -⊢-) [60,60,60] 61)
where *Evt-sat-RG* $\Gamma\ e\ rg \equiv \Gamma \vdash e\ sat_e\ [Pre_f\ rg,\ Rely_f\ rg,\ Guar_f\ rg,\ Post_f\ rg]$

6.4 Proof System for Event Systems

inductive *rghoare-es* :: $'Env \Rightarrow [('l,'k,'s,'prog)\ rgformula\text{-}ess,\ 's\ set,\ ('s \times 's)\ set,\ ('s \times 's)\ set,\ 's\ set] \Rightarrow bool$
$(- \vdash -\ sat_s\ [-,\ -,\ -,\ -]\ [60,60,0,0,0,0]\ 45)$
for Γ :: $'Env$
where
EvtSeq-h: $[\![\ \Gamma \vdash E_e\ ef\ sat_e\ [Pre_e\ ef,\ Rely_e\ ef,\ Guar_e\ ef,\ Post_e\ ef];$
$\Gamma \vdash fst\ esf\ sat_s\ [Pre_f\ (snd\ esf),\ Rely_f\ (snd\ esf),\ Guar_f\ (snd\ esf),\ Post_f\ (snd\ esf)];$
$pre = Pre_e\ ef;\ post = Post_f\ (snd\ esf);$
$rely \subseteq Rely_e\ ef;\ rely \subseteq Rely_f\ (snd\ esf);$
$Guar_e\ ef \subseteq guar;\ Guar_f\ (snd\ esf) \subseteq guar;$
$Post_e\ ef \subseteq Pre_f\ (snd\ esf)]\!]$
$\Longrightarrow \Gamma \vdash (rgf\text{-}EvtSeq\ ef\ esf)\ sat_s\ [pre,\ rely,\ guar,\ post]$

| *EvtSys-h*: $[\![\forall ef \in esf.\ \Gamma \vdash E_e\ ef\ sat_e\ [Pre_e\ ef,\ Rely_e\ ef,\ Guar_e\ ef,\ Post_e\ ef];$
$\forall ef \in esf.\ pre \subseteq Pre_e\ ef;\ \ \forall ef \in esf.\ rely \subseteq Rely_e\ ef;$
$\forall ef \in esf.\ Guar_e\ ef \subseteq guar;\ \forall ef \in esf.\ Post_e\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1 \in esf \wedge ef2 \in esf \longrightarrow Post_e\ ef1 \subseteq Pre_e\ ef2;$
$stable\ pre\ rely;\ \forall s.\ (s,\ s) \in guar]\!]$
$\Longrightarrow \Gamma \vdash rgf\text{-}EvtSys\ esf\ sat_s\ [pre,\ rely,\ guar,\ post]$

| *EvtSys-conseq*: $[\![\ pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\Gamma \vdash esys\ sat_s\ [pre',\ rely',\ guar',\ post']\ ]\!]$
$\Longrightarrow \Gamma \vdash esys\ sat_s\ [pre,\ rely,\ guar,\ post]$

definition *Esys-sat-RG* :: $'Env \Rightarrow ('l,'k,'s,'prog)\ rgformula\text{-}ess \Rightarrow 's\ rgformula \Rightarrow bool$ $((- \dashv\vdash_{es}-)\ [60,60,60]\ 61)$
where *Esys-sat-RG* $\Gamma\ es\ rg \equiv \Gamma \vdash es\ sat_s\ [Pre_f\ rg,\ Rely_f\ rg,\ Guar_f\ rg,\ Post_f\ rg]$

6.5 Proof System for Parallel Event Systems

inductive *rghoare-pes* :: *'Env* ⇒ [(*'l*,*'k*,*'s*,*'prog*) *rgformula-par*, *'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
(- ⊢ - *SAT* [-, -, -, -] [*60*,*60*,*0*,*0*,*0*,*0*] *45*)
for Γ :: *'Env*
where
ParallelESys: ⟦∀ *k*. Γ ⊢ *fst* (*pesf k*) sat_s [Pre_{es} (*pesf k*), $Rely_{es}$ (*pesf k*), $Guar_{es}$ (*pesf k*), $Post_{es}$ (*pesf k*)];
∀ *k*. *pre* ⊆ Pre_{es} (*pesf k*);
∀ *k*. *rely* ⊆ $Rely_{es}$ (*pesf k*);
∀ *k j*. *j*≠*k* ⟶ $Guar_{es}$ (*pesf j*) ⊆ $Rely_{es}$ (*pesf k*);
∀ *k*. $Guar_{es}$ (*pesf k*) ⊆ *guar*;
∀ *k*. $Post_{es}$ (*pesf k*) ⊆ *post*⟧
⟹ Γ ⊢ *pesf SAT* [*pre*, *rely*, *guar*, *post*]

| *ParallelESys-conseq*: ⟦ *pre* ⊆ *pre'*; *rely* ⊆ *rely'*; *guar'* ⊆ *guar*; *post'* ⊆ *post*;
Γ ⊢ *pesf SAT* [*pre'*, *rely'*, *guar'*, *post'*] ⟧
⟹ Γ ⊢ *pesf SAT* [*pre*, *rely*, *guar*, *post*]

lemma *es-sat-eq*: (Γ ⊢ *fst* (*pesf k*) sat_s [Pre_{es} (*pesf k*), $Rely_{es}$ (*pesf k*), $Guar_{es}$ (*pesf k*), $Post_{es}$ (*pesf k*)])
= Γ (*fst* (*pesf k*)) $\vdash_{es}$ (*snd* (*pesf k*))
by (*simp add*:*Esys-sat-RG-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*)

7 Soundness

7.1 Some previous lemmas

7.1.1 event

lemma *assume-e-imp*: ⟦*pre1*⊆*pre*; *rely1*⊆*rely*; *c*∈*assume-e* Γ (*pre1*,*rely1*)⟧ ⟹ *c*∈*assume-e* Γ (*pre*,*rely*)
proof −
assume *p0*: *pre1*⊆*pre*
and *p1*: *rely1*⊆*rely*
and *p3*: *c*∈*assume-e* Γ (*pre1*,*rely1*)
then have *a0*: *gets-e* (*c*!*0*) ∈ *pre1* ∧ (∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*ee*→ *c*!(*Suc i*) ⟶ (*gets-e* (*c*!*i*), *gets-e* (*c*!*Suc i*)) ∈ *rely1*)
by (*simp add*:*assume-e-def*)
show *?thesis*
proof(*simp add*:*assume-e-def*,*rule conjI*)
from *p0 a0* **show** *gets-e* (*c* ! *0*) ∈ *pre* **by** *auto*
next
from *p1 a0* **show** ∀ *i*. *Suc i* < *length c* ⟶ Γ ⊢ *c* ! *i* −*ee*→ *c* ! *Suc i*
⟶ (*gets-e* (*c* ! *i*), *gets-e* (*c* ! *Suc i*)) ∈ *rely*
by *auto*
qed
qed

```
lemma commit-e-imp: ⟦guar1⊆guar; post1⊆post; c∈commit-e Γ (guar1,post1)⟧
⟹ c∈commit-e Γ (guar,post)
  proof –
    assume p0: guar1⊆guar
      and  p1: post1⊆post
      and  p3: c∈commit-e Γ (guar1,post1)
    then have a0: (∀ i. Suc i<length c ⟶
             (∃ t. Γ ⊢ c!i −et−t→ c!(Suc i)) ⟶ (gets-e (c!i), gets-e (c!Suc i)) ∈
guar1) ∧
             (getspc-e (last c) = AnonyEvent fin-com ⟶ gets-e (last c) ∈ post1)
      by (simp add:commit-e-def)
    show ?thesis
      proof(simp add:commit-e-def)
        from p0 p1 a0 show (∀ i. Suc i < length c ⟶ (∃ t. Γ ⊢ c ! i −et−t→ c !
Suc i)
                          ⟶ (gets-e (c ! i), gets-e (c ! Suc i)) ∈ guar) ∧
             (getspc-e (last c) = AnonyEvent fin-com ⟶ gets-e (last c) ∈ post)
          by auto
      qed
  qed
```

7.1.2 event system

```
lemma concat-i-lm[rule-format]: ∀ ls l. concat ls = l ∧ (∀ i<length ls. ls!i ≠ [])⟶
(∀ i. Suc i < length ls ⟶
                    (∃ m n. m ≤ length l ∧ n ≤ length l ∧ m ≤ n ∧ ls!i@[(ls!Suc
i)!0] = take (n − m) (drop m l)))
  proof –
  {
    fix ls
     have ∀ l. concat ls = l ∧ (∀ i<length ls. ls!i ≠ [])⟶ (∀ i. Suc i < length ls
⟶
                    (∃ m n. m ≤ length l ∧ n ≤ length l ∧ m ≤ n ∧ ls!i@[(ls!Suc
i)!0] = take (n − m) (drop m l)))
    proof(induct ls)
      case Nil show ?case by simp
    next
      case (Cons x xs)
      assume a0: ∀ l. concat xs = l ∧ (∀ i<length xs. xs ! i ≠ []) ⟶
                    (∀ i. Suc i < length xs ⟶ (∃ m n. m ≤ length l ∧ n ≤ length
l ∧
                          m ≤ n ∧ xs ! i @ [xs ! Suc i ! 0] = take (n − m) (drop
m l)))
      show ?case
        proof –
        {
          fix l
          assume b0: concat (x # xs) = l
```

and *b1*: $\forall$ *i*<*length* (*x* # *xs*). (*x* # *xs*) ! *i* $\neq$ []
let *?l'* = *concat xs*
from *b0* **have** *b2*: *l* = *x*@*?l'* **by** *simp*
have $\forall$ *i*. *Suc i* < *length* (*x* # *xs*) $\longrightarrow$ ($\exists$ *m n*. *m* $\leq$ *length l* $\wedge$ *n* $\leq$ *length l* $\wedge$
m $\leq$ *n* $\wedge$ (*x* # *xs*) ! *i* @ [(*x* # *xs*) ! *Suc i* ! *0*] = *take* (*n* − *m*) (*drop m l*))
proof −
{
fix *i*
assume *c0*: *Suc i* < *length* (*x* # *xs*)
then have *c1*: *length xs* > *0* **by** *auto*
have $\exists$ *m n*. *m* $\leq$ *length l* $\wedge$ *n* $\leq$ *length l* $\wedge$ *m* $\leq$ *n* $\wedge$
(*x* # *xs*) ! *i* @ [(*x* # *xs*) ! *Suc i* ! *0*] = *take* (*n* − *m*) (*drop m l*)
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
from *b1 c1* **have** *d1*: (*x* # *xs*) ! *1* $\neq$ [] **by** (*metis One-nat-def c0 d0*)
with *b0* **have** *d2*: *x* @ [*xs*!*0* ! *0*] = *take* (*length x* + *1*) (*drop 0 l*)
by (*smt Cons-nth-drop-Suc Nil-is-append-conv One-nat-def append-eq-conv-conj*
c0 concat.simps(*2*) *d0 drop-0 drop-Suc-Cons length-greater-0-conv*
nth-Cons-Suc nth-append self-append-conv2 take-0 take-Suc-conv-app-nth take-add)
then have *d3*: (*x* # *xs*) ! *0* @ [(*x* # *xs*) ! *1* ! *0*] = *take* (*length x* + *1*) (*drop 0 l*)
by *simp*
moreover
have *0* $\leq$ *length l* **using** *calculation* **by** *auto*
moreover
from *b0 d1* **have** *length x* + *1* $\leq$ *length l*
by (*metis Suc-eq-plus1 d2 drop-0 length-append-singleton linear take-all*)
ultimately show *?thesis* **using** *d0* **by** *force*
next
assume *d0*: *i* $\neq$ *0*
moreover
from *b1* **have** *d1*: $\forall$ *i*<*length xs*. *xs* ! *i* $\neq$ [] **by** *auto*
moreover
from *c0* **have** *Suc* (*i* − *1*) < *length xs* **using** *d0* **by** *auto*
ultimately have $\exists$ *m n*. *m* $\leq$ *length ?l'* $\wedge$ *n* $\leq$ *length ?l'* $\wedge$
m $\leq$ *n* $\wedge$ *xs* ! (*i* − *1*) @ [*xs* ! *Suc* (*i* − *1*) ! *0*] = *take* (*n* − *m*) (*drop m ?l'*)
using *a0 d0* **by** *blast*
then obtain *m* **and** *n* **where** *d2*: *m* $\leq$ *length ?l'* $\wedge$ *n* $\leq$ *length ?l'* $\wedge$
m $\leq$ *n* $\wedge$ *xs* ! (*i* − *1*) @ [*xs* ! *Suc* (*i* − *1*) ! *0*] = *take* (*n* − *m*) (*drop m ?l'*)

by *auto*
let $?m' = m + length\ x$
let $?n' = n + length\ x$
from *b0 d2* **have** $?m' \leq length\ l$ **by** *auto*
moreover
from *b0 d2* **have** $?n' \leq length\ l$ **by** *auto*
moreover
from *d2* **have** $?m' \leq ?n'$ **by** *auto*
moreover
have $(x \# xs)\ !\ i\ @\ [(x \# xs)\ !\ Suc\ i\ !\ 0] = take\ (?n' - ?m')\ (drop\ ?m'\ l)$
using *b2 d0 d2* **by** *auto*
ultimately have $?m' \leq length\ l \wedge ?n' \leq length\ l \wedge ?m' \leq ?n' \wedge$
$(x \# xs)\ !\ i\ @\ [(x \# xs)\ !\ Suc\ i\ !\ 0] = take\ (?n' - ?m')\ (drop\ ?m'\ l)$ **by** *simp*
then show *?thesis* **by** *blast*
qed
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *concat-last-lm*: $\forall\, ls\ l.\ concat\ ls = l \wedge length\ ls > 0 \longrightarrow$
$(\exists\, m\ .\ m \leq length\ l \wedge last\ ls = drop\ m\ l)$
proof
fix *ls*
show $\forall\, l.\ concat\ ls = l \wedge length\ ls > 0 \longrightarrow$
$(\exists\, m\ .\ m \leq length\ l \wedge last\ ls = drop\ m\ l)$
proof(*induct ls*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons x xs*)
assume *a0*: $\forall\, l.\ concat\ xs = l \wedge 0 < length\ xs \longrightarrow (\exists\, m \leq length\ l.\ last\ xs = drop\ m\ l)$
show *?case*
proof –
{
fix *l*
assume *b0*: $concat\ (x \# xs) = l$
and *b1*: $0 < length\ (x \# xs)$
let $?l' = concat\ xs$
have $\exists\, m \leq length\ l.\ last\ (x \# xs) = drop\ m\ l$
proof(*cases* $xs = []$)

```
          assume c0: xs = []
          then show ?thesis using b0 by auto
        next
          assume c0: xs ≠ []
          then have c1: length xs > 0 by auto
          with a0 have ∃ m≤length ?l'. last xs = drop m ?l' by auto
          then obtain m where c2: m≤length ?l' ∧ last xs = drop m ?l' by
auto
          with b0 show ?thesis
            by (metis append-eq-conv-conj c0 concat.simps(2)
                drop-all drop-drop last.simps nat-le-linear)
        qed
      }
      then show ?thesis by auto
      qed
    qed
  qed

lemma concat-equiv: ⟦l ≠ []; l = concat lt; ∀ i<length lt. length (lt!i) ≥ 2⟧ ⟹
      ∀ i. i ≤ length l ⟶ (∃ k j. k < length lt ∧ j ≤ length (lt!k) ∧
            drop i l = (drop j (lt!k)) @ concat (drop (Suc k) lt) )
  proof -
    assume p0: l = concat lt
      and  p1: ∀ i<length lt. length (lt!i) ≥ 2
      and  p3: l ≠ []
    then have p4: lt ≠ [] using concat.simps(1) by blast
    show ?thesis
      proof -
      {
        fix i
        assume a0: i ≤ length l
        from a0 have ∃ k j. k < length lt ∧ j ≤ length (lt!k) ∧
                drop i l = (drop j (lt!k)) @ concat (drop (Suc k) lt)
          proof(induct i)
            case 0
            assume b0: 0 ≤ length l
            have drop 0 l = drop 0 (lt ! 0) @ concat (drop (Suc 0) lt)
            by (metis concat.simps(2) drop-0 drop-Suc-Cons list.exhaust nth-Cons-0
p0 p4)
            then show ?case using p4 by blast
          next
            case (Suc m)
           assume b0: m ≤ length l ⟹ ∃ k j. k < length lt ∧ j ≤ length (lt ! k) ∧
                      drop m l = drop j (lt ! k) @ concat (drop (Suc k) lt)
              and  b1: Suc m ≤ length l
            then have ∃ k j. k < length lt ∧ j ≤ length (lt ! k) ∧
                      drop m l = drop j (lt ! k) @ concat (drop (Suc k) lt)
              by auto
            then obtain k and j where b2: k < length lt ∧ j ≤ length (lt ! k) ∧
```

drop m l = *drop j* (*lt* ! *k*) @ *concat* (*drop* (*Suc k*) *lt*) **by** *auto*
show *?case*
proof(*cases j* = *length* (*lt*!*k*))
assume *c0*: *j* = *length* (*lt*!*k*)
with *b2* **have** *c1*: *drop m l* = *concat* (*drop* (*Suc k*) *lt*) **by** *simp*
from *b1* **have** *drop m l* ≠ [] **by** *simp*
with *c1* **have** *c2*: *drop* (*Suc k*) *lt* ≠ [] **by** *auto*
then obtain *lt1* **and** *lts* **where** *c3*: *drop* (*Suc k*) *lt* = *lt1*#*lts*
by (*meson neq-Nil-conv*)
then have *c4*: *drop* (*Suc* (*Suc k*)) *lt* = *lts* **by** (*metis drop-Suc list.sel*(*3*) *tl-drop*)
moreover
from *c3* **have** *c5*: *lt*!*Suc k* = *lt1* **by** (*simp add*: *nth-via-drop*)
ultimately have *drop* (*Suc m*) *l* = *drop 1 lt1* @ *concat lts* **using** *c1 c3*
by (*metis One-nat-def Suc-leI Suc-lessI b2 concat.simps*(*2*)
drop-0 drop-Suc drop-all list.distinct(*1*) *list.size*(*3*)
not-less-eq-eq numeral-2-eq-2 p1 tl-append2 tl-drop zero-less-Suc)
with *c4 c5* **have** *drop* (*Suc m*) *l* = *drop 1* (*lt*!*Suc k*) @ *concat* (*drop* (*Suc* (*Suc k*)) *lt*) **by** *simp*
then show *?thesis* **by** (*metis One-nat-def Suc-leD Suc-leI Suc-lessI c2 b2 drop-all numeral-2-eq-2 p1*)
next
assume *c0*: *j* ≠ *length* (*lt*!*k*)
with *b2* **have** *c1*: *j* < *length* (*lt*!*k*) **by** *auto*
with *b2* **have** *drop* (*Suc m*) *l* = *drop* (*Suc j*) (*lt* ! *k*) @ *concat* (*drop* (*Suc k*) *lt*)
by (*metis c0 drop-Suc drop-eq-Nil le-antisym tl-append2 tl-drop*)
then show *?thesis* **using** *Suc-leI c1 b2* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *rely-take-rely*: ∀ *i*. *Suc i*<*length l* ⟶ Γ ⊢ *l*!*i* −*ese*→ *l*!(*Suc i*)
⟶ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) ∈ *rely* ⟹
∀ *m subl*. *m* ≤ *length l* ∧ *subl* = *take m l* ⟶ (∀ *i*. *Suc i*<*length subl* ⟶ Γ ⊢ *subl*!*i* −*ese*→ *subl*!(*Suc i*)
⟶ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) ∈ *rely*)
proof −
assume *p0*: ∀ *i*. *Suc i*<*length l* ⟶ Γ ⊢ *l*!*i* −*ese*→ *l*!(*Suc i*)
⟶ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) ∈ *rely*
show *?thesis*
proof −
{
fix *m*
have ∀ *subl*. *m* ≤ *length l* ∧ *subl* = *take m l* ⟶ (∀ *i*. *Suc i*<*length subl* ⟶

$\Gamma \vdash$ *subl*!*i* $-ese\rightarrow$ *subl*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) $\in$ *rely*)
proof(*induct m*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
assume *a0*: $\forall$ *subl*. $n \leq$ *length l* $\wedge$ *subl* = *take n l* $\longrightarrow$
($\forall$ *i*. *Suc i* < *length subl* $\longrightarrow \Gamma \vdash$ *subl* ! *i* $-ese\rightarrow$ *subl* ! *Suc i*
$\longrightarrow$
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) $\in$ *rely*)
show *?case*
proof –
{
fix *subl*
assume *b0*: *Suc n* $\leq$ *length l*
and *b1*: *subl* = *take* (*Suc n*) *l*
with *a0* **have** $\forall$ *i*. *Suc i* < *length subl* $\longrightarrow \Gamma \vdash$ *subl* ! *i* $-ese\rightarrow$ *subl* !
Suc i $\longrightarrow$
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) $\in$ *rely*
using *p0* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *rely-drop-rely*: $\forall$ *i*. *Suc i*<*length l* $\longrightarrow \Gamma \vdash$ *l*!*i* $-ese\rightarrow$ *l*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) $\in$ *rely* $\Longrightarrow$
$\forall$ *m subl*. $m \leq$ *length l* $\wedge$ *subl* = *drop m l* $\longrightarrow$ ($\forall$ *i*. *Suc i*<*length subl* $\longrightarrow$
$\Gamma \vdash$ *subl*!*i* $-ese\rightarrow$ *subl*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) $\in$ *rely*)
proof –
assume *p0*: $\forall$ *i*. *Suc i*<*length l* $\longrightarrow \Gamma \vdash$ *l*!*i* $-ese\rightarrow$ *l*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) $\in$ *rely*
show *?thesis*
proof –
{
fix *m*
have $\forall$ *subl*. $m \leq$ *length l* $\wedge$ *subl* = *drop m l* $\longrightarrow$ ($\forall$ *i*. *Suc i*<*length subl*
$\longrightarrow \Gamma \vdash$ *subl*!*i* $-ese\rightarrow$ *subl*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) $\in$ *rely*)
proof(*induct m*)
case *0* **show** *?case* **by** (*simp add*: *p0*)
next
case (*Suc n*)
assume *a0*: $\forall$ *subl*. $n \leq$ *length l* $\wedge$ *subl* = *drop n l* $\longrightarrow$

$(\forall i.\ Suc\ i < length\ subl \longrightarrow \Gamma \vdash subl\ !\ i\ -ese\rightarrow subl\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (subl\ !\ i),\ gets\text{-}es\ (subl\ !\ Suc\ i)) \in rely)$
show *?case*
proof −
{
fix *subl*
assume *b0*: $Suc\ n \leq length\ l$
and *b1*: $subl = drop\ (Suc\ n)\ l$
with *a0* **have** $\forall i.\ Suc\ i < length\ subl \longrightarrow \Gamma \vdash subl\ !\ i\ -ese\rightarrow subl\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (subl\ !\ i),\ gets\text{-}es\ (subl\ !\ Suc\ i)) \in rely$
using *p0* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *rely-takedrop-rely*: $[\![\forall i.\ Suc\ i<length\ l \longrightarrow \Gamma \vdash l!i\ -ese\rightarrow l!(Suc\ i)$
$\longrightarrow (gets\text{-}es\ (l!i),\ gets\text{-}es\ (l!Suc\ i)) \in rely;$
$\exists m\ n.\ m \leq length\ l \wedge n \leq length\ l \wedge m \leq n \wedge subl = take\ (n - m)\ (drop\ m\ l)]\!] \Longrightarrow$
$\forall i.\ Suc\ i<length\ subl \longrightarrow \Gamma \vdash subl!i\ -ese\rightarrow subl!(Suc\ i)$
$\longrightarrow (gets\text{-}es\ (subl!i),\ gets\text{-}es\ (subl!Suc\ i)) \in rely$
proof −
assume *p1*: $\forall i.\ Suc\ i<length\ l \longrightarrow \Gamma \vdash l!i\ -ese\rightarrow l!(Suc\ i)$
$\longrightarrow (gets\text{-}es\ (l!i),\ gets\text{-}es\ (l!Suc\ i)) \in rely$
and *p3*: $\exists m\ n.\ m \leq length\ l \wedge n \leq length\ l \wedge m \leq n \wedge subl = take\ (n - m)\ (drop\ m\ l)$

from *p3* **obtain** *m* **and** *n* **where** *a0*: $m \leq length\ l \wedge n \leq length\ l \wedge m \leq n \wedge subl = take\ (n - m)\ (drop\ m\ l)$
by *auto*
let $?subl1 = drop\ m\ l$
have *a1*: $\forall i.\ Suc\ i<length\ ?subl1 \longrightarrow \Gamma \vdash ?subl1!i\ -ese\rightarrow ?subl1!(Suc\ i)$
$\longrightarrow (gets\text{-}es\ (?subl1!i),\ gets\text{-}es\ (?subl1!Suc\ i)) \in rely$
using *a0 p1 rely-drop-rely* **by** *blast*
show *?thesis* **using** *a0 a1* **by** *simp*
qed

lemma *pre-trans*: $[\![esl \in assume\text{-}es\ \Gamma\ (pre,\ rely);\ \forall i<length\ esl.\ getspc\text{-}es\ (esl!i) = es;\ stable\ pre\ rely]\!]$
$\Longrightarrow \forall i<length\ esl.\ gets\text{-}es\ (esl!i) \in pre$
proof −

```
    assume p0: esl ∈ assume-es Γ (pre, rely)
      and  p2: ∀ i<length esl. getspc-es (esl!i) = es
      and  p3: stable pre rely
    then show ?thesis
      proof -
      {
        fix i
        assume a0: i<length esl
        then have gets-es (esl!i) ∈ pre
          proof(induct i)
            case 0 from p0 show ?case by (simp add:assume-es-def)
          next
            case (Suc j)
            assume b0: j < length esl ⟹ gets-es (esl ! j) ∈ pre
              and  b1: Suc j < length esl
            then have b2: gets-es (esl ! j) ∈ pre by auto

            from p2 b1 have getspc-es (esl ! j) = es by auto
            moreover
            from p2 b1 have getspc-es (esl ! Suc j) = es by auto
                ultimately have Γ ⊢ esl ! j −ese→ esl ! Suc j by (simp add:
eqconf-esetran)
             with p0 b1 have (gets-es (esl!j), gets-es (esl!Suc j)) ∈ rely by (simp
add:assume-es-def)
            with p3 b2 show ?case by (simp add:stable-def)
          qed
      }
      then show ?thesis by auto
      qed
  qed

lemma pre-trans-assume-es:
  ⟦esl ∈ assume-es Γ (pre, rely); n < length esl;
    ∀ j. j ≤ n ⟶ getspc-es (esl ! j) = es; stable pre rely⟧
        ⟹ drop n esl ∈ assume-es Γ (pre, rely)
  proof -
    assume p0: esl ∈ assume-es Γ (pre, rely)
      and  p2: ∀ j. j ≤ n ⟶ getspc-es (esl ! j) = es
      and  p3: stable pre rely
      and  p4: n < length esl
    then show ?thesis
      proof(cases n = 0)
        assume n = 0 with p0 show ?thesis by auto
      next
        assume n ≠ 0
        then have a0: n > 0 by simp
        let ?esl = drop n esl
        let ?esl1 = take (Suc n) esl
        from p0 a0 p4 have ?esl1∈assume-es Γ (pre, rely)
```

    using *assume-es-take-n*[*of Suc n esl* Γ *pre rely*] **by** *simp*
  moreover
  from *p2 a0* **have** ∀ *i*<*length ?esl1*. *getspc-es* (*?esl1* ! *i*) = *es* **by** *simp*
  ultimately
  have ∀ *i*<*length ?esl1*. *gets-es* (*?esl1*!*i*) ∈ *pre*
    using *pre-trans*[*of take* (*Suc n*) *esl* Γ *pre rely es*] *p3* **by** *simp*
  with *a0 p4* **have** *gets-es* (*?esl*!*0*)∈*pre*
    using *Cons-nth-drop-Suc Suc-leI length-take lessI less-or-eq-imp-le min.absorb2 nth-Cons-0 nth-append-length take-Suc-conv-app-nth* **by** *auto*
  moreover
  have ∀ *i*. *Suc i*<*length ?esl* ⟶
      Γ ⊢ *?esl*!*i* −*ese*→ *?esl*!(*Suc i*) ⟶ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *rely*
    proof −
    {
      fix *i*
      assume *b0*: *Suc i*<*length ?esl*
        and *b1*: Γ ⊢ *?esl*!*i* −*ese*→ *?esl*!(*Suc i*)
      from *p0* **have** ∀ *i*. *Suc i*<*length esl* ⟶
        Γ ⊢ *esl*!*i* −*ese*→ *esl*!(*Suc i*) ⟶ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*
          by (*simp add*:*assume-es-def*)
      with *p4 a0 b0 b1* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *rely*
        using *less-imp-le-nat rely-drop-rely* **by** *auto*
    }
    then show *?thesis* **by** *auto*
    qed
  ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
 qed
qed

7.1.3 parallel event system

7.2 State trace equivalence

7.2.1 trace equivalence of program and anonymous event

primrec *lower-anonyevt0* :: (′*l*,′*k*,′*s*,′*prog*) *event* ⇒ ′*s* ⇒ (′*s*,′*prog*) *pconf*
  where *AnonyEv*: *lower-anonyevt0* (*AnonyEvent p*) *s* = (*p*, *s*) |
    BasicEv: *lower-anonyevt0* (*BasicEvent p*) *s* = (*fin-com*, *s*)

definition *lower-anonyevt1* :: (′*l*,′*k*,′*s*,′*prog*) *econf* ⇒ (′*s*,′*prog*) *pconf*
  where *lower-anonyevt1 ec* ≡ *lower-anonyevt0* (*getspc-e ec*) (*gets-e ec*)

definition *lower-evts* :: (′*l*,′*k*,′*s*,′*prog*) *econfs* ⇒ ((′*s*,′*prog*) *pconfs*)
  where *lower-evts ecfs* ≡ *map lower-anonyevt1 ecfs*

lemma *lower-anonyevt-s* : *getspc-e e* = *AnonyEvent P* ⟹ *gets-p* (*lower-anonyevt1 e*) = *gets-e e*
  by (*simp add*: *gets-p-def lower-anonyevt1-def*)

lemma *lower-evts-same-len*: *ps* = *lower-evts es* ⟹ *length ps* = *length es*
apply(*induct ps*) **by**(*simp add*:*lower-evts-def lower-anonyevt1-def*)+

lemma *lower-evts-same-s*: *ps* = *lower-evts* (*es*::('*l*,'*k*,'*s*,'*prog*) *econfs*) ⟹ ∀ *i*<*length ps*. *gets-p* (*ps*!*i*) = *gets-e* (*es*!*i*)
proof(*induct ps arbitrary*:*es*)
  case *Nil*
  then show *?case* **by**(*simp add*:*lower-evts-def lower-anonyevt1-def*)
next
  case (*Cons a ps*)
  assume *p*: (⋀*es*. *ps* = *lower-evts* (*es*::('*l*,'*k*,'*s*,'*prog*) *econfs*) ⟹ ∀ *i*<*length ps*. *gets-p* (*ps* ! *i*) = *gets-e* (*es* ! *i*))
    and *p1*: *a* # *ps* = *lower-evts es*
  {
    fix *i*
    assume *i*: *i*<*length* (*a* # *ps*)
    then have *gets-p* ((*a* # *ps*) ! *i*) = *gets-e* (*es* ! *i*)
    proof(*induct i*)
      case *0*
        then show *?case* **apply** (*simp add*:*gets-p-def gets-e-def*) **using** *p1* **apply**(*case-tac getspc-e* (*es*!*0*))
        apply (*simp add*:*lower-evts-def lower-anonyevt1-def getspc-e-def*)
      apply (*metis AnonyEv gets-e-def getspc-e-def lower-anonyevt1-def map-eq-Cons-D nth-Cons-0 sndI*)
        apply (*simp add*:*lower-evts-def lower-anonyevt1-def getspc-e-def*)
      by (*metis BasicEv gets-e-def getspc-e-def lower-anonyevt1-def map-eq-Cons-D nth-Cons-0 sndI*)
    next
      case (*Suc j*)
      assume *a0*: *Suc j* < *length* (*a* # *ps*)
      hence *a1*: *j* < *length ps* **by** *auto*
        from *p1* **have** *ps* = *lower-evts* (*tl es*) **apply** (*simp add*:*lower-evts-def lower-anonyevt1-def*) **by** *auto*
      moreover
      have *gets-p* ((*a* # *ps*) ! *Suc j*) = *gets-p* (*ps* ! *j*) **by**(*simp add*: *gets-p-def*)
      moreover
    from *p1* **have** *gets-e* (*es* ! *Suc j*) = *gets-e* (*tl es* ! *j*) **using** *lower-evts-same-len*[*of a* # *ps es*] **apply**(*simp add*: *gets-e-def*)
        by (*metis length-0-conv list.simps*(*3*) *local.nth-tl nth-Cons-Suc*)
      ultimately show *?case*
        using *lower-evts-same-len*[*of ps tl es*] *p*[*rule-format*,*of tl es j*] *a1* **by** *auto*
    qed

  }
  then show *?case* **by** *auto*
qed

lemma *equiv-lower-evts0* : $[\![\exists P.\ getspc\text{-}e\ (es\ !\ 0) = AnonyEvent\ P;\ es \in cpts\text{-}ev\ \Gamma]\!] \Longrightarrow lower\text{-}evts\ es \in cpts\text{-}p\ \Gamma$
proof−
  assume *a0*: $es \in cpts\text{-}ev\ \Gamma$ **and** *a1*: $\exists P.\ getspc\text{-}e\ (es\ !\ 0) = AnonyEvent\ P$
  have $\forall es\ P.\ getspc\text{-}e\ (es\ !\ 0) = AnonyEvent\ P \land es \in cpts\text{-}ev\ \Gamma \longrightarrow lower\text{-}evts\ es \in cpts\text{-}p\ \Gamma$
   proof −
   {
    fix *es*
    assume *b0*: $\exists P.\ getspc\text{-}e\ (es\ !\ 0) = AnonyEvent\ P$ **and**
      b1: $es \in cpts\text{-}ev\ \Gamma$
    from *b1 b0* **have** $lower\text{-}evts\ es \in cpts\text{-}p\ \Gamma$
     proof(*induct es*)
      case $(CptsEvOne\ e'\ s'\ x')$
      assume *c0*: $\exists P.\ getspc\text{-}e\ ([(e',\ s',\ x')]\ !\ 0) = AnonyEvent\ P$
      then obtain P **where** $getspc\text{-}e\ ([(e',\ s',\ x')]\ !\ 0) = AnonyEvent\ P$ **by** *auto*
      then have *c1*: $e' = AnonyEvent\ P$ **by** (*simp add:getspc-e-def*)
      then have *c2*: $lower\text{-}anonyevt1\ (e',\ s',\ x') = (P,\ s')$
       by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
      then have *c2*: $lower\text{-}evts\ [(e',\ s',\ x')] = [(P,\ s')]$
       by (*simp add*: *lower-evts-def*)
      then show *?case* **by** (*simp add*: *CptsPOne*)
     next
      case $(CptsEvEnv\ e'\ t'\ x'\ xs'\ s'\ y')$
      assume *c0*: $(e',\ t',\ x')\ \#\ xs' \in cpts\text{-}ev\ \Gamma$ **and**
        c1: $\exists P.\ getspc\text{-}e\ (((e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P \Longrightarrow lower\text{-}evts\ ((e',\ t',\ x')\ \#\ xs') \in cpts\text{-}p\ \Gamma$ **and**
        c2: $\exists P.\ getspc\text{-}e\ (((e',\ s',\ y')\ \#\ (e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P$
      let $?ob = lower\text{-}evts\ ((e',\ s',\ y')\ \#\ (e',\ t',\ x')\ \#\ xs')$
      from *c2* **obtain** P **where** *c-*:$getspc\text{-}e\ (((e',\ s',\ y')\ \#\ (e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P$ **by** *auto*
      then have *c3*: $?ob\ !\ 0 = (P,\ s')$
       by (*simp add*: *lower-evts-def lower-anonyevt1-def lower-anonyevt0-def gets-e-def getspc-e-def*)

       from *c-* **have** *c5*: $(e',\ s',\ y') = (AnonyEvent\ P,\ s',\ y')$ **by** (*simp add:getspc-e-def*)
      then have *c4*: $e' = AnonyEvent\ P$ **by** *simp*
      with *c1* **have** *c6*: $lower\text{-}evts\ ((e',\ t',\ x')\ \#\ xs') \in cpts\text{-}p\ \Gamma$ **by** (*simp add:getspc-e-def*)
      from *c5* **have** *c7*: $?ob = (P,\ s')\ \#\ lower\text{-}evts\ ((e',\ t',\ x')\ \#\ xs')$
      by (*metis* (*no-types*, *lifting*) *c3 list.simps*(*9*) *lower-evts-def nth-Cons-0*)

      from *c4* **have** *c8*: $lower\text{-}evts\ ((e',\ t',\ x')\ \#\ xs') = (P,\ t')\ \#\ lower\text{-}evts\ xs'$
       by (*simp add:lower-evts-def lower-anonyevt1-def lower-anonyevt0-def gets-e-def getspc-e-def*)

with *c6 c7* **show** *?case* **by** (*simp add*: *CptsPEnv*)
next
case (*CptsEvComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *c0*: $\Gamma \vdash$ (*e1*, *s1*, *x1*) $-et-et\rightarrow$ (*e2*, *t1*, *y1*) **and**
c1: (*e2*, *t1*, *y1*) # *xs1* $\in$ *cpts-ev* Γ **and**
c2: $\exists$ *P*. *getspc-e* (((*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P*
$\Longrightarrow$ *lower-evts* ((*e2*, *t1*, *y1*) # *xs1*) $\in$ *cpts-p* Γ **and**
c3: $\exists$ *P*. *getspc-e* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P*
from *c3* **obtain** *P* **where** *c-*:*getspc-e* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P* **by** *auto*
then have *c4*: *e1* = *AnonyEvent P* **by** (*simp add*:*getspc-e-def*)
with *c0* **have** $\exists$ *Q*. *e2* = *AnonyEvent Q*
apply(*clarify*)
apply(*rule etran.cases*)
apply(*simp-all*)+
done
then obtain *Q* **where** *c5*: *e2* = *AnonyEvent Q* **by** *auto*
with *c2* **have** *c6*:*lower-evts* ((*e2*, *t1*, *y1*) # *xs1*) $\in$ *cpts-p* Γ **by** (*simp add*: *getspc-e-def*)
have *c7*: *lower-evts* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) =
(*lower-anonyevt1* (*e1*, *s1*, *x1*)) # *lower-evts* ((*e2*, *t1*, *y1*) # *xs1*)
by (*simp add*: *lower-evts-def*)
have *c7-*: *lower-evts* ((*e2*, *t1*, *y1*) # *xs1*) = *lower-anonyevt1* (*e2*, *t1*, *y1*) # *lower-evts xs1*
by (*simp add*: *lower-evts-def*)
with *c6* **have** *c8*: *lower-anonyevt1* (*e2*, *t1*, *y1*) # *lower-evts xs1* $\in$ *cpts-p* Γ **by** *simp*
from *c4* **have** *c9*: *lower-anonyevt1* (*e1*, *s1*, *x1*) = (*P*, *s1*)
by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
from *c5* **have** *c10*: *lower-anonyevt1* (*e2*, *t1*, *y1*) = (*Q*, *t1*)
by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
from *c0 c4 c5* **have** *c11*: $\Gamma \vdash$ (*AnonyEvent P*, *s1*, *x1*) $-et-et\rightarrow$ (*AnonyEvent Q*, *t1*, *y1*) **by** *simp*
then have $\Gamma \vdash$ (*P*, *s1*) $-c\rightarrow$ (*Q*, *t1*)
apply(*rule etran.cases*)
apply(*simp-all*)
done
with *c8 c9 c10* **have** *lower-anonyevt1* (*e1*, *s1*, *x1*) # *lower-anonyevt1* (*e2*, *t1*, *y1*) # *lower-evts xs1* $\in$ *cpts-p* Γ
using *CptsPComp* **by** *simp*
with *c7 c7-* **show** *?case* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed
with *a0 a1* **show** *?thesis* **by** *blast*
qed

lemma *equiv-lower-evts2* : $es \in$ *cpts-of-ev* Γ (*AnonyEvent P*) s $x \Longrightarrow$ *lower-evts es* $\in$ *cpts-p* $\Gamma \wedge$ (*lower-evts es*) ! *0* = (P,s)
proof −
assume *a0*: $es \in$ *cpts-of-ev* Γ (*AnonyEvent P*) s x
then have *a1*: *es*!*0*=(*AnonyEvent P*,(s,x)) $\wedge$ $es \in$ *cpts-ev* Γ **by** (*simp add*: *cpts-of-ev-def*)
then have *a2*: *getspc-e* (*es* ! *0*) = *AnonyEvent P* **by** (*simp add*:*getspc-e-def*)
with *a1* **have** *a3*: *lower-evts es* $\in$*cpts-p* Γ **using** *equiv-lower-evts0*
by (*simp add*: *equiv-lower-evts0*)
have *a4*: *lower-evts es* ! *0* = *lower-anonyevt1* (*es* ! *0*)
by (*metis a3 cptn-not-empty list.simps*(*8*) *list.size*(*3*) *lower-evts-def neq0-conv not-less0 nth-equalityI nth-map*)
from *a1* **have** *a5*: *lower-anonyevt1* (*es* ! *0*) = (P,s)
by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
with *a4* **have** *a6*: *lower-evts es* ! *0* = (P,s) **by** *simp*
with *a3* **show** *?thesis* **by** *simp*
qed

lemma *equiv-lower-evts* : $es \in$ *cpts-of-ev* Γ (*AnonyEvent P*) s $x \Longrightarrow$ *lower-evts es* $\in$ *cpts-of-p* Γ P s
using *equiv-lower-evts2*[*of es* Γ *P s x*] *cpts-of-p-def*[*of lower-evts es P s* Γ] **by** *simp*

7.2.2 trace between of basic and anonymous events

lemma *evtent-in-cpts1*: $el \in$ *cpts-ev* $\Gamma \wedge$ *el* ! *0* = (*BasicEvent ev*, s, x) $\Longrightarrow$
Suc i < *length el* $\wedge$ $\Gamma \vdash$ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯k→ *el* ! (*Suc i*) $\Longrightarrow$
($\forall j$. *Suc j* $\leq i \longrightarrow$ *getspc-e* (*el* ! *j*) = *BasicEvent ev* $\wedge$ $\Gamma \vdash$ *el* ! *j* −*ee*→ *el* ! (*Suc j*))
proof −
assume *p0*: $el \in$ *cpts-ev* $\Gamma \wedge$ *el* ! *0* = (*BasicEvent ev*, s, x)
assume *p1*: *Suc i* < *length el* $\wedge$ $\Gamma \vdash$ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯k→ *el* ! (*Suc i*)
from *p0* **have** *p01*: $el \in$ *cpts-ev* Γ **and**
p02: *el* ! *0* = (*BasicEvent ev*, s, x) **by** *auto*
from *p1* **have** *p3*: *getspc-e* (*el* ! *i*) = *BasicEvent ev* **by** (*meson ent-spec*)
show $\forall j$. *Suc j* $\leq i \longrightarrow$ *getspc-e* (*el* ! *j*) = *BasicEvent ev* $\wedge$ $\Gamma \vdash$ *el* ! *j* −*ee*→ *el* ! (*Suc j*)
proof −
{
fix j
assume *a0*: *Suc j* $\leq i$
have $\forall k$. $k < i \longrightarrow$ *getspc-e* (*el* ! (i −k −*1*)) = *BasicEvent ev* $\wedge$ $\Gamma \vdash$ *el* ! (i −k −*1*)−*ee*→ *el* ! (i − k)
proof −
{

fix *k*
assume $k < i$
then have *getspc-e* (*el* ! ($i - k - 1$)) = *BasicEvent ev* ∧ Γ ⊢ *el* ! ($i - k - 1$)−*ee*→ *el* ! ($i - k$)
proof(*induct k*)
case *0*
from *p3* **have** *b0*: ¬(∃ *t ec1*. Γ ⊢ *ec1*−*et*−*t*→(*el* ! *i*))
using *no-tran2basic getspc-e-def* **by** (*metis prod.collapse*)
with *p1 p01* **have** *b1*: *getspc-e* (*el* ! ($i - 1$)) = *getspc-e* (*el* ! *i*) **using** *notran-confeqi*
by (*metis 0.prems Suc-diff-1 Suc-lessD*)
with *p3* **show** *?case* **by** (*simp add*: *eqconf-eetran*)
next
case (*Suc m*)
assume *b0*: $m < i$ ⟹ *getspc-e* (*el* ! ($i - m - 1$)) = *BasicEvent ev*
∧ Γ ⊢ *el* ! ($i - m - 1$) −*ee*→ *el* ! ($i - m$) **and**
b1: *Suc m* < *i*
then have *b2*: *getspc-e* (*el* ! ($i - m - 1$)) = *BasicEvent ev* **and**
b3: Γ ⊢ *el* ! ($i - m - 1$) −*ee*→ *el* ! ($i - m$)
using *Suc-lessD* **apply** *blast*
using *Suc-lessD b0 b1* **by** *blast*
have *b4*: *Suc m* = $m + 1$ **by** *auto*
with *b2* **have** ¬(∃ *t ec1*. Γ ⊢ *ec1*−*et*−*t*→(*el* ! (*i* − *Suc m*)))
using *no-tran2basic getspc-e-def* **by** (*metis diff-diff-left prod.collapse*)

with *p1 p02* **have** *b5*: *getspc-e* (*el* ! ((*i* − *Suc m* − *1*))) = *getspc-e* (*el* ! (*i* − *Suc m*))
using *notran-confeqi* **by** (*smt Suc-diff-1 Suc-lessD b1 diff-less less-trans p01*
zero-less-Suc zero-less-diff)
with *b2 b4* **have** *b6*: *getspc-e* (*el* ! ((*i* − *Suc m* − *1*))) = *BasicEvent ev*
by (*metis diff-diff-left*)
from *b5* **have** Γ ⊢ *el* ! (*i* − *Suc m* − *1*) −*ee*→ *el* ! (*i* − *Suc m*) **using** *eqconf-eetran* **by** *simp*
with *b6* **show** *?case* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed

}
then show *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-le-lessD diff-Suc-1 diff-Suc-less*
diff-diff-cancel gr-implies-not0 less-antisym zero-less-Suc)
qed
qed

lemma *evtent-in-cpts2*: *el* ∈ *cpts-ev* Γ ∧ *el* ! *0* = (*BasicEvent ev*, *s*, *x*) ⟹

Suc i < *length el* ∧ Γ ⊢ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*) ⟹
(*gets-e* (*el* ! *i*) ∈ *guard ev* ∧ *drop* (*Suc i*) *el* ∈
cpts-of-ev Γ (*AnonyEvent* (*body ev*)) (*gets-e* (*el* ! (*Suc i*))) ((*getx-e* (*el* ! *i*)) (*k* := *BasicEvent ev*)))
proof −
assume *p0*: *el* ∈ *cpts-ev* Γ ∧ *el* ! *0* = (*BasicEvent ev*, *s*, *x*)
assume *p1*: *Suc i* < *length el* ∧ Γ ⊢ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*)
then have *a2*: *gets-e* (*el* ! *i*) ∈ *guard ev* ∧ *gets-e* (*el* ! *i*) = *gets-e* (*el* ! (*Suc i*))
∧ *getspc-e* (*el* ! (*Suc i*)) = *AnonyEvent* (*body ev*)
∧ *getx-e* (*el* ! (*Suc i*)) = (*getx-e* (*el* ! *i*)) (*k* := *BasicEvent ev*)
by (*meson ent-spec2*)

from *p1* **have** (*drop* (*Suc i*) *el*)!*0* = *el* ! (*Suc i*) **by** *auto*
with *a2* **have** *a3*: (*drop* (*Suc i*) *el*)!*0* = (*AnonyEvent* (*body ev*),(*gets-e* (*el* ! (*Suc i*)),
(*getx-e* (*el* ! *i*)) (*k* := *BasicEvent ev*)))
using *gets-e-def getspc-e-def getx-e-def* **by** (*metis prod.collapse*)
have *a4*: *drop* (*Suc i*) *el* ∈ *cpts-ev* Γ **by** (*simp add*: *cpts-ev-subi p0 p1*)
with *a2 a3* **show** *gets-e* (*el* ! *i*) ∈ *guard ev* ∧ *drop* (*Suc i*) *el* ∈
cpts-of-ev Γ (*AnonyEvent* (*body ev*)) (*gets-e* (*el* ! (*Suc i*))) ((*getx-e* (*el* ! *i*)) (*k* := *BasicEvent ev*))
by (*metis* (*mono-tags*, *lifting*) *CollectI cpts-of-ev-def*)

qed

lemma *no-evtent-in-cpts*: *el* ∈ *cpts-ev* Γ ⟹ *el* ! *0* = (*BasicEvent ev*, *s*, *x*) ⟹
(¬ (∃ *i k*. *Suc i* < *length el* ∧ Γ ⊢ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*))) ⟹
(∀ *j*. *Suc j* < *length el* ⟶ *getspc-e* (*el* ! *j*) = *BasicEvent ev*
∧ Γ ⊢ *el* ! *j* −*ee*→ *el* ! (*Suc j*)
∧ *getspc-e* (*el* ! (*Suc j*)) = *BasicEvent ev*)
proof −
assume *p0*: *el* ∈ *cpts-ev* Γ **and**
p1: *el* ! *0* = (*BasicEvent ev*, *s*, *x*) **and**
p2: ¬ (∃ *i k*. *Suc i* < *length el* ∧ Γ ⊢ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*))
show *?thesis*
proof −
{
fix *j*
assume *Suc j* < *length el*
then have *getspc-e* (*el* ! *j*) = *BasicEvent ev* ∧ Γ ⊢ *el* ! *j* −*ee*→ *el* ! (*Suc j*)
∧ *getspc-e* (*el* ! (*Suc j*)) = *BasicEvent ev*

proof(*induct j*)
case *0*
assume *a0*: *Suc 0* < *length el*
from *p1* **have** *a00*: *getspc-e* (*el* ! *0*) = *BasicEvent ev* **by** (*simp add*:*getspc-e-def*)
from *a0 p2* **have** ¬ (∃ *k*. Γ ⊢ *el* ! *0* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc 0*)) **by** *simp*
with *p0 p1* **have** ¬ (∃ *t*. Γ ⊢ *el* ! *0* −*et*−*t*→ *el* ! (*Suc 0*)) **by** (*metis noevtent-notran*)
with *p0 a0* **have** *a1*: *getspc-e* (*el* ! *0*) = *getspc-e* (*el* ! (*Suc 0*))
using *notran-confeqi* **by** *blast*

with *a00* **have** *a2*: *getspc-e* (*el* ! (*Suc 0*)) = *BasicEvent ev* **by** *simp*
from *a1* **have** Γ ⊢ *el* ! *0* −*ee*→ *el* ! *Suc 0* **using** *getspc-e-def eetran.EnvE*
by (*metis eq-fst-iff*)
then show *?case* **by** (*simp add*: *a00 a2*)
next
case (*Suc m*)
assume *a0*: *Suc m* < *length el* ⟹ *getspc-e* (*el* ! *m*) = *BasicEvent ev* ∧ Γ ⊢ *el* ! *m* −*ee*→ *el* ! *Suc m*
∧ *getspc-e* (*el* ! *Suc m*) = *BasicEvent ev*
assume *a1*: *Suc* (*Suc m*) < *length el*
with *a0* **have** *a2*: *getspc-e* (*el* ! *m*) = *BasicEvent ev* ∧ Γ ⊢ *el* ! *m* −*ee*→ *el* ! *Suc m* **by** *simp*
then have *a3*: *getspc-e* (*el* ! *Suc m*) = *BasicEvent ev* **using** *getspc-e-def* **by** (*metis eetranE fstI*)

then have *a4*: ∃ *s x*. *el* ! *Suc m* = (*BasicEvent ev*, *s*, *x*) **unfolding** *getspc-e-def*
by (*metis fst-conv surj-pair*)
from *a0 a1 p2* **have** ¬ (∃ *k*. Γ ⊢ *el* ! (*Suc m*) −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc* (*Suc m*))) **by** *simp*
with *a4* **have** *a5*: ¬ (∃ *t*. Γ ⊢ *el* ! (*Suc m*) −*et*−*t*→ *el* ! (*Suc* (*Suc m*)))
using *noevtent-notran* **by** *metis*

with *p0 a0 a1* **have** *a6*: *getspc-e* (*el* ! (*Suc m*)) = *getspc-e* (*el* ! (*Suc* (*Suc m*)))
using *notran-confeqi* **by** *blast*
with *a3* **have** *a7*: *getspc-e* (*el* ! (*Suc* (*Suc m*))) = *BasicEvent ev* **by** *simp*
from *a6* **have** Γ ⊢ *el* ! *Suc m* −*ee*→ *el* ! *Suc* (*Suc m*) **using** *getspc-e-def eetran.EnvE*
by (*metis eq-fst-iff*)

with *a3 a7* **show** *?case* **by** *simp*
qed
}
then show *?thesis* **by** *auto*

qed
qed

7.2.3 trace between of event and event system

primrec *rm-evtsys0* :: $('l,'k,'s,'prog)$ *esys* $\Rightarrow$ *'s* $\Rightarrow$ $('l,'k,'s,'prog)$ *x* $\Rightarrow$ $('l,'k,'s,'prog)$ *econf*
where *EvtSeqrm*: *rm-evtsys0* (*EvtSeq e es*) *s x*= (*e*, *s*, *x*) |
EvtSysrm: *rm-evtsys0* (*EvtSys es*) *s x*= (*AnonyEvent fin-com*, *s*, *x*)

definition *rm-evtsys1* :: $('l,'k,'s,'prog)$ *esconf* $\Rightarrow$ $('l,'k,'s,'prog)$ *econf*
where *rm-evtsys1 esc* $\equiv$ *rm-evtsys0* (*getspc-es esc*) (*gets-es esc*) (*getx-es esc*)

definition *rm-evtsys* :: $('l,'k,'s,'prog)$ *esconfs* $\Rightarrow$ $('l,'k,'s,'prog)$ *econfs*
where *rm-evtsys escfs* $\equiv$ *map rm-evtsys1 escfs*

definition *e-eqv-einevtseq* :: $('l,'k,'s,'prog)$ *esconfs* $\Rightarrow$ $('l,'k,'s,'prog)$ *econfs* $\Rightarrow$ $('l,'k,'s,'prog)$ *esys* $\Rightarrow$ *bool*
where *e-eqv-einevtseq esl el es* $\equiv$ *length esl* = *length el* $\wedge$
($\forall$ *i*. *Suc i* $\leq$ *length el* $\longrightarrow$ *gets-e* (*el* ! *i*) = *gets-es* (*esl* ! *i*) $\wedge$
getx-e (*el* ! *i*) = *getx-es* (*esl* ! *i*) $\wedge$
getspc-es (*esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*)

lemma *e-eqv-einevtseq-s* : $[\![$*e-eqv-einevtseq esl el es*; *gets-e e1* = *gets-es es1*; *getx-e e1* = *getx-es es1*;
getspc-es es1 = *EvtSeq* (*getspc-e e1*) *es*$]\!]$ $\Longrightarrow$ *e-eqv-einevtseq* (*es1* # *esl*) (*e1* # *el*) *es*
proof −
assume *p0*: *e-eqv-einevtseq esl el es*
and *p1*: *gets-e e1* = *gets-es es1*
and *p2*: *getx-e e1* = *getx-es es1*
and *p3*: *getspc-es es1* = *EvtSeq* (*getspc-e e1*) *es*
let *?el'* = *e1* # *el*
let *?esl'* = *es1* # *esl*
from *p0* **have** *a1*: *length esl* = *length el* **by** (*simp add*: *e-eqv-einevtseq-def*)
from *p0* **have** *a2*: $\forall$ *i*. *Suc i* $\leq$ *length el* $\longrightarrow$ *gets-e* (*el* ! *i*) = *gets-es* (*esl* ! *i*) $\wedge$
getx-e (*el* ! *i*) = *getx-es* (*esl* ! *i*) $\wedge$
getspc-es (*esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
from *a1* **have** *length* (*es1* # *esl*) = *length* (*e1* # *el*) **by** *simp*
moreover have $\forall$ *i*. *Suc i* $\leq$ *length ?el'* $\longrightarrow$ *gets-e* (*?el'* ! *i*) = *gets-es* (*?esl'* ! *i*) $\wedge$
getx-e (*?el'* ! *i*) = *getx-es* (*?esl'* ! *i*) $\wedge$
getspc-es (*?esl'* ! *i*) = *EvtSeq* (*getspc-e* (*?el'* ! *i*)) *es*
by (*simp add*: *a2 nth-Cons' p1 p2 p3*)
ultimately show *e-eqv-einevtseq ?esl' ?el' es* **by** (*simp add*:*e-eqv-einevtseq-def*)
qed

definition *same-s-x*:: ('l,'k,'s,'prog) *esconfs* ⇒ ('l,'k,'s,'prog) *econfs* ⇒ *bool*
where *same-s-x esl el* ≡ *length esl* = *length el* ∧
(∀ *i*. *Suc i* ≤ *length el* ⟶ *gets-e* (*el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
getx-e (*el* ! *i*) = *getx-es* (*esl* ! *i*))

lemma *rm-evtsys-same-sx*: *same-s-x esl* (*rm-evtsys esl*)
proof(*induct esl*)
case *Nil*
show *?case* **by** (*simp add*:*rm-evtsys-def same-s-x-def*)
next
case (*Cons ec1 esl1*)
assume *a0*: *same-s-x esl1* (*rm-evtsys esl1*)
have *a1*: *rm-evtsys* (*ec1* # *esl1*) = *rm-evtsys1 ec1* # *rm-evtsys esl1* **by** (*simp add*:*rm-evtsys-def*)
obtain *es* **and** *s* **and** *x* **where** *a2*: *ec1* = (*es*, *s*, *x*) **using** *prod-cases3* **by** *blast*
then show *?case*
proof(*induct es*)
case (*EvtSeq x1 es1*)
assume *b0*: *ec1* = (*EvtSeq x1 es1*, *s*, *x*)
then have *b1*: *rm-evtsys1 ec1* # *rm-evtsys esl1* = (*x1*, *s*, *x*) # *rm-evtsys esl1*
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
have *length* (*ec1* # *esl1*) = *length* (*rm-evtsys* (*ec1* # *esl1*)) **by** (*simp add*: *rm-evtsys-def*)
moreover have ∀ *i*. *Suc i* ≤ *length* (*rm-evtsys* (*ec1* # *esl1*)) ⟶
gets-e ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *gets-es* ((*ec1* # *esl1*) ! *i*)
∧ *getx-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *getx-es* ((*ec1* # *esl1*) ! *i*)
proof −
{
fix *i*
assume *c0*: *Suc i* ≤ *length* (*rm-evtsys* (*ec1* # *esl1*))
have *gets-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *gets-es* ((*ec1* # *esl1*) ! *i*)
∧ *getx-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *getx-es* ((*ec1* # *esl1*) ! *i*)
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
with *a0 a1 b0 b1* **show** *?thesis* **using** *gets-e-def gets-es-def getx-e-def getx-es-def*
by (*metis nth-Cons-0 snd-conv*)
next
assume *d0*: *i* ≠ *0*
then have (*rm-evtsys* (*ec1* # *esl1*)) ! *i* = (*rm-evtsys esl1*) ! (*i* − *1*)
by (*simp add*: *a1*)
moreover have (*ec1* # *esl1*) ! *i* = *esl1* ! (*i* − *1*)
by (*simp add*: *d0 nth-Cons'*)

ultimately show *?thesis* **using** *a0 c0 d0 same-s-x-def*
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leI Suc-le-lessD*
Suc-less-eq a1 length-Cons neq0-conv)
qed
}
then show *?thesis* **by** *auto*
qed

ultimately show *?case* **using** *same-s-x-def* **by** *blast*
next
case (*EvtSys xa*)
assume *b0*: *ec1* = (*EvtSys xa*, *s*, *x*)
then have *b1*: *rm-evtsys1 ec1* # *rm-evtsys esl1* = (*AnonyEvent fin-com*, *s*, *x*) # *rm-evtsys esl1*
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
have *length* (*ec1* # *esl1*) = *length* (*rm-evtsys* (*ec1* # *esl1*)) **by** (*simp add*: *rm-evtsys-def*)
moreover have $\forall$ *i*. *Suc i* $\leq$ *length* (*rm-evtsys* (*ec1* # *esl1*)) $\longrightarrow$
gets-e ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *gets-es* ((*ec1* # *esl1*) ! *i*)
$\wedge$ *getx-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *getx-es* ((*ec1* # *esl1*) ! *i*)
proof −
{
fix *i*
assume *c0*: *Suc i* $\leq$ *length* (*rm-evtsys* (*ec1* # *esl1*))
have *gets-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *gets-es* ((*ec1* # *esl1*) ! *i*)
$\wedge$ *getx-e* ((*rm-evtsys* (*ec1* # *esl1*)) ! *i*) = *getx-es* ((*ec1* # *esl1*) ! *i*)
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
with *a0 a1 b0 b1* **show** *?thesis* **using** *gets-e-def gets-es-def getx-e-def getx-es-def*
by (*metis nth-Cons-0 snd-conv*)
next
assume *d0*: *i* $\neq$ *0*
then have (*rm-evtsys* (*ec1* # *esl1*)) ! *i* = (*rm-evtsys esl1*) ! (*i* − *1*)
by (*simp add*: *a1*)
moreover have (*ec1* # *esl1*) ! *i* = *esl1* ! (*i* − *1*)
by (*simp add*: *d0 nth-Cons'*)
ultimately show *?thesis* **using** *a0 c0 d0 same-s-x-def*
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leI Suc-le-lessD*
Suc-less-eq a1 length-Cons neq0-conv)
qed
}
then show *?thesis* **by** *auto*
qed
ultimately show *?case* **using** *same-s-x-def* **by** *blast*
qed

qed

definition *e-sim-es*:: ('l,'k,'s,'prog) *esconfs* ⇒ ('l,'k,'s,'prog) *econfs*
⇒ ('l,'k,'s,'prog) *event set* ⇒ ('l,'s,'prog) *event'* ⇒ *bool*
where *e-sim-es esl el es e* ≡ *length esl* = *length el* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧
getspc-e (*el*!*0*) = *BasicEvent e* ∧
(∀ *i*. *i* < *length el* ⟶ *gets-e* (*el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
getx-e (*el* ! *i*) = *getx-es* (*esl* ! *i*)) ∧
(∀ *i*. *i* > *0* ∧ *i* < *length el* ⟶
(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*))
)

7.3 Soundness of Events

lemma *anony-cfgs0* : ⟦∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*; *es* ∈ *cpts-ev* Γ⟧
⟹ ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof −
assume *a0*: *es* ∈ *cpts-ev* Γ **and** *a1*: ∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*
from *a0 a1* **show** ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof(*induct es*)
case (*CptsEvOne e s x*)
assume *b0*: ∃ *P*. *getspc-e* ([(*e*, *s*, *x*)] ! *0*) = *AnonyEvent P*
show *?case* **using** *b0* **by** *auto*
next
case (*CptsEvEnv e' t' x' xs' s' y'*)
assume *b0*: (*e'*, *t'*, *x'*) # *xs'* ∈ *cpts-ev* Γ **and**
b1: ∃ *P*. *getspc-e* (((*e'*, *t'*, *x'*) # *xs'*) ! *0*) = *AnonyEvent P* ⟹
∀ *i*<*length* ((*e'*, *t'*, *x'*) # *xs'*). ∃ *Q*. *getspc-e* (((*e'*, *t'*, *x'*) # *xs'*) ! *i*) = *AnonyEvent Q* **and**
b2: ∃ *P*. *getspc-e* (((*e'*, *s'*, *y'*) # (*e'*, *t'*, *x'*) # *xs'*) ! *0*) = *AnonyEvent P*
from *b2* **obtain** *P1* **where** *b3*: *getspc-e* (((*e'*, *s'*, *y'*) # (*e'*, *t'*, *x'*) # *xs'*) ! *0*) = *AnonyEvent P1* **by** *auto*
then have *b4*: *e'* = *AnonyEvent P1* **by** (*simp add*: *getspc-e-def*)
with *b1* **have** ∀ *i*<*length* ((*e'*, *t'*, *x'*) # *xs'*). ∃ *Q*. *getspc-e* (((*e'*, *t'*, *x'*) # *xs'*) ! *i*) = *AnonyEvent Q*
by (*simp add*: *getspc-e-def*)
with *b4* **show** *?case* **by** (*metis* (*no-types*, *hide-lams*) *Ex-list-of-length b3 gr0-conv-Suc*
length-Cons length-tl list.sel(*3*) *not-less-eq nth-non-equal-first-eq*)
next
case (*CptsEvComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *b0*: Γ ⊢ (*e1*, *s1*, *x1*) −*et*−*et*→ (*e2*, *t1*, *y1*) **and**

b1: (*e2*, *t1*, *y1*) # *xs1* ∈ *cpts-ev* Γ **and**
b2: ∃ *P*. *getspc-e* (((*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P* ⟹
∀ *i*<*length* ((*e2*, *t1*, *y1*) # *xs1*). ∃ *Q*. *getspc-e* (((*e2*, *t1*, *y1*) # *xs1*) ! *i*) = *AnonyEvent Q* **and**
b3: ∃ *P*. *getspc-e* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P*
from *b3* **obtain** *P1* **where** *b4*: *getspc-e* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P1* **by** *auto*
then have *b5*: *e1* = *AnonyEvent P1* **by** (*simp add*: *getspc-e-def*)
with *b0* **have** ∃ *Q*. *e2* = *AnonyEvent Q*
apply(*clarify*)
apply(*rule etran.cases*)
apply(*simp-all*)+
done
then have ∃ *P*. *getspc-e* (((*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *AnonyEvent P* **by** (*simp add*:*getspc-e-def*)
with *b2* **have** *b6*: ∀ *i*<*length* ((*e2*, *t1*, *y1*) # *xs1*). ∃ *Q*. *getspc-e* (((*e2*, *t1*, *y1*) # *xs1*) ! *i*) = *AnonyEvent Q* **by** *auto*
with *b5* **show** *?case* **by** (*metis* (*no-types*, *hide-lams*) *Ex-list-of-length b3 gr0-conv-Suc*
length-Cons length-tl list.sel(*3*) *not-less-eq nth-non-equal-first-eq*)
qed
qed

lemma *anony-cfgs* : *es* ∈ *cpts-of-ev* Γ (*AnonyEvent P*) *s x* ⟹ ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof −
assume *a0*: *es* ∈ *cpts-of-ev* Γ (*AnonyEvent P*) *s x*
then have *a1*: *es*!*0*=(*AnonyEvent P*,(*s*,*x*)) ∧ *es* ∈ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)
then have ∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P* **by** (*simp add*:*getspc-e-def*)
with *a1* **show** *?thesis* **using** *anony-cfgs0* **by** *blast*
qed

lemma *AnonyEvt-sound*: Γ ⊨ *P* sat_p [*pre*, *rely*, *guar*, *post*] ⟹ Γ ⊨ *AnonyEvent P* sat_e [*pre*, *rely*, *guar*, *post*]
proof −
assume *a0*: Γ ⊨ *P* sat_p [*pre*, *rely*, *guar*, *post*]
then have *a1*: ∀ *s*. *cpts-of-p* Γ *P s* ∩ *assume-p* Γ (*pre*, *rely*) ⊆ *commit-p* Γ (*guar*, *post*)
using *prog-validity-def* **by** *simp*
then have ∀ *s x*. (*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) ∩ *assume-e* Γ (*pre*, *rely*)
⊆ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *s x*
have ∀ *el*. *el*∈(*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) ∩ *assume-e* Γ (*pre*, *rely*) ⟶ *el*∈ *commit-e* Γ (*guar*, *post*)
proof −

{
fix *el*
assume *b0*: *el*∈(*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) ∩ *assume-e* Γ (*pre*, *rely*)
then obtain *pl* **where** *b1*: *pl* = *lower-evts el* **by** *simp*
with *b0* **have** *b2*: *pl* ∈ *cpts-of-p* Γ *P s* **using** *equiv-lower-evts* **by** *auto*
from *b0 b1* **have** *b21*: *pl* ∈ *cpts-p* Γ ∧ *pl*!*0* = (*P*,*s*) **using** *equiv-lower-evts2*[*of el* Γ *P s x*] **by** *auto*
from *b0* **have** *b3*: *el*!*0*=(*AnonyEvent P*,(*s*,*x*)) **and** *b4*: *el* ∈ *cpts-ev* Γ
by (*simp add*:*cpts-of-ev-def*)+
from *b0* **have** *b5*: *el* ∈ *assume-e* Γ (*pre*, *rely*) **by** *simp*
hence *b51*: *gets-e* (*el*!*0*) ∈ *pre* **by** (*simp add*:*assume-e-def*)
from *b1 b21 b3 b51* **have** *b6*: *gets-p* (*pl*!*0*) ∈ *pre* **by** (*simp add*:*gets-p-def gets-e-def*)

have *b7*: ∀ *i*. *Suc i*<*length pl* ⟶
Γ ⊢ *pl*!*i* −*pe*→ *pl*!(*Suc i*) ⟶ (*gets-p* (*pl*!*i*), *gets-p* (*pl*!*Suc i*)) ∈ *rely*
proof −
{
fix *i*
assume *c0*: *Suc i*<*length pl* **and** *c1*: Γ ⊢ *pl*!*i* −*pe*→ *pl*!(*Suc i*)
from *b1 c0* **have** *c2*: *Suc i* < *length el* **by** (*simp add*:*lower-evts-def*)
from *c1* **have** *c3*: *getspc-p* (*pl*!*i*) = *getspc-p* (*pl*!(*Suc i*))
using *getspc-p-def fst-conv petran-simps*
by (*metis prod.collapse*)
from *b1* **have** *c4*: *lower-anonyevt1* (*el*!*i*) = *pl*!*i*
by (*simp add*: *Suc-lessD c2 lower-evts-def*)
from *b1* **have** *c5*: *lower-anonyevt1* (*el*!*Suc i*) = *pl*!*Suc i*
by (*simp add*: *Suc-lessD c2 lower-evts-def*)

from *b0 c2* **have** *c7*: ∃ *Q*. *getspc-e* (*el*!*i*) = *AnonyEvent Q*
by (*meson Int-iff Suc-lessD anony-cfgs*)
then obtain *Q1* **where** *c71*: *getspc-e* (*el*!*i*) = *AnonyEvent Q1* **by** *auto*
from *b0 c2* **have** *c8*: ∃ *Q*. *getspc-e* (*el* ! (*Suc i*)) = *AnonyEvent Q*
by (*meson Int-iff anony-cfgs*)
then obtain *Q2* **where** *c81*: *getspc-e* (*el* ! (*Suc i*)) = *AnonyEvent Q2* **by** *auto*
from *c4 c71* **have** *c9*: *getspc-p* (*pl* ! *i*) = *Q1*
using *lower-anonyevt1-def AnonyEv getspc-p-def* **by** (*metis fst-conv*)
from *c5 c81* **have** *c10*: *getspc-p* (*pl* ! (*Suc i*)) = *Q2*
using *lower-anonyevt1-def AnonyEv getspc-p-def* **by** (*metis fst-conv*)
with *c3 c9* **have** *c11*: *Q1* = *Q2* **by** *simp*

from *c4 c71* **have** *c61*: *gets-p* (*pl*!*i*) = *gets-e* (*el*!*i*)
using *lower-anonyevt1-def AnonyEv gets-p-def* **by** (*metis snd-conv*)

from *c5 c81* **have** *c62*: *gets-p* (*pl*! (*Suc i*)) = *gets-e* (*el*! (*Suc i*))
using *lower-anonyevt1-def AnonyEv gets-p-def* **by** (*metis snd-conv*)

from *c71 c81 c11* **have** *c12*: *getspc-e* (*el*!*i*) = *getspc-e* (*el*!(*Suc i*))
by *simp*
then have *c13*: Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*) **using** *eetran.EnvE getspc-e-def*
by (*metis prod.collapse*)
from *b5 c2* **have** (∀ *i*. *Suc i* < *length el* ⟶ Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i*
⟶ (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely*) **by** (*simp add*:*assume-e-def*)
with *c2 c13* **have** (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely* **by** *auto*

with *c61 c62* **have** (*gets-p* (*pl*!*i*), *gets-p* (*pl*!*Suc i*)) ∈ *rely* **by** *simp*
}
then show *?thesis* **by** *auto*
qed

with *b6* **have** *b8*: *pl* ∈ *assume-p* Γ (*pre*, *rely*) **by** (*simp add*:*assume-p-def*)

with *a1 b2* **have** *b9*: *pl* ∈ *commit-p* Γ (*guar*, *post*) **by** *auto*
then have *b10*: (∀ *i*. *Suc i*<*length el* ⟶
(∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*)) ⟶ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar*)
proof −
{
fix *i*
assume *c0*: *Suc i*<*length el*
assume *c1*: ∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*)
from *b1 c0* **have** *c2*: *Suc i* < *length pl* **by** (*simp add*:*lower-evts-def*)

from *b1* **have** *c3*: *lower-anonyevt1* (*el*!*i*) = *pl*!*i*
by (*simp add*: *Suc-lessD c0 lower-evts-def*)
from *b1* **have** *c4*: *lower-anonyevt1* (*el*!*Suc i*) = *pl*!*Suc i*
by (*simp add*: *Suc-lessD c0 lower-evts-def*)
from *b0 c0* **have** *c7*: ∃ *Q*. *getspc-e* (*el*!*i*) = *AnonyEvent Q*
by (*meson Int-iff Suc-lessD anony-cfgs*)
then obtain *Q1* **where** *c71*: *getspc-e* (*el*!*i*) = *AnonyEvent Q1* **by** *auto*
from *b0 c0* **have** *c8*: ∃ *Q*. *getspc-e* (*el* ! (*Suc i*)) = *AnonyEvent Q*
by (*meson Int-iff anony-cfgs*)
then obtain *Q2* **where** *c81*: *getspc-e* (*el*! (*Suc i*)) = *AnonyEvent Q2* **by** *auto*

have *c5*: Γ ⊢ *pl*!*i* −*c*→ *pl*!(*Suc i*)
proof −
from *c1* **obtain** *t* **where** *d0*: Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*) **by** *auto*
obtain *s1* **and** *x1* **where** *d1*: *s1* = *gets-e* (*el* ! *i*) ∧ *x1* = *getx-e*

(*el* ! *i*) **by** *simp*
obtain *s2* **and** *x2* **where** *d2*: *s2* = *gets-e* (*el* ! (*Suc i*)) ∧ *x2* = *getx-e* (*el* ! (*Suc i*)) **by** *simp*
with *d1 c71 c81* **have** *d21*: *el* ! *i* = (*AnonyEvent Q1*, *s1*, *x1*)
∧ *el* ! (*Suc i*) = (*AnonyEvent Q2*, *s2*, *x2*)
using *gets-e-def getx-e-def getspc-e-def* **by** (*metis prod.collapse*)

with *d0* **have** *d3*: Γ ⊢ (*AnonyEvent Q1*, *s1*, *x1*) −*et*−*t*→ (*AnonyEvent Q2*, *s2*, *x2*) **by** *simp*
then have ∃ *k*. *t* = ((*Cmd CMP*)♯*k*)
apply(*rule etran.cases*)
apply *simp-all*
by *auto*
then obtain *k* **where** *t* = ((*Cmd CMP*)♯*k*) **by** *auto*
with *d3* **have** *d4*: Γ ⊢ (*Q1*,*s1*) −*c*→ (*Q2*, *s2*)
apply(*clarify*)
apply(*rule etran.cases*)
apply *simp-all*+
done
from *c3 d21* **have** *d5*: *pl*!*i* = (*Q1*,*s1*) **by** (*simp add*:*lower-anonyevt1-def getspc-e-def gets-e-def*)
from *c4 d21* **have** *d6*: *pl*! (*Suc i*) = (*Q2*,*s2*) **by** (*simp add*:*lower-anonyevt1-def getspc-e-def gets-e-def*)
with *d4 d5* **show** *?thesis* **by** *simp*
qed
with *b9 c2* **have** *c6*: (*gets-p* (*pl*!*i*), *gets-p* (*pl*!*Suc i*)) ∈ *guar* **by** (*simp add*:*commit-p-def*)

from *c3 c71* **have** *c9*: *gets-e* (*el*!*i*) = *gets-p* (*pl*!*i*) **using** *lower-anonyevt-s* **by** *fastforce*
from *c4 c81* **have** *c10*: *gets-e* (*el*!*Suc i*) = *gets-p* (*pl*!*Suc i*) **using** *lower-anonyevt-s* **by** *fastforce*
from *c6 c9 c10* **have** (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar* **by** *simp*
}
then show *?thesis* **by** *auto*
qed

have *b11*: (*getspc-e* (*last el*) = *AnonyEvent fin-com* ⟶ *gets-e* (*last el*) ∈ *post*)
proof
assume *c0*: *getspc-e* (*last el*) = *AnonyEvent fin-com*
from *b1* **have** *c1*: *last pl* = *lower-anonyevt1* (*last el*)
by (*metis b4 cpts-e-not-empty last-map lower-evts-def*)
from *b9* **have** *c2*: *getspc-p* (*last pl*) = *fin-com* ⟶ *gets-p* (*last pl*) ∈ *post* **by** (*simp add*:*commit-p-def*)
from *c0 c1* **have** *c3*: *getspc-p* (*last pl*) = *fin-com*
by (*simp add*: *getspc-p-def lower-anonyevt1-def*)

with *c2* **have** *c4*: *gets-p* (*last pl*) $\in$ *post* **by** *auto*
from *c0 c1* **have** *gets-p* (*last pl*) = *gets-e* (*last el*)
by (*simp add*: *getspc-p-def lower-anonyevt1-def gets-p-def*)
with *c4* **show** *gets-e* (*last el*) $\in$ *post* **by** *simp*
qed

with *b10* **have** *el* $\in$ *commit-e* Γ (*guar*, *post*) **by** (*simp add*:*commit-e-def*)

}
then show *?thesis* **by** *auto*
qed

then have (*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\subseteq$ *commit-e* Γ (*guar*, *post*) **by** *auto*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*: *evt-validity-def*)
qed

lemma *BasicEvt-sound*:
$[\![$ $\Gamma \models$ (*body ev*) sat_p [*pre* $\cap$ (*guard ev*), *rely*, *guar*, *post*];
stable pre rely; $\forall$ *s*. (*s*, *s*)$\in$*guar*$]\!]$
$\Longrightarrow$ $\Gamma \models$ ((*BasicEvent ev*)::('*l*,'*k*,'*s*,'*prog*) *event*) sat_e [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: $\Gamma \models$ (*body ev*) sat_p [*pre* $\cap$ (*guard ev*), *rely*, *guar*, *post*]
assume *p1*: $\forall$ *s*. (*s*, *s*)$\in$*guar*
assume *p2*: *stable pre rely*
have $\forall$ *s x*. (*cpts-of-ev* Γ ((*BasicEvent ev*)::('*l*,'*k*,'*s*,'*prog*) *event*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
$\subseteq$ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *s x*
have $\forall$ *el*. *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\longrightarrow$ *el*$\in$ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *el*
assume *b0*: *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
then have *b0-1*: *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) **and**
b0-2: *el* $\in$ *assume-e* Γ (*pre*, *rely*) **by** *auto*
from *b0-1* **have** *b1*: *el* ! *0* = (*BasicEvent ev*, (*s*, *x*)) **and**
b2: *el* $\in$ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)+
from *b0-2* **have** *b3*: *gets-e* (*el*!*0*) $\in$ *pre* **and**
b4: ($\forall$ *i*. *Suc i*<*length el* $\longrightarrow$ $\Gamma \vdash$ *el*!*i* −*ee*→ *el*!(*Suc i*) $\longrightarrow$
(*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *rely*) **by** (*simp add*: *assume-e-def*)+

have $el \in$ *commit-e* Γ (*guar*, *post*)
proof(*cases* $\exists i\ k.$ *Suc* $i <$ *length* $el \wedge \Gamma \vdash el\ !\ i\ -et-($*EvtEnt* (*BasicEvent* $ev))\sharp k\rightarrow el\ !\ ($*Suc* $i))$
assume *c0*: $\exists i\ k.$ *Suc* $i <$ *length* $el \wedge \Gamma \vdash el\ !\ i\ -et-($*EvtEnt* (*BasicEvent* $ev))\sharp k\rightarrow el\ !\ ($*Suc* $i)$
then obtain m **and** k **where** *c1*: *Suc* $m <$ *length* $el \wedge \Gamma \vdash el\ !\ m\ -et-($*EvtEnt* (*BasicEvent* $ev))\sharp k\rightarrow el\ !\ ($*Suc* $m)$
by *auto*
with *b1 b2* **have** *c2*: $\forall j.$ *Suc* $j \le m \longrightarrow$ *getspc-e* $(el\ !\ j) =$ *BasicEvent* $ev \wedge \Gamma \vdash el\ !\ j\ -ee\rightarrow el\ !\ ($*Suc* $j)$
by (*meson evtent-in-cpts1*)
from *b1 b2 c1* **have** *c4*: *gets-e* $(el\ !\ m) \in$ *guard* ev **and**
c6: *drop* (*Suc* m) $el \in$ *cpts-of-ev* Γ (*AnonyEvent* (*body* ev)) (*gets-e* $(el\ !\ ($*Suc* $m)))$ ((*getx-e* $(el\ !\ m)$) ($k :=$ *BasicEvent* ev))
using *evtent-in-cpts2*[*of el* Γ *ev s x m k*] **by** *auto*

from *p0*[*rule-format*] *c4* **have** *c7*: $\Gamma \models$ ((*AnonyEvent* (*body* ev))::('*l*,'*k*,'*s*,'*prog*) *event*)
sat_e [*pre* $\cap$ (*guard* ev), *rely*, *guar*, *post*]
by (*simp add*: *AnonyEvt-sound*)

from *b4 c1 c2* **have** *c8*:$\forall j.$ *Suc* $j \le m \longrightarrow$ (*gets-e* $(el\ !\ j)$, *gets-e* $(el\ !\ ($*Suc* $j))) \in$ *rely* **by** *auto*
with *p2 b3* **have** *c9*: $\forall j.\ j \le m \longrightarrow$ *gets-e* $(el\ !\ j) \in$ *pre*
proof −
{
fix j
assume *d0*: $j \le m$
then have *gets-e* $(el\ !\ j) \in$ *pre*
proof(*induct* j)
case *0* **show** *?case* **by** (*simp add*: *b3*)
next
case (*Suc jj*)
assume *e0*: *Suc* $jj \le m$
assume *e1*: $jj \le m \Longrightarrow$ *gets-e* $(el\ !\ jj) \in$ *pre*
from *e0 c8* **have** (*gets-e* $(el\ !\ jj)$, *gets-e* $(el\ !\ ($*Suc* $jj))) \in$ *rely* **by** *auto*
with *p2 e0 e1* **show** *?case* **by** (*meson Suc-leD stable-def*)
qed
}
then show *?thesis* **by** *auto*
qed
from *c1* **have** *c10*: *gets-e* $(el\ !\ m) =$ *gets-e* $(el\ !\ ($*Suc* $m))$ **by** (*meson ent-spec2*)
with *c9* **have** *c11*: *gets-e* $(el\ !\ ($*Suc* $m)) \in$ *pre* **by** *auto*
from *c7* **have** *c12*: $\forall s\ x.$ (*cpts-of-ev* Γ ((*AnonyEvent* (*body* ev))::('*l*,'*k*,'*s*,'*prog*) *event*) $s\ x) \cap$
assume-e Γ (*pre* $\cap$ (*guard* ev), *rely*) $\subseteq$ *commit-e* Γ (*guar*, *post*) **by** (*simp add*:*evt-validity-def*)

```
          have drop (Suc m) el ∈ assume-e Γ (pre ∩ (guard ev), rely)
            proof -
              from c11 have d1: gets-e (drop (Suc m) el ! 0) ∈ pre using c1
by auto
              from c4 c10 have d2: gets-e (drop (Suc m) el ! 0) ∈ guard ev
                using c1 by auto
              from b4 have d3: ∀ i. Suc i < length el − Suc m ⟶
                       Γ ⊢ el ! Suc (m + i) −ee→ el ! Suc (Suc (m + i)) ⟶
                      (gets-e (el ! Suc (m + i)), gets-e (el ! Suc (Suc (m + i))))
∈ rely
                  by simp
              with d1 d2 show ?thesis by (simp add:assume-e-def)
            qed

          with c6 c12 have c13: drop (Suc m) el ∈ commit-e Γ (guar, post)
           by (meson AnonyEvt-sound IntI contra-subsetD evt-validity-def p0)


          have c14: ∀ i. Suc i < length el ⟶ (∃ t. Γ ⊢ el ! i −et−t→ el ! Suc
i)
              ⟶ (gets-e (el ! i), gets-e (el ! Suc i)) ∈ guar
            proof -
            {
              fix i
              assume d0: Suc i < length el and
                     d1: (∃ t. Γ ⊢ el ! i −et−t→ el ! Suc i)
              then have (gets-e (el ! i), gets-e (el ! Suc i)) ∈ guar
                proof(cases Suc i ≤ m)
                  assume e0: Suc i ≤ m
                  with c2 have Γ ⊢ el ! i −ee→ el ! (Suc i) by auto
                  then have ¬(∃ t. Γ ⊢ el ! i −et−t→ el ! Suc i)
                    by (metis eetranE evt-not-eq-in-tran prod.collapse)
                  with d1 show ?thesis by simp
                next
                  assume e0: ¬ Suc i ≤ m
                  then have e1: Suc i > m by auto
                  show ?thesis
                    proof(cases Suc i = m + 1)
                      assume f0: Suc i = m + 1
                      then have f1: i = m by auto
                    with c1 have Γ ⊢ el ! i −et−(EvtEnt (BasicEvent ev))♯k→
el ! (Suc i) by simp
                     then have gets-e (el ! i) = gets-e (el ! (Suc i)) by (meson
ent-spec2)
                      with p1 show ?thesis by auto
                    next
                      assume f0: ¬ Suc i = m + 1
```

with *e1* **have** *f1*: *Suc i* $>$ *Suc m* **by** *auto*
from *c13* **have** *f2*: $\forall$ *i*. *Suc i* $<$ *length* (*drop* (*Suc m*) *el*) $\longrightarrow$
($\exists$ *t*. $\Gamma \vdash$ (*drop* (*Suc m*) *el*) ! *i* $-et-t\rightarrow$ (*drop* (*Suc m*) *el*) ! *Suc i*) $\longrightarrow$
(*gets-e* ((*drop* (*Suc m*) *el*) ! *i*), *gets-e* ((*drop* (*Suc m*) *el*) ! *Suc i*)) $\in$ *guar*
by (*simp add*:*commit-e-def*)
with *d0 d1 f1* **have** (*gets-e* (*drop* (*Suc m*) *el* ! (*i* $-$ *Suc m*)), *gets-e* (*drop* (*Suc m*) *el* ! *Suc* (*i* $-$ *Suc m*))) $\in$ *guar*
proof $-$
from *d0 f1* **have** *g0*: *Suc* (*i* $-$ *Suc m*) $<$ *length* (*drop* (*Suc m*) *el*) **by** *auto*
from *d1 f1* **have** ($\exists$ *t*. $\Gamma \vdash$ *drop* (*Suc m*) *el* ! (*i* $-$ *Suc m*) $-et-t\rightarrow$ *drop* (*Suc m*) *el* ! *Suc* (*i* $-$ *Suc m*))
using *d0* **by** *auto*
with *g0 f2* **show** *?thesis* **by** *simp*
qed
then show *?thesis*
using *c1 f1* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed

from *c13* **have** *c15*: *getspc-e* (*last el*) = *AnonyEvent fin-com* $\longrightarrow$ *gets-e* (*last el*) $\in$ *post*
proof $-$
from *c1* **have** *last* (*drop* (*Suc m*) *el*) = *last el* **by** *simp*
with *c13* **show** *?thesis* **by** (*simp add*:*commit-e-def*)
qed

from *c14 c15* **show** *?thesis* **by** (*simp add*:*commit-e-def*)
next
assume *c0*: $\neg$ ($\exists$ *i k*. *Suc i* $<$ *length el* $\wedge$ $\Gamma \vdash$ *el* ! *i* $-et-$(*EvtEnt* (*BasicEvent ev*))$\sharp k\rightarrow$ *el* ! (*Suc i*))
with *b1 b2* **have** *c1*: $\forall$ *j*. *Suc j* $<$ *length el* $\longrightarrow$ *getspc-e* (*el* ! *j*) = *BasicEvent ev*
$\wedge$ $\Gamma \vdash$ *el* ! *j* $-ee\rightarrow$ *el* ! (*Suc j*)
$\wedge$ *getspc-e* (*el* ! (*Suc j*)) = *BasicEvent ev*
using *no-evtent-in-cpts* **by** *simp*
then have *c2*: ($\forall$ *i*. *Suc i*$<$*length el* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *el*!*i* $-et-t\rightarrow$ *el*!(*Suc i*))
$\longrightarrow$ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar*)
proof $-$
{
fix *i*

assume *Suc i<length el*
and *d0*: $\exists\, t.\ \Gamma \vdash el!i\ -et-t\rightarrow\ el!(Suc\ i)$
with *c1* **have** $\Gamma \vdash el\ !\ i\ -ee\rightarrow\ el\ !\ Suc\ i$ **by** *auto*
then have $\neg\ (\exists\, t.\ \Gamma \vdash el!i\ -et-t\rightarrow\ el!(Suc\ i))$
by (*metis eetranE evt-not-eq-in-tran2 prod.collapse*)
with *d0* **have** *False* **by** *simp*
}
then show *?thesis* **by** *auto*
qed
from *b1 b2* **have** *el* ≠ [] **using** *cpts-e-not-empty* **by** *auto*
with *b1 b2* **obtain** *els* **where** *el* = (*BasicEvent ev*, *s*, *x*) # *els*
by (*metis hd-Cons-tl hd-conv-nth*)
then have *getspc-e* (*last el*) = *BasicEvent ev*
proof(*induct els*)
case *Nil*
assume *el* = [(*BasicEvent ev*, *s*, *x*)]
then have *last el* = (*BasicEvent ev*, *s*, *x*) **by** *simp*
then show *?case* **by** (*simp add*:*getspc-e-def*)
next
case (*Cons els1 elsr*)
assume *d0*: *el* = (*BasicEvent ev*, *s*, *x*) # *els1* # *elsr*
then have *d1*: *length el* > *1* **by** *simp*
with *d0* **obtain** *mm* **where** *d2*: *Suc mm* = *length el* **by** *simp*
with *d1* **obtain** *jj* **where** *d3*: *Suc jj* = *mm* **using** *d0* **by** *auto*
with *d2* **have** *d4*: *last el* = *el* ! *mm*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc drop-eq-Nil last-ConsL last-drop le-eq-less-or-eq lessI*)
with *c1* **have** *getspc-e* (*el* ! (*Suc jj*)) = *BasicEvent ev* **using** *d2 d3* **by** *auto*
with *d3 d4* **show** *?case* **by** *simp*
qed

then have *c3*: *getspc-e* (*last el*) = *AnonyEvent fin-com* ⟶ *gets-e* (*last el*) ∈ *post* **by** *simp*

with *c2* **show** *?thesis* **by** (*simp add*:*commit-e-def*)
qed
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*: *evt-validity-def*)
qed

lemma *ev-seq-sound*:
⟦*pre* ⊆ *pre′*; *rely* ⊆ *rely′*; *guar′* ⊆ *guar*; *post′* ⊆ *post*;

$\Gamma \models ev\ sat_e\ [pre', rely', guar', post']]\!]$
$\Longrightarrow \Gamma \models ev\ sat_e\ [pre, rely, guar, post]$
proof −
assume *p0*: $pre \subseteq pre'$
and *p1*: $rely \subseteq rely'$
and *p2*: $guar' \subseteq guar$
and *p3*: $post' \subseteq post$
and *p4*: $\Gamma \models ev\ sat_e\ [pre', rely', guar', post']$
from *p4* **have** *p5*: $\forall s\ x.\ (cpts\text{-}of\text{-}ev\ \Gamma\ ev\ s\ x) \cap assume\text{-}e\ \Gamma\ (pre', rely') \subseteq commit\text{-}e\ \Gamma\ (guar', post')$
by (*simp add*: *evt-validity-def*)
have $\forall s\ x.\ (cpts\text{-}of\text{-}ev\ \Gamma\ ev\ s\ x) \cap assume\text{-}e\ \Gamma\ (pre, rely) \subseteq commit\text{-}e\ \Gamma\ (guar, post)$
proof −
{
fix *c s x*
assume *a0*: $c \in (cpts\text{-}of\text{-}ev\ \Gamma\ ev\ s\ x) \cap assume\text{-}e\ \Gamma\ (pre, rely)$
then have $c \in (cpts\text{-}of\text{-}ev\ \Gamma\ ev\ s\ x) \wedge c \in assume\text{-}e\ \Gamma\ (pre, rely)$ **by** *simp*
with *p0 p1* **have** $c \in (cpts\text{-}of\text{-}ev\ \Gamma\ ev\ s\ x) \wedge c \in assume\text{-}e\ \Gamma\ (pre', rely')$
using *assume-e-imp*[*of pre pre' rely rely' c*] **by** *simp*
with *p5* **have** $c \in commit\text{-}e\ \Gamma\ (guar', post')$ **by** *auto*
with *p2 p3* **have** $c \in commit\text{-}e\ \Gamma\ (guar, post)$
using *commit-e-imp*[*of guar' guar post' post c*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*evt-validity-def*)
qed

theorem *rgsound-e*:
$\Gamma \vdash Evt\ sat_e\ [pre, rely, guar, post] \Longrightarrow \Gamma \models Evt\ sat_e\ [pre, rely, guar, post]$
apply(*erule rghoare-e.induct*)
apply (*simp add*: *AnonyEvt-sound rgsound-p*)
apply (*meson BasicEvt-sound rgsound-p*)
apply (*simp add*: *ev-seq-sound rgsound-p*)
done

7.4 Soundness of Event Systems

lemma *evtseq-nfin-samelower*: $[\![esl \in cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x; \forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) \neq es]\!]$
$\Longrightarrow (\exists el.\ (el \in cpts\text{-}of\text{-}ev\ \Gamma\ e\ s\ x \wedge length\ esl = length\ el \wedge e\text{-}eqv\text{-}einevtseq\ esl\ el\ es))$
proof −
assume *p0*: $esl \in cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x$
and *p1*: $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) \neq es$
from *p0* **have** *p01*: $esl\ !\ 0 = (EvtSeq\ e\ es, s, x) \wedge esl \in cpts\text{-}es\ \Gamma$ **by** (*simp add*: *cpts-of-es-def*)
then have *p01-1*: $esl\ !\ 0 = (EvtSeq\ e\ es, s, x)$ **by** *simp*

then have *p2*: $\exists$ *e. getspc-es* (*esl* ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
from *p01* **have** *p01-2*: *esl* $\in$ *cpts-es* Γ **by** *simp*
let *?el* = *rm-evtsys esl*
have *a1*: *length esl* = *length ?el* **by** (*simp add*: *rm-evtsys-def*)
moreover have *?el* $\in$ *cpts-of-ev* Γ *e s x*
proof −
from *p01-2 p1 p2* **have** *b1*: *?el* $\in$ *cpts-ev* Γ
proof(*induct esl*)
case (*CptsEsOne es1 s1 x1*)
assume *c0*: $\exists$ *e. getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e es*
then obtain *e1* **where** *c1*: *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e1 es* **by** *auto*
then have *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
then have *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
then have *rm-evtsys* [(*es1*, *s1*, *x1*)] = [(*e1*, *s1*, *x1*)] **by** (*simp add*:*rm-evtsys-def*)
then show *?case* **by** (*simp add*: *cpts-ev.CptsEvOne*)
next
case (*CptsEsEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* $\in$ *cpts-es* Γ
and *c1*: $\forall$ *i. Suc i* $\leq$ *length* ((*es1*, *t1*, *x1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
$\Longrightarrow$$\exists$ *e. getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es*
$\Longrightarrow$ *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) $\in$ *cpts-ev* Γ
and *c11*: $\forall$ *i. Suc i* $\leq$ *length* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*)
$\longrightarrow$ *getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
and *c2*: $\exists$ *e. getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *c2* **obtain** *e1* **where** *c3*: *getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e1 es* **by** *auto*
then have *c4*: *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
from *c11* **have** $\forall$ *i. Suc i* $\leq$ *length* ((*es1*, *t1*, *x1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
by *auto*
with *c1 c4* **have** *c5*: *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) $\in$ *cpts-ev* Γ **by** (*simp add*:*getspc-es-def*)
have *c6*: *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) = (*rm-evtsys1* (*es1*, *t1*, *x1*)) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)
have *c7*: *rm-evtsys* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) =
(*rm-evtsys1* (*es1*, *s1*, *y1*)) # (*rm-evtsys1* (*es1*, *t1*, *x1*)) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)
from *c4* **have** *c8*: *rm-evtsys1* (*es1*, *s1*, *y1*) = (*e1*, *s1*, *y1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
from *c4* **have** *c9*: *rm-evtsys1* (*es1*, *t1*, *x1*) = (*e1*, *t1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)

have *c10*: *rm-evtsys* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) = (*e1*, *s1*, *y1*) # (*e1*, *t1*, *x1*) # *rm-evtsys xs1*
by (*simp add*: *c7 c8 c9*)
have *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) = (*e1*, *t1*, *x1*) # *rm-evtsys xs1*
by (*simp add*: *c6 c9*)
with *c5 c10* **show** *?case* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
next
case (*CptsEsComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *c0*: Γ ⊢ (*es1*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-es* Γ
and *c2*: ∀ *i*. *Suc i* ≤ *length* ((*es2*, *t1*, *y1*) # *xs1*) ⟶ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) ≠ *es*
⟹ ∃ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
⟹ *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*) ∈ *cpts-ev* Γ
and *c3*: ∀ *i*. *Suc i* ≤ *length* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*)
⟶ *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *i*) ≠ *es*
and *c4*: ∃ *e*. *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *c4* **obtain** *e1* **where** *c41*: *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e1 es*
by *auto*
then have *c5*: *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
from *c3* **have** *getspc-es* (*es2*, *t1*, *y1*) ≠ *es* **by** *auto*
then have *c6*: *es2* ≠ *es* **by** (*simp add*:*getspc-es-def*)

with *c0 c5* **have** ∃ *e2*. *es2* = *EvtSeq e2 es* **by** (*meson evtseq-tran-evtsys*)

then obtain *e2* **where** *c7*: *es2* = *EvtSeq e2 es* **by** *auto*
with *c0 c5* **have** ∃ *t*. Γ ⊢ (*e1*,*s1*,*x1*) −*et*−*t*→ (*e2*,*t1*,*y1*) **by** (*simp add*: *evtseq-tran-exist-etran*)
then obtain *t* **where** *c71*: Γ ⊢ (*e1*,*s1*,*x1*) −*et*−*t*→ (*e2*,*t1*,*y1*) **by** *auto*
have *c8*: *rm-evtsys* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) =
(*rm-evtsys1* (*es1*, *s1*, *x1*)) # (*rm-evtsys1* (*es2*, *t1*, *y1*)) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)
have *c9*: *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*) = *rm-evtsys1* (*es2*, *t1*, *y1*) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)

from *c3* **have** *c10*: ∀ *i*. *Suc i* ≤ *length* ((*es2*, *t1*, *y1*) # *xs1*) ⟶ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) ≠ *es*
by *auto*
from *c7* **have** ∃ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
by (*simp add*:*getspc-es-def*)
with *c2 c10* **have** *c11*: *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*) ∈ *cpts-ev* Γ **by** *auto*
from *c5* **have** *c12*: *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)

by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
from *c7* **have** *c13*: *rm-evtsys1* (*es2*, *t1*, *y1*) = (*e2*, *t1*, *y1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
with *c71 c8 c9 c11 c12* **show** *?case* **using** *cpts-ev.CptsEvComp* **by** *fastforce*
qed
moreover have *?el* ! *0*=(*e*,(*s*,*x*))
proof −
from *p01* **have** *rm-evtsys1* (*esl* ! *0*) = (*e*, *s*, *x*)
by (*simp add*: *gets-es-def getspc-es-def getx-es-def rm-evtsys1-def*)
moreover from *a1 b1* **have** *?el* ! *0* = *rm-evtsys1* (*esl* ! *0*) **using** *rm-evtsys-def*
by (*metis cpts-e-not-empty length-greater-0-conv nth-map*)
ultimately show *?thesis* **by** *simp*
qed
ultimately have *?el* ! *0*=(*e*,(*s*,*x*)) ∧ *?el* ∈ *cpts-ev* Γ **by** *auto*
then show *?thesis* **by** (*simp add*: *cpts-of-ev-def*)
qed
moreover from *p01-2 p1 p2* **have** *e-eqv-einevtseq esl ?el es*
proof(*induct esl*)
case (*CptsEsOne es1 s1 x1*)
assume *a0*: ∃ *e*. *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e es*
then obtain *e1* **where** *a1*: *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e1 es* **by** *auto*
then have *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
then have *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
then have *a2*: *rm-evtsys* [(*es1*, *s1*, *x1*)] = [(*e1*, *s1*, *x1*)] **by** (*simp add*:*rm-evtsys-def*)
show *?case*
proof(*simp add*:*e-eqv-einevtseq-def*, *rule conjI*)
show *b0*: *Suc 0* = *length* (*rm-evtsys* [(*es1*, *s1*, *x1*)]) **by** (*simp add*: *a2*)
moreover
from *a2* **have** *gets-e* (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *0*) = *gets-es* ([(*es1*, *s1*, *x1*)] ! *0*)
by (*simp add*: *gets-es-def rm-evtsys1-def gets-e-def*)
moreover
from *a2* **have** *getx-e* (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *0*) = *getx-es* ([(*es1*, *s1*, *x1*)] ! *0*)
by (*simp add*: *getx-es-def rm-evtsys1-def getx-e-def*)
moreover
from *a2* **have** *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq* (*getspc-e* (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *0*)) *es*
using *getspc-es-def getspc-e-def* **by** (*metis a1 fst-conv nth-Cons-0*)
ultimately show ∀ *i*. *Suc i* ≤ *length* (*rm-evtsys* [(*es1*, *s1*, *x1*)]) ⟶
gets-e (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *i*) = *gets-es* ([(*es1*, *s1*, *x1*)] ! *i*) ∧
getx-e (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *i*) = *getx-es* ([(*es1*, *s1*, *x1*)] ! *i*) ∧

getspc-es ([(*es1*, *s1*, *x1*)] ! *i*) = *EvtSeq* (*getspc-e* (*rm-evtsys* [(*es1*, *s1*, *x1*)] ! *i*)) *es*
by (*metis One-nat-def Suc-le-lessD less-one*)
qed
next
case (*CptsEsEnv es1 t1 x1 xs1 s1 y1*)
assume *a0*: (*es1*, *t1*, *x1*) # *xs1* $\in$ *cpts-es* Γ
and *a1*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *t1*, *x1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es* $\Longrightarrow$
$\exists$ *e*. *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es* $\Longrightarrow$
e-eqv-einevtseq ((*es1*, *t1*, *x1*) # *xs1*) (*rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*)) *es*
and *a2*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*)
$\longrightarrow$ *getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
and *a3*: $\exists$ *e*. *getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *a2* **have** *a4*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *t1*, *x1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
by *auto*
from *a3* **obtain** *e1* **where** *a5*: *es1* = *EvtSeq e1 es* **using** *getspc-es-def* **by** (*metis fst-conv nth-Cons-0*)
then have $\exists$ *e*. *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es*
using *getspc-es-def* **by** (*simp add*: *getspc-es-def*)
with *a1 a4* **have** *a6*: *e-eqv-einevtseq* ((*es1*, *t1*, *x1*) # *xs1*) (*rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*)) *es* **by** *simp*
from *a5* **have** *a7*: *rm-evtsys1* (*es1*, *s1*, *y1*) = (*e1*, *s1*, *y1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
have *rm-evtsys* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) =
rm-evtsys1 (*es1*, *s1*, *y1*) # *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) **by** (*simp add*: *rm-evtsys-def*)
with *a6 a7* **show** *?case* **using** *gets-e-def gets-es-def getx-e-def getx-es-def getspc-es-def getspc-e-def e-eqv-einevtseq-s* **by** (*metis a5 fst-conv snd-conv*)
next
case (*CptsEsComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *a0*: $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-es-et\rightarrow$ (*es2*, *t1*, *y1*)
and *a1*: (*es2*, *t1*, *y1*) # *xs1* $\in$ *cpts-es* Γ
and *a2*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es2*, *t1*, *y1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es* $\Longrightarrow$
$\exists$ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es* $\Longrightarrow$
e-eqv-einevtseq ((*es2*, *t1*, *y1*) # *xs1*) (*rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)) *es*
and *a3*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*)
$\longrightarrow$ *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es*
and *a4*: $\exists$ *e*. *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *a3* **have** *a5*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es2*, *t1*, *y1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es*
by *auto*
from *a4* **obtain** *e1* **where** *a6*: *es1* = *EvtSeq e1 es* **using** *getspc-es-def* **by**

(*metis fst-conv nth-Cons-0*)
from *a3* **have** *getspc-es* (*es2*, *t1*, *y1*) $\neq$ *es* **by** *auto*
then have *a7*: *es2* $\neq$ *es* **by** (*simp add*:*getspc-es-def*)
with *a0 a6* **have** $\exists$ *e2*. *es2* = *EvtSeq e2 es* **by** (*meson evtseq-tran-evtsys*)
then obtain *e2* **where** *a8*: *es2* = *EvtSeq e2 es* **by** *auto*
then have *a9*: $\exists$ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
with *a2 a5* **have** *a10*: *e-eqv-einevtseq* ((*es2*, *t1*, *y1*) # *xs1*) (*rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)) *es* **by** *simp*
have *a11*: *rm-evtsys* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) = *rm-evtsys1* (*es1*, *s1*, *x1*) # *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)
by (*simp add*:*rm-evtsys-def*)
from *a6* **have** *a12*: *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
with *a6 a11 a10* **show** *?case* **using** *gets-e-def gets-es-def getx-e-def getx-es-def*

getspc-es-def getspc-e-def e-eqv-einevtseq-s **by** (*metis fst-conv snd-conv*)
qed

ultimately have *?el* $\in$ *cpts-of-ev* Γ *e s x* $\wedge$ *length esl* = *length ?el* $\wedge$ *e-eqv-einevtseq esl ?el es* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *evtseq-fst-finish*:
$[\![$*esl* $\in$ *cpts-es* Γ; *getspc-es* (*esl* ! *0*) = *EvtSeq e es*; *Suc m* $\leq$ *length esl*;
$\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*$]\!]$ $\Longrightarrow$
$\exists$ *i*. (*i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\wedge$ ($\forall$ *j*. *j* < *i* $\longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
proof –
assume *p0*: *esl* $\in$ *cpts-es* Γ
and *p1*: *getspc-es* (*esl* ! *0*) = *EvtSeq e es*
and *p2*: *Suc m* $\leq$ *length esl*
and *p3*: $\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*
have $\forall$ *m*. *esl* $\in$ *cpts-es* Γ $\wedge$ *getspc-es* (*esl* ! *0*) = *EvtSeq e es* $\wedge$ *Suc m* $\leq$ *length esl* $\wedge$
($\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\longrightarrow$
($\exists$ *i*. (*i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\wedge$ ($\forall$ *j*. *j* < *i* $\longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*))
proof –
{
fix *m*
assume *a0*: *esl* $\in$ *cpts-es* Γ
and *a1*: *getspc-es* (*esl* ! *0*) = *EvtSeq e es*
and *a2*: *Suc m* $\leq$ *length esl*
and *a3*: ($\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*)
then have $\exists$ *i*. (*i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\wedge$ ($\forall$ *j*. *j* < *i* $\longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
proof(*induct m*)

case *0* **show** *?case* **using** *0.prems(4)* **by** *auto*
next
case (*Suc n*)
assume *b0*: $esl \in cpts\text{-}es\ \Gamma \Longrightarrow$
$getspc\text{-}es\ (esl\ !\ 0) = EvtSeq\ e\ es \Longrightarrow$
$Suc\ n \leq length\ esl \Longrightarrow$
$\exists i \leq n.\ getspc\text{-}es\ (esl\ !\ i) = es \Longrightarrow$
$\exists i.\ (i \leq n \wedge getspc\text{-}es\ (esl\ !\ i) = es) \wedge (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) \neq es)$
and *b1*: $esl \in cpts\text{-}es\ \Gamma$
and *b2*: $getspc\text{-}es\ (esl\ !\ 0) = EvtSeq\ e\ es$
and *b3*: $Suc\ (Suc\ n) \leq length\ esl$
and *b4*: $\exists i \leq Suc\ n.\ getspc\text{-}es\ (esl\ !\ i) = es$
show *?case*
proof(*cases* $\exists i \leq n.\ getspc\text{-}es\ (esl\ !\ i) = es$)
assume *c0*: $\exists i \leq n.\ getspc\text{-}es\ (esl\ !\ i) = es$
with *b0 b1 b2 b3* **have** $\exists i.\ (i \leq n \wedge getspc\text{-}es\ (esl\ !\ i) = es) \wedge (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) \neq es)$
using *Suc-leD* **by** *blast*
then show *?case* **using** *le-Suc-eq* **by** *blast*
next
assume *c0*: $\neg\ (\exists i \leq n.\ getspc\text{-}es\ (esl\ !\ i) = es)$
with *b4* **have** $getspc\text{-}es\ (esl\ !\ (Suc\ n)) = es$ **using** *le-SucE* **by** *auto*
moreover from *c0* **have** $\forall j.\ j < Suc\ n \longrightarrow getspc\text{-}es\ (esl\ !\ j) \neq es$ **by** *auto*
ultimately show *?case* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed

then show *?thesis* **using** *p0 p1 p2 p3* **by** *blast*
qed

lemma *EventSeq-sound* :
$[\![\ \Gamma \models e\ sat_e\ [pre,\ rely1,\ guar1,\ post1];\ \Gamma \models es\ sat_s\ [pre2,\ rely2,\ guar2,\ post];$
$rely \subseteq rely1;\ rely \subseteq rely2;\ guar1 \subseteq guar;\ guar2 \subseteq guar;\ post1 \subseteq pre2\ ]\!]$
$\Longrightarrow \Gamma \models EvtSeq\ e\ es\ sat_s\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: $\Gamma \models e\ sat_e\ [pre,\ rely1,\ guar1,\ post1]$
and *p1*: $\Gamma \models es\ sat_s\ [pre2,\ rely2,\ guar2,\ post]$
and *p2*: $rely \subseteq rely1$
and *p3*: $rely \subseteq rely2$
and *p4*: $guar1 \subseteq guar$
and *p5*: $guar2 \subseteq guar$
and *p6*: $post1 \subseteq pre2$
then have $\forall s\ x.\ (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}es\ \Gamma\ (guar,\ post)$

proof –
{
fix *s x*
have ∀ *esl. esl*∈(*cpts-of-es* Γ (*EvtSeq e es*) *s x*) ∩ *assume-es* Γ (*pre*, *rely*) ⟶ *esl*∈ *commit-es* Γ (*guar*, *post*)
proof –
{
fix *esl*
assume *a0*: *esl* ∈ (*cpts-of-es* Γ (*EvtSeq e es*) *s x*) ∩ *assume-es* Γ (*pre*, *rely*)
then have *a01*: *esl* ∈ *cpts-of-es* Γ (*EvtSeq e es*) *s x* **by** *simp*
from *a0* **have** *a02*: *esl* ∈ *assume-es* Γ (*pre*, *rely*) **by** *auto*

from *a01* **have** *a01-1*: *esl* ! *0* = (*EvtSeq e es*, *s*, *x*) **by** (*simp add*: *cpts-of-es-def*)
from *a01* **have** *a01-2*: *esl* ∈ *cpts-es* Γ **by** (*simp add*: *cpts-of-es-def*)

have *esl*∈ *commit-es* Γ (*guar*, *post*)
proof(*cases* ∀ *i*. *Suc i* ≤ *length esl* ⟶ *getspc-es* (*esl* ! *i*) ≠ *es*)
assume *b0*: ∀ *i*. *Suc i* ≤ *length esl* ⟶ *getspc-es* (*esl* ! *i*) ≠ *es*
with *a01* **have** ∃ *el*. (*el* ∈ *cpts-of-ev* Γ *e s x* ∧ *length esl* = *length el* ∧ *e-eqv-einevtseq esl el es*)
by (*simp add*: *evtseq-nfin-samelower*)
then obtain *el* **where** *b1*: *el* ∈ *cpts-of-ev* Γ *e s x* ∧ *length esl* = *length el* ∧ *e-eqv-einevtseq esl el es*
by *auto*
have *el* ∈ *assume-e* Γ (*pre*, *rely1*)
proof(*simp add*:*assume-e-def*, *rule conjI*)
from *a02* **have** *c0*: *gets-es* (*esl* ! *0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
moreover
from *b1* **have** *gets-e* (*el* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-e-def*)
moreover
from *a01-1* **have** *gets-es* (*esl* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-es-def*)
ultimately show *gets-e* (*el* ! *0*) ∈ *pre* **by** *simp*
next
show ∀ *i*. *Suc i* < *length el* ⟶ Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i* ⟶
(*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely1*
proof –
{
fix *i*
assume *c0*:*Suc i* < *length el*
and *c1*: Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i*
then have *c2*: *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *eetran-eqconf1*)
moreover from *b1 c0* **have** *getspc-es* (*esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*

by (*simp add*: *e-eqv-einevtseq-def*)
moreover from *b1 c0* **have** *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
ultimately have *c3*: *getspc-es* (*esl* ! *i*) = *getspc-es* (*esl* ! *Suc i*) **by** *simp*

then have $\Gamma \vdash$ *esl* ! *i* $-ese\rightarrow$ *esl* ! *Suc i* **by** (*simp add*: *eqconf-esetran*)
with *a02 b1 c0* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely*
by (*simp add*: *assume-es-def*)
moreover have *gets-es* (*esl*!*i*) = *gets-e* (*el* ! *i*)
by (*metis b1 c0 e-eqv-einevtseq-def less-imp-le-nat*)
moreover have *gets-es* (*esl*!*Suc i*) = *gets-e* (*el* ! *Suc i*)
by (*metis Suc-le-eq b1 c0 e-eqv-einevtseq-def*)
ultimately have (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) $\in$ *rely* **by** *simp*

with *p2* **have** (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) $\in$ *rely1* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
with *p0 b1* **have** *el* $\in$ *commit-e* Γ (*guar1*, *post1*)
by (*meson IntI contra-subsetD evt-validity-def*)
then have $\forall$ *i*. *Suc i*<*length el* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *el*!*i* $-et-t\rightarrow$ *el*!(*Suc i*))
$\longrightarrow$ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar1* **by** (*simp add*:*commit-e-def*)
with *p4* **have** *b2*: $\forall$ *i*. *Suc i*<*length el* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *el*!*i* $-et-t\rightarrow$ *el*!(*Suc i*))
$\longrightarrow$ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar* **by** *auto*
show *?thesis*
proof(*simp add*:*commit-es-def*)
show $\forall$ *i*. *Suc i* < *length esl* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *esl* ! *i* $-es-t\rightarrow$ *esl* ! *Suc i*)
$\longrightarrow$ (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *guar*
proof −
{
fix *i*
assume *c0*: *Suc i* < *length esl*
and *c1*: ($\exists$ *t*. $\Gamma \vdash$ *esl* ! *i* $-es-t\rightarrow$ *esl* ! *Suc i*)
with *b1* **have** *c2*: *getspc-es* (*esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)

from *b1 c0* **have** *c3*: *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)

from *c1* **have** *getspc-es* (*esl* ! *i*) $\neq$ *getspc-es* (*esl* ! *Suc i*)
using *evtsys-not-eq-in-tran-aux getspc-es-def* **by** (*metis surjective-pairing*)
with *c2 c3* **have** *getspc-e* (*el* ! *i*) $\neq$ *getspc-e* (*el* ! *Suc i*) **by** *simp*
then have $\exists$ *t*. $\Gamma \vdash$ (*el* ! *i*) $-et-t\rightarrow$ (*el* ! *Suc i*)
using *b1 c0 cpts-of-ev-def notran-confeqi* **by** *fastforce*
with *b2* **have** (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar*
using *b1 c0* **by** *auto*
moreover have *gets-e* (*el*!*i*) = *gets-es* (*esl* ! *i*)
using *b1 c0 e-eqv-einevtseq-def less-imp-le* **by** *fastforce*
moreover have *gets-e* (*el*!*Suc i*) = *gets-es* (*esl* ! *Suc i*)
using *Suc-leI b1 c0 e-eqv-einevtseq-def* **by** *fastforce*
ultimately have (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *guar* **by** *simp*
}
then show *?thesis* **by** *auto*
qed
qed
next
assume *b0*: $\neg$ ($\forall$ *i*. *Suc i* $\leq$ *length esl* $\longrightarrow$ *getspc-es* (*esl* ! *i*) $\neq$ *es*)
from *a01-1* **have** *b00*: *getspc-es* (*esl* ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
from *b0* **have** $\exists$ *m*. *Suc m* $\leq$ *length esl* $\wedge$ *getspc-es* (*esl* ! *m*) = *es* **by** *auto*
then obtain *m* **where** *b1*: *Suc m* $\leq$ *length esl* $\wedge$ *getspc-es* (*esl* ! *m*) = *es* **by** *auto*
then have $\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es* **by** *auto*
with *a01-1 a01-2 b00 b1* **have** *b2*: $\exists$ *i*. (*i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\wedge$ ($\forall$ *j*. *j* < *i* $\longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
using *evtseq-fst-finish* **by** *blast*
then obtain *n* **where** *b3*: (*n* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *n*) = *es*) $\wedge$ ($\forall$ *j*. *j* < *n* $\longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
by *auto*
with *b00* **have** *b41*: *n* $\neq$ *0* **by** (*metis* (*no-types*, *hide-lams*) *add.commute add.right-neutral*
add-Suc dual-order.irrefl esys.size(*3*) *le-add1 le-imp-less-Suc*)
then have *b4*: *n* > *0* **by** *auto*
then obtain *esl0* **where** *b5*: *esl0* = *take n esl* **by** *simp*
then have *b5-1*: *length esl0* = *n* **using** *b1 b3 less-le-trans* **by** *auto*
obtain *esl1* **where** *b6*: *esl1* = *drop n esl* **by** *simp*
with *b5* **have** *b7*: *esl0* @ *esl1* = *esl* **by** *simp*
from *a01-2 b1 b3 b4 b5* **have** *b8*: *esl0* $\in$ *cpts-es* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-le-lessD cpts-es-take less-trans*)
from *a01-2 b1 b3 b4 b5 b6* **have** *b9*: *esl1* $\in$ *cpts-es* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-le-lessD cpts-es-dropi le-neq-implies-less less-trans*)

have *b10*: *esl0* ! *0* = (*EvtSeq e es*, *s*, *x*) **by** (*simp add*: *a01-1 b4 b5*)
have *b11*: *getspc-es* (*esl1* ! *0*) = *es* **using** *b1 b3 b6* **by** *auto*

from *b3 b5* **have** *b11-1*: ∀ *i*. *i* < *length esl0* ⟶ *getspc-es* (*esl0* ! *i*) ≠ *es* **by** *auto*
moreover from *b8 b10* **have** *esl0* ∈ *cpts-of-es* Γ (*EvtSeq e es*) *s x* **by** (*simp add*:*cpts-of-es-def*)
ultimately have *b12*: ∃ *el*. (*el* ∈ *cpts-of-ev* Γ *e s x* ∧ *length esl0* = *length el* ∧ *e-eqv-einevtseq esl0 el es*)
by (*simp add*: *evtseq-nfin-samelower*)
then obtain *el* **where** *b12-1*: *el* ∈ *cpts-of-ev* Γ *e s x* ∧ *length esl0* = *length el* ∧ *e-eqv-einevtseq esl0 el es*
by *auto*
then have *b12-2*: *el* ∈ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)

from *a02* **have** *b13*: *gets-es* (*esl*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length esl* ⟶
Γ ⊢ *esl*!*i* −*ese*→ *esl*!(*Suc i*) ⟶ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*)
by (*simp add*:*assume-es-def*)
have *b14*: *esl0* ∈ *assume-es* Γ (*pre*, *rely*)
proof(*simp add*:*assume-es-def*, *rule conjI*)
show *gets-es* (*esl0* ! *0*) ∈ *pre* **using** *a01-1 b10 b13* **by** *auto*
next
from *b5 b13* **show** ∀ *i*. *Suc i* < *length esl0* ⟶ Γ ⊢ *esl0* ! *i* −*ese*→ *esl0* ! *Suc i*
⟶ (*gets-es* (*esl0* ! *i*), *gets-es* (*esl0* ! *Suc i*)) ∈ *rely* **by** *auto*
qed
with *p2* **have** *b15*: *esl0* ∈ *assume-es* Γ (*pre*, *rely1*)
by (*simp add*: *assume-es-def subset-iff*)

have *b16*: *el* ∈ *assume-e* Γ (*pre*, *rely1*)
proof(*simp add*:*assume-e-def*, *rule conjI*)
from *a02* **have** *c0*: *gets-es* (*esl* ! *0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
moreover
from *b12-1* **have** *gets-e* (*el* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-e-def*)
moreover
from *a01-1* **have** *gets-es* (*esl* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-es-def*)
ultimately show *gets-e* (*el* ! *0*) ∈ *pre* **by** *simp*
next
show ∀ *i*. *Suc i* < *length el* ⟶ Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i* ⟶
(*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely1*
proof −
{
fix *i*

assume *c0*:*Suc i* < *length el*
and *c1*: Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i*
then have *c2*: *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *eetran-eqconf1*)
moreover from *b12-1 c0* **have** *getspc-es* (*esl0* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
moreover from *b12-1 c0* **have** *getspc-es* (*esl0* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
ultimately have *c3*: *getspc-es* (*esl0* ! *i*) = *getspc-es* (*esl0* ! *Suc i*) **by** *simp*

then have *c4*: Γ ⊢ *esl0* ! *i* −*ese*→ *esl0* ! *Suc i* **by** (*simp add*: *eqconf-esetran*)
with *b14 b12-1 c0* **have** (*gets-es* (*esl0*!*i*), *gets-es* (*esl0*!*Suc i*)) ∈ *rely*
proof −
from *b14* **have** ∀ *i*. *Suc i*<*length esl0* ⟶ Γ ⊢ *esl0*!*i* −*ese*→ *esl0*!(*Suc i*)
⟶ (*gets-es* (*esl0*!*i*), *gets-es* (*esl0*!*Suc i*)) ∈ *rely*
by (*simp add*:*assume-es-def*)
with *b12-1 c0 c4* **show** *?thesis* **by** *simp*
qed

moreover have *gets-es* (*esl0*!*i*) = *gets-e* (*el* ! *i*)
by (*metis b12-1 c0 e-eqv-einevtseq-def less-imp-le-nat*)
moreover have *gets-es* (*esl0*!*Suc i*) = *gets-e* (*el* ! *Suc i*)
using *b12-1 c0* **by** (*simp add*: *b12-1 c0 e-eqv-einevtseq-def Suc-leI*)
ultimately have (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely* **by** *simp*

with *p2* **have** (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely1* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
have *b17*: *el* ∈ *commit-e* Γ (*guar1*, *post1*)
using *b12-1 b16 evt-validity-def p0* **by** *fastforce*
then have *b18*: ∀ *i*. *Suc i*<*length el* ⟶ (∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*))
⟶ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar1* **by** (*simp add*:*commit-e-def*)
with *p4* **have** *b19*: ∀ *i*. *Suc i*<*length el* ⟶ (∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*))
⟶ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar* **by** *auto*

from *b11* **have** $\exists$ *sn xn. esl1* ! *0* = (*es*, *sn*, *xn*) **using** *getspc-es-def*
by (*metis fst-conv surj-pair*)
then obtain *sn* **and** *xn* **where** *b13*: *esl1* ! *0* = (*es*, *sn*, *xn*) **by** *auto*
with *b9* **have** *esl1* $\in$ *cpts-of-es* Γ *es sn xn* **by** (*simp add*:*cpts-of-es-def*)

have $\forall$ *i. Suc i*<*length esl* $\longrightarrow$ ($\exists$ *t.* $\Gamma \vdash$ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*))
$\longrightarrow$ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *guar*
proof −
{
fix *i*
assume *c0*: *Suc i*<*length esl*
and *c1*: $\exists$ *t.* $\Gamma \vdash$ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*)
have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *guar*
proof(*cases Suc i* < *n*)
assume *d0*: *Suc i* < *n*

with *b5 b5-1 b12-1 c0 c1* **have** *d1*: *getspc-es* (*esl0* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
using *e-eqv-einevtseq-def* **by** (*metis less-imp-le-nat*)

with *b5 b5-1 b12-1 c0 c1* **have** *d2*: *getspc-es* (*esl0* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
using *e-eqv-einevtseq-def* **by** (*metis Suc-le-eq d0*)

from *c1* **have** *d3*: *getspc-es* (*esl* ! *i*) $\neq$ *getspc-es* (*esl* ! *Suc i*)
using *evtsys-not-eq-in-tran-aux getspc-es-def* **by** (*metis surjective-pairing*)

with *d1 d2* **have** *getspc-e* (*el* ! *i*) $\neq$ *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *Suc-lessD b5 d0*)
then have $\exists$ *t.* $\Gamma \vdash$ (*el* ! *i*) −*et*−*t*→ (*el* ! *Suc i*)
using *b12-1 b5-1 cpts-of-ev-def d0 notran-confeqi* **by** *fastforce*

with *b19* **have** (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar*
using *b12-1 b5-1 d0* **by** *auto*
moreover have *gets-e* (*el*!*i*) = *gets-es* (*esl0* ! *i*)
using *b12-1 b5-1 d0 e-eqv-einevtseq-def less-imp-le-nat* **by** *fastforce*
moreover have *gets-e* (*el*!*Suc i*) = *gets-es* (*esl0* ! *Suc i*)
using *Suc-leI b12-1 b5-1 d0 e-eqv-einevtseq-def less-imp-le-nat* **by** *fastforce*
ultimately have (*gets-es* (*esl0* ! *i*), *gets-es* (*esl0* ! *Suc i*)) $\in$ *guar* **by** *simp*

then show *?thesis* **by** (*simp add*: *Suc-lessD b5 d0*)
next
assume *d0*: $\neg$ (*Suc i* < *n*)
from *b5-1 b12-1* **have** *d1*: *getspc-es* (*esl0* ! (*n*−*1*)) = *EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*

by (*simp add*: *b12-1 e-eqv-einevtseq-def b4*)
with *b5* **have** *d1-1*: *getspc-es* (*esl* ! (*n*−*1*)) = *EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*
by (*simp add*: *b4*)
then have ∃ *sn1 xn1*. *esl* ! (*n*−*1*) = (*EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*, *sn1*, *xn1*)
using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
then obtain *sn1* **and** *xn1* **where** *d2*: *esl* ! (*n*−*1*) = (*EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*, *sn1*, *xn1*)
by *auto*

from *b4 b5 b5-1 b12-1* **have** *gets-e* (*el* ! (*n* −*1*)) = *gets-es* (*esl0* ! (*n* −*1*)) ∧
getx-e (*el* ! (*n* −*1*)) = *getx-es* (*esl0* ! (*n* −*1*)) **by** (*simp add*:*e-eqv-einevtseq-def*)
with *b5 d2* **have** *d3*: *el* ! (*n* −*1*) = (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*)
using *gets-e-def gets-es-def getx-e-def getx-es-def getspc-e-def*
by (*metis Suc-diff-1 b4 lessI nth-take prod.collapse snd-conv*)

from *b13* **have** *d4*: *esl* ! *n* = (*es*, *sn*, *xn*) **using** *b6 c0 d0* **by** *auto*

from *a01-2 b1 b3* **have** *d5*: *drop* (*n*−*1*) *esl* ∈ *cpts-es* Γ **using** *cpts-es-dropi*
by (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-le-lessD b5 b5-1*
drop-0 less-or-eq-imp-le neq0-conv not-le take-all zero-less-diff)

with *b1 b3 b4 b6 b9 d2 d4* **have** *d6*: ∃ *est*. Γ ⊢ *esl* ! (*n*−*1*) −*es*−*est*→ *esl* ! *n*
using *incpts-es-impl-evnorcomptran cpts-es-not-empty evtseq-ne-es*
by (*smt Suc-diff-1 Suc-le-lessD a01-2 d1-1 esetran-eqconf1 le-neq-implies-less less-trans*)
with *d2* **have** *d7*: ∃ *t*. Γ ⊢ (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*) −*et*−*t*→(*AnonyEvent fin-com*,*sn*, *xn*)
using *evtseq-tran-0-exist-etran* **using** *d4* **by** *fastforce*
with *b4 b5-1 b12-1 b12-2 d3* **have** *d8*:*el* @ [(*AnonyEvent fin-com*,*sn*, *xn*)] ∈ *cpts-ev* Γ
using *cpts-ev-onemore* **by** *fastforce*
let *?el1* = *el* @ [(*AnonyEvent fin-com*,*sn*, *xn*)]

from *d8* **have** *d9*: *?el1* ∈ *cpts-of-ev* Γ *e s x*
by (*metis* (*no-types*, *lifting*) *append-Cons b12-1 b3 b4 b5-1*
cpts-of-ev-def list.size(*3*) *mem-Collect-eq neq-Nil-conv nth-Cons-0*)
moreover from *b16 d7* **have** *?el1* ∈ *assume-e* Γ (*pre*, *rely1*)
proof −

have *gets-e* (*?el1*!*0*) ∈ *pre*
proof −
from *b16* **have** *gets-e* (*el*!*0*) ∈ *pre* **by** (*simp* *add*:*assume-e-def*)
then show *?thesis* **by** (*metis b12-1 b4 b5-1 nth-append*)
qed
moreover
have ∀ *i*. *Suc i*<*length ?el1* ⟶ Γ ⊢ *?el1*!*i* −*ee*→ *?el1*!(*Suc* *i*) ⟶
(*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *rely1*
proof −
{
fix *i*
assume *e0*: *Suc i*<*length ?el1*
and *e1*: Γ ⊢*?el1*!*i* −*ee*→ *?el1*!(*Suc i*)
from *b16* **have** *e2*: ∀ *i*. *Suc i*<*length el* ⟶ Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*) ⟶
(*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely1* **by** (*simp* *add*:*assume-e-def*)
have (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *rely1*
proof(*cases Suc i* < *length ?el1* − *1*)
assume *f0*: *Suc i* < *length ?el1* − *1*
with *e0 e2* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-diff-1*
Suc-less-eq Suc-mono e1 length-append-singleton *nth-append zero-less-Suc*)
next
assume ¬ (*Suc i* < *length ?el1* − *1*)
then have *f0*: *Suc i* ≥ *length ?el1* − *1* **by** *simp*
with *e0* **have** *f1*: *Suc i* = *length ?el1* − *1* **by** *simp*
then have *f2*: *?el1*!(*Suc i*) = (*AnonyEvent fin-com*, *sn*, *xn*) **by** *simp*
from *f1* **have** *f3*: *?el1*!*i* = (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*)
by (*metis b12-1 b5-1 d3 diff-Suc-1 length-append-singleton* *lessI nth-append*)

with *d7 f2* **have** *getspc-e* (*?el1*!*i*) ≠ *getspc-e* (*?el1*!(*Suc* *i*))

using *evt-not-eq-in-tran-aux* **by** (*metis e1 eetran.cases*)
moreover from *e1* **have** *getspc-e* (*?el1*!*i*) = *getspc-e* (*?el1*!(*Suc i*))

using *eetran-eqconf1* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed

ultimately show *?thesis* **by** (*simp add*:*assume-e-def*)
qed
ultimately have *d10*: *?el1* $\in$ *commit-e* Γ (*guar1*, *post1*)
using *evt-validity-def p0* **by** *fastforce*

have *d11*: *getspc-e* (*last ?el1*) = *AnonyEvent fin-com* **by** (*simp add*:*getspc-e-def*)
with *d10* **have** *d12*: *gets-e* (*last ?el1*) $\in$ *post1* **by** (*simp add*: *commit-e-def*)

show *?thesis*
proof(*cases Suc i* = *n*)
assume *g0*: *Suc i* = *n*
from *d10* **have** ($\forall$ *i*. *Suc i*<*length ?el1* $\longrightarrow$ ($\exists$ *t*. Γ $\vdash$*?el1*!*i* $-et-t\rightarrow$ *?el1*!(*Suc i*))
$\longrightarrow$ (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) $\in$ *guar1*) **by** (*simp add*: *commit-e-def*)
with *d7* **have** *g1*: (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) $\in$ *guar1*
by (*metis* (*no-types*, *lifting*) *b12-1 b5-1 d3 diff-Suc-1 g0 length-append-singleton lessI nth-append nth-append-length*)

moreover have *?el1*!(*Suc i*) = (*AnonyEvent fin-com*, *sn*, *xn*)
using *b12-1 b5-1 g0* **by** *auto*
moreover from *g0 b5-1 b12-1* **have** *?el1*!*i* = (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*)
by (*metis b12-1 b5-1 d3 diff-Suc-1 lessI nth-append*)
ultimately have (*sn1*,*sn*) $\in$ *guar1* **by** (*simp add*:*gets-e-def*)
with *p4* **have** (*sn1*,*sn*) $\in$ *guar* **by** *auto*
with *d4 d2* **have** (*gets-es* (*esl* ! (*n* − *1*)), *gets-es* (*esl* ! *Suc* (*n* − *1*))) $\in$ *guar*
by (*simp add*: *gets-es-def b4*)
then show *?thesis* **using** *g0* **by** *auto*
next
assume *Suc i* $\neq$ *n*
then have *g1*: *Suc i* > *n*
using *d0 linorder-neqE-nat* **by** *blast*
from *d4* **have** *g2*: *esl1* ! *0* = (*es*, *sn*, *xn*) **by** (*simp add*: *b13*)
with *b9* **have** *g3*: *esl1* $\in$ *cpts-of-es* Γ *es sn xn* **by** (*simp add*:*cpts-of-es-def*)

have *esl1* $\in$ *assume-es* Γ (*pre2*, *rely2*)
proof(*simp add*:*assume-es-def*, *rule conjI*)
from *d12* **have** *sn* $\in$ *post1* **by** (*simp add*:*gets-e-def*)
with *g2 p6* **show** *gets-es* (*esl1* ! *0*) $\in$ *pre2*
using *gets-es-def* **by** (*metis fst-conv rev-subsetD snd-conv*)

show $\forall i.$ *Suc i* $<$ *length esl1* $\longrightarrow \Gamma \vdash$ *esl1* ! *i* $-ese\rightarrow$ *esl1* ! *Suc i*
$\longrightarrow$ (*gets-es* (*esl1* ! *i*), *gets-es* (*esl1* ! *Suc i*)) $\in$ *rely2*
proof –
{
fix *i*
assume *h0*: *Suc i* $<$ *length esl1*
and *h1*: $\Gamma \vdash$ *esl1* ! *i* $-ese\rightarrow$ *esl1* ! *Suc i*
have *h2*: *esl1* ! *i* = *esl* ! (*n* + *i*) **using** *b5-1 b7* **by** *auto*

have *h3*: *esl1* ! *Suc i* = *esl* ! (*n* + *Suc i*)
by (*metis b5-1 b7 nth-append-length-plus*)
with *h1 h2* **have** *h4*: $\Gamma \vdash$ *esl* ! (*n* + *i*) $-ese\rightarrow$ *esl* ! (*n* + *Suc i*) **by** *simp*

have *Suc* (*n* + *i*) $<$ *length esl* **using** *b5-1 b7 h0* **by** *auto*

with *a02 h4* **have** (*gets-es* (*esl* ! (*n* + *i*)), *gets-es* (*esl* ! (*n* + *Suc i*))) $\in$ *rely*
by (*simp add*:*assume-es-def*)
with *h2 h3* **have** (*gets-es* (*esl1* ! *i*), *gets-es* (*esl1* ! *Suc i*)) $\in$ *rely* **by** *simp*

then have (*gets-es* (*esl1* ! *i*), *gets-es* (*esl1* ! *Suc i*)) $\in$ *rely2*

using *p3* **by** *auto*
}
then show *?thesis* **by** *auto*
qed

qed
with *p1 g3* **have** *g4*: *esl1* $\in$ *commit-es* Γ (*guar2*,*post*)
by (*meson Int-iff es-validity-def subsetCE*)

have *g5*: *esl* ! *i* = *esl1* ! (*i* − *n*)
by (*metis b5-1 b7 g1 not-less-eq nth-append*)
have *g6*: *esl* ! *Suc i* = *esl1* ! (*Suc i* − *n*)
by (*metis b5-1 b7 d0 nth-append*)

have *g7*: *Suc* (*i* − *n*) $<$ *length esl1* **using** *b6 c0 g1* **by** *auto*
from *g4* **have** $\forall i.$ *Suc i*$<$*length esl1* $\longrightarrow$ ($\exists t.$ $\Gamma \vdash$ *esl1*!*i* $-es-t\rightarrow$ *esl1*!(*Suc i*))
$\longrightarrow$ (*gets-es* (*esl1*!*i*), *gets-es* (*esl1*!*Suc i*)) $\in$ *guar2* **by** (*simp add*:*commit-es-def*)
with *g7* **have** (*gets-es* (*esl1*!(*i* − *n*)), *gets-es* (*esl1*!(*Suc i* − *n*))) $\in$ *guar2*
using *Suc-diff-le c1 g1 g5 g6* **by** *auto*
with *g5 g6* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *guar2* **by** *simp*

```
                    then show ?thesis using p5 by auto
                  qed
                qed
              }
              then show ?thesis by auto
              qed

            then show ?thesis by (simp add:commit-es-def)

          qed
        }
        then show ?thesis by auto
        qed
      }
      then show ?thesis by auto
      qed

    then show ?thesis by (simp add: es-validity-def)
  qed

primrec parse-es-cpts-i2 :: ('l,'k,'s,'prog) esconfs ⇒('l,'k,'s,'prog) event set ⇒
                          (('l,'k,'s,'prog) esconfs) list ⇒ (('l,'k,'s,'prog) esconfs) list
  where parse-es-cpts-i2 [] es rlst = rlst |
        parse-es-cpts-i2 (x#xs) es rlst =
            (if getspc-es x = EvtSys es ∧ length xs > 0
                ∧ (getspc-es (xs!0) ≠ EvtSys es) then
              parse-es-cpts-i2 xs es (rlst@[[x]])
            else
                parse-es-cpts-i2 xs es (list-update rlst (length rlst − 1) (last rlst @
[x])) )

lemma concat-list-lemma-take-n [rule-format]:
  ⟦esl = concat lst; i ≤ length lst⟧ ⟹
     ∃ k. k ≤ length esl ∧ take k esl = concat (take i lst)
  proof −
    assume p0: esl = concat lst
      and  p1: i ≤ length lst
    then show ?thesis
      proof(induct i)
        case 0
        have concat (take 0 lst) = take 0 esl by simp
        then show ?case by auto
      next
        case (Suc ii)
        assume a0: esl = concat lst ⟹ ii ≤ length lst
                    ⟹ ∃ k≤length esl. take k esl = concat (take ii lst)
          and  a1: esl = concat lst
          and  a2: Suc ii ≤ length lst
        then have ∃ k≤length esl. take k esl = concat (take ii lst)
```

using *Suc-leD* **by** *blast*
then obtain *k* **where** *a3*: $k \leq length\ esl \land take\ k\ esl = concat\ (take\ ii\ lst)$
by *auto*
from *a2* **have** *a4*: $concat\ (take\ (Suc\ ii)\ lst) = concat\ (take\ ii\ lst)\ @\ lst!ii$
by (*simp add*: *take-Suc-conv-app-nth*)
with *a3* **have** $concat\ (take\ (Suc\ ii)\ lst) = take\ (k + length\ (lst!ii))\ esl$
by (*metis Cons-nth-drop-Suc Suc-le-lessD a2 append-eq-conv-conj append-take-drop-id concat.simps*(*2*) *concat-append p0 take-add*)
then show *?case* **by** (*metis nat-le-linear take-all*)
qed
qed

lemma *concat-list-lemma-take-n2* [*rule-format*]:
$[\![esl = concat\ lst;\ i \leq length\ lst]\!] \Longrightarrow$
$\exists k.\ k \leq length\ esl \land k = length\ (concat\ (take\ i\ lst)) \land take\ k\ esl = concat\ (take\ i\ lst)$
proof −
assume *p0*: $esl = concat\ lst$
and *p1*: $i \leq length\ lst$
then show *?thesis*
proof(*induct i*)
case *0*
have $concat\ (take\ 0\ lst) = take\ 0\ esl$ **by** *simp*
then show *?case* **by** *auto*
next
case (*Suc ii*)
assume *a0*: $esl = concat\ lst \Longrightarrow ii \leq length\ lst$
$\Longrightarrow \exists k \leq length\ esl.\ k = length\ (concat\ (take\ ii\ lst))$
$\land\ take\ k\ esl = concat\ (take\ ii\ lst)$
and *a1*: $esl = concat\ lst$
and *a2*: $Suc\ ii \leq length\ lst$
then have $\exists k \leq length\ esl.\ k = length\ (concat\ (take\ ii\ lst))$
$\land\ take\ k\ esl = concat\ (take\ ii\ lst)$
using *Suc-leD* **by** *blast*
then obtain *k* **where** *a3*: $k \leq length\ esl \land k = length\ (concat\ (take\ ii\ lst))$
$\land\ take\ k\ esl = concat\ (take\ ii\ lst)$
by *auto*
from *a2* **have** *a4*: $concat\ (take\ (Suc\ ii)\ lst) = concat\ (take\ ii\ lst)\ @\ lst!ii$
by (*simp add*: *take-Suc-conv-app-nth*)
with *a3* **have** $concat\ (take\ (Suc\ ii)\ lst) = take\ (k + length\ (lst!ii))\ esl$
by (*metis Cons-nth-drop-Suc Suc-le-lessD a2 append-eq-conv-conj append-take-drop-id concat.simps*(*2*) *concat-append p0 take-add*)
then show *?case* **by** (*metis a2 concat-list-lemma-take-n length-take min.absorb2 p0*)
qed
qed

lemma *concat-list-lemma* [*rule-format*]:
$\forall esl\ lst.\ esl = concat\ lst \land (\forall i < length\ lst.\ length\ (lst!i) > 0) \longrightarrow$

($\forall i$. *Suc i* < *length esl*
$\longrightarrow$ ($\exists k\ j$. *Suc k* < *length lst* $\wedge$ *Suc j* < *length* (*lst*!*k*@[*lst*!(*Suc k*)!*0*])
$\wedge$ *esl*!*i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*j* $\wedge$ *esl*!*Suc i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*Suc j*
$\vee$ *Suc k* = *length lst* $\wedge$ *Suc j* < *length* (*lst*!*k*) $\wedge$ *esl*!*i* = *lst*!*k*!*j* $\wedge$ *esl*!*Suc i* = *lst*!*k*!*Suc j*))
proof −
{
fix *lst*
have $\forall$ *esl*. *esl* = *concat lst* $\wedge$ ($\forall$ *i*<*length lst*. *length* (*lst*!*i*) > *0*)$\longrightarrow$
($\forall i$. *Suc i* < *length esl*
$\longrightarrow$ ($\exists k\ j$. *Suc k* < *length lst* $\wedge$ *Suc j* < *length* (*lst*!*k*@[*lst*!(*Suc k*)!*0*])
$\wedge$ *esl*!*i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*j* $\wedge$ *esl*!*Suc i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*Suc j*
$\vee$ *Suc k* = *length lst* $\wedge$ *Suc j* < *length* (*lst*!*k*) $\wedge$ *esl*!*i* = *lst*!*k*!*j* $\wedge$ *esl*!*Suc i* = *lst*!*k*!*Suc j*))
proof(*induct lst*)
case *Nil* **then show** *?case* **by** *simp*
next
case (*Cons l lt*)
assume *a0*: $\forall$ *esl*. *esl* = *concat lt* $\wedge$ ($\forall$ *i*<*length lt*. *0* < *length* (*lt* ! *i*)) $\longrightarrow$
($\forall i$. *Suc i* < *length esl* $\longrightarrow$
($\exists k\ j$. *Suc k* < *length lt* $\wedge$
Suc j < *length* (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) $\wedge$
esl ! *i* = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *j* $\wedge$ *esl* ! *Suc i* = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *Suc j* $\vee$
Suc k = *length lt* $\wedge$ *Suc j* < *length* (*lt* ! *k*) $\wedge$ *esl* ! *i* = *lt* ! *k* ! *j* $\wedge$ *esl* ! *Suc i* = *lt* ! *k* ! *Suc j*))
{
fix *esl*
assume *b0*: *esl* = *concat* (*l* # *lt*)
and *b1*: $\forall$ *i*<*length* (*l* # *lt*). *0* < *length* ((*l* # *lt*) ! *i*)

{
fix *i*
assume *c0*: *Suc i* < *length esl*
then have $\exists k\ j$. *Suc k* < *length* (*l* # *lt*) $\wedge$
Suc j < *length* ((*l* # *lt*) ! *k* @ [(*l* # *lt*) ! *Suc k* ! *0*]) $\wedge$
esl ! *i* = ((*l* # *lt*) ! *k* @ [(*l* # *lt*) ! *Suc k* ! *0*]) ! *j* $\wedge$
esl ! *Suc i* = ((*l* # *lt*) ! *k* @ [(*l* # *lt*) ! *Suc k* ! *0*]) ! *Suc j* $\vee$
Suc k = *length* (*l* # *lt*) $\wedge$
Suc j < *length* ((*l* # *lt*) ! *k*) $\wedge$ *esl* ! *i* = (*l* # *lt*) ! *k* ! *j* $\wedge$ *esl* ! *Suc i* = (*l* # *lt*) ! *k* ! *Suc j*
proof(*cases lt* = [])
assume *d0*: *lt* = []
with *b0* **have** *esl* = *l* **by** *auto*
with *b0 c0* **have** *Suc 0* = *length* (*l* # []) $\wedge$
Suc i < *length* ((*l* # []) ! *0*) $\wedge$ *esl* ! *i* = (*l* # []) ! *0* ! *i* $\wedge$ *esl* ! *Suc i* = (*l* # []) ! *0* ! *Suc i*

```
        by simp
      with d0 show ?thesis by auto
    next
      assume d0: lt ≠ []
      then show ?thesis
        proof(cases Suc i < length (l@[(l # lt) ! Suc 0!0]))
          assume e0: Suc i < length (l@[(l # lt) ! Suc 0!0])
          with b0 b1 show ?thesis
            by (smt Cons-nth-drop-Suc Suc-lessE Suc-lessI Suc-mono
              cancel-comm-monoid-add-class.diff-cancel concat.simps(2)
       d0 diff-Suc-1 drop-0 drop-Suc-Cons length-Cons length-append-singleton

              length-greater-0-conv nth-Cons-0 nth-append)
        next
          assume e00: ¬(Suc i < length (l@[(l # lt) ! Suc 0!0]))
          then have e0: Suc i ≥ length (l@[(l # lt) ! Suc 0!0]) by simp
          from b0 have ∃esl1. esl = l@esl1 ∧ esl1 = concat lt by simp
          then obtain esl1 where e1: esl = l@esl1 ∧ esl1 = concat lt by
auto
          with a0 b1 have e2: ∀i. Suc i < length esl1 ⟶
            (∃k j. Suc k < length lt ∧
                   Suc j < length (lt ! k @ [lt ! Suc k ! 0]) ∧
                  esl1 ! i = (lt ! k @ [lt ! Suc k ! 0]) ! j ∧ esl1 ! Suc i = (lt
! k @ [lt ! Suc k ! 0]) ! Suc j ∨
                  Suc k = length lt ∧ Suc j < length (lt ! k) ∧ esl1 ! i = lt
! k ! j ∧ esl1 ! Suc i = lt ! k ! Suc j)
            by auto
          from c0 e0 e00 e1 have e3: esl!i = esl1!(i−length l)
            by (simp add: length-append-singleton nth-append)
          from c0 e0 e00 e1 have e4: esl!Suc i = esl1!(Suc i − length l)
            by (simp add: length-append-singleton less-Suc-eq nth-append)
          from c0 e0 e00 e1 have e5: Suc (i−length l) < length esl1
            using Suc-le-mono add.commute le-SucI length-append
            length-append-singleton less-diff-conv2 by auto
          with e2 have ∃k j. Suc k < length lt ∧
                   Suc j < length (lt ! k @ [lt ! Suc k ! 0]) ∧
                  esl1 ! (i−length l) = (lt ! k @ [lt ! Suc k ! 0]) ! j ∧ esl1 !
Suc (i−length l) = (lt ! k @ [lt ! Suc k ! 0]) ! Suc j ∨
                      Suc k = length lt ∧ Suc j < length (lt ! k) ∧ esl1 !
(i−length l) = lt ! k ! j ∧ esl1 ! Suc (i−length l) = lt ! k ! Suc j
            by auto
          then obtain k and j where Suc k < length lt ∧
                   Suc j < length (lt ! k @ [lt ! Suc k ! 0]) ∧
                  esl1 ! (i−length l) = (lt ! k @ [lt ! Suc k ! 0]) ! j ∧ esl1 !
Suc (i−length l) = (lt ! k @ [lt ! Suc k ! 0]) ! Suc j ∨
                      Suc k = length lt ∧ Suc j < length (lt ! k) ∧ esl1 !
(i−length l) = lt ! k ! j ∧ esl1 ! Suc (i−length l) = lt ! k ! Suc j
            by auto
```

```
                  with c0 e0 e1 show ?thesis
                    by (smt Suc-diff-le Suc-le-mono Suc-mono e3 e4 length-Cons
                      length-append-singleton nat-neq-iff nth-Cons-Suc)
                qed
              qed
            }
          }
          then show ?case by auto
        qed
  }
  then show ?thesis by blast
  qed

lemma concat-list-lemma2 [rule-format]:
  ∀ esl lst. esl = concat lst ⟶
        (∀ i < length lst. (take (length (lst!i)) (drop (length (concat (take i lst)))
esl) = lst ! i))
  proof −
  {
    fix lst
    have ∀ esl. esl = concat lst ⟶
        (∀ i < length lst. (take (length (lst!i)) (drop (length (concat (take i lst)))
esl) = lst ! i))
      proof(induct lst)
        case Nil then show ?case by simp
      next
        case (Cons l lt)
        assume a0[rule-format]: ∀ esl. esl = concat lt ⟶
                        (∀ i<length lt. take (length (lt ! i)) (drop (length (concat
(take i lt))) esl) = lt ! i)
        {
          fix esl
          assume b0: esl = concat (l # lt)
          let ?esl = concat lt
          from b0 have b1: esl = l @ ?esl by auto
          {
            fix i
            assume c0: i<length (l # lt)
            have take (length ((l # lt) ! i)) (drop (length (concat (take i (l # lt))))
esl) = (l # lt) ! i
              proof(cases i = 0)
                assume d0: i = 0
                then show ?thesis by (simp add: b0 d0)
              next
                assume d0: i ≠ 0
                 with c0 have take (length (lt ! (i−1))) (drop (length (concat (take
(i−1) lt))) ?esl) = lt ! (i−1)
              using a0[of ?esl i−1] by (metis One-nat-def leI less-Suc0 less-diff-conv2
list.size(4))
```

moreover
from *d0 c0* **have** $lt ! (i - 1) = (l \# lt) ! i$ **by** (*simp add*: *nth-Cons'*)
moreover
from *b0 b1 d0 c0* **have** *drop* (*length* (*concat* (*take* (*i*−*1*) *lt*))) *?esl*
= *drop* (*length* (*concat* (*take i* (*l* # *lt*)))) *esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append drop-Cons'*)
ultimately show *?thesis* **by** *simp*
qed
}
}
then show *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *concat-list-lemma3* [*rule-format*]:
⟦*esl* = *concat lst*; *i* < *length lst*; *length* (*lst*!*i*) > *1*⟧ ⟹
∃ *k j*. *k* = *length* (*concat* (*take i lst*)) ∧ *j* = *length* (*concat* (*take* (*Suc i*) *lst*)) ∧
k ≤ *length esl* ∧ *j* ≤ *length esl* ∧ *k* < *j* ∧ *drop k* (*take j esl*) = *lst* ! *i*
proof −
assume *p0*: *esl* = *concat lst*
and *p1*: *i* < *length lst*
and *p2*: *length* (*lst*!*i*) > *1*
then have *a1*: *take* (*length* (*lst*!*i*)) (*drop* (*length* (*concat* (*take i lst*))) *esl*) = *lst* ! *i*
using *concat-list-lemma2* **by** *auto*
let *?k* = *length* (*concat* (*take i lst*))
let *?j* = *length* (*concat* (*take* (*Suc i*) *lst*))
from *p0 p1 p2* **have** *a10*: *drop ?k* (*take ?j esl*) = *lst* ! *i*
proof −
have *length* (*lst* ! *i*) + *length* (*concat* (*take i lst*)) = *length* (*concat* (*take* (*Suc i*) *lst*))
by (*simp add*: *p1 take-Suc-conv-app-nth*)
then show *?thesis*
by (*metis* (*full-types*) *a1 take-drop*)
qed
have *a2*: *?j* − *?k* = *length* (*lst*!*i*) **by** (*simp add*: *p1 take-Suc-conv-app-nth*)
have *a3*: *?j* = *?k* + *length* (*lst*!*i*) **by** (*simp add*: *p1 take-Suc-conv-app-nth*)
moreover
from *p0 p1* **have** *?k* ≤ *length esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append nat-le-linear take-all*)
moreover
from *p0 p1* **have** *?j* ≤ *length esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append nat-le-linear take-all*)

moreover
from *a3 p2* **have** *?k < ?j* **using** *a2 diff-is-0-eq leI not-less0* **by** *linarith*
ultimately have *?k ≤ length esl ∧ ?j ≤ length esl ∧ ?k < ?j ∧ drop ?k (take ?j esl) = lst ! i*
using *a10* **by** *simp*
then show *?thesis* **by** *blast*
qed

lemma *concat-list-lemma-withnextfst*:
⟦*esl = concat lst*; *Suc i < length lst*; *length (lst!Suc i) > 0*⟧ ⟹
∃ *k j*. *k ≤ length esl ∧ j ≤ length esl ∧ k < j ∧ drop k (take j esl) = lst!i @ [lst!Suc i!0]*
proof −
assume *p0*: *esl = concat lst*
and *p1*: *Suc i < length lst*
and *p2*: *length (lst!Suc i) > 0*
then have ∃ *k*. *k ≤ length esl ∧ take k esl = concat (take (Suc (Suc i)) lst)*
using *concat-list-lemma-take-n[of esl lst Suc (Suc i)]* **by** *simp*
then obtain *k* **where** *a1*: *k ≤ length esl ∧ take k esl = concat (take (Suc (Suc i)) lst)* **by** *auto*

from *p0 p1 p2* **have** ∃ *k*. *k ≤ length esl ∧ take k esl = concat (take (Suc i) lst)*
using *concat-list-lemma-take-n[of esl lst Suc i]* **by** *simp*
then obtain *k2* **where** *a2*: *k2 ≤ length esl ∧ take k2 esl = concat (take (Suc i) lst)* **by** *auto*

with *p0* **have** *a5*: *concat (take (Suc i) lst) @ [lst!Suc i!0] = take (Suc k2) esl*
by *(metis (no-types, lifting) Cons-nth-drop-Suc append-eq-conv-conj append-take-drop-id concat-list-lemma2 drop-eq-Nil length-greater-0-conv less-eq-Suc-le not-less-eq-eq nth-Cons-0 nth-take p1 p2 take-Suc-conv-app-nth take-eq-Nil)*
then have *a3*: *concat (take i lst)@lst!i@[lst!Suc i!0] = take (Suc k2) esl*
by *(metis (no-types, lifting) Suc-lessD append-Nil2 append-eq-appendI concat.simps(1) concat.simps(2) concat-append p1 take-Suc-conv-app-nth)*

from *p0 p1 p2* **have** ∃ *k*. *k ≤ length esl ∧ take k esl = concat (take i lst)*
using *concat-list-lemma-take-n[of esl lst i]* **by** *simp*
then obtain *k1* **where** *a4*: *k1 ≤ length esl ∧ take k1 esl = concat (take i lst)* **by** *auto*

from *a3 a4* **have** *drop k1 (take (Suc k2) esl) = lst!i@[lst!Suc i!0]*
by *(metis append-eq-conv-conj length-take min.absorb2)*
then show *?thesis* **using** *a2 a4 a5*
by *(metis Nil-is-append-conv drop-eq-Nil leI length-take min.absorb2 nat-le-linear not-Cons-self2 take-all)*
qed

lemma *concat-list-lemma-withnextfst2*:

⟦*esl* = *concat lst*; *Suc i* < *length lst*; *length* (*lst*!*Suc i*) > *0*⟧ ⟹
∃ *k j*. *k* = *length* (*concat* (*take i lst*)) ∧ *j* = *Suc* (*length* (*concat* (*take* (*Suc i*) *lst*))) ∧
k ≤ *length esl* ∧ *j* ≤ *length esl* ∧ *k* < *j* ∧ *drop k* (*take j esl*) = *lst*!*i* @ [*lst*!*Suc i*!*0*]
proof −
assume *p0*: *esl* = *concat lst*
and *p1*: *Suc i* < *length lst*
and *p2*: *length* (*lst*!*Suc i*) > *0*
then have ∃ *k*. *k* ≤ *length esl* ∧ *k* = *length* (*concat* (*take* (*Suc* (*Suc i*)) *lst*))
∧ *take k esl* = *concat* (*take* (*Suc* (*Suc i*)) *lst*)
using *concat-list-lemma-take-n2*[*of esl lst Suc* (*Suc i*)] **by** *simp*
then obtain *k* **where** *a1*: *k* ≤ *length esl* ∧ *k* = *length* (*concat* (*take* (*Suc* (*Suc i*)) *lst*))
∧ *take k esl* = *concat* (*take* (*Suc* (*Suc i*)) *lst*) **by** *auto*

from *p0 p1 p2* **have** ∃ *k*. *k* ≤ *length esl* ∧ *k* = *length* (*concat* (*take* (*Suc i*) *lst*))
∧ *take k esl* = *concat* (*take* (*Suc i*) *lst*)
using *concat-list-lemma-take-n2*[*of esl lst Suc i*] **by** *simp*
then obtain *k2* **where** *a2*: *k2* ≤ *length esl* ∧ *k2* = *length* (*concat* (*take* (*Suc i*) *lst*))
∧ *take k2 esl* = *concat* (*take* (*Suc i*) *lst*) **by** *auto*

with *p0* **have** *a5*: *concat* (*take* (*Suc i*) *lst*) @ [*lst*!*Suc i*!*0*] = *take* (*Suc k2*) *esl*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc append-eq-conv-conj*
append-take-drop-id concat-list-lemma2 drop-eq-Nil length-greater-0-conv
less-eq-Suc-le not-less-eq-eq nth-Cons-0 nth-take p1 p2 take-Suc-conv-app-nth
take-eq-Nil)
then have *a3*: *concat* (*take i lst*)@*lst*!*i*@[*lst*!*Suc i*!*0*] = *take* (*Suc k2*) *esl*
by (*metis* (*no-types*, *lifting*) *Suc-lessD append-Nil2 append-eq-appendI*
concat.simps(*1*) *concat.simps*(*2*) *concat-append p1 take-Suc-conv-app-nth*)

from *p0 p1 p2* **have** ∃ *k*. *k* ≤ *length esl* ∧ *k* = *length* (*concat* (*take i lst*))
∧ *take k esl* = *concat* (*take i lst*)
using *concat-list-lemma-take-n2*[*of esl lst i*] **by** *simp*
then obtain *k1* **where** *a4*: *k1* ≤ *length esl* ∧ *k1* = *length* (*concat* (*take i lst*))

∧ *take k1 esl* = *concat* (*take i lst*) **by** *auto*

from *a3 a4* **have** *drop k1* (*take* (*Suc k2*) *esl*) = *lst*!*i*@[*lst*!*Suc i*!*0*]
by (*metis append-eq-conv-conj length-take*)

with *a2 a4 a5* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Nil-is-append-conv*
drop-eq-Nil leI length-append-singleton less-or-eq-imp-le not-Cons-self2
take-all)
qed

lemma *concat-list-lemma-withnextfst3*:

$\llbracket esl = concat\ lst;\ Suc\ i < length\ lst;\ length\ (lst!Suc\ i) > 1\rrbracket \Longrightarrow$
$\exists\, k\ j.\ k = length\ (concat\ (take\ i\ lst)) \wedge j = Suc\ (length\ (concat\ (take\ (Suc\ i)\ lst))) \wedge$
$k \le length\ esl \wedge j < length\ esl \wedge k < j \wedge drop\ k\ (take\ j\ esl) = lst!i\ @\ [lst!Suc\ i!0]$

proof $-$
assume *p0*: $esl = concat\ lst$
and *p1*: $Suc\ i < length\ lst$
and *p2*: $length\ (lst!Suc\ i) > 1$
then have $\exists\, k.\ k \le length\ esl \wedge k = length\ (concat\ (take\ (Suc\ (Suc\ i))\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ (Suc\ (Suc\ i))\ lst)$
using *concat-list-lemma-take-n2*[*of esl lst Suc* (*Suc i*)] **by** *simp*
then obtain *k* **where** *a1*: $k \le length\ esl \wedge k = length\ (concat\ (take\ (Suc\ (Suc\ i))\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ (Suc\ (Suc\ i))\ lst)$ **by** *auto*

from *p0 p1 p2* **have** $\exists\, k.\ k \le length\ esl \wedge k = length\ (concat\ (take\ (Suc\ i)\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ (Suc\ i)\ lst)$
using *concat-list-lemma-take-n2*[*of esl lst Suc i*] **by** *simp*
then obtain *k2* **where** *a2*: $k2 \le length\ esl \wedge k2 = length\ (concat\ (take\ (Suc\ i)\ lst))$
$\wedge\ take\ k2\ esl = concat\ (take\ (Suc\ i)\ lst)$ **by** *auto*

with *p0* **have** *a5*: $concat\ (take\ (Suc\ i)\ lst)\ @\ [lst!Suc\ i!0] = take\ (Suc\ k2)\ esl$
by (*metis One-nat-def Suc-lessD Suc-n-not-le-n append-Nil2 append-take-drop-id*

concat-list-lemma2 concat-list-lemma-withnextfst2 hd-conv-nth
le-neq-implies-less nth-take p1 p2 take-hd-drop)

then have *a3*: $concat\ (take\ i\ lst)@lst!i@[lst!Suc\ i!0] = take\ (Suc\ k2)\ esl$
by (*metis* (*no-types*, *lifting*) *Suc-lessD append-Nil2 append-eq-appendI*
concat.simps(*1*) *concat.simps*(*2*) *concat-append p1 take-Suc-conv-app-nth*)

from *p0 p1 p2* **have** $\exists\, k.\ k \le length\ esl \wedge k = length\ (concat\ (take\ i\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ i\ lst)$
using *concat-list-lemma-take-n2*[*of esl lst i*] **by** *simp*
then obtain *k1* **where** *a4*: $k1 \le length\ esl \wedge k1 = length\ (concat\ (take\ i\ lst))$

$\wedge\ take\ k1\ esl = concat\ (take\ i\ lst)$ **by** *auto*

from *a3 a4* **have** $drop\ k1\ (take\ (Suc\ k2)\ esl) = lst!i@[lst!Suc\ i!0]$
by (*metis append-eq-conv-conj length-take*)

with *a2 a4 a5* **show** *?thesis*
by (*smt One-nat-def append-eq-conv-conj concat-list-lemma2 concat-list-lemma-withnextfst2*

leI length-Cons less-trans list.size(*3*) *nat-neq-iff p0 p1 p2 take-all zero-less-one*)

qed

lemma *parse-es-cpts-i2-concat*:
$\forall$ *esl rlst es. esl*$\in$*cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
$\longrightarrow$ *concat* (*parse-es-cpts-i2 esl es rlst*) = *concat rlst* @ *esl*

proof −
{
fix *esl*
have $\forall$ *rlst es. esl*$\in$*cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ [] $\longrightarrow$ *concat* (*parse-es-cpts-i2 esl es rlst*) = *concat rlst* @ *esl*
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *rlst es. esl1* $\in$ *cpts-es* Γ $\wedge$ *rlst* $\neq$ [] $\longrightarrow$ *concat* (*parse-es-cpts-i2 esl1 es rlst*) = *concat rlst* @ *esl1*
then show *?case*
proof −
{
fix *rlst es*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
have *concat* (*parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*) = *concat rlst* @ (*esc* # *esl1*)
proof(*cases getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
assume *c0*: *getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*
then have *c1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])
by *simp*
from *b0* **have** *c2*: *rlst*@[[*esc*]] $\neq$ [] **by** *simp*
from *b0 c0* **have** *esl1* $\in$ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
with *a0 c2* **have** *c3*: *concat* (*parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])) = *concat* (*rlst*@[[*esc*]]) @ *esl1* **by** *simp*
have *concat rlst* @ (*esc* # *esl1*) = *concat* (*rlst*@[[*esc*]]) @ *esl1* **by** *auto*
with *c1 c3* **show** *?thesis* **by** *presburger*
next
assume *c0*: $\neg$(*getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
then have *c1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* =
parse-es-cpts-i2 esl1 es (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *auto*
show *?thesis*
proof(*cases esl1* = [])
assume *d0*: *esl1* = []
then have *d1*: *parse-es-cpts-i2* (*esc* # []) *es rlst* =
parse-es-cpts-i2 [] *es* (*list-update rlst* (*length rlst* − *1*)

(*last rlst* @ [*esc*])) **by** *simp*
have *d2*: *parse-es-cpts-i2* [] *es* (*list-update rlst* (*length rlst* − *1*)
(*last rlst* @ [*esc*])) =
list-update rlst (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
from *b0* **have** *concat* (*list-update rlst* (*length rlst* − *1*) (*last rlst*
@ [*esc*])) = *concat rlst* @ *esc* # []
by (*metis* (*no-types*, *lifting*) *append-assoc append-butlast-last-id*
append-self-conv concat.simps(*2*) *concat-append length-butlast*
list-update-length)
with *d0 d1 d2* **show** *?thesis* **by** *simp*
next
assume *d0*: ¬(*esl1* = [])
then have *length esl1* > *0* **by** *simp*
with *b0* **have** *d1*: *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *b0* **have** *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])
≠ [] **by** *simp*
with *a0 d1* **have** *d2*: *concat* (*parse-es-cpts-i2 esl1 es* (*list-update*
rlst (*length rlst* − *1*) (*last rlst* @ [*esc*]))) =
concat (*list-update rlst* (*length rlst* − *1*) (*last rlst* @
[*esc*])) @ *esl1* **by** *auto*
from *b0* **have** *d3*: *concat rlst* @ (*esc* # *esl1*) = *concat* (*list-update*
rlst (*length rlst* − *1*) (*last rlst* @ [*esc*])) @ *esl1*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI append-assoc*
append-butlast-last-id
concat.simps(*2*) *concat-append length-butlast list-update-length*
self-append-conv2)

with *c1 d2* **show** *?thesis* **by** *simp*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed

lemma *parse-es-cpts-i2-concat1*:
esl∈*cpts-es* Γ ⟹ *concat* (*parse-es-cpts-i2 esl es* [[]]) = *esl*
by (*simp add*: *parse-es-cpts-i2-concat*)

lemma *parse-es-cpts-i2-lst0*:
∀ *esl l1 l2 es*. *esl*∈*cpts-es* Γ ∧ (*l2*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) ≠ []
⟶ *parse-es-cpts-i2 esl es* (*l1*@*l2*) = *l1*@(*parse-es-cpts-i2 esl es l2*)
proof −
{
fix *esl*
have ∀ *l1 l2 es*. *esl*∈*cpts-es* Γ ∧ (*l2*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) ≠ []

$\longrightarrow$ *parse-es-cpts-i2 esl es* $(l1@l2) = l1@($*parse-es-cpts-i2 esl es* $l2)$

proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *l1 l2 es. esl1* $\in$ *cpts-es* $\Gamma \wedge$ (*l2*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
$\longrightarrow$ *parse-es-cpts-i2 esl1 es* (*l1* @*l2*) = *l1* @ *parse-es-cpts-i2 esl1 es l2*
show *?case*
proof –
{
fix *l1 l2 es*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ
and *b1*: (*l2*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
have *parse-es-cpts-i2* (*esc* # *esl1*) *es* (*l1* @ *l2*) = *l1* @ *parse-es-cpts-i2* (*esc* # *esl1*) *es l2*
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
then have *parse-es-cpts-i2* (*esc* # []) *es* (*l1* @ *l2*) =
parse-es-cpts-i2 [] *es* (*list-update* (*l1* @ *l2*) (*length* (*l1* @ *l2*) − *1*) (*last* (*l1* @ *l2*) @ [*esc*]))
by *simp*
then have *c1*: *parse-es-cpts-i2* (*esc* # []) *es* (*l1* @ *l2*) =
list-update (*l1* @ *l2*) (*length* (*l1* @ *l2*) − *1*) (*last* (*l1* @ *l2*) @ [*esc*])
by *simp*
with *b1* **have** *c2*: *parse-es-cpts-i2* (*esc* # []) *es* (*l1* @ *l2*) =
l1 @ (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*]))
by (*smt append-butlast-last-id append-is-Nil-conv butlast-append butlast-list-update last-appendR last-list-update list-update-nonempty*)
have *l1* @ *parse-es-cpts-i2* (*esc* # []) *es l2* =
l1 @ *parse-es-cpts-i2* [] *es* (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*])) **by** *simp*
then have *l1* @ *parse-es-cpts-i2* (*esc* # []) *es l2* =
l1 @ (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*])) **by** *simp*
with *c0 c2* **show** *?thesis* **by** *simp*
next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* $\in$ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
show *?thesis*
proof(*cases getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
assume *d0*: *getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*
then have *d1*:*parse-es-cpts-i2* (*esc* # *esl1*) *es* (*l1* @ *l2*) =
parse-es-cpts-i2 esl1 es (*l1* @ *l2*@[[*esc*]]) **by** *simp*
from *a0 c1* **have** *d2*: *parse-es-cpts-i2 esl1 es* (*l1* @ *l2*@[[*esc*]]) =

$l1$ @ *parse-es-cpts-i2 esl1 es* ($l2$@[[*esc*]]) **by** *simp*
from *d0* **have** *d3*: $l1$ @ *parse-es-cpts-i2* (*esc* # *esl1*) *es* $l2$ =
$l1$ @ *parse-es-cpts-i2 esl1 es* ($l2$@[[*esc*]]) **by** *simp*
with *d1 d2* **show** *?thesis* **by** *simp*
next
assume *d0*: ¬(*getspc-es esc* = *EvtSys es* ∧ *length esl1* > 0 ∧ *getspc-es* (*esl1*!0) ≠ *EvtSys es*)
then have *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es* ($l1$ @ $l2$) =
parse-es-cpts-i2 esl1 es (*list-update* ($l1$ @ $l2$) (*length* ($l1$ @ $l2$) − 1)
(*last* ($l1$ @ $l2$) @ [*esc*])) **by** *auto*
with *b1* **have** *d2*: *parse-es-cpts-i2* (*esc* # *esl1*) *es* ($l1$ @ $l2$) =
parse-es-cpts-i2 esl1 es ($l1$ @ *list-update* $l2$ (*length* $l2$ − 1) (*last* $l2$ @ [*esc*]))
by (*smt append1-eq-conv append-assoc append-butlast-last-id*
append-is-Nil-conv length-butlast list-update-length)
with *a0 b1 c1* **have** *d3*: *parse-es-cpts-i2* (*esc* # *esl1*) *es* ($l1$ @ $l2$) =
$l1$ @ *parse-es-cpts-i2 esl1 es* (*list-update* $l2$ (*length* $l2$ − 1) (*last* $l2$ @ [*esc*]))
by *auto*
from *d0* **have** $l1$ @ *parse-es-cpts-i2* (*esc* # *esl1*) *es* $l2$ =
$l1$ @ *parse-es-cpts-i2 esl1 es* (*list-update* $l2$ (*length* $l2$ − 1) (*last* $l2$ @ [*esc*]))
by *auto*
with *d3* **show** *?thesis* **by** *simp*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed

lemma *parse-es-cpts-i2-lst*:
∀ *esl l1 l2 es*. *esl*∈*cpts-es* Γ ∧ ($l2$::(('l,'k,'s,'prog) *esconfs*) *list*) ≠ []
⟶ *parse-es-cpts-i2 esl es* ([$l1$]@$l2$) = [$l1$]@(*parse-es-cpts-i2 esl es* $l2$)
using *parse-es-cpts-i2-lst0* **by** *blast*

lemma *parse-es-cpts-i2-fst*: ∀ *esl elst rlst es l*. *esl*∈*cpts-es* Γ ∧ *rlst* = [*l*] ∧ *elst* = *parse-es-cpts-i2 esl es rlst*
⟶ (∃ *i*≤*length* (*elst*!0). *take i* (*elst*!0) = *l*)
proof −
{
fix *esl*

have ∀ *elst rlst es l. esl*∈*cpts-es* Γ ∧ *rlst* = [*l*] ∧ *elst* = *parse-es-cpts-i2 esl es rlst*
⟶ (∃ *i*≤*length* (*elst*!*0*). *take i* (*elst*!*0*) = *l*)
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: ∀ *elst rlst es l. esl1* ∈ *cpts-es* Γ ∧ *rlst* = [*l*] ∧ *elst* = *parse-es-cpts-i2 esl1 es rlst*
⟶ (∃ *i*≤*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l*)
show *?case*
proof −
{
fix *elst rlst es l*
assume *b0*: *esc* # *esl1* ∈ *cpts-es* Γ
and *b1*: *rlst* = [*l*]
and *b2*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*
have ∃ *i*≤*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l*
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
with *b2* **have** *c1*: *elst* = *parse-es-cpts-i2* [] *es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))
by *simp*
then have *elst* = *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b1* **have** *c2*: *elst* = [*l*@[*esc*]] **by** *simp*
then show *?thesis* **by** (*metis butlast-conv-take butlast-snoc linear nth-Cons-0 take-all*)
next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*
by (*meson neq-Nil-conv*)
show *?thesis*
proof(*cases getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
assume *d0*: *getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*
with *c2* **have** *d01*: *getspc-es ec1* ≠ *EvtSys es* **by** *simp*
from *d0* **have** *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])
by *simp*
with *b1 b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) **by** *simp*
from *c1* **have** *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) = [*l*]@*parse-es-cpts-i2 esl1 es* ([[*esc*]])
using *parse-es-cpts-i2-lst* **by** *blast*
with *d2* **have** *elst* = [*l*] @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*
then show *?thesis* **by** *auto*

next
assume *d0*: ¬(*getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
then have *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* =
parse-es-cpts-i2 esl1 es (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *auto*
with *b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))
by *simp*
with *b1* **have** *elst* = *parse-es-cpts-i2 esl1 es* ([*l* @ [*esc*]]) **by** *simp*
with *a0 c1* **have** ∃ *i*≤*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l* @ [*esc*] **by** *simp*
then obtain *i* **where** *i*≤*length* (*elst* ! *0*) ∧ *take i* (*elst* ! *0*) = *l* @ [*esc*] **by** *auto*
then show *?thesis* **by** (*metis* (*no-types*, *lifting*) *butlast-snoc butlast-take diff-le-self dual-order.trans*)
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *parse-es-cpts-i2-start-withlen* [*simp*]:
∀ *esl elst rlst es l*. *esl*∈*cpts-es* Γ ∧ *rlst* ≠ [] ∧ *elst* = *parse-es-cpts-i2 esl es rlst* ⟶
(∀ *i*. *i* ≥ *length rlst* ∧ *i* < *length elst* ⟶
length (*elst*!*i*) ≥ *2* ∧ *getspc-es* (*elst*!*i*!*0*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*1*) ≠ *EvtSys es*)
proof −
{
fix *esl*
have ∀ *elst rlst es l*. *esl*∈*cpts-es* Γ ∧ *rlst* ≠ [] ∧ *elst* = *parse-es-cpts-i2 esl es rlst* ⟶
(∀ *i*. *i* ≥ *length rlst* ∧ *i* < *length elst* ⟶
length (*elst*!*i*) ≥ *2* ∧ *getspc-es* (*elst*!*i*!*0*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*1*) ≠ *EvtSys es*)
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: ∀ *elst rlst es l*. *esl1* ∈ *cpts-es* Γ ∧ *rlst* ≠ [] ∧ *elst* = *parse-es-cpts-i2 esl1 es rlst* ⟶
(∀ *i*. *i* ≥ *length rlst* ∧ *i* < *length elst* ⟶
length (*elst*!*i*) ≥ *2* ∧ *getspc-es* (*elst* ! *i* ! *0*) =

EvtSys es
$\wedge$ *getspc-es* (*elst* ! *i* ! *1*) $\neq$ *EvtSys es*)
then show *?case*
proof −
{
fix *elst rlst es l*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ
and *b1*: *rlst* $\neq$ []
and *b2*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*
have $\forall$ *i*. *i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$ *length* (*elst*!*i*) $\geq$ *2* $\wedge$ *getspc-es* (*elst* ! *i* ! *0*) = *EvtSys es*
$\wedge$ *getspc-es* (*elst* ! *i* ! *1*) $\neq$ *EvtSys es*
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
then have *c1*: *parse-es-cpts-i2* (*esc* # []) *es rlst* =
parse-es-cpts-i2 [] *es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *simp*
have *c2*: *parse-es-cpts-i2* [] *es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))
= *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b2 c0 c1* **have** *elst* = *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b1* **show** *?thesis* **by** *auto*
next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* $\in$ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*
by (*meson neq-Nil-conv*)
show *?thesis*
proof(*cases getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
assume *d0*: *getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*
with *c2* **have** *d01*: *getspc-es ec1* $\neq$ *EvtSys es* **by** *simp*
from *d0* **have** *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])
by *simp*
with *b1 b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]]) **by** *simp*
from *c1* **have** *d4*: *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]]) = *rlst*@*parse-es-cpts-i2 esl1 es* ([[*esc*]])
using *parse-es-cpts-i2-lst0* **by** *blast*
with *d2* **have** *d3*: *elst* = *rlst* @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*
show *?thesis*
proof(*cases esl2* = [])
assume *e0*: *esl2* = []
with *c2* **have** *e1*: *elst* = *rlst* @ *parse-es-cpts-i2* [] *es*
(*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]]

@ [*ec1*]))
using *b2 d1* **by** *auto*
then have *elst* = *rlst* @ (*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]] @ [*ec1*]))
by *simp*
then have *elst* = *rlst* @ ([[*esc*] @ [*ec1*]]) **by** *simp*
with *d0 d01* **show** *?thesis* **using** *leD le-eq-less-or-eq* **by** *auto*
next
assume *e0*: ¬(*esl2* = [])

let *?elst2* = *parse-es-cpts-i2 esl1 es* ([[*esc*]])
from *a0 c1* **have** *e1*: ∀ *i*. *i* ≥ *1* ∧ *i* < *length ?elst2* ⟶
length (*?elst2*!*i*) ≥ *2* ∧ *getspc-es* (*?elst2* ! *i* ! *0*) = *EvtSys es*
∧ *getspc-es* (*?elst2* ! *i* ! *1*) ≠ *EvtSys es*
by (*metis One-nat-def length-Cons list.distinct*(*2*) *list.size*(*3*))

from *c2 d01 d3* **have** *elst* = *rlst* @ *parse-es-cpts-i2 esl2 es*
(*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]] @ [*ec1*])) **by** *simp*
then have *e2*: *elst* = *rlst* @ *parse-es-cpts-i2 esl2 es* [[*esc*]@[*ec1*]] **by** *simp*
with *d3* **have** *e3*: *?elst2* = *parse-es-cpts-i2 esl2 es* [[*esc*]@[*ec1*]] **by** *simp*
from *c1 c2 e0* **have** *esl2*∈*cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
with *e3* **have** *e4*: ∃ *i*≤*length* (*?elst2*!*0*). *take i* (*?elst2*!*0*) = [*esc*]@[*ec1*]
using *parse-es-cpts-i2-fst* **by** *blast*
with *d0 d01 e1 e2 e3* **show** *?thesis*
proof −
{
fix *i*
assume *f0*: *length rlst* ≤ *i* ∧ *i* < *length elst*
have *length* (*elst* ! *i*) ≥ *2* ∧ *getspc-es* (*elst* ! *i* ! *0*) = *EvtSys es*
∧ *getspc-es* (*elst* ! *i* ! *1*) ≠ *EvtSys es*
proof(*cases length rlst* = *i*)
assume *g0*: *length rlst* = *i*
then have *elst* ! *i* = *?elst2*!*0* **by** (*simp add*: *e2 e3 nth-append*)
with *e4* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 butlast-snoc*
butlast-take c2 d0 diff-Suc-1 length-Cons length-append-singleton
length-take lessI list.size(*3*) *min.absorb2 nth-Cons-0*
nth-append-length nth-take)

next
assume *g0*: $\neg$ (*length rlst* = *i*)
with *f0* **have** *length rlst* < *i* $\wedge$ *i* < *length elst* **by** *simp*
with *e1* **show** *?thesis* **by** (*metis Nil-is-append-conv Suc-leI a0 b1*
c1 d4 e2 e3 length-append-singleton)
qed
}
then show *?thesis* **by** *auto*
qed
qed
next
assume *d0*: $\neg$(*getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
then have *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* =
parse-es-cpts-i2 esl1 es (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *auto*
with *b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))
by *simp*
with *a0 c1* **show** *?thesis* **using** *b1* **by** (*metis length-list-update list-update-nonempty*)
qed
qed
}
then show *?thesis* **by** *blast*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *parse-es-cpts-i2-start-withlen0* [*simp*]:
⟦*esl*∈*cpts-es* Γ; *rlst* $\neq$ []; *elst* = *parse-es-cpts-i2 esl es rlst*⟧ $\Longrightarrow$
$\forall$ *i*. *i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$ *length* (*elst*!*i*) $\geq$ *2*
$\wedge$ *getspc-es* (*elst*!*i*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*1*) $\neq$ *EvtSys es*
using *parse-es-cpts-i2-start-withlen* **by** *fastforce*

lemma *parse-es-cpts-i2-fstempty*: ⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
rlst = *parse-es-cpts-i2 esl es* [[]]⟧ $\Longrightarrow$ *rlst*!*0* =[]
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *rlst* = *parse-es-cpts-i2 esl es* [[]]
then have *rlst* = *parse-es-cpts-i2* ((*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*) *es* ([[]]@[[(*EvtSys es*, *s*, *x*)]])
by (*simp add*:*getspc-es-def*)
moreover from *p0 p1* **have** (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs* ∈ *cpts-es* Γ

using *cpts-es-dropi* **by** *force*
ultimately have *rlst* = [[]]@ *parse-es-cpts-i2* ((*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*) *es* ([[(*EvtSys es*, *s*, *x*)]])
using *parse-es-cpts-i2-lst0* **by** *blast*
then show *?thesis* **by** *simp*
qed

lemma *parse-es-cpts-i2-concat3*: ⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
rlst = *parse-es-cpts-i2 esl es* [[]]⟧ ⟹ *concat* (*tl rlst*) = *esl*
using *parse-es-cpts-i2-concat1 parse-es-cpts-i2-fstempty*
by (*smt append-Nil concat.simps*(*1*) *concat.simps*(*2*) *hd-Cons-tl list.distinct*(*1*) *nth-Cons-0*)

lemma *parse-es-cpts-i2-noent-mid0*:
∀ *esl elst l es*. *esl*∈*cpts-es* Γ ∧ *elst* = *parse-es-cpts-i2 esl es* [*l*] ⟶
¬(*length l* > *1* ∧ *getspc-es* (*last l*) = *EvtSys es* ∧ *getspc-es* (*esl*!*0*) ≠ *EvtSys es*) ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length l* ∧
getspc-es (*l*!*j*) = *EvtSys es* ∧ *getspc-es* (*l*!*Suc j*) ≠ *EvtSys es*) ⟶
(∀ *i*. *i* < *length elst* ⟶ ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*))
proof −
{
fix *esl*
have ∀ *elst l es*. *esl*∈*cpts-es* Γ ∧ *elst* = *parse-es-cpts-i2 esl es* [*l*] ⟶
¬(*length l* > *1* ∧ *getspc-es* (*last l*) = *EvtSys es* ∧ *getspc-es* (*esl*!*0*) ≠ *EvtSys es*) ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length l* ∧
getspc-es (*l*!*j*) = *EvtSys es* ∧ *getspc-es* (*l*!*Suc j*) ≠ *EvtSys es*) ⟶
(∀ *i*. *i* < *length elst* ⟶ ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*))
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: ∀ *elst l es*. *esl1*∈*cpts-es* Γ ∧ *elst* = *parse-es-cpts-i2 esl1 es* [*l*] ⟶
¬(*length l* > *1* ∧ *getspc-es* (*last l*) = *EvtSys es* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*) ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length l* ∧
getspc-es (*l*!*j*) = *EvtSys es* ∧ *getspc-es* (*l*!*Suc j*) ≠ *EvtSys*

es) ⟶
(∀ *i*. *i* < *length elst* ⟶ ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*))
then show *?case*
proof −
{
fix *elst l es*
assume *b0*: *esc* # *esl1* ∈ *cpts-es* Γ
and *b1*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*]
and *b2*: ¬ (*length l* > *1* ∧ *getspc-es* (*last l*) = *EvtSys es* ∧ *getspc-es* ((*esc* # *esl1*) ! *0*) ≠ *EvtSys es*)
and *b3*: ¬ (∃*j*>*0*. *Suc j* < *length l* ∧ *getspc-es* (*l* ! *j*) = *EvtSys es* ∧ *getspc-es* (*l* ! *Suc j*) ≠ *EvtSys es*)
have (∀ *i*. *i* < *length elst* ⟶ ¬ (∃*j*>*0*. *Suc j* < *length* (*elst* ! *i*) ∧
getspc-es (*elst* ! *i* ! *j*) = *EvtSys es* ∧ *getspc-es* (*elst* ! *i* ! *Suc j*) ≠ *EvtSys es*))
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
then have *c1*: *parse-es-cpts-i2* (*esc* # []) *es* [*l*] =
parse-es-cpts-i2 [] *es* (*list-update* [*l*] (*length* [*l*] − *1*) (*last* [*l*] @ [*esc*])) **by** *simp*
have *c2*: *parse-es-cpts-i2* [] *es* (*list-update* [*l*] (*length* [*l*] − *1*) (*last* [*l*] @ [*esc*]))
= *list-update* [*l*] (*length* [*l*] − *1*) (*last* [*l*] @ [*esc*]) **by** *simp*
with *b1 c0 c1* **have** *elst* = *list-update* [*l*] (*length* [*l*] − *1*) (*last* [*l*] @ [*esc*]) **by** *simp*
then have *elst* = [*l* @ [*esc*]] **by** *simp*
with *b2 b3* **show** *?thesis* **by** (*smt Suc-eq-plus1-left Suc-lessD Suc-lessI diff-Suc-1*
dual-order.strict-trans last-conv-nth length-Cons length-append-singleton
less-antisym less-one list.size(*3*) *nat-neq-iff nth-Cons-0 nth-append nth-append-length*)

next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*
by (*meson neq-Nil-conv*)
show *?thesis*
proof(*cases getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
assume *d0*: *getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*
with *c2* **have** *d01*: *getspc-es ec1* ≠ *EvtSys es* **by** *simp*
from *d0* **have** *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*] = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]])

by *simp*
with *b1 b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) **by** *simp*
from *c1* **have** *d4*: *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) = [*l*]@*parse-es-cpts-i2 esl1 es* ([[*esc*]])
using *parse-es-cpts-i2-lst0* **by** *blast*
with *d2* **have** *d3*: *elst* = [*l*] @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*
let *?elst1* = *parse-es-cpts-i2 esl1 es* ([[*esc*]])
have ¬(*length* [*esc*] > *1* ∧ *getspc-es* (*last* [*esc*]) = *EvtSys es* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
by *simp*
moreover have ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* [*esc*] ∧
getspc-es ([*esc*]!*j*) = *EvtSys es* ∧ *getspc-es* ([*esc*]!*Suc j*) ≠ *EvtSys es*) **by** *simp*
ultimately have ∀ *i*. *i* < *length ?elst1* ⟶ ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*?elst1*!*i*) ∧
getspc-es (*?elst1*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*?elst1*!*i*!*Suc j*) ≠ *EvtSys es*)
using *a0 c1* **by** *simp*
with *b3 d3* **show** *?thesis* **by** (*smt Nil-is-append-conv Nitpick.size-list-simp*(*2*)
One-nat-def Suc-diff-Suc Suc-less-eq append-Cons append-Nil
diff-Suc-1 diff-Suc-Suc list.sel(*3*) *not-gr0 nth-Cons'*)
next
assume *d0*: ¬(*getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
then have *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*] =
parse-es-cpts-i2 esl1 es (*list-update* [*l*] (*length* [*l*] − *1*) (*last* [*l*] @ [*esc*]))
by *auto*
with *b1* **have** *d1*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*@[*esc*]]) **by** *simp*
show *?thesis*
proof(*cases length esl1* = *0*)
assume *e0*: *length esl1* = *0*
then have *e1*: *esl1* = [] **by** *simp*
with *d1* **have** *elst* = [*l*@[*esc*]] **by** *simp*
with *b2* **show** *?thesis* **using** *e1 c0* **by** *linarith*
next
assume *e0*: ¬(*length esl1* = *0*)
then have *length esl1* > *0* **by** *simp*
with *d0* **have** *e1*: ¬(*getspc-es esc* = *EvtSys es* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*) **by** *simp*
then have ¬ (*1* < *length* (*l*@[*esc*]) ∧ *getspc-es* (*last* (*l*@[*esc*])) = *EvtSys es*
∧ *getspc-es* (*esl1* ! *0*) ≠ *EvtSys es*) **by** *auto*
moreover from *b2 b3* **have** ¬ (∃ *j*>*0*. *Suc j* < *length* (*l*@[*esc*]) ∧ *getspc-es* ((*l*@[*esc*]) ! *j*) = *EvtSys es* ∧

getspc-es ((*l*@[*esc*]) ! *Suc j*) ≠ *EvtSys es*)
by (*metis* (*no-types*, *hide-lams*) *Suc-neq-Zero diff-Suc-1*
last-conv-nth
length-append-singleton less-antisym list.size(*3*) *not-gr0*
not-less-eq
nth-Cons-0 nth-append zero-less-diff)
ultimately show *?thesis* **using** *a0 d1 c1* **by** *blast*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *parse-es-cpts-i2-noent-mid*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *parse-es-cpts-i2 esl es* [[]]⟧ ⟹ ∀ *i*. *i* < *length* (*tl elst*) ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* ((*tl elst*)!*i*) ∧
getspc-es ((*tl elst*)!*i*!*j*) = *EvtSys es* ∧ *getspc-es* ((*tl elst*)!*i*!*Suc*
j) ≠ *EvtSys es*)
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *elst* = *parse-es-cpts-i2 esl es* [[]]
then have ¬(*length* [] > *1* ∧ *getspc-es* (*last* []) = *EvtSys es* ∧ *getspc-es* (*esl*!*0*)
≠ *EvtSys es*) **by** *simp*
moreover have ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* [] ∧
getspc-es ([]!*j*) = *EvtSys es* ∧ *getspc-es* ([]!*Suc j*) ≠ *EvtSys es*)
by *simp*
ultimately have ∀ *i*. *i* < *length elst* ⟶ ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*)
∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠
EvtSys es)
using *p1 p2 parse-es-cpts-i2-noent-mid0* **by** *blast*
then show *?thesis* **by** (*metis* (*no-types*, *lifting*) *List.nth-tl Nitpick.size-list-simp*(*2*)
Suc-mono list.sel(*2*))
qed

lemma *parse-es-cpts-i2-start-aux*: ⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys*
es), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *parse-es-cpts-i2 esl es* [[]]⟧ ⟹
∀ *i*. *i* < *length* (*tl elst*) ⟶ *length* ((*tl elst*)!*i*) ≥ *2* ∧
getspc-es ((*tl elst*)!*i*!*0*) = *EvtSys es* ∧ *getspc-es* ((*tl elst*)!*i*!*1*) ≠ *EvtSys es*

proof −
assume *p0*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p1*: $esl \in cpts\text{-}es\ \Gamma$
and *p2*: $elst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]$
from *p1 p2* **have** *a0*: $\forall i.\ i \geq length\ [[]] \wedge i < length\ elst \longrightarrow length\ (elst!i) \geq 2\ \wedge$
$getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \neq EvtSys\ es$
by (*metis length-Cons list.distinct*(*2*) *list.size*(*3*) *parse-es-cpts-i2-start-withlen0*)

then show *?thesis*
proof −
{
fix *i*
assume *b0*: $i < length\ (tl\ elst)$
from *a0 b0* **have** $length\ (tl\ elst\ !\ i) \geq 2$
by (*metis List.nth-tl Nil-tl Nitpick.size-list-simp*(*2*) *One-nat-def Suc-eq-plus1-left Suc-less-eq le-add1 length-Cons less-nat-zero-code*)
moreover from *a0 b0* **have** $getspc\text{-}es\ (elst!Suc\ i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!Suc\ i!1) \neq EvtSys\ es$
by *force*
moreover from *b0* **have** $(tl\ elst)!i = elst!Suc\ i$ **by** (*simp add*: *List.nth-tl*)
ultimately have $length\ (tl\ elst\ !\ i) \geq 2 \wedge getspc\text{-}es\ ((tl\ elst)!i!0) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ ((tl\ elst)!i!1) \neq EvtSys\ es$ **by** *simp*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *parse-es-cpts-i2-noent-mid-i*:
$[\![esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;\ esl \in cpts\text{-}es\ \Gamma;$
$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]);\ Suc\ i < length\ elst;\ esl1 = elst!i@[elst!Suc\ i!0]]\!] \Longrightarrow$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl1\ \wedge$
$getspc\text{-}es\ (esl1!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl1!Suc\ j) \neq EvtSys\ es)$
proof −
assume *p0*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p1*: $esl \in cpts\text{-}es\ \Gamma$
and *p2*: $elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
and *p3*: $Suc\ i < length\ elst$
and *p4*: $esl1 = elst!i@[elst!Suc\ i!0]$
let $?esl2 = elst!i$
from *p0 p1 p2 p3* **have** $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ ?esl2\ \wedge$
$getspc\text{-}es\ (?esl2!j) = EvtSys\ es \wedge getspc\text{-}es\ (?esl2!Suc\ j) \neq EvtSys\ es)$
using *parse-es-cpts-i2-noent-mid*[*of esl es s x e s1 x1 xs* Γ *elst*]
by (*meson Suc-lessD parse-es-cpts-i2-noent-mid*)
moreover
from *p0 p1 p2 p3* **have** $getspc\text{-}es\ (elst!Suc\ i!0) = EvtSys\ es$

using *parse-es-cpts-i2-start-aux*[*of esl es s x e s1 x1 xs* Γ
parse-es-cpts-i2 esl es [[]]] **by** *blast*
ultimately show *?thesis* **by** (*simp add*: *nth-append p4*)
qed

lemma *parse-es-cpts-i2-drop-cptes*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
∀ *i*. *i* < *length elst* ⟶ *concat* (*drop i elst*)∈*cpts-es* Γ
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
then have *a1*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
{
fix *i*
assume *b0*: *i* < *length elst*
then have *concat* (*drop i elst*)∈*cpts-es* Γ
proof(*induct i*)
case *0* **with** *p1 a1* **show** *?case* **by** *auto*
next
case (*Suc j*)
assume *c0*: *j* < *length elst* ⟹ *concat* (*drop j elst*) ∈ *cpts-es* Γ
and *c1*: *Suc j* < *length elst*
then have *c2*: *concat* (*drop* (*Suc j*) *elst*) = *drop* (*length* (*elst*!*j*)) (*concat*
(*drop j elst*))
by (*metis Cons-nth-drop-Suc Suc-lessD append-eq-conv-conj con-
cat.simps*(*2*))
from *c0 c1* **have** *concat* (*drop j elst*) ∈ *cpts-es* Γ **by** *simp*
with *c1 c2* **show** *?case*
using *cpts-es-dropi2*[*of concat* (*drop j elst*) Γ *length* (*elst* ! *j*)]
by (*smt List.nth-tl Suc-leI Suc-lessE concat-last-lm diff-Suc-1 drop.simps*(*1*)

last-conv-nth last-drop le-less-trans length-0-conv length-Cons length-drop

length-greater-0-conv length-tl lessI numeral-2-eq-2 p1 p2 parse-es-cpts-i2-start-withlen0

zero-less-diff)
qed
}
then show *?thesis* **by** *auto*
qed

lemma *parse-es-cpts-i2-in-cptes-i*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
∀ *i*. *Suc i* < *length elst* ⟶ (*elst*!*i*)@[*elst*!*Suc i*!*0*] ∈*cpts-es* Γ
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*

and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
then have *p3*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
from *p0 p1 p2* **have** *p4*: ∀ *i*. *i* < *length elst* ⟶ *length* (*elst*!*i*) ≥ *2*
using *parse-es-cpts-i2-start-aux*[*of esl es s x e s1 x1 xs* Γ *parse-es-cpts-i2 esl es* [[]]]
by *simp*

{
fix *i*
assume *a0*: *Suc i* < *length elst*
have (*elst*!*i*)@[*elst*!*Suc i*!*0*] ∈*cpts-es* Γ
proof(*cases i* = *0*)
assume *b0*: *i* = *0*
with *a0 p4* **have** *b1*: *length* (*elst*!*1*) ≥ *2* **by** *auto*
from *p3 a0* **have** *esl* = (*elst*!*0*) @ *concat* (*drop 1 elst*)
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD b0 concat.simps*(*2*) *drop-0*)
with *a0* **have** *esl* = (*elst*!*0*) @ ((*elst*!*1*) @ *concat* (*drop 2 elst*))
by (*metis Cons-nth-drop-Suc One-nat-def Suc-1 b0 concat.simps*(*2*))
with *a0 b0 b1* **have** *take* ((*length* (*elst* ! *0*)) + *1*) *esl* = (*elst* ! *0*) @ [*elst*!*Suc 0*!*0*]
by (*smt Cons-nth-drop-Suc Nil-is-append-conv One-nat-def Suc-1 Suc-le-lessD*
append.simps(*1*) *append.simps*(*2*) *append-eq-conv-conj drop-0 length-greater-0-conv*
list.size(*3*) *not-less0 nth-Cons-0 take-0 take-Suc-conv-app-nth take-add*)

with *p1 b0* **show** *?thesis* **using** *cpts-es-take*[*of esl* Γ *length* (*elst* ! *0*)]
by (*metis One-nat-def Suc-lessD add.right-neutral add-Suc-right le-less-linear take-all*)
next
assume *i*≠*0*
then have *b0*: *i* > *0* **by** *simp*
let *?elst* = *drop* (*i* − *1*) *elst*
let *?esl* = *concat ?elst*
from *a0 b0* **have** *b01*: *length ?elst* > *2* **by** *simp*
from *a0 p4 b0* **have** *b1*: *length* (*?elst*!*1*) ≥ *2* **by** *auto*
from *p0 p1 p2 a0 b1* **have** *b2*: *?esl*∈*cpts-es* Γ
using *parse-es-cpts-i2-drop-cptes*[*of esl es s x e s1 x1 xs* Γ *elst*]
One-nat-def Suc-lessD Suc-pred b0 **by** *presburger*
from *p3 a0* **have** *b3*: *?esl* = (*?elst*!*0*) @ *concat* (*drop 1 ?elst*)
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD Suc-pred b0*
concat.simps(*2*) *drop-0 length-drop zero-less-diff*)
with *a0* **have** *?esl* = (*?elst*!*0*) @ ((*?elst*!*1*) @ *concat* (*drop 2 ?elst*))
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-1*
Suc-leI Suc-lessD b0 concat.simps(*2*) *diff-diff-cancel diff-le-self*
diff-less-mono length-drop)
with *b0 b01 b1* **have** *take* ((*length* (*?elst* ! *0*)) + *1*) *?esl* = (*?elst* ! *0*) @

[?elst!1!0]
    by (*smt Cons-nth-drop-Suc Nil-is-append-conv One-nat-def append.simps*(2)

      append-eq-conv-conj drop-0 length-greater-0-conv list.size(3) *not-numeral-le-zero*

        nth-Cons-0 take-0 take-Suc-conv-app-nth take-add)
    with *b2* **show** *?thesis* **using** *cpts-es-take*[*of ?esl* Γ *length* (*?elst* ! *0*)]
        by (*smt Nil-is-append-conv a0 concat-i-lm cpts-es-seg2 list.size*(3)
not-Cons-self2
        not-numeral-le-zero p0 p1 p2 p3 parse-es-cpts-i2-start-aux)
    qed
  }
  then show *?thesis* **by** *auto*
qed

lemma *parse-es-cpts-i2-in-cptes-last*:
 ⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
    elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
    last elst ∈*cpts-es* Γ
 proof −
  assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
   and *p1*: *esl*∈*cpts-es* Γ
   and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
  then have ∀ *i*. *i* < *length elst* ⟶ *concat* (*drop i elst*)∈*cpts-es* Γ
   using *parse-es-cpts-i2-drop-cptes*[*of esl es s x e s1 x1 xs* Γ *elst*] **by** *fastforce*
  then show *?thesis*
   by (*metis* (*no-types*, *lifting*) *append-butlast-last-id append-eq-conv-conj*
    concat.simps(1) *concat.simps*(2) *diff-less length-butlast length-greater-0-conv*

     less-one list.simps(3) *p0 p1 p2 parse-es-cpts-i2-concat3 self-append-conv*)
 qed

lemma *evtsys-fst-ent*:
   ⟦*esl* ∈ *cpts-es* Γ; *getspc-es* (*esl* ! *0*) = *EvtSys es*; *Suc m* ≤ *length esl*; ∃ *i*. *i* ≤
m ∧ *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*⟧
   ⟹ ∃ *i*. (*i* < *m* ∧ *getspc-es* (*esl* ! *i*) = *EvtSys es* ∧ *getspc-es* (*esl* ! *Suc i*)
≠ *EvtSys es*)
      ∧ (∀ *j*. *j* < *i* ⟶ *getspc-es* (*esl* ! *j*) = *EvtSys es*)
 proof −
  assume *p0*: *esl* ∈ *cpts-es* Γ
   and *p1*: *getspc-es* (*esl* ! *0*) = *EvtSys es*
   and *p2*: *Suc m* ≤ *length esl*
   and *p3*: ∃ *i*. *i* ≤ *m* ∧ *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*
  have ∀ *m*. *esl* ∈ *cpts-es* Γ ∧ *getspc-es* (*esl* ! *0*) = *EvtSys es* ∧ *Suc m* ≤ *length*
esl
       ∧ (∃ *i*. *i* ≤ *m* ∧ *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*)
     ⟶ (∃ *i*. (*i* < *m* ∧ *getspc-es* (*esl* ! *i*) = *EvtSys es* ∧ *getspc-es* (*esl* ! *Suc*
i) ≠ *EvtSys es*)

$\wedge\ (\forall j.\ j < i \longrightarrow \textit{getspc-es}\ (\textit{esl}\ !\ j) = \textit{EvtSys es}))$

proof −

{

fix *m*

assume *a0*: $\textit{esl} \in \textit{cpts-es}\ \Gamma$

and *a1*: $\textit{getspc-es}\ (\textit{esl}\ !\ 0) = \textit{EvtSys es}$

and *a2*: $\textit{Suc}\ m \leq \textit{length esl}$

and *a3*: $\exists i.\ i \leq m \wedge \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es}$

then have $\exists i.\ (i < m \wedge \textit{getspc-es}\ (\textit{esl}\ !\ i) = \textit{EvtSys es}$
$\wedge\ \textit{getspc-es}\ (\textit{esl}\ !\ \textit{Suc}\ i) \neq \textit{EvtSys es})$
$\wedge\ (\forall j.\ j < i \longrightarrow \textit{getspc-es}\ (\textit{esl}\ !\ j) = \textit{EvtSys es})$

proof(*induct m*)

case *0* **show** *?case* **using** *0.prems(4) p1* **by** *auto*

next

case (*Suc n*)

assume *b0*: $\textit{esl} \in \textit{cpts-es}\ \Gamma \Longrightarrow$
$\textit{getspc-es}\ (\textit{esl}\ !\ 0) = \textit{EvtSys es} \Longrightarrow$
$\textit{Suc}\ n \leq \textit{length esl} \Longrightarrow$
$\exists i{\leq}n.\ \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es} \Longrightarrow$
$\exists i.\ (i < n \wedge \textit{getspc-es}\ (\textit{esl}\ !\ i) = \textit{EvtSys es}$
$\wedge\ \textit{getspc-es}\ (\textit{esl}\ !\ \textit{Suc}\ i) \neq \textit{EvtSys es})$
$\wedge\ (\forall j{<}i.\ \textit{getspc-es}\ (\textit{esl}\ !\ j) = \textit{EvtSys es})$

and *b1*: $\textit{esl} \in \textit{cpts-es}\ \Gamma$

and *b2*: $\textit{getspc-es}\ (\textit{esl}\ !\ 0) = \textit{EvtSys es}$

and *b3*: $\textit{Suc}\ (\textit{Suc}\ n) \leq \textit{length esl}$

and *b4*: $\exists i{\leq}\textit{Suc}\ n.\ \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es}$

show *?case*

proof(*cases* $\exists i{\leq}n.\ \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es}$)

assume *c0*: $\exists i{\leq}n.\ \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es}$

with *b0 b1 b2 b3* **have** $\exists i.\ (i < n \wedge \textit{getspc-es}\ (\textit{esl}\ !\ i) = \textit{EvtSys es}$
$\wedge\ \textit{getspc-es}\ (\textit{esl}\ !\ \textit{Suc}\ i) \neq \textit{EvtSys es})$
$\wedge\ (\forall j{<}i.\ \textit{getspc-es}\ (\textit{esl}\ !\ j) = \textit{EvtSys es})$ **by** *simp*

then show *?thesis* **using** *less-Suc-eq* **by** *auto*

next

assume *c0*: $\neg(\exists i{\leq}n.\ \textit{getspc-es}\ (\textit{esl}\ !\ i) \neq \textit{EvtSys es})$

with *b4* **have** $\textit{getspc-es}\ (\textit{esl}\ !\ \textit{Suc}\ n) \neq \textit{EvtSys es}$

using *le-SucE* **by** *auto*

moreover from *c0* **have** $\forall j{<}n.\ \textit{getspc-es}\ (\textit{esl}\ !\ j) = \textit{EvtSys es}$ **by** *auto*

moreover from *c0* **have** $\textit{getspc-es}\ (\textit{esl}\ !\ n) = \textit{EvtSys es}$ **by** *auto*

ultimately show *?thesis* **by** *blast*

qed

qed

}

then show *?thesis* **by** *auto*

qed

then show *?thesis* **using** *p0 p1 p2 p3* **by** *blast*

qed

lemma *rm-evtsys-in-cptse0*:
  $[\![esl \in cpts\text{-}es\ \Gamma;\ length\ esl > 0;\ \exists\, e.\ getspc\text{-}es\ (esl!0) = EvtSeq\ e\ (EvtSys\ es);$
  $\neg(\exists\, j.\ Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)\ ]\!]$
  $\Longrightarrow rm\text{-}evtsys\ esl \in cpts\text{-}ev\ \Gamma$
 proof −
  assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
   and *p1*: $length\ esl > 0$
   and *p2*: $\exists\, e.\ getspc\text{-}es\ (esl!0) = EvtSeq\ e\ (EvtSys\ es)$
   and *p3*: $\neg(\exists\, j.\ Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
  have $\forall\, esl\ e\ es\ .esl \in cpts\text{-}es\ \Gamma \wedge length\ esl > 0 \wedge (\exists\, e.\ getspc\text{-}es\ (esl!0) = EvtSeq\ e\ (EvtSys\ es)) \wedge$
  $\neg(\exists\, j.\ Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
   $\longrightarrow rm\text{-}evtsys\ esl \in cpts\text{-}ev\ \Gamma$
   proof −
   {
    fix *esl e es*
    assume *a0*: $esl \in cpts\text{-}es\ \Gamma$
     and *a1*: $length\ esl > 0$
     and *a2*: $\exists\, e.\ getspc\text{-}es\ (esl!0) = EvtSeq\ e\ (EvtSys\ es)$
     and *a3*: $\neg(\exists\, j.\ Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
    from *a0 a1 a2 a3* **have** $rm\text{-}evtsys\ esl \in cpts\text{-}ev\ \Gamma$
     proof(*induct esl*)
      case (*CptsEsOne es1 s x*)
      show *?case*
       proof(*induct es1*)
        case (*EvtSeq x1 es1*)
        have $rm\text{-}evtsys\ [(EvtSeq\ x1\ es1,\ s,\ x)] = [(x1,\ s,\ x)]$
         by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
        then show *?case* **by** (*simp add: cpts-ev.CptsEvOne*)
       next
        case (*EvtSys xa*)
        have $rm\text{-}evtsys\ [(EvtSys\ xa,\ s,\ x)] = [(AnonyEvent\ fin\text{-}com,\ s,\ x)]$
         by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
        then show *?case* **by** (*simp add: cpts-ev.CptsEvOne*)
       qed
     next
      case (*CptsEsEnv es1 t x xs s y*)
      assume *b0*: $(es1,\ t,\ x)\ \#\ xs \in cpts\text{-}es\ \Gamma$
       and *b1*: $0 < length\ ((es1,\ t,\ x)\ \#\ xs) \Longrightarrow$
         $\exists\, e.\ getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ 0) = EvtSeq\ e\ (EvtSys\ es)$
$\Longrightarrow$

$\neg (\exists j.\ Suc\ j < length\ ((es1,\ t,\ x)\ \#\ xs) \wedge$
$getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ j) = EvtSys\ es \wedge$
$getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ Suc\ j) \neq EvtSys\ es) \Longrightarrow$
$rm\text{-}evtsys\ ((es1,\ t,\ x)\ \#\ xs) \in cpts\text{-}ev\ \Gamma$
and $b2$: $0 < length\ ((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs)$
and $b3$: $\exists e.\ getspc\text{-}es\ (((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs)\ !\ 0) = EvtSeq\ e\ (EvtSys\ es)$
and $b4$: $\neg (\exists j.\ Suc\ j < length\ ((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs) \wedge$
$getspc\text{-}es\ (((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs)\ !\ j) = EvtSys\ es \wedge$
$getspc\text{-}es\ (((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs)\ !\ Suc\ j) \neq EvtSys\ es)$
from $b4$ **have** $\neg (\exists j.\ Suc\ j < length\ ((es1,\ t,\ x)\ \#\ xs) \wedge$
$getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ j) = EvtSys\ es \wedge$
$getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ Suc\ j) \neq EvtSys\ es)$ **by** *force*
moreover have $\exists e.\ getspc\text{-}es\ (((es1,\ t,\ x)\ \#\ xs)\ !\ 0) = EvtSeq\ e\ (EvtSys\ es)$
proof –
from $b3$ **obtain** e **where** $getspc\text{-}es\ (((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs)\ !\ 0) = EvtSeq\ e\ (EvtSys\ es)$
by *auto*
then have $es1 = EvtSeq\ e\ (EvtSys\ es)$ **by** (*simp add:getspc-es-def*)
then show *?thesis* **by** (*simp add:getspc-es-def*)
qed
ultimately have $rm\text{-}evtsys\ ((es1,\ t,\ x)\ \#\ xs) \in cpts\text{-}ev\ \Gamma$ **using** *b1 b3* **by** *blast*
then have $b4$: $rm\text{-}evtsys1\ (es1,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs \in cpts\text{-}ev\ \Gamma$ **by** (*simp add:rm-evtsys-def*)
have $b5$: $rm\text{-}evtsys\ ((es1,\ s,\ y)\ \#\ (es1,\ t,\ x)\ \#\ xs) =$
$rm\text{-}evtsys1\ (es1,\ s,\ y)\ \#\ rm\text{-}evtsys1\ (es1,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs$
by (*simp add:rm-evtsys-def*)
from $b4$ **show** *?case*
proof(*induct es1*)
case(*EvtSeq x1 es2*)
assume $c0$: $rm\text{-}evtsys1\ (EvtSeq\ x1\ es2,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs \in cpts\text{-}ev\ \Gamma$
have $rm\text{-}evtsys\ ((EvtSeq\ x1\ es2,\ s,\ y)\ \#\ (EvtSeq\ x1\ es2,\ t,\ x)\ \#\ xs) =$
$(x1,s,y)\ \#\ (x1,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs$
by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
moreover from $c0$ **have** $(x1,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs \in cpts\text{-}ev\ \Gamma$
by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
ultimately show *?case* **by** (*simp add: cpts-ev.CptsEvEnv*)
next
case (*EvtSys xa*)
assume $c0$: $rm\text{-}evtsys1\ (EvtSys\ xa,\ t,\ x)\ \#\ rm\text{-}evtsys\ xs \in cpts\text{-}ev\ \Gamma$

have *rm-evtsys* ((*EvtSys xa*, *s*, *y*) # (*EvtSys xa*, *t*, *x*) # *xs*) =
(*AnonyEvent fin-com*, *s*, *y*) # (*AnonyEvent fin-com*, *t*, *x*) #
rm-evtsys xs
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def*
getx-es-def)
moreover from *c0* **have** (*AnonyEvent fin-com*,*t*, *x*) # *rm-evtsys xs*
$\in$ *cpts-ev* Γ
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def*
getx-es-def)
ultimately show *?case* **by** (*simp add*: *cpts-ev*.*CptsEvEnv*)
qed
next
case (*CptsEsComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *b0*: $\Gamma \vdash$ (*e1*, *s1*, *x1*) $-es-et\rightarrow$ (*e2*, *t1*, *y1*)
and *b1*: (*e2*, *t1*, *y1*) # *xs1* $\in$ *cpts-es* Γ
and *b2*: *0* < *length* ((*e2*, *t1*, *y1*) # *xs1*) $\Longrightarrow$
$\exists$ *e*. *getspc-es* (((*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e* (*EvtSys*
es) $\Longrightarrow$
$\neg$ ($\exists$ *j*. *Suc j* < *length* ((*e2*, *t1*, *y1*) # *xs1*) $\wedge$
getspc-es (((*e2*, *t1*, *y1*) # *xs1*) ! *j*) = *EvtSys es* $\wedge$
getspc-es (((*e2*, *t1*, *y1*) # *xs1*) ! *Suc j*) $\neq$ *EvtSys es*)
$\Longrightarrow$
rm-evtsys ((*e2*, *t1*, *y1*) # *xs1*) $\in$ *cpts-ev* Γ
and *b3*: *0* < *length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*)
and *b4*: $\exists$ *e*. *getspc-es* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *0*) =
EvtSeq e (*EvtSys es*)
and *b5*: $\neg$ ($\exists$ *j*. *Suc j* < *length* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) $\wedge$
getspc-es (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *j*) =
EvtSys es $\wedge$
getspc-es (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) ! *Suc j*)
$\neq$ *EvtSys es*)
have *b6*: *rm-evtsys* ((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) # *xs1*) =
rm-evtsys1 (*e1*, *s1*, *x1*) # *rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys*
xs1
by (*simp add*:*rm-evtsys-def*)
from *b4* **obtain** *e'* **where** *getspc-es* (((*e1*, *s1*, *x1*) # (*e2*, *t1*, *y1*) #
xs1) ! *0*) = *EvtSeq e'* (*EvtSys es*)
by *auto*
then have *b7*: *e1* = *EvtSeq e'* (*EvtSys es*) **by** (*simp add*:*getspc-es-def*)
show *?case*
proof(*cases* $\exists$ *e*. *e2* = *EvtSeq e* (*EvtSys es*))
assume *c0*: $\exists$ *e*. *e2* = *EvtSeq e* (*EvtSys es*)
then obtain *e* **where** *c1*: *e2* = *EvtSeq e* (*EvtSys es*) **by** *auto*
then have *c2*: $\exists$ *e*. *getspc-es* (((*e2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e*
(*EvtSys es*)
by (*simp add*:*getspc-es-def*)
moreover from *b5* **have** $\neg$ ($\exists$ *j*. *Suc j* < *length* ((*e2*, *t1*, *y1*) # *xs1*)
$\wedge$
getspc-es (((*e2*, *t1*, *y1*) # *xs1*) ! *j*) = *EvtSys es* $\wedge$

getspc-es $(((e2,\ t1,\ y1)\ \#\ xs1)\ !\ Suc\ j) \neq EvtSys\ es)$
by *force*
ultimately have *c3*: *rm-evtsys* $((e2,\ t1,\ y1)\ \#\ xs1) \in$ *cpts-ev* Γ
using *b2* **by** *blast*
then have *c5*: *rm-evtsys1* $(e2,\ t1,\ y1)$ # *rm-evtsys xs1* $\in$ *cpts-ev* Γ
by (*simp add*:*rm-evtsys-def*)

from *b0 c1 b7* **have** $\exists\, t.\ \Gamma \vdash (e',\ s1,\ x1)\ -et-t\rightarrow\ (e,\ t1,\ y1)$
using *evtseq-tran-exist-etran* **by** *simp*
then obtain *t* **where** *c8*: $\Gamma \vdash (e',\ s1,\ x1)\ -et-t\rightarrow\ (e,\ t1,\ y1)$ **by**
auto
from *b7* **have** *rm-evtsys1* $(e1,\ s1,\ x1) = (e',\ s1,\ x1)$
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def*
getx-es-def)
moreover from *c1* **have** *rm-evtsys1* $(e2,\ t1,\ y1) = (e,\ t1,\ y1)$
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def*
getx-es-def)
ultimately show *?thesis* **using** *b6 c8 c5* **using** *cpts-ev.CptsEvComp*
by *fastforce*
next
assume *c0*: $\neg(\exists\, e.\ e2 = EvtSeq\ e\ (EvtSys\ es))$
with *b0 b7* **have** *c1*: $e2 = EvtSys\ es$ **by** (*meson evtseq-tran-evtseq*)
then have *c11*: *rm-evtsys1* $(e2,\ t1,\ y1)$ # *rm-evtsys xs1* $\in$ *cpts-ev* Γ
proof −
from *b5* **have** *d0*: $\neg\ (\exists j.\ Suc\ j < length\ ((e2,\ t1,\ y1)\ \#\ xs1)\ \wedge$
getspc-es $(((e2,\ t1,\ y1)\ \#\ xs1)\ !\ j) = EvtSys\ es\ \wedge$
getspc-es $(((e2,\ t1,\ y1)\ \#\ xs1)\ !\ Suc\ j) \neq EvtSys\ es)$ **by**
force
have *d00*: $\forall j.\ j < length\ xs1 \longrightarrow$ *getspc-es* $(xs1!j) = EvtSys\ es$
proof −
{
fix *j*
assume *e0*: $j < length\ xs1$
then have *getspc-es* $(xs1!j) = EvtSys\ es$
proof(*induct j*)
case *0* **from** *b1 c1 d0* **show** *?case*
using *getspc-es-def* **by** (*metis One-nat-def e0 fst-conv*
length-Cons
less-one not-less-eq nth-Cons-0 nth-Cons-Suc)
next
case (*Suc m*)
assume *f0*: $m < length\ xs1 \Longrightarrow$ *getspc-es* $(xs1\ !\ m) =$
EvtSys es
and *f1*: $Suc\ m < length\ xs1$
with *d0* **show** *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

then have *d1*: $\forall j.\ j < $ *length* (*rm-evtsys xs1*) $\longrightarrow$ *getspc-e* ((*rm-evtsys xs1*)!*j*) = *AnonyEvent fin-com*
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def getspc-e-def*)
from *c1* **have** *d2*: *rm-evtsys1* (*e2*, *t1*, *y1*) = (*AnonyEvent fin-com*, *t1*, *y1*)
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def getspc-e-def*)
with *d1* **have** $\forall i.\ i <$ *length* (*rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys xs1*) $\longrightarrow$
getspc-e ((*rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys xs1*)!*i*) = *AnonyEvent fin-com*
using *getspc-e-def less-Suc-eq-0-disj* **by** *force*
moreover have *length* (*rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys xs1*) > 0 **by** *simp*
ultimately show *?thesis* **using** *cpts-ev-same* **by** *blast*

qed
from *b7* **have** *c2*: *rm-evtsys1* (*e1*, *s1*, *x1*) = (e', *s1*, *x1*)
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
from *c1* **have** *c3*: *rm-evtsys1* (*e2*, *t1*, *y1*) = (*AnonyEvent fin-com*, *t1*, *y1*)
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
from *b0 b7 c1* **have** $\exists t.\ \Gamma \vdash$ (e', *s1*, *x1*) $-et-t\rightarrow$ (*AnonyEvent fin-com*, *t1*, *y1*)
using *evtseq-tran-0-exist-etran* **by** *simp*
then obtain *t* **where** $\Gamma \vdash$ (e', *s1*, *x1*) $-et-t\rightarrow$ (*AnonyEvent fin-com*, *t1*, *y1*) **by** *auto*
with *b6 c2 c3 c11* **show** *?thesis* **using** *cpts-ev.CptsEvComp* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p0 p1 p2 p3* **show** *?thesis* **by** *force*
qed

lemma *rm-evtsys-in-cptse*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
$\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))♯*k*$\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬($\exists j.\ j > 0 \wedge$ *Suc j* $<$ *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
⟧ $\Longrightarrow$

el ∈ *cpts-ev* Γ
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
from *p0 p1* **have** *a1*: *?esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
moreover have *a2*: *length ?esl1* > *0* **by** *simp*
moreover have *a3*: ∃ *e*. *getspc-es* (*?esl1* ! *0*) = *EvtSeq e* (*EvtSys es*) **by** (*simp add*:*getspc-es-def*)
moreover from *p1 p3* **have** *a4*: ¬ (∃ *j*. *Suc j* < *length ?esl1* ∧ *getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧ *getspc-es* (*?esl1* ! *Suc j*) ≠ *EvtSys es*) **by** *force*
ultimately have *?esl1* ∈ *cpts-es* Γ **using** *rm-evtsys-in-cptse0* **by** *blast*

with *a1 a2 a3 a4* **have** *a5*: *rm-evtsys ?esl1* ∈ *cpts-ev* Γ **using** *rm-evtsys-in-cptse0* **by** *blast*
have *rm-evtsys ?esl1* = *rm-evtsys1* (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *rm-evtsys xs*
by (*simp add*:*rm-evtsys-def*)
then have *a6*: *rm-evtsys ?esl1* = (*ev*,*s1*,*x1*) # *rm-evtsys xs*
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
from *p2* **have** Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*ev*, *s1*, *x1*)
using *evtsysent-evtent*[*of* Γ *es s x e k ev s1 x1*] **by** *auto*
with *p4 a6* **show** *?thesis* **using** *a5 cpts-ev*.*CptsEvComp* **by** *fastforce*
qed

lemma *fstent-nomident-e-sim-es-aux*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*); *el*∈*cpts-ev* Γ⟧ ⟹
∀ *i*. *i* > *0* ∧ *i* < *length el* ⟶
(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*))
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p3*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)

and *p4*: *el*∈*cpts-ev* Γ
let *?el1* = *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
have *a1*: *length ?esl1* = *length ?el1* **using** *rm-evtsys-same-sx same-s-x-def* **by** *blast*
from *p0 p1* **have** *a2*: *?esl1*∈*cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *p2* **have** *p2-1*: ∀ *j*. *j* > *0* ∧ *Suc j* < *length esl* ⟶
getspc-es (*esl* ! *j*) = *EvtSys es* ⟶ *getspc-es* (*esl* ! *Suc j*) = *EvtSys es*
using *noevtent-inmid-eq* **by** *auto*
have ∀ *i*. *i* < *length ?el1* ⟶
(*getspc-es* (*?esl1*!*i*) = *EvtSys es* ∧ *getspc-e* (*?el1*!*i*) = *AnonyEvent fin-com*)

∨ (*getspc-es* (*?esl1*!*i*) = *EvtSeq* (*getspc-e* (*?el1*!*i*)) (*EvtSys es*))
proof −
{
fix *i*
assume *b0*: *i* < *length ?el1*
then have (*getspc-es* (*?esl1*!*i*) = *EvtSys es* ∧ *getspc-e* (*?el1*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*?esl1*!*i*) = *EvtSeq* (*getspc-e* (*?el1*!*i*)) (*EvtSys es*))
proof(*induct i*)
case *0*
have *getspc-es* (*?esl1*!*0*) = *EvtSeq* (*getspc-e* (*?el1*!*0*)) (*EvtSys es*)
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def gets-es-def getx-es-def EvtSeqrm*
by (*smt fstI length-greater-0-conv list.distinct*(*2*) *nth-Cons-0 nth-map*)
then show *?case* **by** *simp*
next
case (*Suc j*)
assume *c0*: *j* < *length ?el1* ⟹ *getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *j*) = *AnonyEvent fin-com* ∨
getspc-es (*?esl1* ! *j*) =
EvtSeq (*getspc-e* (*?el1* ! *j*)) (*EvtSys es*)
and *c1*: *Suc j* < *length ?el1*
then have *c2*: *getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *j*) = *AnonyEvent fin-com* ∨
getspc-es (*?esl1* ! *j*) =
EvtSeq (*getspc-e* (*?el1* ! *j*)) (*EvtSys es*) **by** *simp*
show *?case*
proof(*cases getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *j*) = *AnonyEvent fin-com*)
assume *d0*: *getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *j*) = *AnonyEvent fin-com*
with *p1 p2-1 a1* **have** *d1*: *getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es*
proof −
from *p1 d0* **have** *getspc-es* (*esl* ! *Suc j*) = *EvtSys es* **by** *simp*
moreover
from *p1 c1* **have** *0* < *Suc j* ∧ *Suc* (*Suc j*) < *length esl*
using *a1* **by** *auto*

ultimately have *getspc-es* (*esl* ! *Suc* (*Suc j*)) = *EvtSys es*
using *p2-1* **by** *simp*
with *p1* **show** *?thesis* **by** *simp*
qed
with *a1 c1* **have** *d2*: *getspc-e* (*?el1* ! *Suc j*) = *AnonyEvent fin-com*
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSysrm **by** (*smt fst-conv nth-map*)
with *d1* **show** *?case* **by** *simp*
next
assume ¬(*getspc-es* (*?esl1* ! *j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *j*) = *AnonyEvent fin-com*)
with *c2* **have** *d0*: *getspc-es* (*?esl1* ! *j*) =
EvtSeq (*getspc-e* (*?el1* ! *j*)) (*EvtSys es*)
by *simp*
obtain *e* **and** *s1* **and** *x1* **where** *d1*: *?el1* ! *j* = (*e*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
with *d0* **have** *d2*: *?esl1* ! *j* = (*EvtSeq e* (*EvtSys es*),*s1*,*x1*)
proof −
have *e1*: *same-s-x ?esl1 ?el1* **using** *rm-evtsys-same-sx* **by** *blast*
from *d0 d1* **have** *getspc-es* (*?esl1* ! *j*) = *EvtSeq e* (*EvtSys es*)
by (*simp add:getspc-es-def getspc-e-def*)
moreover
from *e1* **have** *gets-e* (*?el1* ! *j*) = *gets-es* (*?esl1* ! *j*)
by (*simp add*: *Suc.prems less-or-eq-imp-le same-s-x-def*)
moreover
from *e1* **have** *getx-e* (*?el1* ! *j*) = *getx-es* (*?esl1* ! *j*)
by (*simp add*: *Suc.prems less-or-eq-imp-le same-s-x-def*)
ultimately show *?thesis*
using *d1 getspc-es-def gets-es-def getx-es-def gets-e-def getx-e-def*
by (*metis prod.collapse snd-conv*)
qed
then show *?case*
proof(*cases getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es*)
assume *e0*: *getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es*
then obtain *s2* **and** *x2* **where** *e1*: *?esl1* ! *Suc j* = (*EvtSys es*,
s2,*x2*)
using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
then have *e2*: *?el1* ! *Suc j* = (*AnonyEvent fin-com*, *s2*,*x2*)
using *getspc-es-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSysrm **by** (*metis Suc.prems a1 fst-conv*
nth-map snd-conv)
with *e1* **have** *getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es* ∧
getspc-e (*?el1* ! *Suc j*) = *AnonyEvent fin-com*
using *getspc-es-def getspc-e-def* **by** (*metis fst-conv*)
then show *?thesis* **by** *simp*
next
assume *e0*: *getspc-es* (*?esl1* ! *Suc j*) ≠ *EvtSys es*
with *a1 a2 c1 d2* **have** ∃ *e1*. *getspc-es* (*?esl1* ! *Suc j*) = *EvtSeq*
e1 (*EvtSys es*)

using *evtseq-next-in-cpts getspc-es-def* **by** *fastforce*
then obtain *e1* **where** *e1:getspc-es (?esl1 ! Suc j) = EvtSeq e1 (EvtSys es)* **by** *auto*
with *a1 c1* **have** *getspc-e (?el1 ! Suc j) = e1*
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def gets-es-def getx-es-def EvtSeqrm* **by** *(smt fstI nth-map)*
with *e1* **have** *getspc-es (?esl1 ! Suc j) =*
EvtSeq (getspc-e (?el1 ! Suc j)) (EvtSys es) **by** *simp*
then show *?thesis* **by** *simp*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p1 p2 p3 p4* **show** *?thesis* **by** *(metis (no-types, lifting) Suc-diff-1 Suc-less-SucD length-Cons nth-Cons-pos)*
qed

lemma *fstent-nomident-e-sim-es*:
$[\![$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
$\neg(\exists j.\ j > 0 \wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*)$]\!] \Longrightarrow$
$\exists$ *el e s x*. *el*$\in$*cpts-of-ev* Γ (*BasicEvent e*) *s x* $\wedge$ *e-sim-es esl el es e*
proof −
assume *p0*: *esl*$\in$*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p3*: $\neg(\exists j.\ j > 0 \wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es*
$\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*)
from *p1* **have** $\exists t.\ \Gamma \vdash$ (*EvtSys es*, *s*, *x*) −*es*−*t*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
apply(*induct esl*)
apply(*simp*)
by (*metis esys.distinct*(*1*) *exist-estran p0 p1*)
then obtain *t* **where** *a1*: $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) −*es*−*t*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *auto*
then have $\exists$ *evt e*. *evt* $\in$ *es* $\wedge$ *evt* = *BasicEvent e* $\wedge$ *Act t* = *EvtEnt* (*BasicEvent e*) $\wedge$
$\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) −*et*−*t*→ (*ev*, *s1*, *x1*) **using** *evtsysent-evtent0* **by** *fastforce*
then obtain *evt* **and** *e* **where** *a2*: *evt* $\in$ *es* $\wedge$ *evt* = *BasicEvent e* $\wedge$ *Act t* = *EvtEnt* (*BasicEvent e*) $\wedge$
$\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) −*et*−*t*→ (*ev*, *s1*, *x1*) **by** *auto*
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
let *?el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys ?esl1*
let *?el1* = *rm-evtsys ?esl1*
have *a5*: *?el* = (*BasicEvent e*, *s*, *x*) # *?el1* **by** *simp*
from *p1* **have** *a3*: *esl* = (*EvtSys es*, *s*, *x*) # *?esl1* **by** *simp*

from *a2* **obtain** *at* **and** *ak* **where** $\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) $-et-$(*at*$\sharp$*ak*)$\rightarrow$ (*ev*, *s1*, *x1*)
using *get-actk-def* **by** (*metis actk.cases*)
with *p0 p1 p3 a1 a2* **have** *a4*: *?el* $\in$ *cpts-ev* Γ
using *rm-evtsys-in-cptse* [*of esl* Γ *es s x ev s1 x1 xs*]
by (*metis estran.EvtOccur evtsysent-evtent0 noevtent-notran0*)
moreover have *e-sim-es esl ?el es e*
proof –
from *a3* **have** *b1*: *length esl* = *length ?el* **by** (*simp add*:*rm-evtsys-def*)
moreover
from *p1* **have** *b2*: *getspc-es* (*esl* ! *0*) = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
moreover
have *b3*: *getspc-e* (*?el* ! *0*) = *BasicEvent e* **by** (*simp add*:*getspc-e-def*)
moreover
from *a3 b1* **have** *b4*: $\forall$ *i*. *i* < *length ?el* $\longrightarrow$
gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) $\wedge$
getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
proof –
have *c1*: *same-s-x ?esl1* (*rm-evtsys ?esl1*) **using** *rm-evtsys-same-sx* **by** *auto*
show *?thesis*
proof –
{
fix *i*
have *i* < *length ?el* $\longrightarrow$
gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) $\wedge$
getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
proof(*cases i* = *0*)
assume *i* = *0*
with *p1* **show** *?thesis* **using** *gets-e-def getx-e-def gets-es-def getx-es-def* **by** (*metis nth-Cons-0 snd-conv*)
next
assume *i*$\neq$*0*
with *p1 p3 a3 c1* **show** *?thesis* **by** (*simp add*: *same-s-x-def*)
qed
}
then show *?thesis* **by** *auto*
qed
qed
moreover
have $\forall$ *i*. *i* > *0* $\wedge$ *i* < *length ?el* $\longrightarrow$
(*getspc-es* (*esl*!*i*) = *EvtSys es* $\wedge$ *getspc-e* (*?el*!*i*) = *AnonyEvent fin-com*)
$\vee$ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*)) (*EvtSys es*))
using *p0 p1 p3 a4* **by** (*meson fstent-nomident-e-sim-es-aux*)
ultimately show *?thesis* **by** (*simp add*:*e-sim-es-def*)
qed
ultimately show *?thesis* **using** *cpts-of-ev-def* **by** (*smt mem-Collect-eq nth-Cons'*)

qed

lemma *fstent-nomident-e-sim-es2*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*); *el*∈*cpts-ev* Γ⟧ ⟹
e-sim-es esl el es e
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es*
∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *p5*: *el*∈*cpts-ev* Γ
from *p2* **have** *a2*: Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*ev*, *s1*, *x1*)
using *evtsysent-evtent*[*of* Γ *es s x e k ev s1 x1*] **by** *auto*
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
let *?el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys ?esl1*
let *?el1* = *rm-evtsys ?esl1*
have *a5*: *?el* = (*BasicEvent e*, *s*, *x*) # *?el1* **by** *simp*
from *p1* **have** *a3*: *esl* = (*EvtSys es*, *s*, *x*) # *?esl1* **by** *simp*
from *p0 p1 p2 p3 p4 a2* **have** *a4*: *?el* ∈ *cpts-ev* Γ
using *rm-evtsys-in-cptse* **by** *metis*
show *?thesis*
proof −
from *a3* **have** *b1*: *length esl* = *length ?el* **by** (*simp add*:*rm-evtsys-def*)
moreover
from *p1* **have** *b2*: *getspc-es* (*esl* ! *0*) = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
moreover
have *b3*: *getspc-e* (*?el* ! *0*) = *BasicEvent e* **by** (*simp add*:*getspc-e-def*)
moreover
from *a3 b1* **have** *b4*: ∀*i*. *i* < *length ?el* ⟶
gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
proof −
have *c1*: *same-s-x ?esl1* (*rm-evtsys ?esl1*) **using** *rm-evtsys-same-sx* **by** *auto*
show *?thesis*
proof −
{
fix *i*

have *i* < *length ?el* ⟶
gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
proof(*cases i* = *0*)
assume *i* = *0*
with *p1* **show** *?thesis* **using** *gets-e-def getx-e-def gets-es-def getx-es-def* **by** (*metis nth-Cons-0 snd-conv*)
next
assume *i*≠*0*
with *p1 p3 a3 c1* **show** *?thesis* **by** (*simp add*: *same-s-x-def*)
qed
}
then show *?thesis* **by** *auto*
qed
qed
moreover
have ∀ *i*. *i* > *0* ∧ *i* < *length ?el* ⟶
(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*?el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*)) (*EvtSys es*))
using *p0 p1 p3 a4* **by** (*meson fstent-nomident-e-sim-es-aux*)
ultimately show *?thesis* **using** *e-sim-es-def* **using** *p4* **by** *blast*
qed

qed

lemma *e-sim-es-same-assume*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*⟶ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*);

e-sim-es esl el es e; *esl*∈*assume-es* Γ (*pre*,*rely*)⟧
⟹ *el*∈*assume-e* Γ (*pre*,*rely*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*⟶ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es*
∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *a1*: *e-sim-es esl el es e*
and *b0*: *esl*∈*assume-es* Γ (*pre*,*rely*)
from *p3* **have** *p3-1*: ∀ *j*. *j* > *0* ∧ *Suc j* < *length esl* ⟶ *getspc-es* (*esl* ! *j*) = *EvtSys es*

$\longrightarrow$ *getspc-es* (*esl* ! *Suc j*) = *EvtSys es* **using** *noevtent-inmid-eq* **by** *auto*

let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
let *?el1* = *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
from *p4* **have** *a2*: *el* = (*BasicEvent e*, *s*, *x*) # (*ev*,*s1*,*x1*) # *rm-evtsys xs*
by (*simp add*: *gets-es-def getspc-es-def getx-es-def rm-evtsys1-def rm-evtsys-def*)

from *p1 a2* **have** *a3*: *length esl* = *length el* **by** (*simp add*:*rm-evtsys-def*)

from *b0* **have** *b1*: *gets-es* (*esl*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length esl* $\longrightarrow$
Γ ⊢ *esl*!*i* −*ese*→ *esl*!(*Suc i*) $\longrightarrow$ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*)
by (*simp add*:*assume-es-def*)
then show *?thesis*
proof −
from *p1 p4 b1* **have** *gets-e* (*el*!*0*) ∈ *pre* **using** *gets-es-def gets-e-def*
by (*metis nth-Cons-0 snd-conv*)
moreover
have ∀ *i*. *Suc i*<*length el* $\longrightarrow$ Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*)
$\longrightarrow$ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely*
proof −
{
fix *i*
assume *c0*: *Suc i*<*length el*
and *c1*: Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*)
with *a2* **have** ¬(Γ ⊢ *el*!*0* −*ee*→ *el*!*1*)
by (*metis* (*no-types*, *lifting*) *One-nat-def eetran-eqconf evtsysent-evtent0*

no-tran2basic nth-Cons-0 nth-Cons-Suc p2)

with *c1* **have** *c2*: *i* ≠ *0* **by** (*metis One-nat-def*)
with *a1* **have** *c3*: (*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*))
using *e-sim-es-def Suc-lessD c0* **by** *blast*
from *c1* **have** *c4*: *getspc-e* (*el*!*i*) = *getspc-e* (*el*!*Suc i*)
by (*simp add*: *eetran-eqconf1*)
from *a1 c0 a3* **have** *c5*: *gets-es* (*esl*!*i*) = *gets-e* (*el*!*i*)
∧ *gets-es* (*esl*!*Suc i*) = *gets-e* (*el*!*Suc i*) **by** (*simp add*:*e-sim-es-def*)
from *a1 c0 a3* **have** *c6*:
(*getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*Suc i*)) (*EvtSys es*))
using *e-sim-es-def* **by** *blast*
have (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely*
proof(*cases getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent*

fin-com)
assume *d0*: *getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*
with *c2 p3-1 c0 a3* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSys es* **by** *auto*
with *d0* **have** Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)

with *b1 c0 a3* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** *auto*
then show *?thesis* **using** *c5* **by** *simp*
next
assume ¬(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
with *c3* **have** *d0*: *getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*)
by *simp*
let *?ei* = *getspc-e* (*el*!*i*)
show *?thesis*
proof(*cases ?ei* = *AnonyEvent fin-com*)
assume *e0*: *?ei* = *AnonyEvent fin-com*
with *c1* **have** *e1*: *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*
using *eetran-eqconf1* **by** *fastforce*
show *?thesis*
proof(*cases getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
assume *f0*: *getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*
with *d0* **have** *getspc-e* (*el*!*i*) ≠ *AnonyEvent fin-com*
proof −
let *?esl′* = *drop i esl*
from *p0* **have** *?esl′*∈*cpts-es* Γ
by (*metis Suc-lessD a3 c0 c2 cpts-es-dropi old.nat.exhaust*)
moreover
from *c0 a3* **have** *length ?esl′* > *1*
by *auto*
moreover
from *d0* **have** *getspc-es* (*?esl′*!*0*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*)
using *a3 c0* **by** *auto*
moreover
from *f0* **have** *getspc-es* (*?esl′*!*1*) = *EvtSys es*
using *a3 c0* **by** *fastforce*
ultimately show *?thesis* **using** *not-anonyevt-none-in-evtseq1* **by** *blast*
qed
with *e0* **show** *?thesis* **by** *simp*
next
assume ¬(*getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
with *c6* **have** *f0*: *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e*

(*el*!*Suc i*)) (*EvtSys es*)
by *simp*
with *c4* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*) **by** *simp*
with *d0* **have** *getspc-es* (*esl*!*Suc i*) = *getspc-es* (*esl*!*i*) **by** *simp*
then have Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)
with *b1* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*
by (*simp add*: *a3 c0*)
with *c5* **show** *?thesis* **by** *simp*
qed
next
assume *e0*: *?ei* ≠ *AnonyEvent fin-com*
with *c4 c6* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*Suc i*)) (*EvtSys es*)
by *simp*
with *c4 d0* **have** *getspc-es* (*esl*!*Suc i*) = *getspc-es* (*esl*!*i*) **by** *simp*
then have Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)

with *b1* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*
by (*simp add*: *a3 c0*)
with *c5* **show** *?thesis* **by** *simp*
qed
qed
}
then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add*:*assume-e-def*)
qed
qed

lemma *e-sim-es-same-commit*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*);

e-sim-es esl el es e; *el*∈*commit-e* Γ (*guar*,*post*)⟧
⟹ *esl*∈*commit-es* Γ (*guar*,*post*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es*
∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*),

s1,x1) # *xs*)
and *a1*: *e-sim-es esl el es e*
and *b3*: *el*∈*commit-e* Γ (*guar,post*)
from *p3* **have** *p3-1*: ∀ *j*. *j* > *0* ∧ *Suc j* < *length esl* ⟶ *getspc-es* (*esl* ! *j*) = *EvtSys es*
⟶ *getspc-es* (*esl* ! *Suc j*) = *EvtSys es* **using** *noevtent-inmid-eq* **by** *auto*
from *p0 p1 p2 p3 p4* **have** *a0*: *el* ∈ *cpts-ev* Γ **using** *rm-evtsys-in-cptse* **by** *metis*
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1,x1*) # *xs*
let *?el1* = *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1,x1*) # *xs*)
from *p4* **have** *a2*: *el* = (*BasicEvent e*, *s*, *x*) # (*ev,s1,x1*) # *rm-evtsys xs*
by (*simp add*: *gets-es-def getspc-es-def getx-es-def rm-evtsys1-def rm-evtsys-def*)

from *p1 a2* **have** *a3*: *length esl* = *length el* **by** (*simp add*:*rm-evtsys-def*)

from *b3* **have** *b4*: ∀ *i*. *Suc i*<*length el* ⟶
(∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*)) ⟶ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar*
by (*simp add*:*commit-e-def*)
then show *esl*∈*commit-es* Γ (*guar,post*)
proof −
have ∀ *i*. *Suc i*<*length esl* ⟶ (∃ *t*. Γ ⊢ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*))
⟶ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
proof −
{
fix *i*
assume *c0*: *Suc i*<*length esl*
and *c1*: ∃ *t*. Γ ⊢ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*)

have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
from *p2* **have** Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*ev*, *s1*, *x1*)
using *evtsysent-evtent* **by** *fastforce*
with *a2 b4* **have** (*s*, *s1*) ∈ *guar* **using** *gets-e-def*
by (*metis a3 c0 d0 fst-conv nth-Cons-0 nth-Cons-Suc snd-conv*)
with *p1* **show** *?thesis* **by** (*simp add*: *gets-es-def d0*)
next
assume *d0*: *i* ≠ *0*
then show *?thesis*
proof(*cases getspc-es* (*esl*!*i*) = *EvtSys es*)
assume *e0*: *getspc-es* (*esl*!*i*) = *EvtSys es*
with *p3-1 c0 d0* **have** *e1*: *getspc-es* (*esl*!*Suc i*) = *EvtSys es* **by** *simp*
from *c1* **obtain** *t* **where** Γ ⊢ *esl* ! *i* −*es*−*t*→ *esl* ! *Suc i* **by** *auto*
then have *getspc-es* (*esl*!*i*) ≠ *getspc-es* (*esl*!*Suc i*)
using *evtsys-not-eq-in-tran-aux1* **by** *blast*
with *e0 e1* **show** *?thesis* **by** *simp*

next
assume *e0*: *getspc-es* (*esl*!*i*) ≠ *EvtSys es*
from *p0 p1 c0* **have** *getspc-es* (*esl*!*i*) = *EvtSys es* ∨
(∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*))
using *evtsys-all-es-in-cpts getspc-es-def*
by (*metis Suc-lessD fst-conv length-Cons nth-Cons-0 zero-less-Suc*)

with *e0* **have** ∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*) **by** *simp*
then obtain *e* **where** *e1*: *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*) **by** *auto*
from *p0 p1 c0* **have** *e0-1*: *getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∨
(∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*))
using *evtsys-all-es-in-cpts getspc-es-def*
by (*metis fst-conv length-greater-0-conv list.distinct*(*1*) *nth-Cons-0*)

obtain *esi* **and** *si* **and** *xi* **and** *esi′* **and** *si′* **and** *xi′*
where *e2*: *esl*!*i* = (*esi*,*si*,*xi*) ∧ *esl*!(*Suc i*) = (*esi′*,*si′*,*xi′*)
by (*metis prod.collapse*)
with *c1* **obtain** *t* **where** *e3*: Γ ⊢ (*esi*,*si*,*xi*) −*es*−*t*→ (*esi′*,*si′*,*xi′*) **by** *auto*

from *e0-1* **show** *?thesis*
proof
assume *f0*: *getspc-es* (*esl*!*Suc i*) = *EvtSys es*
with *e1 e2 e3* **have** ∃ *t*. Γ ⊢ (*e*, *si*, *xi*) −*et*−*t*→ (*AnonyEvent fin-com*, *si′*,*xi′*)
by (*simp add*: *evtseq-tran-0-exist-etran getspc-es-def*)
then obtain *et* **where** *f1*: Γ ⊢ (*e*, *si*, *xi*) −*et*−*et*→ (*AnonyEvent fin-com*, *si′*,*xi′*)
by *auto*
from *p1 p4 a3 c0 d0 e1 e2* **have** *f2*:*el*!*i* = (*e*, *si*, *xi*)
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSeqrm
by (*smt Suc-lessD fst-conv list.simps*(*9*) *nth-Cons-Suc nth-map old.nat.exhaust snd-conv*)
moreover
from *p1 p4 a3 c0 d0 e2 f0* **have** *f3*:*el*!*Suc i* = (*AnonyEvent fin-com*, *si′*,*xi′*)
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSysrm
by (*smt List.nth-tl Suc-lessE diff-Suc-1 fst-conv*
length-tl list.sel(*3*) *nth-map snd-conv*)
ultimately have (*si*,*si′*)∈*guar* **using** *b4 f1 a3 c0 gets-e-def*
by (*metis fst-conv snd-conv*)

with *e2* **show** *?thesis* **by** (*simp add*:*gets-es-def*)
next
assume *f0*: ∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e* (*EvtSys es*)

then obtain e' **where** $f1$: *getspc-es* $(esl!Suc\ i) = EvtSeq\ e'$ $(EvtSys\ es)$
by *auto*
with $e1\ e2\ e3$ **have** $\exists t.\ \Gamma \vdash (e,\ si,\ xi)\ -et-t\rightarrow\ (e',\ si',\ xi')$
by (*simp add*: *evtseq-tran-exist-etran getspc-es-def*)
moreover
from $p1\ p4\ a3\ c0\ d0\ e1\ e2$ **have** $f2$:$el!i = (e,\ si,\ xi)$
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def gets-es-def getx-es-def EvtSeqrm*
by (*smt Suc-lessD fst-conv list.simps*(*9*) *nth-Cons-Suc nth-map old.nat.exhaust snd-conv*)
moreover
from $p1\ p4\ a3\ c0\ d0\ e2\ f1$ **have** $f3$:$el!Suc\ i = (e',\ si',xi')$
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def gets-es-def getx-es-def EvtSeqrm*
by (*smt Suc-lessD fst-conv less-Suc-eq-0-disj list.simps*(*9*) *nth-Cons-Suc nth-map snd-conv*)
ultimately have $(si,si')\in guar$ **using** $b4\ f1\ a3\ c0$ *gets-e-def*
by (*metis fst-conv snd-conv*)

with $e2$ **show** *?thesis* **by** (*simp add*:*gets-es-def*)
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*commit-es-def*)
qed
qed

lemma *rm-evtsys-assum-comm*:
$[\![esl\in cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$\Gamma \vdash (EvtSys\ es,\ s,\ x)\ -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1);$
$\neg(\exists j.\ j > 0 \land Suc\ j < length\ esl \land getspc\text{-}es\ (esl!j) = EvtSys\ es \land getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es);$
$el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs);$

$el\in assume\text{-}e\ \Gamma\ (pre,rely) \longrightarrow el\in commit\text{-}e\ \Gamma\ (guar,post)\ ]\!]$
$\Longrightarrow esl\in assume\text{-}es\ \Gamma\ (pre,rely) \longrightarrow esl\in commit\text{-}es\ \Gamma\ (guar,post)$
proof −
assume $p0$: $esl\in cpts\text{-}es\ \Gamma$
and $p1$: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and $p2$: $\Gamma \vdash (EvtSys\ es,\ s,\ x)\ -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)$
and $p3$: $\neg(\exists j.\ j > 0 \land Suc\ j < length\ esl \land getspc\text{-}es\ (esl!j) = EvtSys\ es$
$\land getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$

and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *p5*: *el*∈*assume-e* Γ (*pre*,*rely*) ⟶ *el*∈*commit-e* Γ (*guar*,*post*)
from *p3* **have** *p3-1*: ∀ *j*. *j* > *0* ∧ *Suc j* < *length esl* ⟶ *getspc-es* (*esl* ! *j*) = *EvtSys es*
⟶ *getspc-es* (*esl* ! *Suc j*) = *EvtSys es* **using** *noevtent-inmid-eq* **by** *auto*
from *p0 p1 p2 p3 p4* **have** *a0*: *el* ∈ *cpts-ev* Γ **using** *rm-evtsys-in-cptse* **by** *metis*
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
let *?el1* = *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
from *p0 p1 p2 p3 p4 a0* **have** *a1*: *e-sim-es esl el es e*
using *fstent-nomident-e-sim-es2* **by** *metis*
from *p4* **have** *a2*: *el* = (*BasicEvent e*, *s*, *x*) # (*ev*,*s1*,*x1*) # *rm-evtsys xs*
by (*simp add*: *gets-es-def getspc-es-def getx-es-def rm-evtsys1-def rm-evtsys-def*)

from *p1 a2* **have** *a3*: *length esl* = *length el* **by** (*simp add*:*rm-evtsys-def*)
show *?thesis*
proof
assume *b0*: *esl*∈*assume-es* Γ (*pre*,*rely*)
with *p0 p1 p2 p3 p4 a1* **have** *b2*: *el*∈*assume-e* Γ (*pre*,*rely*) **using** *e-sim-es-same-assume* **by** *metis*
with *p5* **have** *b3*: *el*∈*commit-e* Γ (*guar*,*post*) **by** *simp*
with *p0 p1 p2 p3 p4 a1* **show** *esl*∈*commit-es* Γ (*guar*,*post*) **using** *e-sim-es-same-commit* **by** *metis*
qed
qed

lemma *EventSys-sound-aux1*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es* Γ;*length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*;
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)⟧
⟹ ∃ *m*∈*es*. (*esl*∈*assume-es* Γ (*Pre m*,*Rely m*) ⟶ *esl*∈*commit-es* Γ (*Guar m*,*Post m*))
∧ (∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *a0*: *length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*
and *c41*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *c1*: *esl*∈*cpts-es* Γ

from *a0 c1* **have** *c2*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
by (*simp add*:*fst-esys-snd-eseq-exist*)
then obtain *s* **and** *x* **and** *ev* **and** *s1* **and** *x1* **and** *xs* **where** *c3*:
esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*

with *c1* **have** $\exists\, e\ k.\ \Gamma \vdash (\mathit{EvtSys}\ es,\ s,\ x)\ -es-(\mathit{EvtEnt}\ (\mathit{BasicEvent}\ e))\sharp k\rightarrow (\mathit{EvtSeq}\ ev\ (\mathit{EvtSys}\ es),\ s1,x1)$
using *fst-esys-snd-eseq-exist-evtent2* **by** *fastforce*
then obtain *e* **and** *k* **where** *c4*:
$\Gamma \vdash (\mathit{EvtSys}\ es,\ s,\ x)\ -es-(\mathit{EvtEnt}\ (\mathit{BasicEvent}\ e))\sharp k\rightarrow (\mathit{EvtSeq}\ ev\ (\mathit{EvtSys}\ es),\ s1,x1)$
by *auto*
let $?el = (\mathit{BasicEvent}\ e,\ s,\ x)\ \#\ \mathit{rm\text{-}evtsys}\ ((\mathit{EvtSeq}\ ev\ (\mathit{EvtSys}\ es),\ s1,x1)\ \#\ xs)$

from *c1 c3 c4 c41* **have** *c5*: $?el \in \mathit{cpts\text{-}ev}\ \Gamma$ **using** *rm-evtsys-in-cptse* **by** *metis*
from *c4* **have** $\exists\, ei{\in}es.\ ei = \mathit{BasicEvent}\ e$ **using** *evtsysent-evtent* **by** *metis*
then obtain *ei* **where** *c6*: $ei{\in}es \wedge ei = \mathit{BasicEvent}\ e$ **by** *auto*
from *c3 c4 c6* **have** *c61*: $\Gamma \vdash esl!0-es-(\mathit{EvtEnt}\ ei)\sharp k\rightarrow esl!1$ **by** *simp*
have *c8*: $?el{\in}\mathit{assume\text{-}e}\ \Gamma\ (\mathit{Pre}\ ei,\ \mathit{Rely}\ ei) \longrightarrow ?el{\in}\mathit{commit\text{-}e}\ \Gamma\ (\mathit{Guar}\ ei,\mathit{Post}\ ei)$
proof
assume *d0*: $?el{\in}\mathit{assume\text{-}e}\ \Gamma\ (\mathit{Pre}\ ei,\ \mathit{Rely}\ ei)$
moreover
from *p0 c6* **have** *d1*: $\Gamma \models ei\ sat_e\ [\mathit{Pre}\ ei,\ \mathit{Rely}\ ei,\ \mathit{Guar}\ ei,\ \mathit{Post}\ ei]$ **by** *auto*
moreover
from *c5* **have** $?el{\in}\mathit{cpts\text{-}of\text{-}ev}\ \Gamma\ (\mathit{BasicEvent}\ e)\ s\ x$ **by** (*simp add*:*cpts-of-ev-def*)
ultimately show $?el{\in}\mathit{commit\text{-}e}\ \Gamma\ (\mathit{Guar}\ ei,\mathit{Post}\ ei)$ **using** *evt-validity-def c6*
by *fastforce*
qed
with *c1 c3 c4 c41* **have** *c7*: $esl{\in}\mathit{assume\text{-}es}\ \Gamma\ (\mathit{Pre}\ ei,\ \mathit{Rely}\ ei) \longrightarrow esl{\in}\mathit{commit\text{-}es}\ \Gamma\ (\mathit{Guar}\ ei,\mathit{Post}\ ei)$
using *rm-evtsys-assum-comm* **by** *metis*
then show *?thesis* **using** *c6 c61* **by** *blast*
qed

lemma *EventSys-sound-aux1-forall*:
$[\![\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [\mathit{Pre}\ ef,\ \mathit{Rely}\ ef,\ \mathit{Guar}\ ef,\ \mathit{Post}\ ef];$
$esl{\in}\mathit{cpts\text{-}es}\ \Gamma;\mathit{length}\ esl \geq 2 \wedge \mathit{getspc\text{-}es}\ (esl!0) = \mathit{EvtSys}\ es \wedge \mathit{getspc\text{-}es}\ (esl!1) \neq \mathit{EvtSys}\ es;$
$\neg(\exists\, j.\ j > 0 \wedge \mathit{Suc}\ j < \mathit{length}\ esl \wedge \mathit{getspc\text{-}es}\ (esl!j) = \mathit{EvtSys}\ es \wedge \mathit{getspc\text{-}es}\ (esl!\mathit{Suc}\ j) \neq \mathit{EvtSys}\ es)]\!]$
$\Longrightarrow \forall\, m{\in}es.\ (\exists\, k.\ \Gamma \vdash esl!0-es-(\mathit{EvtEnt}\ m)\sharp k\rightarrow esl!1)$
$\longrightarrow (esl{\in}\mathit{assume\text{-}es}\ \Gamma\ (\mathit{Pre}\ m,\mathit{Rely}\ m) \longrightarrow esl{\in}\mathit{commit\text{-}es}\ \Gamma\ (\mathit{Guar}\ m,\mathit{Post}\ m))$
proof −
assume *p0*: $\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [\mathit{Pre}\ ef,\ \mathit{Rely}\ ef,\ \mathit{Guar}\ ef,\ \mathit{Post}\ ef]$
and *a0*: $\mathit{length}\ esl \geq 2 \wedge \mathit{getspc\text{-}es}\ (esl!0) = \mathit{EvtSys}\ es \wedge \mathit{getspc\text{-}es}\ (esl!1) \neq \mathit{EvtSys}\ es$
and *c41*: $\neg(\exists\, j.\ j > 0 \wedge \mathit{Suc}\ j < \mathit{length}\ esl \wedge \mathit{getspc\text{-}es}\ (esl!j) = \mathit{EvtSys}\ es \wedge \mathit{getspc\text{-}es}\ (esl!\mathit{Suc}\ j) \neq \mathit{EvtSys}\ es)$
and *c1*: $esl{\in}\mathit{cpts\text{-}es}\ \Gamma$

then show *?thesis*
proof –
{
fix *m*
assume *c01*: *m*∈*es*
and *c02*: ∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*
from *a0 c1* **have** *c2*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
by (*simp add:fst-esys-snd-eseq-exist*)
then obtain *s* **and** *x* **and** *ev* **and** *s1* **and** *x1* **and** *xs* **where** *c3*:
esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*
with *c02* **have** ∃ *k*. Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt m*)♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *simp*
then obtain *k* **where** *c4*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt m*)♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *auto*
then have ∃ *e*. *m* = *BasicEvent e* **by** (*meson evtent-is-basicevt*)
then obtain *e* **where** *c40*: *m* = *BasicEvent e* **by** *auto*
let *?el* = (*m*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
from *c1 c3 c4 c40 c41* **have** *c5*: *?el* ∈ *cpts-ev* Γ **using** *rm-evtsys-in-cptse* **by** *metis*

from *c3 c4 c40* **have** *c61*: Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1* **by** *simp*
have *c8*: *?el*∈*assume-e* Γ (*Pre m*, *Rely m*) ⟶ *?el*∈*commit-e* Γ (*Guar m*,*Post m*)
proof
assume *d0*: *?el*∈*assume-e* Γ (*Pre m*, *Rely m*)
moreover
from *p0 c01 c40* **have** *d1*: Γ ⊨ *m* sat_e [*Pre m*, *Rely m*, *Guar m*, *Post m*] **by** *auto*
moreover
from *c5 c40* **have** *?el*∈*cpts-of-ev* Γ (*BasicEvent e*) *s x* **by** (*simp add*:*cpts-of-ev-def*)
ultimately show *?el*∈*commit-e* Γ (*Guar m*,*Post m*) **using** *evt-validity-def c40*
by *fastforce*
qed
with *c1 c3 c4 c40 c41* **have** *c7*: *esl*∈*assume-es* Γ (*Pre m*, *Rely m*) ⟶ *esl*∈*commit-es* Γ (*Guar m*,*Post m*)
using *rm-evtsys-assum-comm* **by** *metis*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *EventSys-sound-seg-aux0-exist*:
⟦*esl*∈*cpts-es* Γ;*length esl* ≥ *2*; *getspc-es* (*esl*!*0*) = *EvtSys es*; *getspc-es* (*esl*!*1*) ≠ *EvtSys es*⟧
⟹ ∃ *m*∈*es*. (∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
proof –

assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *length esl* ≥ *2*
and *p2*: *getspc-es* (*esl*!*0*) = *EvtSys es*
and *p3*: *getspc-es* (*esl*!*1*) ≠ *EvtSys es*
then have *a1*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
by (*simp add*:*fst-esys-snd-eseq-exist*)
then obtain *s* **and** *x* **and** *ev* **and** *s1* **and** *x1* **and** *xs* **where** *a2*:
esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*
with *p0 a1* **have** ∃ *e k*. Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
using *fst-esys-snd-eseq-exist-evtent2* **by** *fastforce*
then obtain *e* **and** *k* **where** *a3*:
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
by *auto*
from *a3* **have** ∃ *i*∈*es*. *i* = *BasicEvent e* **using** *evtsysent-evtent* **by** *metis*
then obtain *ei* **where** *c6*: *ei*∈ *es* ∧ *ei* = *BasicEvent e* **by** *auto*
then show *?thesis* **using** *One-nat-def a2 a3 nth-Cons-0 nth-Cons-Suc* **by** *force*

qed

lemma *EventSys-sound-seg-aux0-forall*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es* Γ;*length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*;
getspc-es (*last esl*) = *EvtSys es*;
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)⟧
⟹ ∀ *ei*∈*es*. (∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt ei*)♯*k*→*esl*!*1*)
⟶ (*esl*∈*assume-es* Γ (*Pre ei*,*Rely ei*) ⟶ *esl*∈*commit-es* Γ (*Guar ei*,*Post ei*)
∧ *gets-es* (*last esl*) ∈ *Post ei*)
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *a0*: *length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*
and *p6*: *getspc-es* (*last esl*) = *EvtSys es*
and *c41*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *c1*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof−
{
fix *ei*
assume *c01*: *ei*∈*es*
and *c02*: ∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt ei*)♯*k*→*esl*!*1*

from *a0 c1* **have** *c2*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq*

ev (EvtSys es), s1,x1) # xs
by (simp add:fst-esys-snd-eseq-exist)
then obtain s **and** x **and** ev **and** s1 **and** x1 **and** xs **where** c3:
esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs **by** auto
with c02 **have** $\exists$ k. $\Gamma \vdash$ (EvtSys es, s, x) −es−(EvtEnt ei)♯k→ (EvtSeq ev (EvtSys es), s1,x1) **by** simp
then obtain k **where** c4: $\Gamma \vdash$ (EvtSys es, s, x) −es−(EvtEnt ei)♯k→ (EvtSeq ev (EvtSys es), s1,x1) **by** auto
then have $\exists$ e. ei = BasicEvent e **by** (meson evtent-is-basicevt)
then obtain e **where** c6: ei = BasicEvent e **by** auto
let ?el = (ei, s, x) # rm-evtsys ((EvtSeq ev (EvtSys es), s1,x1) # xs)
from c1 c3 c4 c6 c41 **have** c5: ?el $\in$ cpts-ev Γ **using** rm-evtsys-in-cptse **by** metis

from c3 c4 c6 **have** c61: $\Gamma \vdash$ esl!0−es−(EvtEnt ei)♯k→esl!1 **by** simp
have c8: ?el∈assume-e Γ (Pre ei, Rely ei) $\longrightarrow$?el∈commit-e Γ (Guar ei,Post ei)
proof
assume d0: ?el∈assume-e Γ (Pre ei, Rely ei)
moreover
from p0 c01 c6 **have** d1: $\Gamma \models$ ei sat_e [Pre ei, Rely ei, Guar ei, Post ei] **by** auto
moreover
from c5 c6 **have** ?el∈cpts-of-ev Γ (BasicEvent e) s x **by** (simp add:cpts-of-ev-def)
ultimately show ?el∈commit-e Γ (Guar ei,Post ei) **using** evt-validity-def c6
by fastforce
qed
with c1 c3 c4 c41 c6 **have** c7: esl∈assume-es Γ (Pre ei, Rely ei) $\longrightarrow$ esl∈commit-es Γ (Guar ei,Post ei)
using rm-evtsys-assum-comm **by** metis
moreover
have esl∈assume-es Γ (Pre ei, Rely ei) $\longrightarrow$ gets-es (last esl) $\in$ Post ei
proof
assume d0: esl∈assume-es Γ (Pre ei, Rely ei)
from c1 c3 c4 c41 c5 c6 **have** d2: e-sim-es esl ?el es e **using** fstent-nomident-e-sim-es2 **by** metis
with c1 c3 c4 c41 c5 c6 d0 **have** d3: ?el∈assume-e Γ (Pre ei, Rely ei)
using e-sim-es-same-assume **by** metis
with c8 **have** d1: ?el∈commit-e Γ (Guar ei,Post ei) **by** auto

have d4: getspc-e (last ?el) = AnonyEvent fin-com
proof −
from a0 d2 **have** e1: length ?el = length esl **by** (simp add: e-sim-es-def)

with d2 **have** $\forall$ i. i > 0 $\wedge$ i < length ?el $\longrightarrow$
(getspc-es (esl!i) = EvtSys es $\wedge$ getspc-e (?el!i) =

AnonyEvent fin-com)
$\vee$ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*))
(*EvtSys es*))
by (*simp add*: *e-sim-es-def*)
with *a0 e1* **have** (*getspc-es* (*last esl*) = *EvtSys es* $\wedge$ *getspc-e* (*last*
?el) = *AnonyEvent fin-com*)
$\vee$ (*getspc-es* (*last esl*) = *EvtSeq* (*getspc-e* (*last*
?el)) (*EvtSys es*))
by (*metis* (*no-types*, *lifting*) *c3 diff-less last-conv-nth length-greater-0-conv*
length-tl
list.sel(*3*) *list.simps*(*3*) *zero-less-one*)
with *p6* **show** *?thesis* **by** *simp*
qed
with *d1* **have** *gets-e* (*last ?el*) $\in$ *Post ei* **by** (*simp add*: *commit-e-def*)
moreover
from *a0 d2* **have** *gets-e* (*last ?el*) = *gets-es* (*last esl*) **using** *e-sim-es-def*
proof −
from *a0 d2* **have** *e1*: *length ?el* = *length esl* **by** (*simp add*: *e-sim-es-def*)
with *d2* **have** $\forall$ *i*. *i* < *length ?el* $\longrightarrow$ *gets-e* (*?el* ! *i*) = *gets-es* (*esl* !
i) $\wedge$
getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
by (*simp add*: *e-sim-es-def*)
with *a0 e1* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *c3 diff-less last-conv-nth length-greater-0-conv*
length-tl
list.sel(*3*) *list.simps*(*3*) *zero-less-one*)
qed
ultimately show *gets-es* (*last esl*) $\in$ *Post ei* **by** *simp*
qed

ultimately have (*esl*$\in$*assume-es* Γ (*Pre ei*,*Rely ei*) $\longrightarrow$ *esl*$\in$*commit-es* Γ
(*Guar ei*,*Post ei*)
$\wedge$ *gets-es* (*last esl*) $\in$ *Post ei*) **by** *simp*
}
then show *?thesis* **by** *auto*
qed
qed

lemma *EventSys-sound-seg-aux0*:
$[\![\forall$ *ef*$\in$*es*. $\Gamma \models$ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl$\in$*cpts-es* Γ;*length esl* $\geq$ *2* $\wedge$ *getspc-es* (*esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*1*)
$\neq$ *EvtSys es*;
getspc-es (*last esl*) = *EvtSys es*;
$\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es*
(*esl*!*Suc j*) $\neq$ *EvtSys es*)$]\!]$
$\Longrightarrow \exists$ *m*$\in$*es*. (*esl*$\in$*assume-es* Γ (*Pre m*,*Rely m*) $\longrightarrow$ *esl*$\in$*commit-es* Γ (*Guar*
m,*Post m*)
$\wedge$ *gets-es* (*last esl*) $\in$ *Post m*)
$\wedge$ ($\exists$ *k*. $\Gamma \vdash$ *esl*!*0*−*es*−(*EvtEnt m*)$\sharp$*k*$\rightarrow$*esl*!*1*)

proof −
assume *p0*: $\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *p1*: $length\ esl \geq 2 \wedge getspc\text{-}es\ (esl!0) = EvtSys\ es \wedge getspc\text{-}es\ (esl!1) \neq EvtSys\ es$
and *p2*: $getspc\text{-}es\ (last\ esl) = EvtSys\ es$
and *p3*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
and *p4*: $esl{\in}cpts\text{-}es\ \Gamma$
then have $\exists m{\in}es.\ (\exists k.\ \Gamma \vdash esl!0{-}es{-}(EvtEnt\ m)\sharp k{\rightarrow}esl!1)$
using *EventSys-sound-seg-aux0-exist*[*of esl* Γ *es*] **by** *simp*
then obtain *m* **where** *a1*: $m{\in}\ es \wedge (\exists k.\ \Gamma \vdash esl!0{-}es{-}(EvtEnt\ m)\sharp k{\rightarrow}esl!1)$ **by** *auto*
with *p0 p1 p2 p3 p4* **have** $(esl{\in}assume\text{-}es\ \Gamma\ (Pre\ m,Rely\ m) \longrightarrow esl{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
$\wedge\ gets\text{-}es\ (last\ esl) \in Post\ m)$
using *EventSys-sound-seg-aux0-forall* [*of es* Γ *Pre Rely Guar Post esl*] **by** *simp*
with *a1* **show** *?thesis* **by** *auto*
qed

lemma *EventSys-sound-aux-i-forall*:
$[\![\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall ef{\in}es.\ pre \subseteq Pre\ ef;\ \ \forall ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall ef{\in}es.\ Post\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$esl{\in}cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$esl{\in}assume\text{-}es\ \Gamma\ (pre,rely);$
$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])]\!]$
$\Longrightarrow \forall i.\ Suc\ i < length\ elst \longrightarrow$
$(\forall ei{\in}es.\ (\exists k.\ \Gamma \vdash (elst!i@[(elst!Suc\ i)!0])!0{-}es{-}(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei)$
proof −
assume *p0*: $\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *p1*: $\forall ef{\in}es.\ pre \subseteq Pre\ ef$
and *p2*: $\forall ef{\in}es.\ rely \subseteq Rely\ ef$
and *p3*: $\forall ef{\in}es.\ Guar\ ef \subseteq guar$
and *p4*: $\forall ef{\in}es.\ Post\ ef \subseteq post$
and *p5*[*rule-format*]: $\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and *p8*: $esl{\in}cpts\text{-}es\ \Gamma$
and *p9*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p10*: $esl{\in}assume\text{-}es\ \Gamma\ (pre,rely)$
and *p11*: $elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
from *p9 p8 p11* **have** *a0*[*rule-format*]: $\forall i.\ i < length\ elst \longrightarrow length\ (elst!i) \geq 2 \wedge$
$getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \neq EvtSys\ es$
using *parse-es-cpts-i2-start-aux* **by** *metis*

from *p9 p8 p11* **have** *a1*: $\forall i.\ i < length\ elst \longrightarrow$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ (elst!i) \wedge$
$getspc\text{-}es\ (elst!i!j) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!Suc\ j) \neq EvtSys\ es)$
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: *concat elst = esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
show *?thesis*
proof −
{
fix *i*
assume *b0*: *Suc i < length elst*
then have $\forall ei{\in}es.\ (\exists k.\ \Gamma \vdash (elst!i@[(elst!Suc\ i)!0])!0 -es-(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei$
proof(*induct i*)
case *0*
assume *c0*: *Suc 0 < length elst*
let *?els = elst ! 0 @ [elst ! Suc 0 ! 0]*
have *c1*: *?els* $\in$ *cpts-es* Γ
proof −
from *a0* **have** *c11*: $\forall i{<}length\ elst.\ elst\ !\ i \neq []$
using *list.size(3) not-numeral-le-zero* **by** *force*
with *a2 c0* **have** $\exists m\ n.\ m \le length\ esl \wedge n \le length\ esl \wedge m \le n \wedge ?els = take\ (n - m)\ (drop\ m\ esl)$
using *concat-i-lm* **by** *blast*
then obtain *m* **and** *n* **where** *d1*: $m \le length\ esl \wedge n \le length\ esl \wedge m \le n$
$\wedge\ ?els = take\ (n - m)\ (drop\ m\ esl)$ **by** *auto*
have *?els* $\neq$ [] **by** *simp*
with *p8 d1* **show** *?thesis* **by** (*simp add*: *cpts-es-seg2*)
qed

have *c2*: *getspc-es (last ?els) = EvtSys es* **by** (*simp add*: *a0 c0*)
have *c3*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ ?els \wedge getspc\text{-}es\ (?els!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (?els!Suc\ j) \neq EvtSys\ es)$
proof −
from *a0* **have** *getspc-es (elst ! Suc 0 ! 0) = EvtSys es* **using** *c0* **by** *blast*
with *a1* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD Suc-lessE c0 diff-Suc-1 diff-is-0-eq′ length-append-singleton nth-Cons-0 nth-append*)
qed
from *a0* **have** *c4*: $2 \le length\ ?els \wedge getspc\text{-}es\ (?els\ !\ 0) = EvtSys\ es \wedge getspc\text{-}es\ (?els\ !\ 1) \neq EvtSys\ es$

by (*metis* (*no-types*, *hide-lams*) *Suc-1 Suc-eq-plus1-left Suc-le-lessD Suc-lessD add.right-neutral c0 length-append-singleton not-less nth-append*)
with *p0 c1 c2 c3* **have** *c5*: $\forall$ *ei*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?els*!*0*−*es*−(*EvtEnt ei*)$\sharp$*k*$\rightarrow$*?els*!*1*)
$\longrightarrow$ (*?els*$\in$*assume-es* Γ (*Pre ei*,*Rely ei*) $\longrightarrow$ *?els*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)
$\wedge$ *gets-es* (*last ?els*) $\in$ *Post ei*)
using *EventSys-sound-seg-aux0-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *auto*

from *p10 a2* **have** *?els*$\in$*assume-es* Γ (*pre*,*rely*)
proof −
from *a0* **have** *d1*: $\forall$ *i*<*length elst*. *elst* ! *i* $\neq$ []
using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 c0* **have** $\exists$ *m n*. *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n* $\wedge$ *?els* = *take* (*n* − *m*) (*drop m esl*)
using *concat-i-lm* **by** *blast*
moreover
from *p10* **have** $\forall$ *i*. *Suc i*<*length esl* $\longrightarrow$ $\Gamma \vdash$ *esl*!*i* −*ese*$\rightarrow$ *esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have $\forall$ *i*. *Suc i*<*length ?els* $\longrightarrow$ $\Gamma \vdash$ *?els*!*i* −*ese*$\rightarrow$ *?els*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) $\in$ *rely*
using *rely-takedrop-rely* **by** *blast*
moreover
have *gets-es* (*?els*!*0*) $\in$ *pre*
proof −
from *a2* **have** *?els*!*0* = *esl*!*0*
by (*metis* (*no-types*, *lifting*) *Suc-lessD d1*
c0 concat.simps(*2*) *cpts-es-not-empty hd-append2*
length-greater-0-conv list.collapse nth-Cons-0 p8 snoc-eq-iff-butlast)
moreover
from *p10* **have** *gets-es* (*esl*!*0*) $\in$ *pre* **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

with *p1 p2 c5* **have** $\forall$ *ei*$\in$*es*. *?els*$\in$*assume-es* Γ (*Pre ei*, *Rely ei*) **using** *assume-es-imp*
by *metis*
with *c5* **show** *?case* **by** *auto*
next
case (*Suc j*)
let *?elstjj* = *elst* ! *j* @ [*elst* ! *Suc j* ! *0*]

let *?els = elst ! Suc j @ [elst ! Suc (Suc j) ! 0]*
assume *c01*: *Suc j < length elst*
$\Longrightarrow$ $\forall$ *ei*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?elstjj ! 0 −es−EvtEnt ei♯k→ ?elstjj ! 1*) $\longrightarrow$
?elstjj $\in$ *commit-es* Γ (*Guar ei*, *Post ei*) $\wedge$ *gets-es* (*elst ! Suc j ! 0*) $\in$ *Post ei*
and *c02*: *Suc* (*Suc j*) < *length elst*
then show *?case*
proof−
{
fix *ei*
assume *d0*: *ei*$\in$*es*
and *d1*: $\exists$ *k*. $\Gamma \vdash$ *?els ! 0 −es−EvtEnt ei♯k→ ?els ! 1*

from *c02 a0*[*of j*] **have** $\exists$ *m*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?elstjj!0−es−*(*EvtEnt m*)*♯k→?elstjj!1*)
using *EventSys-sound-seg-aux0-exist*[*of ?elstjj* Γ *es*] *p8 p9 p11*
by (*smt One-nat-def Suc-1 Suc-le-lessD Suc-lessD le-SucI length-append-singleton*
nth-append parse-es-cpts-i2-in-cptes-i)

then obtain *ei′* **where** *c03*: *ei′*$\in$*es* $\wedge$ ($\exists$ *k*. $\Gamma \vdash$ *?elstjj!0−es−*(*EvtEnt ei′*)*♯k→?elstjj!1*)
by *auto*
with *c01 c02* **have** *c04*: *?elstjj* $\in$ *commit-es* Γ (*Guar ei′*, *Post ei′*)
$\wedge$ *gets-es* (*elst ! Suc j ! 0*) $\in$ *Post ei′*
by *auto*

have *c1*: *?els* $\in$ *cpts-es* Γ
proof −
from *a0* **have** *c11*: $\forall$ *i*<*length elst*. *elst ! i* $\neq$ []
using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 c02* **have** $\exists$ *m n*. *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n* $\wedge$ *?els = take* (*n* − *m*) (*drop m esl*)
using *concat-i-lm* **by** *blast*
then obtain *m* **and** *n* **where** *d1*: *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n*
$\wedge$ *?els = take* (*n* − *m*) (*drop m esl*) **by** *auto*
have *?els* $\neq$ [] **by** *simp*
with *p8 d1* **show** *?thesis* **by** (*simp add*: *cpts-es-seg2*)
qed

have *c2*: *getspc-es* (*last ?els*) = *EvtSys es* **by** (*simp add*: *a0 c02*)
have *c3*: ¬($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length ?els* $\wedge$ *getspc-es* (*?els!j*) = *EvtSys es*
$\wedge$ *getspc-es* (*?els!Suc j*) $\neq$ *EvtSys es*)
proof −
from *a0* **have** *getspc-es* (*elst ! Suc* (*Suc j*) *! 0*) = *EvtSys es*

using *c02* **by** *blast*
with *a1* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD*
Suc-lessE c02 diff-Suc-1 diff-is-0-eq′ length-append-singleton nth-Cons-0 nth-append)
qed
from *a0* **have** *c4*: $2 \leq$ *length ?els* $\wedge$ *getspc-es* (*?els* ! *0*) = *EvtSys es* $\wedge$ *getspc-es* (*?els* ! *1*) $\neq$ *EvtSys es*
by (*metis* (*no-types*, *hide-lams*) *Suc-1 Suc-eq-plus1-left Suc-le-lessD*

Suc-lessD add.right-neutral c02 length-append-singleton not-less nth-append)

with *p0 c1 c2 c3 d0 d1* **have** *c5*: (*?els*$\in$*assume-es* Γ (*Pre ei*,*Rely ei*) $\longrightarrow$ *?els*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)
$\wedge$ *gets-es* (*last ?els*) $\in$ *Post ei*)
using *EventSys-sound-seg-aux0-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *blast*
from *p10 a2* **have** *?els*$\in$*assume-es* Γ (*Pre ei*,*rely*)
proof −
from *a0* **have** *d1*: $\forall$ *i*<*length elst*. *elst* ! *i* $\neq$ []
using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 c02* **have** $\exists$ *m n*. *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n* $\wedge$ *?els* = *take* (*n* − *m*) (*drop m esl*)
using *concat-i-lm* **by** *blast*
moreover
from *p10* **have** $\forall$ *i*. *Suc i*<*length esl* $\longrightarrow$ $\Gamma \vdash$ *esl*!*i* −*ese*→ *esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have $\forall$ *i*. *Suc i*<*length ?els* $\longrightarrow$ $\Gamma \vdash$ *?els*!*i* −*ese*→ *?els*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) $\in$ *rely*
using *rely-takedrop-rely* **by** *blast*
moreover
have *gets-es* (*?els*!*0*) $\in$ *Pre ei*
proof −
from *p5*[*of ei′ ei*] *d0 c03 c04* **have** *gets-es* (*elst* ! *Suc j* ! *0*) $\in$ *Pre ei*
by *blast*
then show *?thesis* **by** (*simp add*: *Suc-lessD c02 d1 nth-append*)
qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

with *p2* **have** *?els*$\in$*assume-es* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp*[*of Pre ei Pre ei rely Rely ei*]
d0 order-refl **by** *auto*

with *c5* **have** *c6*: *?els*∈*commit-es* Γ (*Guar ei*,*Post ei*) ∧ *gets-es* (*last ?els*) ∈ *Post ei* **by** *simp*
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *EventSys-sound-aux-i*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
∀ *ef*∈*es*. *pre* ⊆ *Pre ef*; ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*;
∀ *ef*∈*es*. *Guar ef* ⊆ *guar*; ∀ *ef*∈*es*. *Post ef* ⊆ *post*;
∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*;
esl∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es* Γ (*pre*,*rely*);
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧
⟹ ∀ *i*. *Suc i* < *length elst* ⟶
(∃ *m*∈*es*. *elst*!*i*@[(*elst*!*Suc i*)!*0*]∈*commit-es* Γ (*Guar m*,*Post m*)
∧ *gets-es* ((*elst*!*Suc i*)!*0*) ∈ *Post m*
∧ (∃ *k*. Γ ⊢ (*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*0*−*es*−(*EvtEnt m*)♯*k*→(*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*1*))
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *p1*: ∀ *ef*∈*es*. *pre* ⊆ *Pre ef*
and *p2*: ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*
and *p3*: ∀ *ef*∈*es*. *Guar ef* ⊆ *guar*
and *p4*: ∀ *ef*∈*es*. *Post ef* ⊆ *post*
and *p5*: ∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*
and *p8*: *esl*∈*cpts-es* Γ
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*∈*assume-es* Γ (*pre*,*rely*)
and *p11*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
from *p9 p8 p11* **have** *a0*[*rule-format*]: ∀ *i*. *i* < *length elst* ⟶ *length* (*elst*!*i*) ≥ *2* ∧
getspc-es (*elst*!*i*!*0*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*1*) ≠ *EvtSys es*
using *parse-es-cpts-i2-start-aux* **by** *metis*
from *p9 p8 p11* **have** *a1*: ∀ *i*. *i* < *length elst* ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*)
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
show *?thesis*
proof −

{
fix i
assume $b0$: $Suc\ i < length\ elst$
with $a0[of\ i]$ **have** $\exists\, m{\in}es.\ (\exists\, k.\ \Gamma \vdash elst!i!0 -es-(EvtEnt\ m)\sharp k\rightarrow elst!i!1)$
using $EventSys\text{-}sound\text{-}seg\text{-}aux0\text{-}exist[of\ elst!i@[(elst!Suc\ i)!0]\ \Gamma\ es]$
$parse\text{-}es\text{-}cpts\text{-}i2\text{-}in\text{-}cptes\text{-}i[of\ esl\ es\ s\ x\ e\ s1\ x1\ xs\ \Gamma\ elst]$
by (*smt Suc-1 Suc-le-lessD Suc-lessD le-SucI length-append-singleton length-greater-0-conv list.size(3) not-numeral-le-zero nth-append p11 p8 p9*)
then obtain m **where** $b1$: $m{\in}es \wedge (\exists\, k.\ \Gamma \vdash elst!i!0 -es-(EvtEnt\ m)\sharp k\rightarrow elst!i!1)$ **by** *auto*
with *p0 p1 p2 p3 p4 p5 p8 p9 p10 p11 b0*
have $b2[rule\text{-}format]$: $\forall\, i.\ Suc\ i < length\ elst \longrightarrow (\forall\, ei{\in}es.$
$(\exists\, k.\ \Gamma \vdash (elst\ !\ i\ @\ [elst\ !\ Suc\ i\ !\ 0])\ !\ 0\ -es-EvtEnt\ ei\sharp k\rightarrow (elst\ !\ i\ @\ [elst\ !\ Suc\ i\ !\ 0])\ !\ 1) \longrightarrow$
$elst\ !\ i\ @\ [elst\ !\ Suc\ i\ !\ 0] \in commit\text{-}es\ \Gamma\ (Guar\ ei,\ Post\ ei) \wedge gets\text{-}es\ (elst\ !\ Suc\ i\ !\ 0) \in Post\ ei)$
using $EventSys\text{-}sound\text{-}aux\text{-}i\text{-}forall[of\ es\ \Gamma\ Pre\ Rely\ Guar\ Post\ pre\ rely\ guar\ post\ esl\ s\ x\ e\ s1\ x1\ xs\ elst]$
by *fastforce*
from *b0 b1* $b2[of\ i\ m]$ **have** $elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ m$
by (*metis (no-types, lifting) Suc-1 Suc-le-lessD Suc-lessD a0 length-greater-0-conv*

list.size(3) not-numeral-le-zero nth-append)
with *b1* **have** $\exists\, m{\in}es.\ elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ m$
$\wedge\ (\exists\, k.\ \Gamma \vdash (elst!i@[(elst!Suc\ i)!0])!0 -es-(EvtEnt\ m)\sharp k\rightarrow (elst!i@[(elst!Suc\ i)!0])!1)$
by (*smt One-nat-def Suc-lessD a0 b0 lessI less-le-trans nth-append numeral-2-eq-2*)

}
then show *?thesis* **by** *auto*
qed
qed

lemma *EventSys-sound-aux-last-forall*:
$[\![\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall\, ef{\in}es.\ pre \subseteq Pre\ ef;\ \forall\, ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall\, ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall\, ef{\in}es.\ Post\ ef \subseteq post;$
$\forall\, ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$esl{\in}cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$esl{\in}assume\text{-}es\ \Gamma\ (pre,rely);$
$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])]\!]$
$\Longrightarrow \forall\, ei{\in}es.\ (\exists\, k.\ \Gamma \vdash (last\ elst)!0 -es-(EvtEnt\ ei)\sharp k\rightarrow (last\ elst)!1)$

$\longrightarrow$ *last elst*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)

proof –

assume *p0*: $\forall$ *ef*$\in$*es*. $\Gamma \models$ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]

and *p1*: $\forall$ *ef*$\in$*es*. *pre* $\subseteq$ *Pre ef*

and *p2*: $\forall$ *ef*$\in$*es*. *rely* $\subseteq$ *Rely ef*

and *p3*: $\forall$ *ef*$\in$*es*. *Guar ef* $\subseteq$ *guar*

and *p4*: $\forall$ *ef*$\in$*es*. *Post ef* $\subseteq$ *post*

and *p5*: $\forall$ *ef1 ef2*. *ef1*$\in$*es* $\wedge$ *ef2*$\in$*es* $\longrightarrow$ *Post ef1* $\subseteq$ *Pre ef2*

and *p8*: *esl*$\in$*cpts-es* Γ

and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*

and *p10*: *esl*$\in$*assume-es* Γ (*pre*,*rely*)

and *p11*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])

from *p9 p8 p11* **have** *a0*[*rule-format*]: $\forall$ *i*. *i* < *length elst* $\longrightarrow$ *length* (*elst*!*i*) $\geq$ *2* $\wedge$

getspc-es (*elst*!*i*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*1*) $\neq$ *EvtSys es*

using *parse-es-cpts-i2-start-aux* **by** *metis*

from *p9 p8 p11* **have** *a1*: $\forall$ *i*. *i* < *length elst* $\longrightarrow$

$\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length* (*elst*!*i*) $\wedge$

getspc-es (*elst*!*i*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*Suc j*) $\neq$ *EvtSys es*)

using *parse-es-cpts-i2-noent-mid* **by** *metis*

from *p9 p8 p11* **have** *a2*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*

with *p9* **have** *a3*: *elst* $\neq$ [] **by** *auto*

show *?thesis*

proof –

{

fix *ei*

assume *a01*: *ei*$\in$*es*

and *a02*: $\exists$ *k*. $\Gamma \vdash$ (*last elst*)!*0*–*es*–(*EvtEnt ei*)$\sharp$*k*$\rightarrow$(*last elst*)!*1*

have *last elst*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)

proof(*cases length elst* = *1*)

assume *b0*: *length elst* = *1*

from *a2 b0* **have** *b1*: *last elst* = *esl*

by (*metis* (*no-types*, *lifting*) *One-nat-def a3 append-butlast-last-id append-self-conv2 concat.simps*(*1*) *concat.simps*(*2*) *diff-Suc-1 length-0-conv length-butlast self-append-conv*)

let *?els* = *elst* ! *0*

from *p8 a2 b0* **have** *c1*: *?els* $\in$ *cpts-es* Γ **using** *b1 a3 last-conv-nth* **by** *fastforce*

from *a1 b0* **have** *c3*: $\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length ?els* $\wedge$ *getspc-es* (*?els*!*j*) = *EvtSys es*

$\wedge$ *getspc-es* (*?els*!*Suc j*) $\neq$ *EvtSys es*) **by** *simp*

from *a0 b0* **have** *c4*: *2* $\leq$ *length ?els* $\wedge$ *getspc-es* (*?els* ! *0*) = *EvtSys es* $\wedge$ *getspc-es* (*?els* ! *1*) $\neq$ *EvtSys es*

by *simp*

with *p0 c1 c3* **have** *c5*: $\forall$ *m*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?els*!*0*–*es*–(*EvtEnt m*)$\sharp$*k*$\rightarrow$*?els*!*1*)

$\longrightarrow$ (*?els*∈*assume-es* Γ (*Pre m*,*Rely m*) $\longrightarrow$ *?els*∈*commit-es* Γ (*Guar m*,*Post m*))
using *EventSys-sound-aux1-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** *?els*∈*assume-es* Γ (*pre*,*rely*)
proof −

from *a2 b0* **have** ∃ *m n*. *m* ≤ *length esl* ∧ *last elst* = (*drop m esl*)
using *concat-last-lm* **using** *b1* **by** *auto*
moreover
from *p10* **have** ∀ *i*. *Suc i*<*length esl* $\longrightarrow$ Γ ⊢ *esl*!*i* −*ese*→ *esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have ∀ *i*. *Suc i*<*length ?els* $\longrightarrow$ Γ ⊢ *?els*!*i* −*ese*→ *?els*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) ∈ *rely*
using *a3 b0 b1 last-conv-nth* **by** *force*
moreover
have *gets-es* (*?els*!*0*) ∈ *pre*
proof −
from *a2* **have** *?els*!*0* = *esl*!*0*
using *a3 b0 b1 last-conv-nth* **by** *fastforce*
moreover
from *p10* **have** *gets-es* (*esl*!*0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

with *p1 p2 a01* **have** *?els*∈*assume-es* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp*[*of pre Pre ei rely Rely ei elst ! 0*] **by** *simp*
with *a01 a02 c5* **have** *c6*: *?els*∈*commit-es* Γ (*Guar ei*,*Post ei*)
by (*simp add*: *a3 b0 last-conv-nth*)
with *c5* **show** *?thesis* **using** *a3 b0 last-conv-nth* **by** (*metis One-nat-def diff-Suc-1*)
next
assume *length elst* ≠ *1*
with *a3* **have** *b0*: *length elst* > *1* **by** (*simp add*: *Suc-lessI*)
let *?els* = *last elst*
from *p8 a2 b0* **have** *c1*: *?els* ∈ *cpts-es* Γ
proof −
from *a2 b0* **have** ∃ *m* . *m* ≤ *length esl* ∧ *?els* = *drop m esl*
by (*simp add*: *concat-last-lm a3*)

then obtain *m* **where** *d1*: *m* ≤ *length esl* ∧ *?els* = *drop m esl* **by** *auto*
with *a0* **have** *m* < *length esl*
by (*metis One-nat-def a3 diff-less drop-all last-conv-nth le-less-linear*

length-greater-0-conv list.size(*3*) *not-less-eq not-numeral-le-zero*)
with *p8 d1* **show** *?thesis* **using** *cpts-es-dropi*
by (*metis drop-0 le-0-eq le-SucE zero-induct*)
qed

from *a1 b0* **have** *c3*: $\neg(\exists j.\ j > 0 \wedge$ *Suc j* $<$ *length ?els* $\wedge$ *getspc-es* (*?els*!*j*) = *EvtSys es*
$\wedge$ *getspc-es* (*?els*!*Suc j*) $\neq$ *EvtSys es*)
by (*metis One-nat-def Suc-lessD a3 diff-less last-conv-nth zero-less-one*)
from *a0 b0* **have** *c4*: *2* $\leq$ *length ?els* $\wedge$ *getspc-es* (*?els* ! *0*) = *EvtSys es* $\wedge$ *getspc-es* (*?els* ! *1*) $\neq$ *EvtSys es*
by (*simp add*: *a3 last-conv-nth*)

with *p0 c1 c3* **have** *c5*: $\forall$ *m*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?els*!*0*$-$*es*$-$(*EvtEnt m*)$\sharp$*k*$\rightarrow$*?els*!*1*)

$\longrightarrow$ (*?els*$\in$*assume-es* Γ (*Pre m*,*Rely m*) $\longrightarrow$ *?els*$\in$*commit-es* Γ (*Guar m*,*Post m*))
using *EventSys-sound-aux1-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** *c6*: *?els*$\in$*assume-es* Γ (*Pre ei*,*rely*)
proof $-$
from *a2 b0* **have** $\exists$ *m* . *m* $\leq$ *length esl* $\wedge$ *?els* = *drop m esl*
by (*simp add*: *concat-last-lm a3*)
moreover
from *p10* **have** $\forall$ *i*. *Suc i*$<$*length esl* $\longrightarrow \Gamma \vdash$ *esl*!*i* $-$*ese*$\rightarrow$ *esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have $\forall$ *i*. *Suc i*$<$*length ?els* $\longrightarrow \Gamma \vdash$ *?els*!*i* $-$*ese*$\rightarrow$ *?els*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) $\in$ *rely*
using *a3 b0 last-conv-nth* **by** *force*
moreover
have *gets-es* (*?els*!*0*) $\in$ *Pre ei*
proof $-$
from *p0 p1 p2 p3 p4 p5 p8 p9 p10 p11*
have *c1*[*rule-format*]: $\forall$ *i*. *Suc i* $<$ *length elst* $\longrightarrow$
($\forall$ *ei*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ (*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*0*$-$*es*$-$(*EvtEnt ei*)$\sharp$*k*$\rightarrow$(*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*1*)
$\longrightarrow$ *elst*!*i*@[(*elst*!*Suc i*)!*0*]$\in$*commit-es* Γ (*Guar ei*,*Post ei*)
$\wedge$ *gets-es* ((*elst*!*Suc i*)!*0*) $\in$ *Post ei*)
using *EventSys-sound-aux-i-forall*[*of es* Γ *Pre Rely Guar Post pre rely guar*
post esl s x e s1 x1 xs elst] **by** *blast*
let *?els1* = *elst*!(*length elst* $-$ *2*)@[(*elst*!(*length elst* $-$ *1*))!*0*]
have *d1*: *?els1* $\in$ *cpts-es* Γ
proof $-$
from *a0* **have** *c11*: $\forall$ *i*$<$*length elst*. *elst* ! *i* $\neq$ []

using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 b0* **have** $\exists$ *m n*. *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n* $\wedge$ *?els1* = *take* (*n* − *m*) (*drop m esl*)
using *concat-i-lm*[*of elst esl length elst* − *2*]
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-diff-1*
Suc-diff-Suc a3 length-greater-0-conv lessI)
then obtain *m* **and** *n* **where** *d1*: *m* $\leq$ *length esl* $\wedge$ *n* $\leq$ *length esl* $\wedge$ *m* $\leq$ *n*
$\wedge$ *?els1* = *take* (*n* − *m*) (*drop m esl*) **by** *auto*
have *?els1* $\neq$ [] **by** *simp*
with *p8 d1* **show** *?thesis* **by** (*simp add*: *cpts-es-seg2*)
qed
moreover
have *length ?els1* > *2* **using** *a0*[*of length elst* − *2*]
by (*simp add*: *a3*)
moreover
have *getspc-es* (*?els1* ! *0*) = *EvtSys es* $\wedge$ *getspc-es* (*?els1* ! *1*) $\neq$ *EvtSys es*
using *a0*[*of length elst* − *2*] **by** (*metis* (*no-types*, *lifting*) *One-nat-def*

Suc-lessD Suc-less-SucD b0 calculation(*2*) *diff-less*
length-append-singleton nth-append numeral-2-eq-2 zero-less-numeral)

ultimately have $\exists$ *m*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?els1*!*0*−*es*−(*EvtEnt m*)$\sharp$*k*$\rightarrow$*?els1*!*1*)
using *EventSys-sound-seg-aux0-exist*[*of ?els1* Γ *es*] **by** *simp*
then obtain *m* **where** *d2*: *m*$\in$*es* $\wedge$ ($\exists$ *k*. $\Gamma \vdash$ *?els1*!*0*−*es*−(*EvtEnt m*)$\sharp$*k*$\rightarrow$*?els1*!*1*)
by *auto*
then have *gets-es* (*elst* ! (*length elst* − *1*) ! *0*) $\in$ *Post m*
using *c1*[*of length elst* − *2 m*] **by** (*metis* (*no-types*, *lifting*) *One-nat-def*
Suc-diff-Suc Suc-lessD b0 diff-less le-imp-less-Suc le-numeral-extra(*3*) *numeral-2-eq-2*)

then have *gets-es* (*last elst* ! *0*) $\in$ *Post m*
by (*simp add*: *a3 last-conv-nth*)
with *p5 a01 d2* **show** *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed
moreover
from *p1 p2* **have** *rely* $\subseteq$ *Rely ei* **by** (*simp add*: *a01*)
ultimately have *?els*$\in$*assume-es* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp* **by** *blast*
with *c5* **have** *c6*: *?els*$\in$*commit-es* Γ (*Guar ei*,*Post ei*) **using** *a01 a02* **by** *blast*

with *c5* **show** *?thesis* **using** *a3 b0 last-conv-nth* **by** *blast*
qed

}
then show *?thesis* **by** *auto* **qed**
qed

lemma *EventSys-sound-aux-last*:
$[\![\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall ef{\in}es.\ pre \subseteq Pre\ ef;\ \ \forall ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall ef{\in}es.\ Post\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$esl{\in}cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$esl{\in}assume\text{-}es\ \Gamma\ (pre,rely);$
$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])]\!]$
$\Longrightarrow \exists m{\in}es.\ last\ elst{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
$\wedge\ (\exists k.\ \Gamma \vdash (last\ elst)!0{-}es{-}(EvtEnt\ m)\sharp k{\rightarrow}(last\ elst)!1)$
proof −
assume *p0*: $\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *p1*: $\forall ef{\in}es.\ pre \subseteq Pre\ ef$
and *p2*: $\forall ef{\in}es.\ rely \subseteq Rely\ ef$
and *p3*: $\forall ef{\in}es.\ Guar\ ef \subseteq guar$
and *p4*: $\forall ef{\in}es.\ Post\ ef \subseteq post$
and *p5*: $\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and *p8*: $esl{\in}cpts\text{-}es\ \Gamma$
and *p9*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p10*: $esl{\in}assume\text{-}es\ \Gamma\ (pre,rely)$
and *p11*: $elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
from *p9 p8 p11* **have** *a0*[*rule-format*]: $\forall i.\ i < length\ elst \longrightarrow length\ (elst!i)$
$\geq 2\ \wedge$
$getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \neq EvtSys\ es$
using *parse-es-cpts-i2-start-aux* **by** *metis*
from *p9 p8 p11* **have** *a1*: $\forall i.\ i < length\ elst \longrightarrow$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ (elst!i)\ \wedge$
$getspc\text{-}es\ (elst!i!j) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!Suc\ j) \neq EvtSys$
$es)$
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: $concat\ elst = esl$ **using** *parse-es-cpts-i2-concat3* **by**
metis
with *p9* **have** *a3*: $elst \neq []$ **by** *auto*
from *p8 p9 p11 a0*[*of length elst − 1*] **have** $\exists m{\in}es.\ (\exists k.\ \Gamma \vdash last\ elst!0{-}es{-}(EvtEnt$
$m)\sharp k{\rightarrow}last\ elst!1)$
using *EventSys-sound-seg-aux0-exist*[*of last elst* Γ *es*]
parse-es-cpts-i2-in-cptes-last[*of esl es s x e s1 x1 xs* Γ *elst*]
by (*metis a3 diff-less last-conv-nth length-greater-0-conv less-one*)
then obtain *m* **where** *b1*: $m{\in}es \wedge (\exists k.\ \Gamma \vdash last\ elst!0{-}es{-}(EvtEnt\ m)\sharp k{\rightarrow}last$
$elst!1)$ **by** *auto*
with *p0 p1 p2 p3 p4 p5 p8 p9 p10 p11*
have $last\ elst{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
using *EventSys-sound-aux-last-forall*[*of es* Γ *Pre Rely Guar Post pre*
rely guar post esl s x e s1 x1 xs elst] **by** *blast*
with *b1* **show** *?thesis* **by** *auto*

qed

lemma *EventSys-sound-0*:
$[\![\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall ef{\in}es.\ pre \subseteq Pre\ ef;\ \ \forall ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall ef{\in}es.\ Post\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$stable\ pre\ rely;\ \forall s.\ (s,\ s){\in}guar;$
$esl{\in}cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$esl{\in}assume\text{-}es\ \Gamma\ (pre,rely)]\!]$
$\Longrightarrow \forall i.\ Suc\ i{<}length\ esl \longrightarrow (\exists t.\ \Gamma \vdash esl!i\ {-es-t\rightarrow}\ esl!(Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof −
assume $p0$: $\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and $p1$: $\forall ef{\in}es.\ pre \subseteq Pre\ ef$
and $p2$: $\forall ef{\in}es.\ rely \subseteq Rely\ ef$
and $p3$: $\forall ef{\in}es.\ Guar\ ef \subseteq guar$
and $p4$: $\forall ef{\in}es.\ Post\ ef \subseteq post$
and $p5$: $\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and $p6$: $stable\ pre\ rely$
and $p7$: $\forall s.\ (s,\ s){\in}guar$
and $p8$: $esl{\in}cpts\text{-}es\ \Gamma$
and $p9$: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and $p10$: $esl{\in}assume\text{-}es\ \Gamma\ (pre,rely)$
let $?elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
from *p9 p8* **have** $a0$: $concat\ ?elst = esl$ **using** *parse-es-cpts-i2-concat3* **by** *metis*

from *p9 p8* **have** $a1$: $\forall i.\ i < length\ ?elst \longrightarrow length\ (?elst!i) \geq 2\ \wedge$
$getspc\text{-}es\ (?elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (?elst!i!1) \neq EvtSys\ es$
using *parse-es-cpts-i2-start-aux* **by** *metis*

from *p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10*
have $\forall i.\ Suc\ i < length\ ?elst \longrightarrow$
$(\exists m{\in}es.\ ?elst!i@[(?elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
$\wedge\ gets\text{-}es\ ((?elst!Suc\ i)!0) \in Post\ m)$
using *EventSys-sound-aux-i*
[*of es* Γ *Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs ?elst*] **by** *blast*
then have $a2$: $\forall i.\ Suc\ i < length\ ?elst \longrightarrow$
$(\exists m{\in}es.\ ?elst!i@[(?elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m))$ **by** *auto*

from *p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10*
have $a3$: $\exists m{\in}es.\ last\ ?elst{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$
using *EventSys-sound-aux-last*
[*of es* Γ *Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs ?elst*] **by** *blast*
then obtain m **where** $a4$: $m{\in}es \wedge last\ ?elst{\in}commit\text{-}es\ \Gamma\ (Guar\ m,Post\ m)$

by *auto*
  show *?thesis*
   proof −
   {
    fix *i*
    assume *b0*: *Suc i* < *length esl*
     and *b1*: ∃ *t*. Γ ⊢ *esl* ! *i* −*es*−*t*→ *esl* ! *Suc i*
    from *p9* **have** *b01*: *esl* ≠ [] **by** *simp*
    moreover
    from *a1* **have** *b3*: ∀ *i*<*length ?elst*. *length* (*?elst*!*i*) ≥ *2* **by** *simp*
    ultimately have ∃ *k j*. *k* < *length ?elst* ∧ *j* ≤ *length* (*?elst*!*k*) ∧
         drop i esl = (*drop j* (*?elst*!*k*)) @ *concat* (*drop* (*Suc k*) *?elst*)
     using *concat-equiv* [*of esl ?elst*] *a0 b0* **by** *auto*
   then obtain *k* **and** *j* **where** *b2*: *k* < *length ?elst* ∧ *j* ≤ *length* (*?elst*!*k*) ∧
        drop i esl = (*drop j* (*?elst*!*k*)) @ *concat* (*drop* (*Suc k*) *?elst*) **by** *auto*
    have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
     proof(*cases k* = *length ?elst* − *1*)
      assume *c0*: *k* = *length ?elst* − *1*
      with *b2* **have** *c1*: *drop i esl* = *drop j* (*last ?elst*)
       by (*metis* (*no-types*, *lifting*) *Nitpick.size-list-simp*(*2*) *Suc-leI b01*
         a0 concat.simps(*1*) *drop-all last-conv-nth length-tl self-append-conv*)
      with *b0 b01* **have** *c2*: *drop j* (*last ?elst*) ≠ [] **by** *auto*
      with *b2 c0* **have** *c3*: *j* < *length* (*last ?elst*) **by** *auto*
      with *c1* **have** *c4*: *esl* ! *i* = (*last ?elst*) ! *j*
       by (*metis Suc-lessD b0 hd-drop-conv-nth*)
      from *c1 c3* **have** *c5*: *esl* ! *Suc i* = (*last ?elst*) ! *Suc j*
       by (*metis Cons-nth-drop-Suc Suc-lessD b0 list.sel*(*3*) *nth-via-drop*)
      from *a4* **have** ∀ *i*. *Suc i*<*length* (*last ?elst*) ⟶ (∃ *t*. Γ ⊢ (*last ?elst*)!*i*
−*es*−*t*→ (*last ?elst*)!(*Suc i*))
        ⟶ (*gets-es* ((*last ?elst*)!*i*), *gets-es* ((*last ?elst*)!*Suc i*)) ∈ *Guar m*
       by (*simp add*: *commit-es-def*)
      with *b1 c3 c4 c5* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) ∈ *Guar m*
      by (*metis Cons-nth-drop-Suc b0 c1 length-drop list.sel*(*3*) *zero-less-diff*)

      with *p3 a4* **show** *?thesis* **by** *auto*
     next
      assume *c00*: *k* ≠ *length ?elst* − *1*
      with *b2* **have** *c0*: *k* < *length ?elst* − *1* **by** *auto*
      show *?thesis*
       proof(*cases j* = *length* (*?elst*!*k*))
        assume *d0*: *j* = *length* (*?elst*!*k*)
        with *b2* **have** *d1*: *drop i esl* = *concat* (*drop* (*Suc k*) *?elst*) **by** *auto*
        from *b3 c0* **have** *d2*: *length* (*?elst* ! (*Suc k*)) ≥ *2* **by** *auto*
        from *c0* **have** *concat* (*drop* (*Suc k*) *?elst*) = *?elst* ! (*Suc k*) @ *concat*
(*drop* (*Suc* (*Suc k*)) *?elst*)
          by (*metis* (*no-types*, *hide-lams*) *Cons-nth-drop-Suc List.nth-tl*
concat.simps(*2*) *drop-Suc length-tl*)
        with *d1* **have** *d3*: *drop i esl* = *?elst* ! (*Suc k*) @ *concat* (*drop* (*Suc*
(*Suc k*)) *?elst*) **by** *simp*

with *b0 c0 d2* **have** *d4*: *esl* ! *i* = *?elst* ! (*Suc k*) ! *0*
by (*metis* (*no-types*, *hide-lams*) *Cons-nth-drop-Suc One-nat-def*
Suc-1
less-or-eq-imp-le not-less not-less-eq-eq nth-Cons-0 nth-append)

from *b0 c0 d2 d3* **have** *d5*: *esl* ! *Suc i* = *?elst* ! (*Suc k*) ! *1*
by (*metis* (*no-types*, *hide-lams*) *Cons-nth-drop-Suc One-nat-def*
Suc-1 Suc-le-lessD Suc-lessD nth-Cons-0 nth-Cons-Suc nth-append)

from *c0* **have** *Suc k* < *length ?elst* **by** *auto*
show *?thesis*
proof(*cases Suc k* = *length ?elst* − *1*)
assume *e0*: *Suc k* = *length ?elst* − *1*
with *d4* **have** *e1*: *esl* ! *i* = (*last ?elst*) ! *0*
by (*metis a0 b01 concat.simps*(*1*) *last-conv-nth*)
from *e0 d4* **have** *e2*: *esl* ! *Suc i* = (*last ?elst*) ! *1*
by (*metis a0 b01 concat.simps*(*1*) *d5 last-conv-nth*)
from *a4* **have** $\forall$ *i*. *Suc i*<*length* (*last ?elst*) ⟶ ($\exists$ *t*. Γ ⊢ (*last*
?elst)!*i* −*es*−*t*→ (*last ?elst*)!(*Suc i*))
⟶ (*gets-es* ((*last ?elst*)!*i*), *gets-es* ((*last ?elst*)!*Suc i*)) ∈
Guar m
by (*simp add*: *commit-es-def*)
with *b1 e1 e2* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) ∈ *Guar*
m
by (*metis One-nat-def Suc-1 Suc-le-lessD a0 b01 concat.simps*(*1*)
d2 e0 last-conv-nth)
with *p3 a4* **show** *?thesis* **by** *auto*
next
assume *Suc k* ≠ *length ?elst* − *1*
with *c0* **have** *e0*: *Suc k* < *length ?elst* − *1* **by** *auto*
let *?els'* = *?elst*!(*Suc k*)@[(*?elst*!*Suc* (*Suc k*))!*0*]
from *e0* **have** *Suc* (*Suc k*) < *length ?elst* **by** *auto*
with *a2* **have** $\exists$ *m*∈*es*. *?els'*∈*commit-es* Γ (*Guar m*,*Post m*)
by *blast*
then obtain *m* **where** *e1*: *m*∈*es* ∧ *?els'*∈*commit-es* Γ (*Guar*
m,*Post m*)
by *auto*
then have *e2*: $\forall$ *i*. *Suc i*<*length ?els'* ⟶ ($\exists$ *t*. Γ ⊢ *?els'*!*i* −*es*−*t*→
?els'!(*Suc i*))
⟶ (*gets-es* (*?els'*!*i*), *gets-es* (*?els'*!*Suc i*)) ∈ *Guar m*
by (*simp add*: *commit-es-def*)
from *d4* **have** *e3*: *esl* ! *i* = *?els'* ! *0*
by (*metis* (*no-types*, *lifting*) *Suc-le-eq d2 dual-order.strict-trans*
lessI nth-append numeral-2-eq-2)
from *d5* **have** *e4*: *esl* ! *Suc i* = *?els'* ! *1*
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-le-lessD d2 nth-append*)
from *b1 e3 e4* **have** *e5*: $\exists$ *t*. Γ ⊢ *?els'*!*0* −*es*−*t*→ *?els'*!*1* **by** *simp*
have *length ?els'* > *1* **using** *d2* **by** *auto*
with *e2 e5* **have** (*gets-es* (*?els'*!*0*), *gets-es* (*?els'*!*1*)) ∈ *Guar m*

by *simp*
with *e3 e4* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *Guar m* **by** *simp*
with *p3 e1* **show** *?thesis* **by** *auto*
qed
next
assume *d00*: $j \neq$ *length* (*?elst*!*k*)
with *b2* **have** *d0*: $j <$ *length* (*?elst*!*k*) **by** *auto*
with *b2* **have** *d1*: *esl* ! *i* = (*?elst*!*k*) ! *j*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-lessD append-Cons b0 list.inject*)
from *b0 b2 d0* **have** *d2*: *drop* (*Suc i*) *esl* = (*drop* (*Suc j*) (*?elst*!*k*)) @ *concat* (*drop* (*Suc k*) *?elst*)
by (*metis* (*no-types*, *lifting*) *d00 drop-Suc drop-eq-Nil le-antisym tl-append2 tl-drop*)
show *?thesis*
proof(*cases j* = *length* (*?elst*!*k*) − *1*)
assume *e0*: *j* = *length* (*?elst*!*k*) − *1*
let *?els'* = *?elst*!*k*@[(*?elst*!(*Suc k*))!*0*]
from *d1 d0* **have** *e1*: *esl* ! *i* = *last* (*?elst*!*k*)
by (*metis e0 gr-implies-not0 last-conv-nth length-0-conv*)

from *b2 e0* **have** *e2*: *drop* (*Suc i*) *esl* = *concat* (*drop* (*Suc k*) *?elst*)
by (*simp add*: *d2*)
with *c0* **have** *e3*: *drop* (*Suc i*) *esl* = *?elst*!*Suc k* @ *concat* (*drop* (*Suc* (*Suc k*)) *?elst*)
by (*metis Cons-nth-drop-Suc Suc-lessI c00 b2 concat.simps*(*2*) *diff-Suc-1*)
from *b3 c0* **have** *length* (*?elst* ! (*Suc k*)) $\geq$ *2* **by** *auto*
with *e3* **have** *e4*: *esl* ! *Suc i* = *?elst*!(*Suc k*)!*0*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-leD Suc-n-not-le-n b0 hd-append2 hd-conv-nth hd-drop-conv-nth list.size*(*3*))
with *e0* **have** *e5*: *esl* ! *Suc i* = *?els'* ! *Suc j*
by (*metis Suc-pred' d0 gr-implies-not0 linorder-neqE-nat nth-append-length*)
from *e0 e1* **have** *e6*: *esl* ! *i* = *?els'* ! *j*
by (*metis* (*no-types*, *lifting*) *d0 d1 nth-append*)

from *c0 a2* **have** $\exists$ *m*$\in$*es*. *?els'*$\in$*commit-es* Γ (*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e7*: *m*$\in$*es* $\wedge$
?els'$\in$*commit-es* Γ (*Guar m*,*Post m*)
by *auto*
then have *e8*: $\forall$ *i*. *Suc i*<*length ?els'* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *?els'*!*i* −*es*−*t*→ *?els'*!(*Suc i*))
$\longrightarrow$ (*gets-es* (*?els'*!*i*), *gets-es* (*?els'*!*Suc i*)) $\in$ *Guar m*
by (*simp add*: *commit-es-def*)

from *b1 e5 e6* **have** *e9*: $\exists\, t.\ \Gamma \vdash$ *?els'*!*j* $-es-t\rightarrow$ *?els'*!*Suc j* **by** *simp*
have *Suc j* $<$ *length ?els'* **using** *e0 d0* **by** *auto*
with *e8 e9* **have** (*gets-es* (*?els'*!*j*), *gets-es* (*?els'*!*Suc j*)) $\in$ *Guar m* **by** *simp*
with *e5 e6* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *Guar m* **by** *simp*
with *p3 e7* **show** *?thesis* **by** *auto*

next
assume *e0*: *j* $\neq$ *length* (*?elst*!*k*) $-$ *1*
with *d0* **have** *e00*: *j* $<$ *length* (*?elst*!*k*) $-$ *1* **by** *auto*
with *b0 d2* **have** *e1*: *esl* ! *Suc i* = (*?elst*!*k*) ! *Suc j*
by (*metis* (*no-types*, *lifting*) *List.nth-tl Suc-diff-Suc drop-Suc drop-eq-Nil hd-conv-nth hd-drop-conv-nth leD length-drop length-tl nth-append zero-less-Suc*)

let *?els'* = *?elst*!*k*@[(*?elst*!(*Suc k*))!*0*]
from *c0 a2* **have** $\exists$ *m*$\in$*es*. *?els'*$\in$*commit-es* Γ (*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e2*: *m*$\in$*es* $\wedge$ *?els'*$\in$*commit-es* Γ (*Guar m*,*Post m*)
by *auto*
then have *e3*: $\forall$ *i*. *Suc i*$<$*length ?els'* $\longrightarrow$ ($\exists\, t.\ \Gamma \vdash$ *?els'*!*i* $-es-t\rightarrow$ *?els'*!(*Suc i*))
$\longrightarrow$ (*gets-es* (*?els'*!*i*), *gets-es* (*?els'*!*Suc i*)) $\in$ *Guar m*
by (*simp add*: *commit-es-def*)
from *d1 e00* **have** *e4*: *esl* ! *i* = *?els'* ! *j*
by (*simp add*: *d0 nth-append*)
from *e1 e00* **have** *e5*: *esl* ! *Suc i* = *?els'* ! *Suc j*
by (*simp add*: *Suc-lessI nth-append*)
from *b1 e5 e4* **have** *e6*: $\exists\, t.\ \Gamma \vdash$ *?els'*!*j* $-es-t\rightarrow$ *?els'*!*Suc j* **by** *simp*
have *Suc j* $<$ *length ?els'* **using** *e00* **by** *auto*
with *e3 e4 e6* **have** (*gets-es* (*?els'*!*j*), *gets-es* (*?els'*!*Suc j*)) $\in$ *Guar m* **by** *simp*
with *e4 e5* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *Guar m* **by** *simp*
with *p3 e2* **show** *?thesis* **by** *auto*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed

qed

lemma *EventSys-sound* :
$[\![\forall ef \in es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall ef \in es.\ pre \subseteq Pre\ ef;\ \ \forall ef \in es.\ rely \subseteq Rely\ ef;$
$\forall ef \in es.\ Guar\ ef \subseteq guar;\ \forall ef \in es.\ Post\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1 \in es \wedge ef2 \in es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$stable\ pre\ rely;\ \forall s.\ (s,\ s) \in guar\ ]\!]$
$\Longrightarrow \Gamma \models EvtSys\ es\ sat_s\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: $\forall ef \in es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *p1*: $\forall ef \in es.\ pre \subseteq Pre\ ef$
and *p2*: $\forall ef \in es.\ rely \subseteq Rely\ ef$
and *p3*: $\forall ef \in es.\ Guar\ ef \subseteq guar$
and *p4*: $\forall ef \in es.\ Post\ ef \subseteq post$
and *p5*: $\forall ef1\ ef2.\ ef1 \in es \wedge ef2 \in es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and *p6*: *stable pre rely*
and *p7*: $\forall s.\ (s,\ s) \in guar$
then have $\forall s\ x.\ (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSys\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely) \subseteq$ $commit\text{-}es\ \Gamma\ (guar,\ post)$
proof−
{
fix *s x*
have $\forall esl.\ esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSys\ es)\ s\ x) \cap assume\text{-}es\ \ \Gamma\ (pre,\ rely)$ $\longrightarrow esl \in\ commit\text{-}es\ \Gamma\ (guar,\ post)$
proof −
{
fix *esl*
assume *a0*: $esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSys\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely)$
then have *a1*: $esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSys\ es)\ s\ x)$ **by** *simp*
then have *a1-1*: $esl!0 = (EvtSys\ es,\ s,\ x)$ **by** (*simp add*:*cpts-of-es-def*)
from *a1* **have** *a1-2*: $esl \in cpts\text{-}es\ \Gamma$ **by** (*simp add*:*cpts-of-es-def*)
from *a0* **have** *a2*: $esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$ **by** *simp*
then have $\forall i.\ Suc\ i < length\ esl \longrightarrow (\exists t.\ \Gamma \vdash esl!i\ -es-t\rightarrow\ esl!(Suc\ i))$ $\longrightarrow$
$(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof −
{
fix *i*
assume *b0*: $Suc\ i < length\ esl$
and *b1*: $\exists t.\ \Gamma \vdash esl!i\ -es-t\rightarrow\ esl!(Suc\ i)$
then obtain *t* **where** *b2*: $\Gamma \vdash esl!i\ -es-t\rightarrow\ esl!(Suc\ i)$ **by** *auto*
from *a1-2 b0 b1* **have** $(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof(*cases* $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) = EvtSys$ es)
assume *c0*: $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) = EvtSys$ es

with *b0* **have** $getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$ **by** *simp*
moreover from *b0 c0* **have** $getspc\text{-}es\ (esl\ !\ (Suc\ i)) = EvtSys\ es$ **by** *simp*

ultimately have $\neg(\exists\, t.\ \Gamma \vdash$ *esl*!*i* $-es-t\rightarrow$ *esl*!(*Suc i*))
using *evtsys-not-eq-in-tran2 getspc-es-def* **by** (*metis surjective-pairing*)

with *b1* **show** *?thesis* **by** *simp*
next
assume *c0*: $\neg$ ($\forall\, i.$ *Suc i* $\leq$ *length esl* $\longrightarrow$ *getspc-es* (*esl* ! *i*) = *EvtSys es*)
then obtain *m* **where** *c1*: *Suc m* $\leq$ *length esl* $\wedge$ *getspc-es* (*esl* ! *m*) $\neq$ *EvtSys es*
by *auto*
from *a1-1* **have** *c2*: *getspc-es* (*esl*!*0*) = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
from *c1* **have** $\exists\, i.$ *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) $\neq$ *EvtSys es* **by** *auto*
with *a1-2 a1-1 c1 c2* **have** $\exists\, i.$ (*i* < *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *EvtSys es*
$\wedge$ *getspc-es* (*esl* ! *Suc i*) $\neq$ *EvtSys es*)
$\wedge$ ($\forall\, j.$ *j* < *i* $\longrightarrow$ *getspc-es* (*esl* ! *j*) = *EvtSys es*)
using *evtsys-fst-ent* **by** *blast*
then obtain *n* **where** *c3*: (*n* < *m* $\wedge$ *getspc-es* (*esl* ! *n*) = *EvtSys es*
$\wedge$ *getspc-es* (*esl* ! *Suc n*) $\neq$ *EvtSys es*)
$\wedge$ ($\forall\, j.$ *j* < *n* $\longrightarrow$ *getspc-es* (*esl* ! *j*) = *EvtSys es*) **by** *auto*
with *b1* **have** *c4*: *i* $\geq$ *n*
proof –
{
assume *d0*: *i* < *n*
with *c3* **have** *getspc-es* (*esl* ! *i*) = *EvtSys es* **by** *simp*
moreover from *c3 d0* **have** *getspc-es* (*esl* ! *Suc i*) = *EvtSys es*
using *Suc-lessI* **by** *blast*
ultimately have $\neg(\exists\, t.\ \Gamma \vdash$ *esl*!*i* $-es-t\rightarrow$ *esl*!*Suc i*)
using *evtsys-not-eq-in-tran getspc-es-def* **by** (*metis surjective-pairing*)
with *b1* **have** *False* **by** *simp*
}
then show *?thesis* **using** *leI* **by** *auto*
qed

let *?esl* = *drop n esl*
from *c1 c3* **have** *c5*: *length ?esl* $\geq$ *2*
by (*metis One-nat-def Suc-eq-plus1-left Suc-le-eq length-drop less-diff-conv less-trans-Suc numeral-2-eq-2*)
from *c1 c3* **have** *c6*: *getspc-es* (*?esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*?esl*!*1*) $\neq$ *EvtSys es*
by *force*

from *a1-2 c1 c3* **have** *c7*: *?esl* $\in$ *cpts-es* Γ **using** *cpts-es-dropi*
by (*metis* (*no-types*, *lifting*) *b0 c4 drop-0 dual-order.strict-trans*

le-0-eq le-SucE le-imp-less-Suc zero-induct)
from *c5 c6 c7* **have** $\exists$ *s x ev s1 x1 xs. ?esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
using *fst-esys-snd-eseq-exist* **by** *blast*
then obtain *s* **and** *x* **and** *e* **and** *s1* **and** *x1* **and** *xs* **where** *c8*:
?esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
by *auto*

let *?elst* = *tl* (*parse-es-cpts-i2 ?esl es* [[]])
from *c8 c7* **have** *c9*: *concat ?elst* = *?esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
have *c10*: *?esl*$\in$*assume-es* Γ (*pre*,*rely*)
proof(*cases n* = *0*)
assume *d0*: *n* = *0*
then have *?esl* = *esl* **by** *simp*
with *a2* **show** *?thesis* **by** *simp*
next
assume *d0*: *n* $\neq$ *0*
let *?eslh* = *take* (*n* + *1*) *esl*
from *a2* **have** *d1*: $\forall$ *i. Suc i*<*length ?esl* $\longrightarrow$ $\Gamma \vdash$ *?esl*!*i* $-ese\rightarrow$ *?esl*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
have *gets-es* (*?esl*!*0*) $\in$ *pre*
proof $-$
from *a2 d0* **have** *gets-es* (*?eslh*!*0*) $\in$ *pre* **by** (*simp add*:*assume-es-def*)
moreover
from *a2* **have** $\forall$ *i. Suc i*<*length ?eslh* $\longrightarrow$ $\Gamma \vdash$ *?eslh*!*i* $-ese\rightarrow$ *?eslh*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*?eslh*!*i*), *gets-es* (*?eslh*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have *?eslh* $\in$ *assume-es* Γ (*pre*, *rely*) **by** (*simp add*:*assume-es-def*)
moreover
from *c3* **have** $\forall$ *i*<*length ?eslh. getspc-es* (*?eslh*!*i*) = *EvtSys es*
by (*metis Suc-eq-plus1 length-take less-antisym min-less-iff-conj nth-take*)
ultimately have $\forall$ *i*<*length ?eslh. gets-es* (*?eslh*!*i*) $\in$ *pre*
using *p6 pre-trans* **by** *blast*
with *d0* **have** *gets-es* (*?eslh* ! *n*) $\in$ *pre*
using *b0 c4* **by** *auto*
then show *?thesis* **by** (*simp add*: *c8 nth-via-drop*)
qed
with *d1* **show** *?thesis* **by** (*simp add*:*assume-es-def*)
qed

from *p0 p1 p2 p3 p4 p5 p6 p7 c7 c8 c10*

have *c11*: $\forall i.$ *Suc i<length ?esl* $\longrightarrow$ ($\exists t.$ $\Gamma \vdash$ *?esl!i* $-es-t\rightarrow$ *?esl!(Suc i)*) $\longrightarrow$
(*gets-es* (*?esl!i*), *gets-es* (*?esl!Suc i*)) $\in$ *guar*
using *EventSys-sound-0*
[*of es* Γ *Pre Rely Guar Post pre rely guar post ?esl s x e s1 x1 xs*] **by** *simp*

from *b0 c4* **have** *c12*: *esl* ! *i* = *?esl* ! (*i* − *n*) **by** *auto*
moreover
from *b0 c4* **have** *c13*: *esl* ! *Suc i* = *?esl* ! *Suc* (*i* − *n*) **by** *auto*
moreover
from *b0 c4* **have** *Suc* (*i* − *n*) < *length ?esl* **by** *auto*
moreover
from *b1 c12 c13* **have** $\exists t.$ $\Gamma \vdash$ *?esl* ! (*i* − *n*) $-es-t\rightarrow$ *?esl* ! *Suc* (*i* − *n*) **by** *simp*
ultimately
have (*gets-es* (*?esl* ! (*i* − *n*)), *gets-es* (*?esl* ! *Suc* (*i* − *n*))) $\in$ *guar*
using *c11* **by** *simp*

with *c12 c13* **show** *?thesis* **by** *simp*

qed

}
then show *?thesis* **by** *auto*
qed
then have *esl*$\in$*commit-es* Γ (*guar*, *post*) **by** (*simp add*:*commit-es-def*)
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *blast*
qed

then show $\Gamma \models$ *EvtSys es sat*$_s$ [*pre*, *rely*, *guar*, *post*] **by** (*simp add*:*es-validity-def*)
qed

lemma *esys-seq-sound*:
$[\![$*pre* $\subseteq$ *pre'*; *rely* $\subseteq$ *rely'*; *guar'* $\subseteq$ *guar*; *post'* $\subseteq$ *post*;
$\Gamma \models$ *esys sat*$_s$ [*pre'*, *rely'*, *guar'*, *post'*]$]\!]$
$\Longrightarrow \Gamma \models$ *esys sat*$_s$ [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: *pre* $\subseteq$ *pre'*
and *p1*: *rely* $\subseteq$ *rely'*
and *p2*: *guar'* $\subseteq$ *guar*
and *p3*: *post'* $\subseteq$ *post*
and *p4*: $\Gamma \models$ *esys sat*$_s$ [*pre'*, *rely'*, *guar'*, *post'*]
from *p4* **have** *p5*: $\forall s\ x.$ (*cpts-of-es* Γ *esys s x*) $\cap$ *assume-es* Γ (*pre'*, *rely'*) $\subseteq$ *commit-es* Γ (*guar'*, *post'*)

by (*simp add*: *es-validity-def*)
have $\forall s\ x.$ (*cpts-of-es* Γ *esys s x*) $\cap$ *assume-es* Γ (*pre*, *rely*) $\subseteq$ *commit-es* Γ (*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: $c\in$(*cpts-of-es* Γ *esys s x*) $\cap$ *assume-es* Γ (*pre*, *rely*)
then have $c\in$(*cpts-of-es* Γ *esys s x*) $\wedge$ $c\in$*assume-es* Γ (*pre*, *rely*) **by** *simp*
with *p0 p1* **have** $c\in$(*cpts-of-es* Γ *esys s x*) $\wedge$ $c\in$*assume-es* Γ (*pre′*, *rely′*)
using *assume-es-imp*[*of pre pre′ rely rely′ c*] **by** *simp*
with *p5* **have** $c\in$*commit-es* Γ (*guar′*, *post′*) **by** *auto*
with *p2 p3* **have** $c\in$*commit-es* Γ (*guar*, *post*)
using *commit-es-imp*[*of guar′ guar post′ post c*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*es-validity-def*)
qed

lemma *EventSys-sound′*:
assumes *p0*: $\forall ef\in esf.\ \Gamma \vdash E_e\ ef\ sat_e\ [Pre_e\ ef,\ Rely_e\ ef,\ Guar_e\ ef,\ Post_e\ ef]$
and *p1*: $\forall ef\in esf.\ pre \subseteq Pre_e\ ef$
and *p2*: $\forall ef\in esf.\ rely \subseteq Rely_e\ ef$
and *p3*: $\forall ef\in esf.\ Guar_e\ ef \subseteq guar$
and *p4*: $\forall ef\in esf.\ Post_e\ ef \subseteq post$
and *p5*: $\forall ef1\ ef2.\ ef1\in esf \wedge ef2\in esf \longrightarrow Post_e\ ef1 \subseteq Pre_e\ ef2$
and *p6*: *stable pre rely*
and *p7*: $\forall s.\ (s, s) \in guar$
shows $\Gamma \models$ *evtsys-spec* (*rgf-EvtSys esf*) sat_s [*pre*, *rely*, *guar*, *post*]
proof −
let *?es* = *Domain esf*
let *?RG* = $\lambda e.$ *SOME rg*. (*e*,*rg*)$\in$*esf*
have *a1*: $\forall e\in ?es.\ \exists ef\in esf.\ ?RG\ e = snd\ ef$ **by** (*metis Domain.cases snd-conv someI*)

let *?Pre* = *pre-rgf* $\circ$ *?RG*
let *?Rely* = *rely-rgf* $\circ$ *?RG*
let *?Guar* = *guar-rgf* $\circ$ *?RG*
let *?Post* = *post-rgf* $\circ$ *?RG*
from *p0* **have** *a2*: $\forall i\in esf.\ \Gamma \models E_e\ i\ sat_e\ [Pre_e\ i,\ Rely_e\ i,\ Guar_e\ i,\ Post_e\ i]$
by (*simp add*: *rgsound-e*)
have $\forall ef\in ?es.\ \Gamma \models ef\ sat_e\ [?Pre\ ef,\ ?Rely\ ef,\ ?Guar\ ef,\ ?Post\ ef]$
by (*metis* (*mono-tags*, *lifting*) *Domain.cases* E_e*-def* $Guar_e$*-def* $Post_e$*-def* Pre_e*-def* $Rely_e$*-def a2 comp-apply fst-conv snd-conv someI-ex*)
moreover
have $\forall ef\in ?es.\ pre \subseteq ?Pre\ ef$ **by** (*metis* Pre_e*-def a1 comp-def p1*)
moreover
have $\forall ef\in ?es.\ rely \subseteq ?Rely\ ef$ **by** (*metis* $Rely_e$*-def a1 comp-apply p2*)
moreover

have $\forall$ *ef*$\in$*?es*. *?Guar ef* $\subseteq$ *guar* **by** (*metis Guar*$_e$*-def a1 comp-apply p3*)
moreover
have $\forall$ *ef*$\in$*?es*. *?Post ef* $\subseteq$ *post* **by** (*metis Post*$_e$*-def a1 comp-apply p4*)
moreover
have $\forall$ *ef1 ef2*. *ef1* $\in$ *?es* $\wedge$ *ef2* $\in$ *?es* $\longrightarrow$ *?Post ef1* $\subseteq$ *?Pre ef2*
by (*metis* (*mono-tags*, *lifting*) *Post*$_e$*-def Pre*$_e$*-def a1 comp-def p5*)
ultimately have $\Gamma \models$ *EvtSys* (*Domain esf*) *sat*$_s$ [*pre*, *rely*, *guar*, *post*]
using *p6 p7 EventSys-sound* [*of ?es* Γ *?Pre ?Rely ?Guar ?Post pre rely guar post*] **by** *simp*
then show $\Gamma \models$ *evtsys-spec* (*rgf-EvtSys esf*) *sat*$_s$ [*pre*, *rely*, *guar*, *post*] **by** *simp*
qed

theorem *rgsound-es*: $\Gamma \vdash$ (*esf*::(*'l*,*'k*,*'s*,*'prog*) *rgformula-ess*) *sat*$_s$ [*pre*, *rely*, *guar*, *post*]
$\Longrightarrow \Gamma \models$ *evtsys-spec esf sat*$_s$ [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-es.induct*)
apply *auto*[*1*]
using *EventSeq-sound rgsound-e* **apply** *smt*
using *EventSys-sound'* **apply** *blast*
using *esys-seq-sound* **apply** *blast*
done

7.5 Soundness of Parallel Event Systems

lemma *conjoin-comm-imp-rely-n*[*rule-format*]:
$[\![\forall$ *k*. *pre* $\subseteq$ *Pre k*; $\forall$ *k*. *rely* $\subseteq$ *Rely k*;
$\forall$ *k j*. *j*$\neq$*k* $\longrightarrow$ *Guar j* $\subseteq$ *Rely k*;
$\forall$ *k*. *cs k* $\in$ *commit-es* Γ (*Guar k*, *Post k*);
c $\in$ *cpts-of-pes* Γ *pes s x*; *c*$\in$*assume-pes* Γ (*pre*, *rely*); Γ *c* $\propto$ *cs*$]\!] \Longrightarrow$
$\forall$ *n k*. *n* $\leq$ *length* (*cs k*) $\wedge$ *n* > *0* $\longrightarrow$ *take n* (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
proof –
assume *p1*: $\forall$ *k*. *pre* $\subseteq$ *Pre k*
and *p2*: $\forall$ *k*. *rely* $\subseteq$ *Rely k*
and *p3*: $\forall$ *k j*. *j*$\neq$*k* $\longrightarrow$ *Guar j* $\subseteq$ *Rely k*
and *p4*: *c* $\in$ *cpts-of-pes* Γ *pes s x*
and *p5*: *c* $\in$ *assume-pes* Γ (*pre*, *rely*)
and *p6*: Γ *c* $\propto$ *cs*
and *p0*: $\forall$ *k*. *cs k* $\in$ *commit-es* Γ (*Guar k*, *Post k*)
from *p6* **have** *p8*: $\forall$ *k*. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
from *p4 p6* **have** *p7*: $\forall$ *k*. *cs k* $\in$ *cpts-of-es* Γ (*pes k*) *s x* **using** *conjoin-imp-cptses-k* **by** *auto*
then have *p9*: $\forall$ *k*. *cs k* $\in$ *cpts-es* Γ $\wedge$ *cs k* !*0* = (*pes k*,*s*,*x*) **by** (*simp add*:*cpts-of-es-def*)
from *p6* **have** *p10*: $\forall$ *k j*. *j* < *length c* $\longrightarrow$ *gets* (*c*!*j*) = *gets-es* ((*cs k*)!*j*) **by** (*simp add*:*conjoin-def same-state-def*)
{

fix *n*
have $\forall k.\ n \leq$ *length* (*cs k*) $\wedge$ $n > 0 \longrightarrow$ *take n* (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
proof(*induct n*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc m*)
assume *b0*: $\forall k.\ m \leq$ *length* (*cs k*) $\wedge$ $0 < m \longrightarrow$ *take m* (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
{
fix *k*
assume *c0*: *Suc m* $\leq$ *length* (*cs k*) $\wedge$ $0 <$ *Suc m*
from *p7* **have** *c2*: *length* (*cs k*) > 0
by (*metis* (*no-types*, *lifting*) *cpts-es-not-empty cpts-of-es-def gr0I length-0-conv mem-Collect-eq*)
from *p6* **have** *c3*: *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)

let *?esl* = *take* (*Suc m*) (*cs k*)

have *take* (*Suc m*) (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
proof(*cases m* = *0*)
assume *d0*: *m* = *0*
have *gets-es* (*take* (*Suc m*) (*cs k*)!*0*) $\in$ *Pre k*
proof −
from *p6 c2 c3* **have** *gets* (*c*!*0*) = *gets-es* ((*cs k*)!*0*)
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** *gets* (*c*!*0*) $\in$ *pre* **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
from *d0* **have** *d1*: *length* (*take* (*Suc m*) (*cs k*)) = *1*
using *One-nat-def c2 gr0-implies-Suc length-take min-0R min-Suc-Suc* **by** *fastforce*
moreover
from *d1* **have** $\forall i.$ *Suc i* $<$ *length* (*take* (*Suc m*) (*cs k*))
$\longrightarrow \Gamma \vdash$ (*take* (*Suc m*) (*cs k*)) ! *i* −*ese*→ (*take* (*Suc m*) (*cs k*)) ! *Suc i*
$\longrightarrow$ (*gets-es* ((*take* (*Suc m*) (*cs k*)) ! *i*), *gets-es* ((*take* (*Suc m*) (*cs k*)) ! *Suc i*)) $\in$ *rely*
by *auto*
moreover
have *assume-es* Γ (*Pre k*, *Rely k*) = {*c*. *gets-es* (*c* ! *0*) $\in$ *Pre k* $\wedge$
($\forall i.$ *Suc i* $<$ *length c* $\longrightarrow \Gamma \vdash$ *c* ! *i* −*ese*→ *c* ! *Suc i*
$\longrightarrow$ (*gets-es* (*c* ! *i*), *gets-es* (*c* ! *Suc i*)) $\in$ *Rely k*)} **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **using** *Suc-neq-Zero less-one mem-Collect-eq* **by** *auto*

next
assume $m \neq 0$
then have *dd0*: $m > 0$ **by** *simp*
with *b0 c0* **have** *dd1*: $take\ m\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
by *simp*

have $gets\text{-}es\ (?esl\ !\ 0) \in Pre\ k$
proof −
from *p6 c2 c3* **have** $gets\ (c!0) = gets\text{-}es\ ((cs\ k)!0)$
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** $gets\ (c!0) \in pre$ **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
have $\forall i.\ Suc\ i<length\ ?esl \longrightarrow$
$\Gamma \vdash ?esl!i\ -ese\rightarrow\ ?esl!(Suc\ i) \longrightarrow$
$(gets\text{-}es\ (?esl!i),\ gets\text{-}es\ (?esl!Suc\ i)) \in Rely\ k$
proof −
{
fix *i*
assume *d0*: $Suc\ i<length\ ?esl$
and *d1*: $\Gamma \vdash ?esl!i\ -ese\rightarrow\ ?esl!Suc\ i$
then have *d2*: $?esl!i = (cs\ k)!i \wedge ?esl!Suc\ i = (cs\ k)!\ Suc\ i$
by *auto*
from *p6 c3 d0* **have** *d4*: $(\exists t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$
$-es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i))))$
$\vee$
$(\Gamma \vdash (c!i)\ -pese\rightarrow\ (c!Suc\ i) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow$
$((cs\ k)!\ Suc\ i))))$
by (*simp add*:*conjoin-def compat-tran-def*)
from *d1* **have** *d5*: $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$
by (*simp add*: *d2*)
from *d4* **have** $(gets\text{-}es\ (?esl!i),\ gets\text{-}es\ (?esl!Suc\ i)) \in Rely\ k$
proof
assume *e0*: $\exists t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$
$-es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$
then obtain *ct* **and** k' **where** *e1*: $(\Gamma \vdash (c!i)\ -pes-(ct\sharp k')\rightarrow$
$(c!Suc\ i)) \wedge$
$(\Gamma \vdash ((cs\ k')!i)\ -es-(ct\sharp k')\rightarrow\ ((cs\ k')!\ Suc\ i))$ **by** *auto*
with *p6 p8 d0 d5* **have** *e2*: $k \neq k'$
using *conjoin-def*[*of* Γ *c cs*] *same-spec-def*[*of c cs*]
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$ **by**

auto
with *d0* **have** $\Gamma \vdash (?esl!i) -ese\rightarrow (?esl!\ Suc\ i)$ **by** *auto*
then show *?thesis*
proof(*cases* $i < m - 1$)
assume *f0*: $i < m - 1$
with *d2* **have** *f1*:*take* (*Suc m*) (*cs k*) ! *i* = *take m* (*cs k*) ! *i*
by (*simp add*: *diff-less-Suc less-trans-Suc*)

from *f0* **have** *f2*: *take* (*Suc m*) (*cs k*) ! *Suc i* = *take m* (*cs k*) ! *Suc i*
by (*simp add*: *d2 gr-implies-not0 nat-le-linear*)
from *dd1* **have** ∀ *i*. *Suc i*<*length* (*take m* (*cs k*)) ⟶
Γ ⊢ (*take m* (*cs k*))!*i* −*ese*→ (*take m* (*cs k*))!(*Suc i*) ⟶
(*gets-es* ((*take m* (*cs k*))!*i*), *gets-es* ((*take m* (*cs k*))!*Suc i*)) ∈ *Rely k*
by (*simp add*:*assume-es-def*)
with *dd0 f0* **have** (*gets-es* (*take m* (*cs k*) ! *i*), *gets-es* (*take m* (*cs k*) ! *Suc i*)) ∈ *Rely k*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-mono Suc-pred d0 d1 f1 f2 length-take min-less-iff-conj*)
with *f1 f2* **show** *?thesis* **by** *simp*
next
assume ¬(*i* < *m* − *1*)
with *d0* **have** *f0*: *i* = *m* − *1*
by (*simp add*: *c0 dd0 less-antisym min.absorb2*)
let *?esl2* = *take* (*Suc m*) (*cs k′*)

from *b0 c0 dd0* **have** *take m* (*cs k′*) ∈ *assume-es* Γ (*Pre k′*, *Rely k′*)
by (*metis Suc-leD p8*)
moreover
from *e1 f0* **have** ¬(Γ ⊢ *cs k′* ! (*m*−*1*) −*ese*→ *cs k′* !*m*)
using *Suc-pred′ dd0 es-tran-not-etran1* **by** *fastforce*
ultimately have *f1*: *take* (*Suc m*) (*cs k′*) ∈ *assume-es* Γ (*Pre k′*, *Rely k′*)
using *assume-es-one-more*[*of cs k′* Γ *m Pre k′ Rely k′*] *p8 p9 c0 dd0*
by (*simp add*: *Suc-le-eq*)
from *p7* **have** *cs k′* ∈ *cpts-of-es* Γ (*pes k′*) *s x* **by** *simp*
with *p8 c0 dd0* **have** *f2*: *?esl2*∈*cpts-of-es* Γ (*pes k′*) *s x*
using *cpts-es-take*[*of cs k′* Γ *m*] *cpts-of-es-def*[*of* Γ *pes k′ s x*]
by (*simp add*: *Suc-le-lessD*)
from *p0 p8 c0* **have** *?esl2*∈*commit-es* Γ (*Guar k′*, *Post k′*)
using *commit-es-take-n*[*of Suc m cs k′* Γ *Guar k′ Post k′*] **by** *auto*
then have ∀ *i*. *Suc i*<*length ?esl2* ⟶
(∃ *t*. Γ ⊢ *?esl2*!*i* −*es*−*t*→ *?esl2*!(*Suc i*)) ⟶
(*gets-es* (*?esl2*!*i*), *gets-es* (*?esl2*!*Suc i*)) ∈ *Guar k′*

by (*simp add*:*commit-es-def*)

with *p8 e1 f0 c0 dd0* **have** (*gets-es* (*?esl2* ! ($m-1$)), *gets-es* (*?esl2* ! *m*))∈*Guar k'*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred diff-less-Suc length-take lessI min.absorb2 nth-take*)
with *p3 p10 c0 f0 e2* **show** *?thesis*
by (*smt Suc-diff-1 Suc-leD c3 dd0 le-less-linear not-less-eq-eq nth-take subsetCE*)
qed
next
assume *e0*: ((Γ ⊢ (*c*!*i*) −*pese*→ (*c*!*Suc i*)) ∧ (∀ *k*. (Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*))))
from *p5* **have** ∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*pese*→ *c*!(*Suc i*) ⟶
(*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely*
by (*simp add*:*assume-pes-def*)
moreover
from *p8 c0 d0* **have** *e1*:*Suc i* < *length c* **by** *simp*
ultimately have (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely* **using** *e0* **by** *simp*
with *p2* **have** (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *Rely k* **by** *auto*
with *p8 p10 c0 d0* **show** *?thesis*
using *Suc-lessD e1 d2* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed
}
then show *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *conjoin-comm-imp-rely*:
⟦∀ *k*. *pre* ⊆ *Pre k*; ∀ *k*. *rely* ⊆ *Rely k*;
∀ *k j*. *j*≠*k* ⟶ *Guar j* ⊆ *Rely k*;
∀ *k*. *cs k* ∈ *commit-es* Γ (*Guar k*, *Post k*);
c ∈ *cpts-of-pes* Γ *pes s x*; *c*∈*assume-pes* Γ (*pre*, *rely*); Γ *c* ∝ *cs*⟧ ⟹
∀ *k*. (*cs k*) ∈ *assume-es* Γ (*Pre k*, *Rely k*)
proof −
assume *a1*: ∀ *k*. *pre* ⊆ *Pre k*
assume *a2*: ∀ *k*. *rely* ⊆ *Rely k*
assume *a3*: ∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*
assume *a4*: ∀ *k*. *cs k* ∈ *commit-es* Γ (*Guar k*, *Post k*)
assume *a5*: *c* ∈ *cpts-of-pes* Γ *pes s x*

assume *a6*: $c \in$ *assume-pes* Γ (*pre*, *rely*)
assume *a7*: Γ $c \propto cs$
have *f8*: $c \neq []$
using *a5 cpts-of-pes-def* **by** *force*
from *a7* **have** *p8*: $\forall k$. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
{
fix *k*
have (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
using *a1 a2 a3 a4 a5 a6 a7 p8 f8*
conjoin-comm-imp-rely-n[*of pre Pre rely Rely Guar cs* Γ *Post c pes s x length* (*cs k*) *k*] **by** *force*
}
then show *?thesis* **by** *simp*
qed

lemma *cpts-es-sat-rely*[*rule-format*]:
⟦$\forall k$. $\Gamma \models$ (*pes k*) sat_s [*Pre k*, *Rely k*, *Guar k*, *Post k*];
$\forall k$. *pre* $\subseteq$ *Pre k*;
$\forall k$. *rely* $\subseteq$ *Rely k*;
$\forall k\ j$. $j{\neq}k \longrightarrow$ *Guar j* $\subseteq$ *Rely k*;
$c \in$ *cpts-of-pes* Γ *pes s x*; $c{\in}$*assume-pes* Γ (*pre*, *rely*);
Γ $c \propto cs$; $\forall k$. *cs k* $\in$ *cpts-of-es* Γ (*pes k*) *s x*⟧ $\Longrightarrow$
$\forall n\ k$. $n \leq$ *length* (*cs k*) $\wedge$ $n > 0 \longrightarrow$ *take n* (*cs k*)$\in$*assume-es* Γ (*Pre k*, *Rely k*)
proof –
assume *p0*: $\forall k$. $\Gamma \models$ (*pes k*) sat_s [*Pre k*, *Rely k*, *Guar k*, *Post k*]
and *p1*: $\forall k$. *pre* $\subseteq$ *Pre k*
and *p2*: $\forall k$. *rely* $\subseteq$ *Rely k*
and *p3*: $\forall k\ j$. $j{\neq}k \longrightarrow$ *Guar j* $\subseteq$ *Rely k*
and *p4*: $c \in$ *cpts-of-pes* Γ *pes s x*
and *p5*: $c \in$ *assume-pes* Γ (*pre*, *rely*)
and *p6*: Γ $c \propto cs$
and *p7*: $\forall k$. *cs k* $\in$ *cpts-of-es* Γ (*pes k*) *s x*
from *p6* **have** *p8*: $\forall k$. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
from *p7* **have** *p9*: $\forall k$. *cs k* $\in$ *cpts-es* Γ **using** *cpts-of-es-def mem-Collect-eq* **by** *fastforce*
from *p6* **have** *p10*: $\forall k\ j$. $j <$ *length c* $\longrightarrow$ *gets* (*c*!*j*) = *gets-es* ((*cs k*)!*j*) **by** (*simp add*:*conjoin-def same-state-def*)
{
fix *n*
have $\forall k$. $n \leq$ *length* (*cs k*) $\wedge$ $n > 0 \longrightarrow$ *take n* (*cs k*)$\in$*assume-es* Γ (*Pre k*, *Rely k*)
proof(*induct n*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc m*)
assume *b0*: $\forall k$. $m \leq$ *length* (*cs k*) $\wedge$ $0 < m \longrightarrow$ *take m* (*cs k*) $\in$ *assume-es*

Γ (*Pre k*, *Rely k*)

{
fix *k*
assume *c0*: *Suc m* $\leq$ *length* (*cs k*) $\wedge$ *0* < *Suc m*
from *p7* **have** *c2*: *length* (*cs k*) > *0*
by (*metis* (*no-types*, *lifting*) *cpts-es-not-empty cpts-of-es-def gr0I length-0-conv mem-Collect-eq*)
from *p6* **have** *c3*: *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)

let *?esl* = *take* (*Suc m*) (*cs k*)
have *?esl* $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
proof(*cases m* = *0*)
assume *d0*: *m* = *0*
have *gets-es* (*take* (*Suc m*) (*cs k*)!*0*) $\in$ *Pre k*
proof −
from *p6 c2 c3* **have** *gets* (*c*!*0*) = *gets-es* ((*cs k*)!*0*)
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** *gets* (*c*!*0*) $\in$ *pre* **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
from *d0* **have** *d1*: *length* (*take* (*Suc m*) (*cs k*)) = *1*
using *One-nat-def c2 gr0-implies-Suc length-take min-0R min-Suc-Suc* **by** *fastforce*
moreover
from *d1* **have** $\forall$ *i*. *Suc i* < *length* (*take* (*Suc m*) (*cs k*))
$\longrightarrow$ $\Gamma \vdash$ (*take* (*Suc m*) (*cs k*)) ! *i* −*ese*$\rightarrow$ (*take* (*Suc m*) (*cs k*)) ! *Suc i*
$\longrightarrow$ (*gets-es* ((*take* (*Suc m*) (*cs k*)) ! *i*), *gets-es* ((*take* (*Suc m*) (*cs k*)) ! *Suc i*)) $\in$ *rely*
by *auto*
moreover
have *assume-es* Γ (*Pre k*, *Rely k*) = {*c*. *gets-es* (*c* ! *0*) $\in$ *Pre k* $\wedge$
($\forall$ *i*. *Suc i* < *length c* $\longrightarrow$ $\Gamma \vdash$ *c* ! *i* −*ese*$\rightarrow$ *c* ! *Suc i*
$\longrightarrow$ (*gets-es* (*c* ! *i*), *gets-es* (*c* ! *Suc i*)) $\in$ *Rely k*)} **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **using** *Suc-neq-Zero less-one mem-Collect-eq* **by** *auto*
next
assume *m* $\neq$ *0*
then have *dd0*: *m* > *0* **by** *simp*
with *b0 c0* **have** *dd1*: *take m* (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*) **by** *simp*

have *gets-es* (*?esl* ! *0*) $\in$ *Pre k*
proof −

from *p6 c2 c3* **have** *gets* $(c!0)$ = *gets-es* $((cs\ k)!0)$
  by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** *gets* $(c!0) \in$ *pre* **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
have $\forall i.$ *Suc i<length ?esl* $\longrightarrow$
$\Gamma \vdash$ *?esl*!*i* $-ese\rightarrow$ *?esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *Rely k*
proof −
{
fix *i*
assume *d0*: *Suc i<length ?esl*
and *d1*: $\Gamma \vdash$ *?esl*!*i* $-ese\rightarrow$ *?esl*!*Suc i*
then have *d2*: *?esl*!*i* = (*cs k*)!*i* $\wedge$ *?esl*!*Suc i* = (*cs k*)! *Suc i*
by *auto*
from *p6 c3 d0* **have** *d4*: ($\exists$ *t k*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\wedge$
($\forall$ *k t*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*i*
$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc i*) $\wedge$
($\forall$ *k'*. *k'* $\neq$ *k* $\longrightarrow$ ($\Gamma \vdash$ *cs k'*!*i* $-ese\rightarrow$ *cs k'*! *Suc i*))))
$\vee$
(($\Gamma \vdash$ (*c*!*i*) $-pese\rightarrow$ (*c*!*Suc i*)) $\wedge$ ($\forall$ *k*. ($\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$
((*cs k*)! *Suc i*))))
by (*simp add*:*conjoin-def compat-tran-def*)
from *d1* **have** *d5*: $\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*)
by (*simp add*: *d2*)
from *d4* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *Rely k*
proof
assume *e0*: $\exists$ *t k*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\wedge$
($\forall$ *k t*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*i*
$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc i*) $\wedge$
($\forall$ *k'*. *k'* $\neq$ *k* $\longrightarrow$ ($\Gamma \vdash$ *cs k'*!*i* $-ese\rightarrow$ *cs k'*! *Suc i*)))
then obtain *ct* **and** *k'* **where** *e1*: ($\Gamma \vdash$ (*c*!*i*) $-pes-(ct\sharp k')\rightarrow$
(*c*!*Suc i*)) $\wedge$
($\Gamma \vdash$ ((*cs k'*)!*i*) $-es-(ct\sharp k')\rightarrow$ ((*cs k'*)! *Suc i*)) **by** *auto*
with *p6 p8 d0 d5* **have** *e2*: *k* $\neq$ *k'*
using *conjoin-def*[*of* Γ *c cs*] *same-spec-def*[*of c cs*]
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: $\Gamma \vdash$ ((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*) **by**
auto
with *d0* **have** $\Gamma \vdash$ (*?esl*!*i*) $-ese\rightarrow$ (*?esl*! *Suc i*) **by** *auto*
then show *?thesis*
proof(*cases i* < *m* − *1*)
assume *f0*: *i* < *m* − *1*
with *d2* **have** *f1*:*take* (*Suc m*) (*cs k*) ! *i* = *take m* (*cs k*) ! *i*
by (*simp add*: *diff-less-Suc less-trans-Suc*)

from *f0* **have** *f2*: *take* (*Suc m*) (*cs k*) ! *Suc i* = *take m* (*cs k*) ! *Suc i*
by (*simp add*: *d2 gr-implies-not0 nat-le-linear*)
from *dd1* **have** $\forall$ *i*. *Suc i*<*length* (*take m* (*cs k*)) $\longrightarrow$
$\Gamma \vdash$ (*take m* (*cs k*))!*i* −*ese*→ (*take m* (*cs k*))!(*Suc i*) $\longrightarrow$
(*gets-es* ((*take m* (*cs k*))!*i*), *gets-es* ((*take m* (*cs k*))!*Suc i*)) $\in$ *Rely k*
by (*simp add*:*assume-es-def*)
with *dd0 f0* **have** (*gets-es* (*take m* (*cs k*) ! *i*), *gets-es* (*take m* (*cs k*) ! *Suc i*)) $\in$ *Rely k*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-mono Suc-pred d0 d1 f1 f2 length-take min-less-iff-conj*)
with *f1 f2* **show** *?thesis* **by** *simp*
next
assume $\neg$(*i* < *m* − *1*)
with *d0* **have** *f0*: *i* = *m* − *1*
by (*simp add*: *c0 dd0 less-antisym min.absorb2*)
let *?esl2* = *take* (*Suc m*) (*cs k*′)

from *b0 c0 dd0* **have** *take m* (*cs k*′) $\in$ *assume-es* Γ (*Pre k*′, *Rely k*′)
by (*metis Suc-leD p8*)
moreover
from *e1 f0* **have** $\neg$($\Gamma \vdash$ *cs k*′ ! (*m*−*1*) −*ese*→ *cs k*′ !*m*)
using *Suc-pred*′ *dd0 es-tran-not-etran1* **by** *fastforce*
ultimately have *f1*: *take* (*Suc m*) (*cs k*′) $\in$ *assume-es* Γ (*Pre k*′, *Rely k*′)
using *assume-es-one-more*[*of cs k*′ Γ *m Pre k*′ *Rely k*′] *p8 p9 c0 dd0*
by (*simp add*: *Suc-le-eq*)
from *p7* **have** *cs k*′ $\in$ *cpts-of-es* Γ (*pes k*′) *s x* **by** *simp*
with *p8 c0 dd0* **have** *f2*: *?esl2*$\in$*cpts-of-es* Γ (*pes k*′) *s x*
using *cpts-es-take*[*of cs k*′ Γ *m*] *cpts-of-es-def*[*of* Γ *pes k*′ *s x*]
by (*simp add*: *Suc-le-lessD*)
from *p0* **have** *f3*: $\Gamma \models$ *pes k*′ sat_s [*Pre k*′, *Rely k*′, *Guar k*′, *Post k*′] **by** *simp*
with *f1 f2* **have** *?esl2*$\in$*commit-es* Γ (*Guar k*′, *Post k*′)
using *es-validity-def*[*of* Γ *pes k*′ *Pre k*′ *Rely k*′ *Guar k*′ *Post k*′]
by *auto*
then have $\forall$ *i*. *Suc i*<*length ?esl2* $\longrightarrow$
($\exists$ *t*. $\Gamma \vdash$ *?esl2*!*i* −*es*−*t*→ *?esl2*!(*Suc i*)) $\longrightarrow$
(*gets-es* (*?esl2*!*i*), *gets-es* (*?esl2*!*Suc i*)) $\in$ *Guar k*′
by (*simp add*:*commit-es-def*)

with *p8 e1 f0 c0 dd0* **have** (*gets-es* (*?esl2* ! (*m*−*1*)), *gets-es* (*?esl2* ! *m*))$\in$*Guar k*′
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred diff-less-Suc*

```
length-take lessI min.absorb2 nth-take)
                      with p3 p10 c0 f0 e2 show ?thesis
                      by (smt Suc-diff-1 Suc-leD c3 dd0 le-less-linear not-less-eq-eq
nth-take subsetCE)
                    qed
                next
                assume e0: ((Γ ⊢ (c!i) −pese→ (c!Suc i)) ∧ (∀ k. (Γ ⊢ ((cs k)!i)
−ese→ ((cs k)! Suc i))))
                  from p5 have ∀ i. Suc i<length c ⟶
                                Γ ⊢ c!i −pese→ c!(Suc i) ⟶
                                (gets (c!i), gets (c!Suc i)) ∈ rely
                    by (simp add:assume-pes-def)
                  moreover
                  from p8 c0 d0 have e1:Suc i < length c by simp
                 ultimately have (gets (c!i), gets (c!Suc i)) ∈ rely using e0 by
simp
                  with p2 have (gets (c!i), gets (c!Suc i)) ∈ Rely k by auto
                  with p8 p10 c0 d0 show ?thesis
                    using Suc-lessD e1 d2 by auto
                qed
            }
            then show ?thesis by auto
            qed

          ultimately show ?thesis by (simp add:assume-es-def)
        qed

      }
      then show ?case by auto
    qed
  }
  then show ?thesis by auto
  qed

lemma es-tran-sat-guar-aux:
  ⟦∀ k. Γ ⊨ (pes k) sat_s [Pre k, Rely k, Guar k, Post k];
      ∀ k. pre ⊆ Pre k;
      ∀ k. rely ⊆ Rely k;
      ∀ k j. j≠k ⟶ Guar j ⊆ Rely k;
      c ∈ cpts-of-pes Γ pes s x; c∈assume-pes Γ (pre, rely);
      Γ c ∝ cs; ∀ k. cs k ∈ cpts-of-es Γ (pes k) s x ⟧
      ⟹∀ k i m. m ≤ length c ⟶ Suc i < length (take m (cs k)) ⟶ (∃ t.(Γ ⊢
(take m (cs k))!i−es−t→((take m (cs k))!Suc i)))
             ⟶ (gets-es ((take m (cs k))!i),gets-es ((take m (cs k))!Suc i)) ∈
Guar k
  proof −
    assume p0: ∀ k. Γ ⊨ (pes k) sat_s [Pre k, Rely k, Guar k, Post k]
      and p1: ∀ k. pre ⊆ Pre k
      and p2: ∀ k. rely ⊆ Rely k
```

and *p3*: ∀ *k j*. *j*≠*k* ⟶ *Guar j* ⊆ *Rely k*
and *p4*: *c* ∈ *cpts-of-pes* Γ *pes s x*
and *p5*: *c* ∈ *assume-pes* Γ (*pre*, *rely*)
and *p6*: Γ *c* ∝ *cs*
and *p7*: ∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes k*) *s x*
from *p6* **have** *p8*: ∀ *k*. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
{
fix *k i m*
assume *a0*: *m* ≤ *length c*
and *a1*: *Suc i* < *length* (*take m* (*cs k*))
and *a2*: ∃ *t*.(Γ ⊢ (*take m* (*cs k*))!*i*−*es*−*t*→((*take m* (*cs k*))!*Suc i*))
have (*gets-es* ((*take m* (*cs k*))!*i*),*gets-es* ((*take m* (*cs k*))!*Suc i*)) ∈ *Guar k*
proof(*cases m* = *0*)
assume *m* = *0* **with** *a1* **show** *?thesis* **by** *auto*
next
assume *m* ≠ *0*
then have *b0*: *m* > *0* **by** *simp*
let *?esl* = *take m* (*cs k*)
from *p7* **have** *cs k* ∈ *cpts-of-es* Γ (*pes k*) *s x* **by** *simp*
then have *cs k*!*0*=(*pes k*,*s*,*x*) ∧ *cs k* ∈ *cpts-es* Γ **by** (*simp add*:*cpts-of-es-def*)
with *b0* **have** *?esl*!*0*=(*pes k*,*s*,*x*) ∧ *?esl* ∈ *cpts-es* Γ
by (*metis Suc-pred a0 cpts-es-take leD not-less-eq nth-take p8*)
then have *r1*: *?esl* ∈ *cpts-of-es* Γ (*pes k*) *s x* **by** (*simp add*:*cpts-of-es-def*)
from *p0 p1 p2 p3 p4 p5 p6 p7*
have ∀ *n*. *n* ≤ *length* (*cs k*) ∧ *n* > *0* ⟶ *take n* (*cs k*)∈*assume-es* Γ (*Pre k*, *Rely k*)
using *cpts-es-sat-rely*[*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*] **by** *auto*
with *p8 a0 b0* **have** *r2*: *?esl*∈*assume-es* Γ (*Pre k*, *Rely k*) **by** *auto*

from *p0* **have** (*cpts-of-es* Γ (*pes k*) *s x*) ∩ *assume-es* Γ (*Pre k*, *Rely k*) ⊆ *commit-es* Γ (*Guar k*, *Post k*)
by (*simp add*:*es-validity-def*)
with *r1 r2* **have** *?esl* ∈ *commit-es* Γ (*Guar k*, *Post k*)
using *IntI subsetCE* **by** *auto*
then have ∀ *i*. *Suc i*<*length ?esl* ⟶
(∃ *t*. Γ ⊢ *?esl*!*i* −*es*−*t*→ *?esl*!(*Suc i*)) ⟶ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *Guar k*
by (*simp add*:*commit-es-def*)
with *a1 a2* **show** *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *es-tran-sat-guar*:

$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ \ Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);$
$\Gamma\ c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x\ ]\!]$
$\Longrightarrow \forall k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists t.(\Gamma \vdash (cs\ k)!i{-}es{-}t{\rightarrow}(cs\ k)!Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((cs\ k)!i),gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$

proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow\ \ Guar\ j \subseteq Rely\ k$
and *p4*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
then have $\forall k\ i\ m.\ m \leq length\ c \longrightarrow Suc\ i < length\ (take\ m\ (cs\ k)) \longrightarrow (\exists t.(\Gamma \vdash (take\ m\ (cs\ k))!i{-}es{-}t{\rightarrow}((take\ m\ (cs\ k))!Suc\ i)))$
$\longrightarrow (gets\text{-}es\ ((take\ m\ (cs\ k))!i),gets\text{-}es\ ((take\ m\ (cs\ k))!Suc\ i)) \in Guar\ k$
using *es-tran-sat-guar-aux* [*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*] **by** *simp*
moreover
from *p6* **have** $\forall k.\ length\ c = length\ (cs\ k)$ **by** (*simp add*:*conjoin-def same-length-def*)
ultimately show *?thesis* **by** *auto*
qed

lemma *conjoin-es-sat-assume*:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ \ Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);$
$\Gamma\ c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x\ ]\!]$
$\Longrightarrow \forall k.\ cs\ k \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$

proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*[*rule-format*]: $\forall k\ j.\ j{\neq}k \longrightarrow\ \ Guar\ j \subseteq Rely\ k$
and *p4*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
from *p6* **have** *p11*[*rule-format*]: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)
from *p7* **have** *p12*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma$ **using** *cpts-of-es-def mem-Collect-eq*

by *fastforce*
  with *p11* **have** $c \neq Nil$ **using** *cpts-es-not-empty length-0-conv* **by** *auto*
  then have *p13*: *length* $c > 0$ **by** *auto*
  {
   fix *k*
   have *cs k* $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
    using *p0 p1 p2 p3 p4 p5 p6 p7 p13 p11*
     cpts-es-sat-rely[*of* Γ *pes Pre Rely Guar Post pre rely c s x cs length* (*cs k*) *k*] **by** *force*
  }
  then show *?thesis* **by** *auto*
 qed

lemma *pes-tran-sat-guar*:
 $[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
  $\forall k.\ pre \subseteq Pre\ k;$
  $\forall k.\ rely \subseteq Rely\ k;$
  $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
  $\forall k.\ Guar\ k \subseteq guar;$
  $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely)]\!]$
  $\Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}\ c!(Suc\ i))$
   $\longrightarrow (gets\ (c!i), gets\ (c!Suc\ i)) \in guar$
 proof –
  assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
   and *p1*: $\forall k.\ pre \subseteq Pre\ k$
   and *p2*: $\forall k.\ rely \subseteq Rely\ k$
   and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k$
   and *p4*: $\forall k.\ Guar\ k \subseteq guar$
   and *p5*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
   and *p6*: $c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely)$
   {
    fix *i*
    assume *a0*: $Suc\ i < length\ c$
     and *a1*: $\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}\ c!(Suc\ i)$
    from *p5* **have** $\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$
     by (*meson cpt-imp-exist-conjoin-cs*)
    then obtain *cs* **where** *a2*: $(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$ **by** *auto*
    then have *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
    with *a0* **have** *a3*: $(\exists t\ k.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
          $(\forall k\ t.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
          $(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i))))$
        $\vee$
        $((\Gamma \vdash (c!i)\ {-pese\rightarrow}\ (c!Suc\ i)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ {-ese\rightarrow}\ ((cs\ k)!\ Suc\ i))))$
     by (*simp add*:*compat-tran-def*)
    from *a1* **have** $\neg(\Gamma \vdash (c!i)\ {-pese\rightarrow}\ (c!Suc\ i))$
     using *pes-tran-not-etran1* **by** *blast*

with *a3* **have** $\exists\, t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$
$-es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$
by *simp*
then obtain t **and** k **where** *a4*: $(\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$
$-es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$
by *auto*
from *p0 p1 p2 p3 p4 p5 p6 a2* **have**
$\forall\, k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists\, t.(\Gamma \vdash (cs\ k)!i-es-t\rightarrow(cs\ k)!Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$
using *es-tran-sat-guar* [*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*] **by** *simp*
then have *a5*: $Suc\ i < length\ (cs\ k) \longrightarrow (\exists\, t.(\Gamma \vdash (cs\ k)!i-es-t\rightarrow(cs\ k)!Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$ **by** *simp*
from *a2* **have** *a6*: $length\ c = length\ (cs\ k)$ **by** (*simp add*:*conjoin-def same-length-def*)
with *a0 a4 a5* **have** *a7*: $(gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$ **by** *auto*
from *a0 a2* **have** *a8*: $gets\text{-}es\ ((cs\ k)!i) = gets\ (c!i)$ **by** (*simp add*:*conjoin-def same-state-def*)
from *a0 a2* **have** *a9*: $gets\text{-}es\ ((cs\ k)!Suc\ i) = gets\ (c!Suc\ i)$ **by** (*simp add*:*conjoin-def same-state-def*)
with *a7 a8* **have** $(gets\ (c!i), gets\ (c!Suc\ i)) \in Guar\ k$ **by** *auto*
with *p4* **have** $(gets\ (c!i), gets\ (c!Suc\ i)) \in guar$ **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *parallel-sound*:
$[\![\forall\, k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall\, k.\ pre \subseteq Pre\ k;$
$\forall\, k.\ rely \subseteq Rely\ k;$
$\forall\, k\ j.\ j \neq k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$\forall\, k.\ Guar\ k \subseteq guar;$
$\forall\, k.\ Post\ k \subseteq post]\!]$
$\Longrightarrow \Gamma \models pes\ SAT\ [pre,\ rely,\ guar,\ post]$
proof –
assume *p0*: $\forall\, k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall\, k.\ pre \subseteq Pre\ k$
and *p2*: $\forall\, k.\ rely \subseteq Rely\ k$
and *p3*: $\forall\, k\ j.\ j \neq k \longrightarrow\ Guar\ j \subseteq Rely\ k$
and *p4*: $\forall\, k.\ Guar\ k \subseteq guar$
and *p5*: $\forall\, k.\ Post\ k \subseteq post$
have $\forall\, s\ x.\ (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}pes\ \Gamma\ (guar,\ post)$

proof −
{
fix *c s x*
assume *a0*: $c \in (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely)$
then have *a1*: $c \in (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \wedge c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$ **by** *simp*
with *p0 p1 p2 p3 p4* **have** $\forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}\ c!(Suc\ i))$
$\longrightarrow (gets\ (c!i), gets\ (c!Suc\ i)) \in guar$
using *pes-tran-sat-guar* [*of* Γ *pes Pre Rely Guar Post pre rely guar c s x*] **by** *simp*
then have $c \in commit\text{-}pes\ \Gamma\ (guar,\ post)$
by (*simp add*: *commit-pes-def*)
}
then show *?thesis* **by** *auto*
qed

then show *?thesis* **by** (*simp add*:*pes-validity-def*)
qed

lemma *parallel-seq-sound*:
$[\![pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\Gamma \models pes\ SAT\ [pre',\ rely',\ guar',\ post']]\!]$
$\Longrightarrow \Gamma \models pes\ SAT\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: $pre \subseteq pre'$
and *p1*: $rely \subseteq rely'$
and *p2*: $guar' \subseteq guar$
and *p3*: $post' \subseteq post$
and *p4*: $\Gamma \models pes\ SAT\ [pre',\ rely',\ guar',\ post']$
from *p4* **have** *p5*: $\forall s\ x.\ (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre',\ rely') \subseteq commit\text{-}pes\ \Gamma\ (guar',\ post')$
by (*simp add*: *pes-validity-def*)
have $\forall s\ x.\ (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}pes\ \Gamma\ (guar,\ post)$
proof −
{
fix *c s x*
assume *a0*: $c \in (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely)$
then have $c \in (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \wedge c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$ **by** *simp*
with *p0 p1* **have** $c \in (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \wedge c \in assume\text{-}pes\ \Gamma\ (pre',\ rely')$
using *assume-pes-imp*[*of pre pre′ rely rely′ c*] **by** *simp*
with *p5* **have** $c \in commit\text{-}pes\ \Gamma\ (guar',\ post')$ **by** *auto*
with *p2 p3* **have** $c \in commit\text{-}pes\ \Gamma\ (guar,\ post)$
using *commit-pes-imp*[*of guar′ guar post′ post c*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*pes-validity-def*)

qed

lemma *parallel-sound'*:
assumes *p0*: $\forall k.\ \Gamma \vdash$ *fst* ((*pes*::'*k* $\Rightarrow$ ('*l*,'*k*,'*s*,'*prog*) *rgformula-es*) *k*) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
and *p1*: $\forall k.$ *pre* $\subseteq$ Pre_{es} (*pes k*)
and *p2*: $\forall k.$ *rely* $\subseteq$ $Rely_{es}$ (*pes k*)
and *p3*: $\forall k\, j.\ j \neq k \longrightarrow Guar_{es}$ (*pes j*) $\subseteq Rely_{es}$ (*pes k*)
and *p4*: $\forall k.\ Guar_{es}$ (*pes k*) $\subseteq$ *guar*
and *p5*: $\forall k.\ Post_{es}$ (*pes k*) $\subseteq$ *post*
shows $\Gamma \models$ *paresys-spec pes SAT* [*pre*, *rely*, *guar*, *post*]
proof –
from *p0* **have** $\forall k.\ \Gamma \models$ *evtsys-spec* (*fst* (*pes k*)) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
proof –
{
fix *k*
from *p0* **have** $\Gamma \vdash$ *fst* (*pes k*) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
by *simp*
then have $\Gamma \models$ *evtsys-spec* (*fst* (*pes k*)) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
using *rgsound-es* [*of* Γ *fst* (*pes k*) Pre_{es} (*pes k*) $Rely_{es}$ (*pes k*) $Guar_{es}$ (*pes k*) $Post_{es}$ (*pes k*)]
by *simp*
}
then show *?thesis* **by** *auto*
qed
with *p1 p2 p3 p4 p5* **show** $\Gamma \models$ *paresys-spec pes SAT* [*pre*, *rely*, *guar*, *post*]
using *parallel-sound* [*of* Γ *paresys-spec pes* $Pre_{es}\circ$*pes* $Rely_{es}\circ$*pes* $Guar_{es}\circ$*pes* $Post_{es}\circ$*pes*
pre rely guar post] **by** (*simp add*:*paresys-spec-def*)
qed

theorem *rgsound-pes*: $\Gamma \vdash$ *rgf-par SAT* [*pre*, *rely*, *guar*, *post*] $\Longrightarrow \Gamma \models$ *paresys-spec rgf-par SAT* [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-pes.induct*)

using *parallel-sound'* **apply** *blast*
using *parallel-seq-sound* **apply** *blast*
done

end

end

8 Rely-guarantee-based Safety Reasoning

theory *PiCore-RG-Invariant*

```
imports PiCore-Hoare
begin

type-synonym 's invariant = 's ⇒ bool

context event-hoare
begin

definition invariant-presv-pares::'Env ⇒ 's invariant ⇒ ('l,'k,'s,'prog) paresys ⇒
's set ⇒ ('s × 's) set ⇒ bool
  where invariant-presv-pares Γ invar pares init R ≡
        ∀ s0 x0 pesl. s0∈init ∧ pesl ∈ (cpts-of-pes Γ pares s0 x0 ∩ assume-pes Γ
(init, R))
                      ⟶ (∀ i<length pesl. invar (gets (pesl!i)))


theorem invariant-theorem:
  assumes parsys-sat-rg: Γ ⊢ pesf SAT [init, R, G, pst]
    and  stb-rely: stable (Collect invar) R
    and  stb-guar: stable (Collect invar) G
    and  init-in-invar: init ⊆ (Collect invar)
  shows invariant-presv-pares Γ invar (paresys-spec pesf) init R
proof −
  from parsys-sat-rg have Γ ⊨ paresys-spec pesf SAT [init, R, G, pst] using
rgsound-pes by fast
  hence cpts-pes: ∀ s x. (cpts-of-pes Γ (paresys-spec pesf) s x) ∩ assume-pes Γ
(init, R) ⊆ commit-pes Γ (G, pst)
    by (simp add:pes-validity-def)
  show ?thesis
  proof −
  {
    fix s0 x0 pesl
    assume a0: s0∈init
      and  a1: pesl∈cpts-of-pes Γ (paresys-spec pesf) s0 x0 ∩ assume-pes Γ (init,
R)
     from a1 have a3: pesl!0 = (paresys-spec pesf, s0, x0) ∧ pesl∈cpts-pes Γ by
(simp add:cpts-of-pes-def)
    from a1 cpts-pes have pesl-in-comm: pesl ∈ commit-pes Γ (G, pst) by auto
    {
      fix i
      assume b0: i<length pesl
      then have gets (pesl!i) ∈ (Collect invar)
      proof(induct i)
        case 0
        with a3 have gets (pesl!0) = s0 by (simp add:gets-def)
        with a0 init-in-invar show ?case by auto
      next
        case (Suc ni)
        assume c0: ni < length pesl ⟹ gets (pesl ! ni) ∈ (Collect invar)
```

and *c1*: *Suc ni* < *length pesl*
then have *c2*: *gets* (*pesl* ! *ni*) ∈ (*Collect invar*) **by** *auto*
from *c1* **have** *c3*: *ni* < *length pesl* **by** *simp*
with *c0* **have** *c4*: *gets* (*pesl* ! *ni*) ∈ (*Collect invar*) **by** *simp*
from *a3 c1* **have** Γ ⊢ *pesl* ! *ni* −*pese*→ *pesl* ! *Suc ni* ∨ (∃ *et*. Γ ⊢ *pesl* ! *ni* −*pes*−*et*→ *pesl* ! *Suc ni*)
using *incpts-pes-impl-evnorcomptran* **by** *blast*
then show *?case*
proof
assume *d0*: Γ ⊢ *pesl* ! *ni* −*pese*→ *pesl* ! *Suc ni*
then show *?thesis* **using** *c3 c4 a1 c1 stb-rely* **by**(*simp add*:*assume-pes-def stable-def*)
next
assume ∃ *et*. Γ ⊢ *pesl* ! *ni* −*pes*−*et*→ *pesl* ! *Suc ni*
then obtain *et* **where** *d0*: Γ ⊢ *pesl* ! *ni* −*pes*−*et*→ *pesl* ! *Suc ni* **by** *auto*
then show *?thesis* **using** *c3 c4 c1 pesl-in-comm stb-guar* **apply**(*simp add*:*commit-pes-def stable-def*)
by *blast*
qed
qed
}
}
then show *?thesis* **using** *invariant-presv-pares-def* **by** *blast*
qed
qed

end

end

9 messaging system

theory *dmbus*
imports ../*picore*/*PiCore-RG-Invariant*
begin

9.1 model

record (′*a*,′*b*) *Config* = *writer* :: ′*a set*
readers :: ′*b set*

locale *dmsg-bus* = *event-hoare ptran petran fin-com cpts-p cpts-of-p prog-validity assume-p*
commit-p rghoare-p
for *ptran* :: ′*Env* ⇒ ((′*prog* × ′*s*) × ′*prog* × ′*s*) *set*
and *petran* :: ′*Env* ⇒ (′*s*,′*prog*) *pconf* ⇒ (′*s*,′*prog*) *pconf* ⇒ *bool*
and *fin-com* :: ′*prog*
and *cpts-p* :: ′*Env* ⇒ (′*s*,′*prog*) *pconfs set*
and *cpts-of-p* :: ′*Env* ⇒ ′*prog* ⇒ ′*s* ⇒ ((′*s*,′*prog*) *pconfs*) *set*

and *prog-validity* :: $'Env \Rightarrow 'prog \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
and *assume-p* :: $'Env \Rightarrow ('s\ set \times ('s \times 's)\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$
and *commit-p* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$
and *rghoare-p* :: $'Env \Rightarrow ['prog, 's\ set, ('s \times 's)\ set, ('s \times 's)\ set, 's\ set] \Rightarrow bool$
+
fixes *conf* :: $'sys \Rightarrow ('buf,'buf)\ Config$
fixes *buf-writer* :: $'s \Rightarrow 'buf \Rightarrow 'sys\ option$
fixes *buf-readers* :: $'s \Rightarrow 'buf \Rightarrow 'sys\ set$
fixes *buf-msg* :: $'s \Rightarrow ('buf \Rightarrow 'mtype)$
fixes *local-vars* :: $'s \Rightarrow 'sys \Rightarrow 'v$
fixes *bufs* :: $'s \Rightarrow 'buf\ set$
assumes *singlewrite* : $\forall\ sys1\ sys2\ b.\ sys1 \neq sys2 \wedge b \in writer\ (conf\ sys1) \longrightarrow b \notin writer\ (conf\ sys2)$
assumes *writerstb*: $\forall\ s\ sys\ .\{b.\ \ buf\text{-}writer\ s\ b = Some(sys)\} \subseteq writer\ (conf\ sys)$
assumes *readerstb* :$\forall\ s\ sys\ .\{b.\ sys \in buf\text{-}readers\ s\ b\} \subseteq readers\ (conf\ sys)$
begin

definition *get-conf-wrt-bufs* $sys = writer(conf\ sys)$

definition *get-conf-rd-bufs* $sys = readers(conf\ sys)$

definition *get-conf-rd-set* $b = \{sys.\ b \in readers(conf\ sys)\}$

definition *get-wrt-bufs* $s\ sys \equiv \{b.\ \ buf\text{-}writer\ s\ b = Some(sys)\}$

definition *get-rd-bufs* $s\ sys \equiv \{b.\ sys \in buf\text{-}readers\ s\ b\}$

definition *isCreate*:: $'s \Rightarrow 'buf \Rightarrow bool$
where *isCreate* $s\ b \equiv (if\ b \notin bufs\ s\ then\ True\ else\ False)$

definition *create-buf* $s\ b\ sys\ sysSet\ mtype \equiv$
$SOME\ t.\ \forall\ b'\ .(b'=b \wedge b \notin (bufs\ s) \longrightarrow buf\text{-}writer\ t = (buf\text{-}writer\ s)(b := Some(sys)) \wedge$
$buf\text{-}readers\ t = (buf\text{-}readers\ s)(b := \ sysSet) \wedge buf\text{-}msg\ t\ b = mtype \wedge bufs\ t = insert\ b\ (bufs\ s))$
$\wedge (b' \neq b \longrightarrow buf\text{-}writer\ t = buf\text{-}writer\ s \wedge buf\text{-}readers\ s = buf\text{-}readers\ t$
$\wedge buf\text{-}msg\ s\ b' = buf\text{-}msg\ t\ b \wedge bufs\ t = bufs\ s)$

definition *remove-buf* $s\ b\ sys \equiv$
$SOME\ t.\ \forall\ b'\ .b' = b \wedge b' \in (get\text{-}wrt\text{-}bufs\ s\ sys) \wedge b \in (bufs\ s) \longrightarrow$
$buf\text{-}writer\ t = (buf\text{-}writer\ s)(b := None) \wedge$
$buf\text{-}readers\ t = (buf\text{-}readers\ s)(b := \{\}) \wedge bufs\ t = (bufs\ s) - \{b\} \wedge$
$(b' \neq b \wedge b' \notin (get\text{-}wrt\text{-}bufs\ s\ sys) \longrightarrow buf\text{-}writer\ t = buf\text{-}writer\ s \wedge$
$buf\text{-}readers\ t = buf\text{-}readers\ s \wedge buf\text{-}msg\ s\ b' = buf\text{-}msg\ t\ b \wedge bufs\ t = bufs\ s)$

definition *bufs-stb* :: $'buf\ set \Rightarrow ('s \times 's)\ set$
where *bufs-stb* $bs \equiv \{(s,t).\ \forall\ b'\ .\ b' \in bs \wedge b' \in bufs\ s \longrightarrow buf\text{-}msg\ s\ b' = buf\text{-}msg$

t b'}

definition *bufs-stb-cpl* :: *'buf set* ⇒ (*'s* × *'s*) *set*
where *bufs-stb-cpl bs* ≡ {(*s*,*t*). ∀ *b'* .*b'*∉ *bs* ∧ *b'* ∈ *bufs s* − *bs* ⟶ *buf-msg s b'* = *buf-msg t b'*}

definition *assm-bufs-stb-sys* :: *'s* ⇒*'sys* ⇒ (*'s* × *'s*) *set*
where *assm-bufs-stb-sys s sys* ≡ *bufs-stb* (*get-wrt-bufs s sys*)

definition *guar-bufs-stb-sys* :: *'s* ⇒*'sys* ⇒ (*'s* × *'s*) *set*
where *guar-bufs-stb-sys s sys* ≡ *bufs-stb-cpl* (*get-wrt-bufs s sys*)

definition *assm-lvars-stb-sys* :: *'sys* ⇒ (*'s* × *'s*) *set*
where *assm-lvars-stb-sys sys* ≡ {(*s*,*t*). *local-vars s sys* = *local-vars t sys*}

definition *guar-lvars-stb-sys* :: *'sys* ⇒ (*'s* × *'s*) *set*
where *guar-lvars-stb-sys sys* ≡ {(*s*,*t*). ∀ *sys'*≠*sys*. *local-vars s sys'* = *local-vars t sys'*}

lemma *guar-in-rely-bufs*: *sys1* ≠ *sys2* ⟹ *guar-bufs-stb-sys s sys1* ⊆ *assm-bufs-stb-sys s sys2*
apply(*simp add*:*assm-bufs-stb-sys-def guar-bufs-stb-sys-def*)
apply(*simp add*:*bufs-stb-def bufs-stb-cpl-def get-wrt-bufs-def*)
by *fastforce*

lemma *guar-in-rely-lvars*: *sys1* ≠ *sys2* ⟹ *guar-lvars-stb-sys sys1* ⊆ *assm-lvars-stb-sys sys2*
apply(*simp add*:*assm-lvars-stb-sys-def guar-lvars-stb-sys-def*)
by *auto*

lemma *dsingleread*:∀ *sys1 sys2 b*. *sys1* ≠ *sys2* ∧*b* ∈ (*get-wrt-bufs s sys1*)⟶*b* ∉(*get-wrt-bufs s sys2*)
apply(*simp add*:*get-wrt-bufs-def*)
done

end

9.2 rely-guar reasoning

locale *dmsg-bus-rg* = *dmsg-bus ptran petran fin-com cpts-p cpts-of-p prog-validity assume-p commit-p rghoare-p conf buf-writer buf-readers buf-msg local-vars bufs*

for *ptran* :: *'Env* ⇒ ((*'prog* × *'s*) × *'prog* × *'s*) *set*
and *petran* :: *'Env* ⇒ (*'s*,*'prog*) *pconf* ⇒ (*'s*,*'prog*) *pconf* ⇒ *bool*
and *fin-com* :: *'prog*
and *cpts-p* :: *'Env* ⇒ (*'s*,*'prog*) *pconfs set*
and *cpts-of-p* :: *'Env* ⇒ *'prog* ⇒ *'s* ⇒ ((*'s*,*'prog*) *pconfs*) *set*
and *prog-validity* :: *'Env* ⇒ *'prog* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*

and *assume-p* :: $'Env \Rightarrow ('s\ set \times ('s \times 's)\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$
and *commit-p* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$
and *rghoare-p* :: $'Env \Rightarrow ['prog,\ 's\ set,\ ('s \times 's)\ set,\ ('s \times 's)\ set,\ 's\ set] \Rightarrow bool$
and *conf* :: $'sys \Rightarrow ('buf,'buf)\ Config$
and *buf-writer* :: $'s \Rightarrow 'buf \Rightarrow 'sys\ option$
and *buf-readers* :: $'s \Rightarrow 'buf \Rightarrow 'sys\ set$
and *buf-msg* :: $'s \Rightarrow ('buf \Rightarrow 'm)$
and *local-vars* :: $'s \Rightarrow 'sys \Rightarrow 'v$
and *bufs* :: $'s \Rightarrow 'buf\ set$
+
fixes *inv* :: $'s \Rightarrow bool$
and *whole-sys-spec* :: $'s \Rightarrow 'sys \Rightarrow ('l,'sys,'s,'prog)$ *rgformula-es*
and *init* :: $'s\ set$

assumes *gcond* : $\forall sys\ s.\ Guar_{es}$ (*whole-sys-spec s sys*) $\subseteq$
(*guar-bufs-stb-sys s sys* $\cap$ *guar-lvars-stb-sys sys*)
assumes *rcond* : $\forall sys\ s.$ (*assm-bufs-stb-sys s sys* $\cap$ *assm-lvars-stb-sys sys*)
$\subseteq Rely_{es}$ (*whole-sys-spec s sys*)
assumes *syssat*: $\forall sys.\ \Gamma \vdash fst$ (*whole-sys-spec s sys*) sat_s
[Pre_{es} (*whole-sys-spec s sys*),
$Rely_{es}$ (*whole-sys-spec s sys*),
$Guar_{es}$ (*whole-sys-spec s sys*),
$Post_{es}$ (*whole-sys-spec s sys*)]
assumes *inv-stb-rely*: $\forall sys.$ *stable* (*Collect inv*) ($Rely_{es}$ (*whole-sys-spec s sys*))
assumes *inv-stb-guar*: $\forall sys.$ *stable* (*Collect inv*) ($Guar_{es}$ (*whole-sys-spec s sys*))
assumes *init*: $\forall sys.\ init \subseteq Pre_{es}$ (*whole-sys-spec s sys*)
assumes *init-inv*: $init \subseteq$ (*Collect inv*)
begin

definition $P \equiv (\bigcap sys\ s.\ Pre_{es}$ (*whole-sys-spec s sys*))
definition $R \equiv (\bigcap sys\ s.\ Rely_{es}$ (*whole-sys-spec s sys*))
definition $G \equiv (\bigcup sys\ s.\ Guar_{es}$ (*whole-sys-spec s sys*))
definition $Q \equiv (\bigcup sys\ s.\ Post_{es}$ (*whole-sys-spec s sys*))

lemma *inv-stb-R*: *stable* (*Collect inv*) *R*
apply(*simp add*:*R-def*) **using** *inv-stb-rely*
by(*simp add*: *stable-def*)

lemma *inv-stb-G*: *stable* (*Collect inv*) *G*
apply(*simp add*:*G-def*) **using** *inv-stb-guar*
apply(*simp add*: *stable-def*)
by *auto*

lemma *wholesys-sat-RG*: $\forall s\ .\Gamma \vdash$ (*whole-sys-spec s*) *SAT* [*P*, *R*, *G*, *Q*]
apply(*rule allI*)
apply(*rule ParallelESys*)
using *syssat* **apply** *fast*
apply(*simp add*:*P-def*) **apply** *auto*[*1*]

apply(*simp add*:*R-def*) **apply** *auto*[*1*]
using *gcond rcond guar-in-rely-bufs guar-in-rely-lvars* **apply** *fast*
apply(*simp add*:*G-def*) **apply** *auto*[*1*]
apply(*simp add*:*Q-def*) **apply** *auto*[*1*]
done

lemma *inv*:*invariant-presv-pares* Γ *inv* (*paresys-spec* (*whole-sys-spec s*)) *init R*
apply(*rule invariant-theorem*[**where** *G*=*G* **and** *pst* = *Q*]) **defer**
using *inv-stb-R* **apply** *fast*
using *inv-stb-G* **apply** *fast*
using *init-inv* **apply** *fast*
using *wholesys-sat-RG*[*of* Γ] *init P-def R-def G-def Q-def*
ParallelESys-conseq [*of init P R R G G Q Q* Γ (*whole-sys-spec s*)]
by *auto*
end

end

10 Syntax of SIMP language

theory *SIMP-lang*
imports *Main*
begin

type-synonym *'s bexp* = *'s set*

datatype *'s prog* =
Basic 's ⇒*'s*
| *Seq 's prog 's prog*
| *Cond 's bexp 's prog 's prog*
| *While 's bexp 's prog*
| *Await 's bexp 's prog*
| *Nondt* (*'s* × *'s*) *set*

end

11 Operational Semantics of SIMP language

theory *SIMP-semantics*
imports *SIMP-lang*
begin

type-synonym *'s imp-pconf* = ((*'s prog*) *option*) × *'s*

inductive-set
ptran :: (*'s imp-pconf* × *'s imp-pconf*) *set*
and *ptran'* :: *'s imp-pconf* ⇒ *'s imp-pconf* ⇒ *bool* (- −*impc*→ - [*81*,*81*] *80*)
and *ptrans* :: *'s imp-pconf* ⇒ *'s imp-pconf* ⇒ *bool* (- −*impc*∗→ - [*81*,*81*] *80*)

where
  P *−impc→* $Q \equiv (P,Q) \in$ *ptran*
| P *−impc∗→* $Q \equiv (P,Q) \in$*ptran*ˆ∗

| *Basic*: (*Some* (*Basic f*), *s*) *−impc→* (*None*, *f s*)
| *Seq1*: (*Some P0*, *s*) *−impc→* (*None*, *t*) $\Longrightarrow$ (*Some* (*Seq P0 P1*), *s*) *−impc→* (*Some P1*, *t*)
| *Seq2*: (*Some P0*, *s*) *−impc→* (*Some P2*, *t*) $\Longrightarrow$ (*Some*(*Seq P0 P1*), *s*) *−impc→* (*Some*(*Seq P2 P1*), *t*)
| *CondT*: *s*∈*b* $\Longrightarrow$ (*Some*(*Cond b P1 P2*), *s*) *−impc→* (*Some P1*, *s*)
| *CondF*: *s*∉*b* $\Longrightarrow$ (*Some*(*Cond b P1 P2*), *s*) *−impc→* (*Some P2*, *s*)
| *WhileF*: *s*∉*b* $\Longrightarrow$ (*Some*(*While b P*), *s*) *−impc→* (*None*, *s*)
| *WhileT*: *s*∈*b* $\Longrightarrow$ (*Some*(*While b P*), *s*) *−impc→* (*Some*(*Seq P* (*While b P*)), *s*)
| *Await*: ⟦*s*∈*b*; (*Some P*, *s*) *−impc∗→* (*None*, *t*)⟧ $\Longrightarrow$ (*Some*(*Await b P*), *s*) *−impc→* (*None*, *t*)
| *Nondt*: (*s*,*t*)∈*r* $\Longrightarrow$ (*Some*(*Nondt r*), *s*) *−impc→* (*None*, *t*)

monos *rtrancl-mono*

11.1 Lemmas

11.1.1 programs

lemma *list-eq-if* [*rule-format*]:
  ∀ *ys*. *xs*=*ys* ⟶ (*length xs* = *length ys*) ⟶ (∀ *i*<*length xs*. *xs*!*i*=*ys*!*i*)
  by (*induct xs*) *auto*

lemma *list-eq*: (*length xs* = *length ys* ∧ (∀ *i*<*length xs*. *xs*!*i*=*ys*!*i*)) = (*xs*=*ys*)
apply(*rule iffI*)
 apply *clarify*
 apply(*erule nth-equalityI*)
 apply *simp*+
done

lemma *nth-tl*: ⟦ *ys*!*0*=*a*; *ys*≠[] ⟧ $\Longrightarrow$ *ys*=(*a*#(*tl ys*))
  by (*cases ys*) *simp-all*

lemma *nth-tl-if* [*rule-format*]: *ys*≠[] ⟶ *ys*!*0*=*a* ⟶ *P ys* ⟶ *P* (*a*#(*tl ys*))
  by (*induct ys*) *simp-all*

lemma *nth-tl-onlyif* [*rule-format*]: *ys*≠[] ⟶ *ys*!*0*=*a* ⟶ *P* (*a*#(*tl ys*)) ⟶ *P ys*
  by (*induct ys*) *simp-all*

lemma *seq-not-eq1*: *Seq c1 c2*≠*c1*
  by (*induct c1*) *auto*

lemma *seq-not-eq2*: *Seq c1 c2*≠*c2*
  by (*induct c2*) *auto*

lemma *if-not-eq1*: *Cond b c1 c2* ≠*c1*

by (*induct c1*) *auto*

lemma *if-not-eq2*: *Cond b c1 c2*$\neq$*c2*
by (*induct c2*) *auto*

lemmas *seq-and-if-not-eq* [*simp*] = *seq-not-eq1 seq-not-eq2*
seq-not-eq1 [*THEN not-sym*] *seq-not-eq2* [*THEN not-sym*]
if-not-eq1 if-not-eq2 if-not-eq1 [*THEN not-sym*] *if-not-eq2* [*THEN not-sym*]

lemma *prog-not-eq-in-ctran-aux*:
assumes *c*: (P,s) *−impc→* (Q,t)
shows $P \neq Q$ **using** *c*
by (*induct x1* $\equiv$ (P,s) *x2* $\equiv$ (Q,t) *arbitrary*: *P s Q t*) *auto*

lemma *prog-not-eq-in-ctran* [*simp*]: $\neg$ (P,s) *−impc→* (P,t)
apply *clarify*
apply(*drule prog-not-eq-in-ctran-aux*)
apply *simp*
done

end

12 Computation of SIMP language

theory *SIMP-computation*
imports *SIMP-semantics*
begin

inductive-set
petran :: (*'s imp-pconf* $\times$ *'s imp-pconf*) *set*
and *petran'* :: *'s imp-pconf* $\Rightarrow$ *'s imp-pconf* $\Rightarrow$ *bool* (- *−imppe→* - [*81*,*81*] *80*)
where
P −imppe→ Q $\equiv$ $(P,Q) \in$ *petran*
| *EnvP*: (P, s) *−imppe→* (P, t)

lemma *petranE*: *p −imppe→ p'* $\Longrightarrow$ ($\bigwedge P\ s\ t.\ p = (P, s) \Longrightarrow p' = (P, t) \Longrightarrow Q$)
$\Longrightarrow Q$
by (*induct p*, *induct p'*, *erule petran.cases*, *blast*)

lemma *petran-eq*: (P,s)*−imppe→*(Q,t) $\Longrightarrow P = Q$
apply(*rule petran.cases*) **apply** *simp+*
done

type-synonym *'s imp-pconfs* = *'s imp-pconf list*

inductive-set *cpts-p* :: *'s imp-pconfs set*
where
CptsPOne: $[(P,s)] \in$ *cpts-p*

| *CptsPEnv*: $(P,\ t)\#xs \in$ *cpts-p* $\Longrightarrow (P,s)\#(P,t)\#xs \in$ *cpts-p*
| *CptsPComp*: $[\![(P,s)$ $-impc\rightarrow$ (Q,t); $(Q,\ t)\#xs \in$ *cpts-p*$]\!] \Longrightarrow (P,s)\#(Q,t)\#xs \in$ *cpts-p*

thm *cpts-p.simps*
definition *cpts-of-p* :: $('s$ *prog*) *option* $\Rightarrow$ $'s \Rightarrow$ $('s$ *imp-pconfs*) *set* **where**
cpts-of-p P s $\equiv$ $\{l.\ l!0=(P,s) \wedge l \in$ *cpts-p*$\}$

lemma *cptn-not-empty* [*simp*]:$[] \notin$ *cpts-p*
apply(*force elim*:*cpts-p.cases*)
done

12.1 Modular definition of program computations

definition *lift* :: $'s$ *prog* $\Rightarrow$ $'s$ *imp-pconf* $\Rightarrow$ $'s$ *imp-pconf* **where**
lift Q $\equiv$ $\lambda(P,\ s)$. (*if P=None then* (*Some Q*,*s*) *else* (*Some*(*Seq* (*the P*) *Q*), *s*))

inductive-set *cpt-p-mod* :: $('s$ *imp-pconfs*) *set*
where
CptPModOne: $[(P,\ s)] \in$ *cpt-p-mod*
| *CptPModEnv*: $(P,\ t)\#xs \in$ *cpt-p-mod* $\Longrightarrow (P,\ s)\#(P,\ t)\#xs \in$ *cpt-p-mod*
| *CptPModNone*: $[\![$(*Some P*, *s*) $-impc\rightarrow$ (*None*, *t*); (*None*, *t*)#*xs* $\in$ *cpt-p-mod* $]\!]$ $\Longrightarrow$ (*Some P*,*s*)#(*None*, *t*)#*xs* $\in$*cpt-p-mod*

| *CptPModCondT*: $[\![$(*Some P0*, *s*)#*ys* $\in$ *cpt-p-mod*; $s \in b$ $]\!]$ $\Longrightarrow$ (*Some*(*Cond b P0 P1*), *s*)#(*Some P0*, *s*)#*ys* $\in$ *cpt-p-mod*
| *CptPModCondF*: $[\![$(*Some P1*, *s*)#*ys* $\in$ *cpt-p-mod*; $s \notin b$ $]\!]$ $\Longrightarrow$ (*Some*(*Cond b P0 P1*), *s*)#(*Some P1*, *s*)#*ys* $\in$ *cpt-p-mod*

| *CptPModSeq1*: $[\![$(*Some P0*, *s*)#*xs* $\in$ *cpt-p-mod*; *zs=map* (*lift P1*) *xs* $]\!]$
$\Longrightarrow$ (*Some*(*Seq P0 P1*), *s*)#*zs* $\in$ *cpt-p-mod*
| *CptPModSeq2*:
$[\![$(*Some P0*, *s*)#*xs* $\in$ *cpt-p-mod*; *fst*(*last* ((*Some P0*, *s*)#*xs*)) = *None*;
(*Some P1*, *snd*(*last* ((*Some P0*, *s*)#*xs*)))#*ys* $\in$ *cpt-p-mod*;
zs=(*map* (*lift P1*) *xs*)@*ys* $]\!]$ $\Longrightarrow$ (*Some*(*Seq P0 P1*), *s*)#*zs* $\in$ *cpt-p-mod*

| *CptPModWhile1*:
$[\![$ (*Some P*, *s*)#*xs* $\in$ *cpt-p-mod*; $s \in b$; *zs=map* (*lift* (*While b P*)) *xs* $]\!]$
$\Longrightarrow$ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* $\in$ *cpt-p-mod*
| *CptPModWhile2*:
$[\![$ (*Some P*, *s*)#*xs* $\in$ *cpt-p-mod*; *fst*(*last* ((*Some P*, *s*)#*xs*))=*None*; $s \in b$;
zs=(*map* (*lift* (*While b P*)) *xs*)@*ys*;
(*Some*(*While b P*), *snd*(*last* ((*Some P*, *s*)#*xs*)))#*ys* $\in$ *cpt-p-mod*$]\!]$
$\Longrightarrow$ (*Some*(*While b P*), *s*)#(*Some*(*Seq P* (*While b P*)), *s*)#*zs* $\in$ *cpt-p-mod*

12.2 Lemmas

12.2.1 Programs

lemma *tl-in-cptn*: $\llbracket$ *a#xs* ∈*cpts-p*; *xs*≠[] $\rrbracket \Longrightarrow$ *xs*∈*cpts-p*
  by (*force elim*: *cpts-p.cases*)

lemma *tl-zero*[*rule-format*]:
  P (*ys*!*Suc j*) ⟶ *Suc j*<*length ys* ⟶ *ys*≠[] ⟶ *P* (*tl*(*ys*)!*j*)
  by (*induct ys*) *simp-all*

12.3 Equivalence of Sequential and Modular Definitions of Programs.

lemma *last-length*: ((*a#xs*)!(*length xs*))=*last* (*a#xs*)
  by (*induct xs*) *auto*

lemma *div-seq* [*rule-format*]: *list* ∈ *cpt-p-mod* ⟹
 (∀ *s P Q zs*. *list*=(*Some* (*Seq P Q*), *s*)#*zs* ⟶
  (∃ *xs*. (*Some P*, *s*)#*xs* ∈ *cpt-p-mod* ∧ (*zs*=(*map* (*lift Q*) *xs*) ∨
  (*fst*(((*Some P*, *s*)#*xs*)!*length xs*)=*None* ∧
  (∃ *ys*. (*Some Q*, *snd*(((*Some P*, *s*)#*xs*)!*length xs*))#*ys* ∈ *cpt-p-mod*
  ∧ *zs*=(*map* (*lift* (*Q*)) *xs*)@*ys*)))))
apply(*erule cpt-p-mod.induct*)
apply *simp-all*
    apply *clarify*
    apply(*force intro*:*CptPModOne*)
   apply *clarify*
   apply(*erule-tac x*=*Pa* **in** *allE*)
   apply(*erule-tac x*=*Q* **in** *allE*)
   apply *simp*
   apply *clarify*
   apply(*erule disjE*)
    apply(*rule-tac x*=(*Some Pa*,*t*)#*xsa* **in** *exI*)
    apply(*rule conjI*)
     apply *clarify*
     apply(*erule CptPModEnv*)
    apply(*rule disjI1*)
    apply(*simp add*:*lift-def*)
   apply *clarify*
   apply(*rule-tac x*=(*Some Pa*,*t*)#*xsa* **in** *exI*)
   apply(*rule conjI*)
    apply(*erule CptPModEnv*)
   apply(*rule disjI2*)
   apply(*rule conjI*)
    apply(*case-tac xsa*,*simp*,*simp*)
   apply(*rule-tac x*=*ys* **in** *exI*)
   apply(*rule conjI*)
    apply *simp*
   apply(*simp add*:*lift-def*)

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply *clarify*
apply(*rule-tac x=xs* **in** *exI*)
apply *simp*
apply *clarify*
apply(*rule-tac x=xs* **in** *exI*)
apply(*simp add*: *last-length*)
done

lemma *cpts-onlyif-cpt-p-mod-aux* [*rule-format*]:
$\forall s\ Q\ t\ xs\ .((Some\ a,\ s),\ (Q,\ t)) \in ptran \longrightarrow (Q,\ t)\ \#\ xs \in cpt\text{-}p\text{-}mod$
$\longrightarrow (Some\ a,\ s)\ \#\ (Q,\ t)\ \#\ xs \in cpt\text{-}p\text{-}mod$
apply(*induct a*)
apply *simp-all*

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*rule Basic*,*simp*)
apply *clarify*
apply(*erule ptran.cases*,*simp-all*)

apply(*rule-tac xs*=[(*None*,*ta*)] **in** *CptPModSeq2*)
apply(*erule CptPModNone*)
apply(*rule CptPModOne*)
apply *simp*
apply *simp*
apply(*simp add*:*lift-def*)

apply(*erule-tac x=sa* **in** *allE*)
apply(*erule-tac x=Some P2* **in** *allE*)
apply(*erule allE*,*erule impE*, *assumption*)
apply(*drule div-seq*,*simp*)
apply *clarify*
apply(*erule disjE*)
apply *clarify*
apply(*erule allE*,*erule impE*, *assumption*)
apply(*erule-tac CptPModSeq1*)
apply(*simp add*:*lift-def*)
apply *clarify*
apply(*erule allE*,*erule impE*, *assumption*)
apply(*erule-tac CptPModSeq2*)
apply (*simp add*:*last-length*)
apply (*simp add*:*last-length*)
apply(*simp add*:*lift-def*)

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)

apply(*force elim*: *CptPModCondT*)
apply(*force elim*: *CptPModCondF*)

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*erule WhileF*,*simp*)
apply(*drule div-seq*,*force*)
apply *clarify*
apply (*erule disjE*)
apply(*force elim*:*CptPModWhile1*)
apply *clarify*
apply(*force simp add*:*last-length elim*:*CptPModWhile2*)

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*erule Await*,*simp*+)

apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*erule Nondt*,*simp*+)
done

lemma *cpts-onlyif-cpt-p-mod* [*rule-format*]: $c \in cpts\text{-}p \Longrightarrow c \in cpt\text{-}p\text{-}mod$
apply(*erule cpts-p.induct*)
apply(*rule CptPModOne*)
apply(*erule CptPModEnv*)
apply(*case-tac P*)
apply *simp*
apply(*erule ptran.cases CptPModEnv*,*simp-all*)
apply(*force elim*:*cpts-onlyif-cpt-p-mod-aux*)
done

lemma *lift-is-cptn*: $c \in cpts\text{-}p \Longrightarrow map\ (lift\ P)\ c \in cpts\text{-}p$
apply(*erule cpts-p.induct*)
apply(*force simp add*:*lift-def CptsPOne*)
apply(*force intro*:*CptsPEnv simp add*:*lift-def*)
apply(*force simp add*:*lift-def intro*:*CptsPComp Seq2 Seq1 elim*:*ptran.cases*)
done

lemma *cptn-append-is-cptn* [*rule-format*]:
$\forall\, b\ a.\ b\#c1 \in cpts\text{-}p \longrightarrow a\#c2 \in cpts\text{-}p \longrightarrow (b\#c1)!length\ c1 = a \longrightarrow b\#c1@c2 \in cpts\text{-}p$
apply(*induct c1*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp-all*)
apply(*force intro*:*CptsPEnv*)
apply(*force elim*:*CptsPComp*)

done

lemma *last-lift*: $[\![xs\neq[]; fst(xs!(length\ xs - (Suc\ 0)))=None]\!]$
$\Longrightarrow fst((map\ (lift\ P)\ xs)!(length\ (map\ (lift\ P)\ xs)-\ (Suc\ 0)))=(Some\ P)$
by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp add*:*lift-def*)

lemma *last-fst* [*rule-format*]: $P((a\#x)!length\ x) \longrightarrow \neg P\ a \longrightarrow P\ (x!(length\ x - (Suc\ 0)))$
by (*induct x*) *simp-all*

lemma *last-fst-esp*:
$fst(((Some\ a,s)\#xs)!(length\ xs))=None \Longrightarrow fst(xs!(length\ xs - (Suc\ 0)))=None$
apply(*erule last-fst*)
apply *simp*
done

lemma *last-snd*: $xs\neq[] \Longrightarrow$
$snd(((map\ (lift\ P)\ xs))!(length\ (map\ (lift\ P)\ xs) - (Suc\ 0)))=snd(xs!(length\ xs - (Suc\ 0)))$
by (*cases* (*xs* ! (*length xs* − (*Suc 0*)))) (*simp-all add*:*lift-def*)

lemma *Cons-lift*: $(Some\ (Seq\ P\ Q),\ s)\ \#\ (map\ (lift\ Q)\ xs) = map\ (lift\ Q)\ ((Some\ P,\ s)\ \#\ xs)$
by (*simp add*:*lift-def*)

lemma *Cons-lift-append*:
$(Some\ (Seq\ P\ Q),\ s)\ \#\ (map\ (lift\ Q)\ xs)\ @\ ys = map\ (lift\ Q)\ ((Some\ P,\ s)\ \#\ xs)@\ ys$
by (*simp add*:*lift-def*)

lemma *lift-nth*: $i<length\ xs \Longrightarrow map\ (lift\ Q)\ xs\ !\ i = lift\ Q\ \ (xs!\ i)$
by (*simp add*:*lift-def*)

lemma *snd-lift*: $i<\ length\ xs \Longrightarrow snd(lift\ Q\ (xs\ !\ i))=\ snd\ (xs\ !\ i)$
by (*cases xs*!*i*) (*simp add*:*lift-def*)

lemma *cpts-if-cpt-p-mod*: $c \in cpt\text{-}p\text{-}mod \Longrightarrow c \in cpts\text{-}p$
apply(*erule cpt-p-mod.induct*)
apply(*rule CptsPOne*)
apply(*erule CptsPEnv*)
apply(*erule CptsPComp*,*simp*)
apply(*rule CptsPComp*)
apply(*erule CondT*,*simp*)
apply(*rule CptsPComp*)
apply(*erule CondF*,*simp*)
— Seq1
apply(*erule cpts-p.cases*,*simp-all*)
apply(*rule CptsPOne*)
apply *clarify*

```
 apply(drule-tac P=P1 in lift-is-cptn)
 apply(simp add:lift-def)
 apply(rule CptsPEnv,simp)
apply clarify
apply(simp add:lift-def)
apply(rule conjI)
 apply clarify
 apply(rule CptsPComp)
  apply(rule Seq1,simp)
 apply(drule-tac P=P1 in lift-is-cptn)
 apply(simp add:lift-def)
apply clarify
apply(rule CptsPComp)
 apply(rule Seq2,simp)
apply(drule-tac P=P1 in lift-is-cptn)
apply(simp add:lift-def)
— Seq2
apply(rule cptn-append-is-cptn)
  apply(drule-tac P=P1 in lift-is-cptn)
  apply(simp add:lift-def)
 apply simp
apply(simp split: if-split-asm)
apply(frule-tac P=P1 in last-lift)
 apply(rule last-fst-esp)
 apply (simp add:last-length)
apply(simp add:Cons-lift lift-def split-def last-conv-nth)
— While1
apply(rule CptsPComp)
 apply(rule WhileT,simp)
apply(drule-tac P=While b P in lift-is-cptn)
apply(simp add:lift-def)
— While2
apply(rule CptsPComp)
 apply(rule WhileT,simp)
apply(rule cptn-append-is-cptn)
  apply(drule-tac P=While b P in lift-is-cptn)
  apply(simp add:lift-def)
 apply simp
apply(simp split: if-split-asm)
apply(frule-tac P=While b P in last-lift)
 apply(rule last-fst-esp,simp add:last-length)
apply(simp add:Cons-lift lift-def split-def last-conv-nth)
  done

theorem cpts-iff-cpt-p-mod: (c ∈ cpts-p) = (c ∈ cpt-p-mod)
apply(rule iffI)
 apply(erule cpts-onlyif-cpt-p-mod)
apply(erule cpts-if-cpt-p-mod)
done
```

end

13 Rely-guarantee Validity of SIMP language

```
theory SIMP-validity
imports SIMP-computation
begin

type-synonym 's pconfs = 's imp-pconf list

definition gets-p :: 's imp-pconf ⇒ 's where
  gets-p conf ≡ snd conf

definition getspc-p :: 's imp-pconf ⇒ ('s prog) option where
  getspc-p conf ≡ fst conf

definition assume-p :: ('s set × ('s × 's) set) ⇒ ('s pconfs) set where
  assume-p ≡ λ(pre, rely). {c. gets-p (c!0) ∈ pre ∧ (∀ i. Suc i<length c ⟶
             c!i −imppe→ c!(Suc i) ⟶ (gets-p (c!i), gets-p (c!Suc i)) ∈ rely)}

definition commit-p :: (('s × 's) set × 's set) ⇒ ('s pconfs) set where
  commit-p ≡ λ(guar, post). {c. (∀ i. Suc i<length c ⟶
             c!i −impc→ c!(Suc i) ⟶ (gets-p (c!i), gets-p (c!Suc i)) ∈ guar) ∧
             (getspc-p (last c) = None ⟶ gets-p (last c) ∈ post)}

definition prog-validity :: 's prog option ⇒ 's set ⇒ ('s × 's) set ⇒ ('s × 's) set
⇒ 's set ⇒ bool
                 (⊨_I - sat_p [-, -, -, -] [60,0,0,0,0] 45) where
  ⊨_I P sat_p [pre, rely, guar, post] ≡
   ∀ s. cpts-of-p P s ∩ assume-p(pre, rely) ⊆ commit-p(guar, post)


lemma assume-p-imp: ⟦pre1⊆pre; rely1⊆rely; c∈assume-p(pre1,rely1)⟧ ⟹ c∈assume-p(pre,rely)
  proof −
    assume p0: pre1⊆pre
      and  p1: rely1⊆rely
      and  p3: c∈assume-p(pre1,rely1)
    then have a0: gets-p (c!0) ∈ pre1 ∧ (∀ i. Suc i<length c ⟶
             c!i −imppe→ c!(Suc i) ⟶ (gets-p (c!i), gets-p (c!Suc i)) ∈ rely1)
      by (simp add:assume-p-def)
    show ?thesis
      proof(simp add:assume-p-def,rule conjI)
        from p0 a0 show gets-p (c ! 0) ∈ pre by auto
      next
        from p1 a0 show ∀ i. Suc i < length c ⟶ c ! i −imppe→ c ! Suc i
                           ⟶ (gets-p (c ! i), gets-p (c ! Suc i)) ∈ rely
          by auto
      qed
```

qed

lemma *commit-p-imp*: $[\![guar1 \subseteq guar;\ post1 \subseteq post;\ c \in commit\text{-}p(guar1,post1)]\!] \Longrightarrow c \in commit\text{-}p(guar,post)$

proof −

assume *p0*: $guar1 \subseteq guar$

and *p1*: $post1 \subseteq post$

and *p3*: $c \in commit\text{-}p(guar1,post1)$

then have *a0*: $(\forall i.\ Suc\ i < length\ c \longrightarrow c!i\ {-impc\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar1) \wedge (getspc\text{-}p\ (last\ c) = None \longrightarrow gets\text{-}p\ (last\ c) \in post1)$

by (*simp add*:*commit-p-def*)

show *?thesis*

proof(*simp add*:*commit-p-def*)

from *p0 p1 a0* **show** $(\forall i.\ Suc\ i < length\ c \longrightarrow c!i\ {-impc\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar) \wedge (getspc\text{-}p\ (last\ c) = None \longrightarrow gets\text{-}p\ (last\ c) \in post)$

by *auto*

qed

qed

end

14 Rely-guarantee Proof Rules of SIMP language

theory *SIMP-hoare*
imports *SIMP-validity*
begin

declare *Un-subset-iff* [*simp del*] *sup.bounded-iff* [*simp del*]

definition *stable* :: $'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow bool$ **where**
$stable \equiv \lambda f\ g.\ (\forall x\ y.\ x \in f \longrightarrow (x,\ y) \in g \longrightarrow y \in f)$

inductive *rghoare-p* :: $['s\ prog\ option,\ 's\ set,\ ('s \times 's)\ set,\ ('s \times 's)\ set,\ 's\ set] \Rightarrow bool$
$(\vdash_I\ \text{-}\ sat_p\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$
where
Basic: $[\![\ pre \subseteq \{s.\ f\ s \in post\};\ \{(s,t).\ s \in pre \wedge (t = f\ s)\} \subseteq guar;\ stable\ pre\ rely;\ stable\ post\ rely\ ]\!] \Longrightarrow \vdash_I Some\ (Basic\ f)\ sat_p\ [pre,\ rely,\ guar,\ post]$

| *Seq*: $[\![\ \vdash_I Some\ P\ sat_p\ [pre,\ rely,\ guar,\ mid];\ \vdash_I Some\ Q\ sat_p\ [mid,\ rely,\ guar,\ post]\ ]\!] \Longrightarrow \vdash_I Some\ (Seq\ P\ Q)\ sat_p\ [pre,\ rely,\ guar,\ post]$

| *Cond*: $[\![\ stable\ pre\ rely;\ \vdash_I Some\ P1\ sat_p\ [pre \cap b,\ rely,\ guar,\ post];$

$\vdash_I$ *Some P2 sat*$_p$ [*pre* $\cap$ $-b$, *rely*, *guar*, *post*]; $\forall$ *s*. (*s*,*s*)$\in$*guar* $]\!]$
$\Longrightarrow$ $\vdash_I$ *Some* (*Cond b P1 P2*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *While*: $[\![$ *stable pre rely*; (*pre* $\cap$ $-b$) $\subseteq$ *post*; *stable post rely*;
$\vdash_I$ *Some P sat*$_p$ [*pre* $\cap$ *b*, *rely*, *guar*, *pre*]; $\forall$ *s*. (*s*,*s*)$\in$*guar* $]\!]$
$\Longrightarrow$ $\vdash_I$ *Some* (*While b P*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *Await*: $[\![$ *stable pre rely*; *stable post rely*;
$\forall$ *V*. $\vdash_I$ *Some P sat*$_p$ [*pre* $\cap$ *b* $\cap$ $\{V\}$, $\{(s, t).\ s = t\}$,
UNIV, $\{s.\ (V, s) \in$ *guar*$\}$ $\cap$ *post*] $]\!]$
$\Longrightarrow$ $\vdash_I$ *Some* (*Await b P*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *Nondt*: $[\![$ *pre* $\subseteq$ $\{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in$ *post*$) \wedge (\exists t.\ (s,t) \in r)\}$; $\{(s,t).\ s \in$ *pre* $\wedge$ $(s,t)\in r\}$ $\subseteq$ *guar*;
stable pre rely; *stable post rely* $]\!]$
$\Longrightarrow$ $\vdash_I$ *Some* (*Nondt r*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *None-hoare*: $[\![$ *stable pre rely*; *pre* $\subseteq$ *post* $]\!]$
$\Longrightarrow$ $\vdash_I$ *None sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *Conseq*: $[\![$ *pre* $\subseteq$ *pre'*; *rely* $\subseteq$ *rely'*; *guar'* $\subseteq$ *guar*; *post'* $\subseteq$ *post*;
$\vdash_I$ *P sat*$_p$ [*pre'*, *rely'*, *guar'*, *post'*] $]\!]$
$\Longrightarrow$ $\vdash_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post*]

| *Unprecond*: $[\![$ $\vdash_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post*]; $\vdash_I$ *P sat*$_p$ [*pre'*, *rely*, *guar*, *post*] $]\!]$
$\Longrightarrow$ $\vdash_I$ *P sat*$_p$ [*pre* $\cup$ *pre'*, *rely*, *guar*, *post*]

| *Intpostcond*: $[\![$ $\vdash_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post*]; $\vdash_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post'*]
$]\!]$
$\Longrightarrow$ $\vdash_I$ *P sat*$_p$ [*pre*, *rely*, *guar*, *post* $\cap$ *post'*]

| *Allprecond*: $\forall$ *v*$\in$*U*. $\vdash_I$ *P sat*$_p$ [$\{v\}$, *rely*, *guar*, *post*]
$\Longrightarrow$ $\vdash_I$ *P sat*$_p$ [*U*, *rely*, *guar*, *post*]

| *Emptyprecond*: $\vdash_I$ *P sat*$_p$ [$\{\}$, *rely*, *guar*, *post*]

lemma *Id* = $\{(s, t).\ s = t\}$
by *auto*

lemma *Seq2*: $[\![$ $\vdash_I$ *Some P sat*$_p$ [*pre*, *rely*, *guar*, *mida*]; *mida* $\subseteq$ *midb*; $\vdash_I$ *Some Q sat*$_p$ [*midb*, *rely*, *guar*, *post*] $]\!]$
$\Longrightarrow$ $\vdash_I$ *Some* (*Seq P Q*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]
using *Seq*[*of P pre rely guar mida Q post*]
Conseq[*of mida midb rely rely guar guar post post*]
by *blast*

end

15 The Soundness of RG Rules of SIMP language

theory *SIMP-sound*
imports *SIMP-hoare*
begin

15.1 Some previous lemmas

15.1.1 program

lemma *tl-of-assum-in-assum*:
 $(P,\ s)\ \#\ (P,\ t)\ \#\ xs \in \mathit{assume\text{-}p}\ (\mathit{pre},\ \mathit{rely}) \Longrightarrow \mathit{stable\ pre\ rely}$
 $\Longrightarrow (P,\ t)\ \#\ xs \in \mathit{assume\text{-}p}\ (\mathit{pre},\ \mathit{rely})$
apply(*simp add*:*assume-p-def*)
apply *clarify*
apply(*rule conjI*)
 apply(*erule-tac x=0* **in** *allE*)
 apply(*simp* (*no-asm-use*)*only*:*stable-def*)
 apply(*erule allE*,*erule allE*,*erule impE*,*assumption*,*erule mp*)
 apply(*simp add*:*EnvP*)
apply(*simp add*:*getspc-p-def gets-p-def*)
apply *clarify*
apply (*fastforce*)
done

lemma *etran-in-comm*:
 $(P,\ t)\ \#\ xs \in \mathit{commit\text{-}p}(\mathit{guar},\ \mathit{post}) \Longrightarrow (P,\ s)\ \#\ (P,\ t)\ \#\ xs \in \mathit{commit\text{-}p}(\mathit{guar},$
$\mathit{post})$
apply(*simp add*:*commit-p-def*)
apply(*simp add*:*getspc-p-def gets-p-def*)
apply *clarify*
apply(*case-tac i*,*fastforce+*)
done

lemma *ctran-in-comm*:
 $[\![(s,\ s) \in \mathit{guar};\ (Q,\ s)\ \#\ xs \in \mathit{commit\text{-}p}(\mathit{guar},\ \mathit{post})]\!]$
 $\Longrightarrow (P,\ s)\ \#\ (Q,\ s)\ \#\ xs \in \mathit{commit\text{-}p}(\mathit{guar},\ \mathit{post})$
apply(*simp add*:*commit-p-def*)
apply(*simp add*:*getspc-p-def gets-p-def*)
apply *clarify*
apply(*case-tac i*,*fastforce+*)
done

lemma *takecptn-is-cptn* [*rule-format*, *elim*!]:
 $\forall j.\ c \in \mathit{cpts\text{-}p} \longrightarrow \mathit{take}\ (\mathit{Suc}\ j)\ c \in \mathit{cpts\text{-}p}$
apply(*induct c*)
 apply(*force elim*: *cpts-p.cases*)
apply *clarify*
apply(*case-tac j*)
 apply *simp*

apply(*rule CptsPOne*)
apply *simp*
apply(*force intro*:*cpts-p.intros elim*:*cpts-p.cases*)
done

lemma *dropcptn-is-cptn* [*rule-format*,*elim*!]:
$\forall j$<*length c*. $c \in$ *cpts-p* $\longrightarrow$ *drop j c* $\in$ *cpts-p*
apply(*induct c*)
apply(*force elim*: *cpts-p.cases*)
apply *clarify*
apply(*case-tac j*,*simp*+)
apply(*erule cpts-p.cases*)
apply *simp*
apply *force*
apply *force*
done

lemma *tl-of-cptn-is-cptn*: $[\![x$ # $xs \in$ *cpts-p*; $xs \neq [\,]]\!] \Longrightarrow xs \in$ *cpts-p*
apply(*subgoal-tac* $1 <$ *length* (x # xs))
apply(*drule dropcptn-is-cptn*,*simp*+)
done

lemma *not-ctran-None* [*rule-format*]:
$\forall s$. (*None*, s)#$xs \in$ *cpts-p* $\longrightarrow$ ($\forall i$<*length xs*. ((*None*, s)#xs)!i $-$*imppe*$\rightarrow$ xs!i)
apply(*induct xs*,*simp*+)
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply *simp*
apply(*case-tac i*,*simp*)
apply(*rule EnvP*)
apply *simp*
apply(*force elim*:*ptran.cases*)
done

lemma *cptn-not-empty* [*simp*]:[] $\notin$ *cpts-p*
apply(*force elim*:*cpts-p.cases*)
done

lemma *etran-or-ctran* [*rule-format*]:
$\forall m\ i$. $x \in$*cpts-p* $\longrightarrow m \leq$ *length x*
$\longrightarrow$ ($\forall i$. *Suc i* $< m \longrightarrow \neg$ x!i $-$*impc*$\rightarrow$ x!*Suc i*) $\longrightarrow$ *Suc i* $< m$
$\longrightarrow$ x!i $-$*imppe*$\rightarrow$ x!*Suc i*
apply(*induct x*,*simp*)
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply(*case-tac i*,*simp*)
apply(*rule EnvP*)
apply *simp*
apply(*erule-tac* x=$m - 1$ **in** *allE*)

apply(*case-tac m*,*simp*,*simp*)
apply(*subgoal-tac* (∀ *i*. *Suc i* < *nata* ⟶ (((*P*, *t*) # *xs*) ! *i*, *xs* ! *i*) ∉ *ptran*))
apply *force*
apply *clarify*
apply(*erule-tac x*=*Suc ia* **in** *allE*,*simp*)
apply(*erule-tac x*=*0* **and** *P*=λ*j*. *H j* ⟶ (*J j*) ∉ *ptran* **for** *H J* **in** *allE*,*simp*)
done

lemma *etran-or-ctran2* [*rule-format*]:
∀ *i*. *Suc i*<*length x* ⟶ *x*∈*cpts-p* ⟶ (*x*!*i* −*impc*→ *x*!*Suc i* ⟶ ¬ *x*!*i* −*imppe*→ *x*!*Suc i*)
∨ (*x*!*i* −*imppe*→ *x*!*Suc i* ⟶ ¬ *x*!*i* −*impc*→ *x*!*Suc i*)
apply(*induct x*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply(*case-tac i*,*simp*+)
apply(*case-tac i*,*simp*)
apply(*force elim*:*petran.cases*)
apply *simp*
done

lemma *etran-or-ctran2-disjI1*:
⟦ *x*∈*cpts-p*; *Suc i*<*length x*; *x*!*i* −*impc*→ *x*!*Suc i*⟧ ⟹ ¬ *x*!*i* −*imppe*→ *x*!*Suc i*
by(*drule etran-or-ctran2*,*simp-all*)

lemma *etran-or-ctran2-disjI3*:
⟦ *x*∈*cpts-p*; *Suc i*<*length x*; ¬ *x*!*i* −*impc*→ *x*!*Suc i*⟧ ⟹ *x*!*i* −*imppe*→ *x*!*Suc i*
apply(*induct x arbitrary*:*i*)
apply *simp*
apply *clarify*
apply(*rule cpts-p.cases*)
apply *simp*+
using *less-Suc-eq-0-disj petran.intros* **apply** *force*
apply(*case-tac i*,*simp*)
by *simp*

lemma *etran-or-ctran2-disjI2*:
⟦ *x*∈*cpts-p*; *Suc i*<*length x*; *x*!*i* −*imppe*→ *x*!*Suc i*⟧ ⟹ ¬ *x*!*i* −*impc*→ *x*!*Suc i*
by(*drule etran-or-ctran2*,*simp-all*)

lemma *not-ctran-None2* [*rule-format*]:
⟦ (*None*, *s*) # *xs* ∈*cpts-p*; *i*<*length xs*⟧ ⟹ ¬ ((*None*, *s*) # *xs*) ! *i* −*impc*→ *xs* ! *i*
apply(*frule not-ctran-None*,*simp*)
apply(*case-tac i*,*simp*)
apply(*force elim*:*petranE*)
apply *simp*
apply(*rule etran-or-ctran2-disjI2*,*simp-all*)

apply(*force intro*:*tl-of-cptn-is-cptn*)
done

lemma *Ex-first-occurrence* [*rule-format*]: P (n::*nat*) $\longrightarrow$ ($\exists\, m.\ P\ m \wedge (\forall\, i{<}m.\ \neg$ $P\ i$))
apply(*rule nat-less-induct*)
apply *clarify*
apply(*case-tac* $\forall\, m.\ m{<}n \longrightarrow \neg\ P\ m$)
apply *auto*
done

lemma *stability* [*rule-format*]:
$\forall\, j\ k.\ x \in$ *cpts-p* $\longrightarrow$ *stable p rely* $\longrightarrow j{\le}k \longrightarrow k{<}$*length* $x \longrightarrow$ *snd*($x!j$)$\in p$ $\longrightarrow$
($\forall\, i.$ (*Suc* i)$<$*length* x $\longrightarrow$
($x!i$ $-$*imppe*$\rightarrow$ $x!$(*Suc* i)) $\longrightarrow$ (*snd*($x!i$), *snd*($x!$(*Suc* i))) $\in$ *rely*) $\longrightarrow$
($\forall\, i.\ j{\le}i \wedge i{<}k \longrightarrow x!i$ $-$*imppe*$\rightarrow$ $x!$*Suc* i) $\longrightarrow$ *snd*($x!k$)$\in p$ $\wedge$ *fst*($x!j$)=*fst*($x!k$)
apply(*induct* x)
apply *clarify*
apply(*force elim*:*cpts-p.cases*)
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply *simp*
apply(*case-tac* k,*simp*,*simp*)
apply(*case-tac* j,*simp*)
apply(*erule-tac* x=*0* **in** *allE*)
apply(*erule-tac* x=*nat* **and** P=$\lambda j.$ ($0{\le}j$) $\longrightarrow$ ($J\ j$) **for** J **in** *allE*,*simp*)
apply(*subgoal-tac* $t{\in}p$)
apply(*subgoal-tac* ($\forall\, i.\ i <$ *length xs* $\longrightarrow$ ((P, t) # *xs*) ! i $-$*imppe*$\rightarrow$ *xs* ! i $\longrightarrow$ (*snd* (((P, t) # *xs*) ! i), *snd* (*xs* ! i)) $\in$ *rely*))
apply *clarify*
apply(*erule-tac* x=*Suc* i **and** P=$\lambda j.$ ($H\ j$) $\longrightarrow$ ($J\ j$)$\in$*petran* **for** H J **in** *allE*,*simp*)
apply *clarify*
apply(*erule-tac* x=*Suc* i **and** P=$\lambda j.$ ($H\ j$) $\longrightarrow$ ($J\ j$) $\longrightarrow$ ($T\ j$)$\in$*rely* **for** H J T **in** *allE*,*simp*)
apply(*erule-tac* x=*0* **and** P=$\lambda j.$ ($H\ j$) $\longrightarrow$ ($J\ j$)$\in$*petran* $\longrightarrow$ $T\ j$ **for** H J T **in** *allE*,*simp*)
apply(*simp*(*no-asm-use*) *only*:*stable-def*)
apply(*erule-tac* x=s **in** *allE*)
apply(*erule-tac* x=t **in** *allE*)
apply *simp*
apply(*erule mp*)
apply(*erule mp*)
apply(*rule EnvP*)
apply *simp*
apply(*erule-tac* x=*nata* **in** *allE*)
apply(*erule-tac* x=*nat* **and** P=$\lambda j.$ ($s{\le}j$) $\longrightarrow$ ($J\ j$) **for** s J **in** *allE*,*simp*)
apply(*subgoal-tac* ($\forall\, i.\ i <$ *length xs* $\longrightarrow$ ((P, t) # *xs*) ! i $-$*imppe*$\rightarrow$ *xs* ! i $\longrightarrow$ (*snd* (((P, t) # *xs*) ! i), *snd* (*xs* ! i)) $\in$ *rely*))

apply *clarify*
apply(*erule-tac* x=*Suc* i **and** $P=\lambda j.$ (H j) $\longrightarrow$ (J j)$\in$*petran* **for** H J **in** *allE*,*simp*)
apply *clarify*
apply(*erule-tac* x=*Suc* i **and** $P=\lambda j.$ (H j) $\longrightarrow$ (J j) $\longrightarrow$ (T j)$\in$*rely* **for** H J T **in** *allE*,*simp*)
apply(*case-tac* k,*simp*,*simp*)
apply(*case-tac* j)
apply(*erule-tac* x=*0* **and** $P=\lambda j.$ (H j) $\longrightarrow$ (J j)$\in$*petran* **for** H J **in** *allE*,*simp*)
apply(*erule petran.cases*,*simp*)
apply(*erule-tac* x=*nata* **in** *allE*)
apply(*erule-tac* x=*nat* **and** $P=\lambda j.$ ($s\leq j$) $\longrightarrow$ (J j) **for** s J **in** *allE*,*simp*)
apply(*subgoal-tac* ($\forall i.$ i < *length* xs $\longrightarrow$ ((Q, t) # xs) ! i −*imppe*→ xs ! i $\longrightarrow$ (*snd* (((Q, t) # xs) ! i), *snd* (xs ! i)) $\in$ *rely*))
apply *clarify*
apply(*erule-tac* x=*Suc* i **and** $P=\lambda j.$ (H j) $\longrightarrow$ (J j)$\in$*petran* **for** H J **in** *allE*,*simp*)
apply *clarify*
apply(*erule-tac* x=*Suc* i **and** $P=\lambda j.$ (H j) $\longrightarrow$ (J j) $\longrightarrow$ (T j)$\in$*rely* **for** H J T **in** *allE*,*simp*)
done

15.2 Soundness of Programs

15.2.1 Soundness of the Basic rule

lemma *unique-ctran-Basic* [*rule-format*]:
$\forall s$ $i.$ $x \in$ *cpts-p* $\longrightarrow$ x ! *0* = (*Some* (*Basic* f), s) $\longrightarrow$
Suc i<*length* x $\longrightarrow$ x!i −*impc*→ x!*Suc* i $\longrightarrow$
($\forall j.$ *Suc* j<*length* x $\longrightarrow$ $i\neq j$ $\longrightarrow$ x!j −*imppe*→ x!*Suc* j)
apply(*induct* x,*simp*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply(*case-tac* i,*simp*+)
apply *clarify*
apply(*case-tac* j,*simp*)
apply(*rule EnvP*)
apply *simp*
apply *clarify*
apply *simp*
apply(*case-tac* i)
apply(*case-tac* j,*simp*,*simp*)
apply(*erule ptran.cases*,*simp-all*)
apply(*force elim*: *not-ctran-None*)
apply(*ind-cases* ((*Some* (*Basic* f), *sa*), Q, t) $\in$ *ptran* **for** *sa* Q t)
apply *simp*
apply(*drule-tac* i=*nat* **in** *not-ctran-None*,*simp*)
apply(*erule petranE*,*simp*)
done

lemma *exists-ctran-Basic-None* [*rule-format*]:

$\forall s\ i.\ x \in \textit{cpts-p} \longrightarrow x\ !\ 0 = (\textit{Some}\ (\textit{Basic}\ f),\ s)$
$\longrightarrow i{<}\textit{length}\ x \longrightarrow \textit{fst}(x!i){=}\textit{None} \longrightarrow (\exists j{<}i.\ x!j\ {-}\textit{impc}{\rightarrow}\ x!\textit{Suc}\ j)$
apply(*induct x*,*simp*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply(*case-tac i*,*simp*,*simp*)
apply(*erule-tac x=nat* **in** *allE*,*simp*)
apply *clarify*
apply(*rule-tac x=Suc j* **in** *exI*,*simp*,*simp*)
apply *clarify*
apply(*case-tac i*,*simp*,*simp*)
apply(*rule-tac x=0* **in** *exI*,*simp*)
done

lemma *Basic-sound*:
$[\![\textit{pre} \subseteq \{s.\ f\ s \in \textit{post}\};\ \{(s,\ t).\ s \in \textit{pre} \wedge t = f\ s\} \subseteq \textit{guar};$
$\textit{stable pre rely};\ \textit{stable post rely}]\!]$
$\Longrightarrow \models_I \textit{Some}\ (\textit{Basic}\ f)\ \textit{sat}_p\ [\textit{pre},\ \textit{rely},\ \textit{guar},\ \textit{post}]$
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*simp add:commit-p-def*)
apply(*simp add:getspc-p-def gets-p-def*)
apply(*rule conjI*)
apply *clarify*
apply(*simp add:cpts-of-p-def assume-p-def gets-p-def*)
apply *clarify*
apply(*frule-tac j=0* **and** *k=i* **and** *p=pre* **in** *stability*)
apply *simp-all*
apply(*erule-tac x=ia* **in** *allE*,*simp*)
apply(*erule-tac i=i* **and** *f=f* **in** *unique-ctran-Basic*,*simp-all*)
apply(*erule subsetD*,*simp*)
apply(*case-tac x!i*)
apply *clarify*
apply(*drule-tac s=Some (Basic f)* **in** *sym*,*simp*)
apply(*thin-tac* $\forall j.\ H\ j$ **for** *H*)
apply(*force elim:ptran.cases*)
apply *clarify*
apply(*simp add:cpts-of-p-def*)
apply *clarify*
apply(*frule-tac i=length x − 1* **and** *f=f* **in** *exists-ctran-Basic-None*,*simp+*)
apply(*case-tac x*,*simp+*)
apply(*rule last-fst-esp*,*simp add:last-length*)
apply (*case-tac x*,*simp+*)
apply(*simp add:assume-p-def gets-p-def*)
apply *clarify*
apply(*frule-tac j=0* **and** *k=j* **and** *p=pre* **in** *stability*)
apply *simp-all*
apply(*erule-tac x=i* **in** *allE*,*simp*)

apply(*erule-tac i=j* **and** *f=f* **in** *unique-ctran-Basic*,*simp-all*)
apply(*case-tac x*!*j*)
apply *clarify*
apply *simp*
apply(*drule-tac s=Some* (*Basic f*) **in** *sym*,*simp*)
apply(*case-tac x*!*Suc j*,*simp*)
apply(*rule ptran.cases*,*simp*)
apply(*simp-all*)
apply(*drule-tac c=sa* **in** *subsetD*,*simp*)
apply *clarify*
apply(*frule-tac j=Suc j* **and** *k=length x* − *1* **and** *p=post* **in** *stability*,*simp-all*)
apply(*case-tac x*,*simp*+)
apply(*erule-tac x=i* **in** *allE*)
apply(*erule-tac i=j* **and** *f=f* **in** *unique-ctran-Basic*,*simp-all*)
apply *arith*+
apply(*case-tac x*)
apply(*simp add*:*last-length*)+
done

15.2.2 Soundness of the Await rule

lemma *unique-ctran-Await* [*rule-format*]:
$\forall s\ i.\ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s) \longrightarrow$
$Suc\ i{<}length\ x \longrightarrow x!i\ {-impc\rightarrow}\ x!Suc\ i \longrightarrow$
$(\forall j.\ Suc\ j{<}length\ x \longrightarrow i{\neq}j \longrightarrow x!j\ {-imppe\rightarrow}\ x!Suc\ j)$
apply(*induct x*,*simp*+)
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
apply(*case-tac i*,*simp*+)
apply *clarify*
apply(*case-tac j*,*simp*)
apply(*rule EnvP*)
apply *simp*
apply *clarify*
apply *simp*
apply(*case-tac i*)
apply(*case-tac j*,*simp*,*simp*)
apply(*erule ptran.cases*,*simp-all*)
apply(*force elim*: *not-ctran-None*)
apply(*ind-cases* ((*Some* (*Await b c*), *sa*), *Q*, *t*) ∈ *ptran* **for** *sa Q t*,*simp*)
apply(*drule-tac i=nat* **in** *not-ctran-None*,*simp*)
apply(*erule petranE*,*simp*)
done

lemma *exists-ctran-Await-None* [*rule-format*]:
$\forall s\ i.\ \ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s)$
$\longrightarrow i{<}length\ x \longrightarrow fst(x!i){=}None \longrightarrow (\exists j{<}i.\ x!j\ {-impc\rightarrow}\ x!Suc\ j)$
apply(*induct x*,*simp*+)
apply *clarify*

apply(*erule cpts-p.cases*,*simp*)
 apply(*case-tac i*,*simp+*)
 apply(*erule-tac x=nat* **in** *allE*,*simp*)
 apply *clarify*
 apply(*rule-tac x=Suc j* **in** *exI*,*simp*,*simp*)
apply *clarify*
apply(*case-tac i*,*simp*,*simp*)
apply(*rule-tac x=0* **in** *exI*,*simp*)
done

lemma *Star-imp-cptn*:
  $(P,\ s)\ -impc{*}\!\to\ (R,\ t) \Longrightarrow \exists\, l \in cpts\text{-}of\text{-}p\ P\ s.\ (last\ l)=(R,\ t)$
  $\wedge\ (\forall\, i.\ Suc\ i<length\ l \longrightarrow l!i\ -impc\!\to\ l!Suc\ i)$
apply (*erule converse-rtrancl-induct2*)
 apply(*rule-tac x=*[(*R*,*t*)] **in** *bexI*)
  apply *simp*
 apply(*simp add*:*cpts-of-p-def*)
 apply(*rule CptsPOne*)
apply *clarify*
apply(*rule-tac x=*(*a*, *b*)#*l* **in** *bexI*)
 apply (*rule conjI*)
  apply(*case-tac l*,*simp add*:*cpts-of-p-def*)
  apply(*simp add*:*last-length*)
 apply *clarify*
apply(*case-tac i*,*simp*)
apply(*simp add*:*cpts-of-p-def*)
apply *force*
apply(*simp add*:*cpts-of-p-def*)
 apply(*case-tac l*)
 apply(*force elim*:*cpts-p.cases*)
apply *simp*
apply(*erule CptsPComp*)
apply *clarify*
done

lemma *Await-sound*:
  $[\![stable\ pre\ rely;\ stable\ post\ rely;$
  $\forall\, V.\ \vdash_I\ Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
  $\qquad UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post] \ \wedge$
  $\models_I\ Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
  $\qquad UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post]\ ]\!]$
  $\Longrightarrow\ \models_I\ Some\ (Await\ b\ P)\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*simp add*:*commit-p-def*)
apply(*rule conjI*)
 apply *clarify*
 apply(*simp add*:*cpts-of-p-def assume-p-def gets-p-def getspc-p-def*)
 apply *clarify*

```
 apply(frule-tac j=0 and k=i and p=pre in stability,simp-all)
   apply(erule-tac x=ia in allE,simp)
  apply(subgoal-tac x∈ cpts-of-p (Some(Await b P)) s)
  apply(erule-tac i=i in unique-ctran-Await,force,simp-all)
  apply(simp add:cpts-of-p-def)

 apply(erule ptran.cases,simp-all)
 apply(drule Star-imp-cptn)
 apply clarify
 apply(erule-tac x=sa in allE)
 apply clarify
 apply(erule-tac x=sa in allE)
 apply(drule-tac c=l in subsetD)
  apply (simp add:cpts-of-p-def)
  apply clarify
 apply(erule-tac x=ia and P=λi. H i ⟶ (J i, I i)∈ptran for H J I in allE,simp)
  apply(erule petranE,simp)
 apply simp
apply clarify
apply (simp add:gets-p-def getspc-p-def)
apply(simp add:cpts-of-p-def)
apply clarify
apply(frule-tac i=length x − 1 in exists-ctran-Await-None,force)
  apply (case-tac x,simp+)
 apply(rule last-fst-esp,simp add:last-length)
 apply(case-tac x, simp+)
apply clarify
apply(simp add:assume-p-def gets-p-def getspc-p-def)
apply clarify
apply(frule-tac j=0 and k=j and p=pre in stability,simp-all)
  apply(erule-tac x=i in allE,simp)
 apply(erule-tac i=j in unique-ctran-Await,force,simp-all)
apply(case-tac x!j)
apply clarify
apply simp
apply(drule-tac s=Some (Await b P) in sym,simp)
apply(case-tac x!Suc j,simp)
apply(rule ptran.cases,simp)
apply(simp-all)
apply(drule Star-imp-cptn)
apply clarify
apply(erule-tac x=sa in allE)
apply clarify
apply(erule-tac x=sa in allE)
apply(drule-tac c=l in subsetD)
 apply (simp add:cpts-of-p-def)
 apply clarify
 apply(erule-tac x=i and P=λi. H i ⟶ (J i, I i)∈ptran for H J I in allE,simp)
 apply(erule petranE,simp)
```

apply *simp*
apply *clarify*
apply(*frule-tac j=Suc j* **and** *k=length x − 1* **and** *p=post* **in** *stability*,*simp-all*)
 apply(*case-tac x*,*simp+*)
 apply(*erule-tac x=i* **in** *allE*)
apply(*erule-tac i=j* **in** *unique-ctran-Await*,*force*,*simp-all*)
 apply *arith+*
apply(*case-tac x*)
apply(*simp add*:*last-length*)+
done

15.2.3 Soundness of the Conditional rule

lemma *Cond-sound*:
  $[\![$ *stable pre rely*; $\models_I$ *Some P1 sat*$_p$ [*pre* ∩ *b*, *rely*, *guar*, *post*];
  $\models_I$ *Some P2 sat*$_p$ [*pre* ∩ − *b*, *rely*, *guar*, *post*]; ∀ *s*. (*s*,*s*)∈*guar*$]\!]$
  $\Longrightarrow$ $\models_I$ *Some* (*Cond b P1 P2*) *sat*$_p$ [*pre*, *rely*, *guar*, *post*]
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*simp add*:*cpts-of-p-def commit-p-def*)
apply(*simp add*:*getspc-p-def gets-p-def*)
apply(*case-tac* ∃ *i*. *Suc i*<*length x* ∧ *x*!*i* −*impc*→ *x*!*Suc i*)
 prefer *2*
 apply *simp*
 apply *clarify*
 apply(*frule-tac j=0* **and** *k=length x − 1* **and** *p=pre* **in** *stability*,*simp+*)
     apply(*case-tac x*,*simp+*)
    apply(*simp add*:*assume-p-def gets-p-def*)
   apply(*simp add*:*assume-p-def gets-p-def*)
  apply(*erule-tac m=length x* **in** *etran-or-ctran*,*simp+*)
 apply(*case-tac x*, (*simp add*:*last-length*)+)
apply(*erule exE*)
apply(*drule-tac n=i* **and** *P*=λ*i*. *H i* ∧ (*J i*, *I i*) ∈ *ptran* **for** *H J I* **in** *Ex-first-occurrence*)
apply *clarify*
apply (*simp add*:*assume-p-def gets-p-def*)
apply(*frule-tac j=0* **and** *k=m* **and** *p=pre* **in** *stability*,*simp+*)
 apply(*erule-tac m=Suc m* **in** *etran-or-ctran*,*simp+*)
apply(*erule ptran.cases*,*simp-all*)
 apply(*erule-tac x=sa* **in** *allE*)
 apply(*drule-tac c=drop* (*Suc m*) *x* **in** *subsetD*)
  apply *simp*
  apply *clarify*
 apply *simp*
 apply *clarify*
 apply(*case-tac i*≤*m*)
  apply(*drule le-imp-less-or-eq*)
  apply(*erule disjE*)
   apply(*erule-tac x=i* **in** *allE*, *erule impE*, *assumption*)
   apply *simp+*

apply(*erule-tac x=i* − (*Suc m*) **and** *P*=λ*j*. *H j* ⟶ *J j* ⟶ (*I j*)∈*guar* **for** *H J I* **in** *allE*)
apply(*subgoal-tac* (*Suc m*)+(*i* − *Suc m*) ≤ *length x*)
apply(*subgoal-tac* (*Suc m*)+*Suc* (*i* − *Suc m*) ≤ *length x*)
apply(*rotate-tac* −*2*)
apply *simp*
apply *arith*
apply *arith*
apply(*case-tac length* (*drop* (*Suc m*) *x*),*simp*)
apply(*erule-tac x=sa* **in** *allE*)
back
apply(*drule-tac c=drop* (*Suc m*) *x* **in** *subsetD*,*simp*)
apply *clarify*
apply *simp*
apply *clarify*
apply(*case-tac i*≤*m*)
apply(*drule le-imp-less-or-eq*)
apply(*erule disjE*)
apply(*erule-tac x=i* **in** *allE*, *erule impE*, *assumption*)
apply *simp*
apply *simp*
apply(*erule-tac x=i* − (*Suc m*) **and** *P*=λ*j*. *H j* ⟶ *J j* ⟶ (*I j*)∈*guar* **for** *H J I* **in** *allE*)
apply(*subgoal-tac* (*Suc m*)+(*i* − *Suc m*) ≤ *length x*)
apply(*subgoal-tac* (*Suc m*)+*Suc* (*i* − *Suc m*) ≤ *length x*)
apply(*rotate-tac* −*2*)
apply *simp*
apply *arith*
apply *arith*
done

15.2.4 Soundness of the Sequential rule

inductive-cases *Seq-cases* [*elim*!]: (*Some* (*Seq P Q*), *s*) −*impc*→ *t*

lemma *last-lift-not-None*: *fst* ((*lift Q*) ((*x*#*xs*)!(*length xs*))) ≠ *None*
apply(*subgoal-tac length xs*<*length* (*x* # *xs*))
apply(*drule-tac Q=Q* **in** *lift-nth*)
apply(*erule ssubst*)
apply (*simp add*:*lift-def*)
apply(*case-tac* (*x* # *xs*) ! *length xs*,*simp*)
apply *simp*
done

lemma *Seq-sound1* [*rule-format*]:
x∈ *cpt-p-mod* ⟹ ∀ *s P*. *x* !*0*=(*Some* (*Seq P Q*), *s*) ⟶
(∀ *i*<*length x*. *fst*(*x*!*i*)≠*Some Q*) ⟶
(∃ *xs*∈ *cpts-of-p* (*Some P*) *s*. *x*=*map* (*lift Q*) *xs*)
apply(*erule cpt-p-mod.induct*)

```
apply(unfold cpts-of-p-def)
apply safe
apply simp-all
    apply(simp add:lift-def)
    apply(rule-tac x=[(Some Pa, sa)] in exI,simp add:CptsPOne)
   apply(subgoal-tac (∀ i < Suc (length xs). fst (((Some (Seq Pa Q), t) # xs) ! i)
≠ Some Q))
    apply clarify
    apply(rule-tac x=(Some Pa, sa) #(Some Pa, t) # zs in exI,simp)
    apply(rule conjI,erule CptsPEnv)
    apply(simp (no-asm-use) add:lift-def)
   apply clarify
   apply(erule-tac x=Suc i in allE, simp)
  apply(ind-cases ((Some (Seq Pa Q), sa), None, t) ∈ ptran for Pa sa t)
 apply(rule-tac x=(Some P, sa) # xs in exI, simp add:cpts-iff-cpt-p-mod lift-def)
apply(erule-tac x=length xs in allE, simp)
apply(simp only:Cons-lift-append)
apply(subgoal-tac length xs < length ((Some P, sa) # xs))
 apply(simp only :nth-append length-map last-length nth-map)
 apply(case-tac last((Some P, sa) # xs))
 apply(simp add:lift-def)
apply simp
done

lemma Seq-sound2 [rule-format]:
  x ∈ cpts-p ⟹ ∀ s P i. x!0=(Some (Seq P Q), s) ⟶ i<length x
  ⟶ fst(x!i)=Some Q ⟶
  (∀ j<i. fst(x!j)≠(Some Q)) ⟶
  (∃ xs ys. xs ∈ cpts-of-p (Some P) s ∧ length xs=Suc i
   ∧ ys ∈ cpts-of-p (Some Q) (snd(xs !i)) ∧ x=(map (lift Q) xs)@tl ys)
apply(erule cpts-p.induct)
apply(unfold cpts-of-p-def)
apply safe
apply simp-all
 apply(case-tac i,simp+)
 apply(erule allE,erule impE,assumption,simp)
 apply clarify
 apply(subgoal-tac (∀ j < nat. fst (((Some (Seq Pa Q), t) # xs) ! j) ≠ Some
Q),clarify)
  prefer 2
  apply force
 apply(case-tac xsa,simp,simp)
 apply(rename-tac list)
 apply(rule-tac x=(Some Pa, sa) #(Some Pa, t) # list in exI,simp)
 apply(rule conjI,erule CptsPEnv)
 apply(simp (no-asm-use) add:lift-def)
 apply(rule-tac x=ys in exI,simp)
apply(ind-cases ((Some (Seq Pa Q), sa), t) ∈ ptran for Pa sa t)
 apply simp
```

```
 apply(rule-tac x=(Some Pa, sa)#[(None, ta)] in exI,simp)
 apply(rule conjI)
  apply(drule-tac xs=[] in CptsPComp,force simp add:CptsPOne,simp)
 apply(case-tac i, simp+)
 apply(case-tac nat,simp+)
 apply(rule-tac x=(Some Q,ta)#xs in exI,simp add:lift-def)
 apply(case-tac nat,simp+)
 apply(force)
apply(case-tac i, simp+)
apply(case-tac nat,simp+)
apply(erule-tac x=Suc nata in allE,simp)
apply clarify
apply(subgoal-tac (∀j<Suc nata. fst (((Some (Seq P2 Q), ta) # xs) ! j) ≠ Some
Q),clarify)
 prefer 2
 apply clarify
 apply force
apply(rule-tac x=(Some Pa, sa)#(Some P2, ta)#(tl xsa) in exI,simp)
apply(rule conjI,erule CptsPComp)
apply(rule nth-tl-if,force,simp+)
apply(rule-tac x=ys in exI,simp)
apply(rule conjI)
apply(rule nth-tl-if,force,simp+)
 apply(rule tl-zero,simp+)
 apply force
apply(rule conjI,simp add:lift-def)
apply(subgoal-tac lift Q (Some P2, ta) =(Some (Seq P2 Q), ta))
 apply(simp add:Cons-lift del:list.map)
 apply(rule nth-tl-if)
   apply force
  apply simp+
apply(simp add:lift-def)
done


lemma last-lift-not-None2: fst ((lift Q) (last (x#xs))) ≠ None
apply(simp only:last-length [THEN sym])
apply(subgoal-tac length xs<length (x # xs))
 apply(drule-tac Q=Q in  lift-nth)
 apply(erule ssubst)
 apply (simp add:lift-def)
 apply(case-tac (x # xs) ! length xs,simp)
apply simp
done

lemma Seq-sound:
  ⟦⊨_I Some P sat_p [pre, rely, guar, mid]; ⊨_I Some Q sat_p [mid, rely, guar, post]⟧
  ⟹ ⊨_I Some (Seq P Q) sat_p [pre, rely, guar, post]
apply(unfold prog-validity-def)
```

apply *clarify*
apply(*case-tac* $\exists$ *i*<*length x*. *fst*(*x*!*i*)=*Some Q*)
prefer *2*
apply (*simp add*:*cpts-of-p-def cpts-iff-cpt-p-mod*)
apply *clarify*
apply(*frule-tac Seq-sound1*,*force*)
apply *force*
apply *clarify*
apply(*erule-tac x*=*s* **in** *allE*,*simp*)
apply(*drule-tac c*=*xs* **in** *subsetD*,*simp add*:*cpts-of-p-def cpts-iff-cpt-p-mod*)
apply(*simp add*:*assume-p-def gets-p-def*)
apply *clarify*
apply(*erule-tac P*=λ*j*. *H j* $\longrightarrow$ *J j* $\longrightarrow$ *I j* **for** *H J I* **in** *allE*,*erule impE*, *assumption*)
apply(*simp add*:*snd-lift*)
apply(*erule mp*)
apply(*force elim*:*petranE intro*:*EnvP simp add*:*lift-def*)
apply(*simp add*:*commit-p-def*)
apply(*rule conjI*)
apply *clarify*
apply(*erule-tac P*=λ*j*. *H j* $\longrightarrow$ *J j* $\longrightarrow$ *I j* **for** *H J I* **in** *allE*,*erule impE*, *assumption*)
apply(*simp add*:*snd-lift getspc-p-def gets-p-def*)
apply(*erule mp*)
apply(*case-tac* (*xs*!*i*))
apply(*case-tac* (*xs*! *Suc i*))
apply(*case-tac fst*(*xs*!*i*))
apply(*erule-tac x*=*i* **in** *allE*, *simp add*:*lift-def*)
apply(*case-tac fst*(*xs*!*Suc i*))
apply(*force simp add*:*lift-def*)
apply(*force simp add*:*lift-def*)
apply *clarify*
apply(*case-tac xs*,*simp add*:*cpts-of-p-def*)
apply *clarify*
apply (*simp del*:*list.map*)
apply (*rename-tac list*)
apply(*subgoal-tac* (*map* (*lift Q*) ((*a*, *b*) # *list*))$\neq$[])
apply(*drule last-conv-nth*)
apply (*simp del*:*list.map*)
apply(*simp add*:*getspc-p-def gets-p-def*)
apply(*simp only*:*last-lift-not-None*)
apply *simp*

apply(*erule exE*)
apply(*drule-tac n*=*i* **and** *P*=λ*i*. *i* < *length x* $\wedge$ *fst* (*x* ! *i*) = *Some Q* **in** *Ex-first-occurrence*)
apply *clarify*
apply (*simp add*:*cpts-of-p-def*)
apply *clarify*
apply(*frule-tac i*=*m* **in** *Seq-sound2*,*force*)

```
  apply simp+
apply clarify
apply(simp add:commit-p-def)
apply(erule-tac x=s in allE)
apply(drule-tac c=xs in subsetD,simp)
 apply(case-tac xs=[],simp)
 apply(simp add:cpts-of-p-def assume-p-def nth-append gets-p-def getspc-p-def)
 apply clarify
 apply(erule-tac x=i in allE)
  back
 apply(simp add:snd-lift)
 apply(erule mp)
 apply(force elim:petranE intro:EnvP simp add:lift-def)
apply simp
apply clarify
apply(erule-tac x=snd(xs!m) in allE)
apply(simp add:getspc-p-def gets-p-def)
apply(drule-tac c=ys in subsetD,simp add:cpts-of-p-def assume-p-def)
 apply(case-tac xs≠[])
 apply(drule last-conv-nth,simp)
 apply(rule conjI)
  apply(simp add:gets-p-def)
  apply(erule mp)
  apply(case-tac xs!m)
  apply(case-tac fst(xs!m),simp)
  apply(simp add:lift-def nth-append)
 apply clarify
 apply(simp add:gets-p-def)
 apply(erule-tac x=m+i in allE)
 back
 back
 apply(case-tac ys,(simp add:nth-append)+)
 apply (case-tac i, (simp add:snd-lift)+)

  apply(erule mp)
  apply(case-tac xs!m)
  apply(force elim: intro:EnvP simp add:lift-def)
 apply simp
apply simp
apply clarify
apply(rule conjI,clarify)
 apply(case-tac i<m,simp add:nth-append)
  apply(simp add:snd-lift)
  apply(erule allE, erule impE, assumption, erule mp)
  apply(case-tac (xs ! i))
  apply(case-tac (xs ! Suc i))
  apply(case-tac fst(xs ! i),force simp add:lift-def)
  apply(case-tac fst(xs ! Suc i))
   apply (force simp add:lift-def)
```

apply (*force simp add*:*lift-def*)
apply(*erule-tac x*=*i*−*m* **in** *allE*)
back
back
apply(*subgoal-tac Suc* (*i* − *m*) < *length ys*,*simp*)
prefer *2*
apply *arith*
apply(*simp add*:*nth-append snd-lift*)
apply(*rule conjI*,*clarify*)
apply(*subgoal-tac i*=*m*)
prefer *2*
apply *arith*
apply *clarify*
apply(*simp add*:*cpts-of-p-def*)
apply(*rule tl-zero*)
apply(*erule mp*)
apply(*case-tac lift Q* (*xs*!*m*),*simp add*:*snd-lift*)
apply(*case-tac xs*!*m*,*case-tac fst*(*xs*!*m*),*simp add*:*lift-def snd-lift*)
apply(*case-tac ys*,*simp*+)
apply(*simp add*:*lift-def*)
apply *simp*
apply *force*
apply *clarify*
apply(*rule tl-zero*)
apply(*rule tl-zero*)
apply (*subgoal-tac i*−*m*=*Suc*(*i*−*Suc m*))
apply *simp*
apply(*erule mp*)
apply(*case-tac ys*,*simp*+)
apply *force*
apply *arith*
apply *force*
apply *clarify*
apply(*case-tac* (*map* (*lift Q*) *xs* @ *tl ys*)≠[])
apply(*drule last-conv-nth*)
apply(*simp add*: *snd-lift nth-append*)
apply(*rule conjI*,*clarify*)
apply(*case-tac ys*,*simp*+)
apply *clarify*
apply(*case-tac ys*,*simp*+)
done

15.2.5 Soundness of the While rule

lemma *last-append*[*rule-format*]:
∀ *xs*. *ys*≠[] ⟶ ((*xs*@*ys*)!(*length* (*xs*@*ys*) − (*Suc 0*)))=(*ys*!(*length ys* − (*Suc 0*)))
apply(*induct ys*)
apply *simp*
apply *clarify*

apply (*simp add*:*nth-append*)
done

lemma *assum-after-body*:
 $\llbracket \models_I$ *Some P sat*$_p$ [*pre* $\cap$ *b*, *rely*, *guar*, *pre*];
 (*Some P*, *s*) # *xs* $\in$ *cpt-p-mod*; *fst* (*last* ((*Some P*, *s*) # *xs*)) = *None*; *s* $\in$ *b*;
 (*Some* (*While b P*), *s*) # (*Some* (*Seq P* (*While b P*)), *s*) #
  map (*lift* (*While b P*)) *xs* @ *ys* $\in$ *assume-p* (*pre*, *rely*)$\rrbracket$
 $\Longrightarrow$ (*Some* (*While b P*), *snd* (*last* ((*Some P*, *s*) # *xs*))) # *ys* $\in$ *assume-p* (*pre*, *rely*)
apply(*simp add*:*assume-p-def prog-validity-def cpts-of-p-def cpts-iff-cpt-p-mod gets-p-def*)
apply *clarify*
apply(*erule-tac x*=*s* **in** *allE*)
apply(*drule-tac c*=(*Some P*, *s*) # *xs* **in** *subsetD*,*simp*)
 apply *clarify*
 apply(*erule-tac x*=*Suc i* **in** *allE*)
 apply *simp*
 apply(*simp add*:*Cons-lift-append nth-append snd-lift del*:*list*.*map*)
 apply(*erule mp*)
 apply(*erule petranE*,*simp*)
 apply(*case-tac fst*(((*Some P*, *s*) # *xs*) ! *i*))
  apply(*force intro*:*EnvP simp add*:*lift-def*)
 apply(*force intro*:*EnvP simp add*:*lift-def*)
apply(*rule conjI*)
 apply *clarify*
 apply(*simp add*:*commit-p-def last-length*)
apply *clarify*
apply(*rule conjI*)
 apply(*simp add*:*commit-p-def getspc-p-def gets-p-def*)
apply *clarify*
apply(*erule-tac x*=*Suc*(*length xs* + *i*) **in** *allE*,*simp*)
apply(*case-tac i*, *simp add*:*nth-append Cons-lift-append snd-lift last-conv-nth lift-def split-def*)
apply(*simp add*:*Cons-lift-append nth-append snd-lift*)
done

lemma *While-sound-aux* [*rule-format*]:
 $\llbracket$ *pre* $\cap$ − *b* $\subseteq$ *post*; $\models_I$ *Some P sat*$_p$ [*pre* $\cap$ *b*, *rely*, *guar*, *pre*]; $\forall$ *s*. (*s*, *s*) $\in$ *guar*;
  stable pre rely; *stable post rely*; *x* $\in$ *cpt-p-mod* $\rrbracket$
 $\Longrightarrow$ $\forall$ *s xs*. *x*=(*Some*(*While b P*),*s*)#*xs* $\longrightarrow$ *x*$\in$*assume-p*(*pre*, *rely*) $\longrightarrow$ *x* $\in$ *commit-p* (*guar*, *post*)
apply(*erule cpt-p-mod*.*induct*)
apply *safe*
apply (*simp-all del*:*last*.*simps*)

apply(*simp add*:*commit-p-def getspc-p-def gets-p-def*)

apply(*rule etran-in-comm*)
apply(*erule mp*)

apply(*erule tl-of-assum-in-assum*,*simp*)

apply(*ind-cases* ((*Some* (*While b P*), *s*), *None*, *t*) ∈ *ptran* **for** *s t*)
apply(*simp add*:*commit-p-def*)
apply(*simp add*:*cpts-iff-cpt-p-mod* [*THEN sym*])
apply(*rule conjI*,*clarify*)
 apply(*force simp add*:*assume-p-def getspc-p-def gets-p-def*)
apply(*simp add*: *getspc-p-def gets-p-def*)
apply *clarify*
apply(*rule conjI*, *clarify*)
 apply(*case-tac i*,*simp*,*simp*)
 apply(*force simp add*:*not-ctran-None2*)
apply(*subgoal-tac* ∀ *i*. *Suc i* < *length* ((*None*, *t*) # *xs*) ⟶ (((*None*, *t*) # *xs*) ! *i*, ((*None*, *t*) # *xs*) ! *Suc i*)∈ *petran*)
 prefer *2*
 apply *clarify*
 apply(*rule-tac m*=*length* ((*None*, *s*) # *xs*) **in** *etran-or-ctran*,*simp*+)
 apply(*erule not-ctran-None2*,*simp*)
 apply *simp*+
apply(*frule-tac j*=*0* **and** *k*=*length* ((*None*, *s*) # *xs*) − *1* **and** *p*=*post* **in** *stability*,*simp*+)
   apply(*force simp add*:*assume-p-def subsetD gets-p-def*)
  apply(*simp add*:*assume-p-def*)
  apply *clarify*
  apply(*erule-tac x*=*i* **in** *allE*,*simp*)
  apply (*simp add*:*gets-p-def*)
  apply(*erule-tac x*=*Suc i* **in** *allE*,*simp*)
 apply *simp*
apply *clarify*
apply (*simp add*:*last-length*)

apply(*thin-tac P* = *While b P* ⟶ *Q* **for** *Q*)
apply(*rule ctran-in-comm*,*simp*)
apply(*simp add*:*Cons-lift del*:*list.map*)
apply(*simp add*:*commit-p-def del*:*list.map*)
apply(*rule conjI*)
 apply *clarify*
 apply(*case-tac fst*(((*Some P*, *sa*) # *xs*) ! *i*))
  apply(*case-tac* ((*Some P*, *sa*) # *xs*) ! *i*)
  apply (*simp add*:*lift-def*)
  apply(*ind-cases* (*Some* (*While b P*), *ba*) −*impc*→ *t* **for** *ba t*)
   apply (*simp add*:*gets-p-def*)
  apply (*simp add*:*gets-p-def*)
 apply(*simp add*:*snd-lift gets-p-def del*:*list.map*)
 apply(*simp only*:*prog-validity-def cpts-of-p-def cpts-iff-cpt-p-mod*)
 apply(*erule-tac x*=*sa* **in** *allE*)
 apply(*drule-tac c*=(*Some P*, *sa*) # *xs* **in** *subsetD*)
  apply (*simp add*:*assume-p-def gets-p-def del*:*list.map*)
  apply *clarify*

```
  apply(erule-tac x=Suc ia in allE,simp add:snd-lift del:list.map)
  apply(erule mp)
  apply(case-tac fst(((Some P, sa) # xs) ! ia))
   apply(erule petranE,simp add:lift-def)
   apply(rule EnvP)
  apply(erule petranE,simp add:lift-def)
  apply(rule EnvP)
 apply (simp add:commit-p-def getspc-p-def gets-p-def del:list.map)
 apply clarify
 apply(erule allE,erule impE,assumption)
 apply(erule mp)
 apply(case-tac ((Some P, sa) # xs) ! i)
 apply(case-tac xs!i)
 apply(simp add:lift-def)
 apply(case-tac fst(xs!i))
  apply force
 apply force

apply clarify
apply(subgoal-tac (map (lift (While b P)) ((Some P, sa) # xs))≠[])
 apply(drule last-conv-nth)
 apply (simp add:getspc-p-def gets-p-def del:list.map)
 apply(simp only:last-lift-not-None)
apply simp

apply(thin-tac P = While b P ⟶ Q for Q)
apply(rule ctran-in-comm,simp del:last.simps)

apply(subgoal-tac (Some (While b P), snd (last ((Some P, sa) # xs))) # ys ∈
assume-p (pre, rely))
 apply (simp del:last.simps)
 prefer 2
 apply(erule assum-after-body)
  apply (simp del:last.simps)+

apply(simp add:commit-p-def getspc-p-def gets-p-def del:list.map last.simps)
apply(rule conjI)
 apply clarify
 apply(simp only:Cons-lift-append)
 apply(case-tac i<length xs)
  apply(simp add:nth-append del:list.map last.simps)
  apply(case-tac fst(((Some P, sa) # xs) ! i))
   apply(case-tac ((Some P, sa) # xs) ! i)
   apply (simp add:lift-def del:last.simps)
   apply(ind-cases (Some (While b P), ba) −impc→ t for ba t)
    apply simp
   apply simp
  apply(simp add:snd-lift del:list.map last.simps)
  apply(thin-tac ∀i. i < length ys ⟶ P i for P)
```

```
  apply(simp only:prog-validity-def cpts-of-p-def cpts-iff-cpt-p-mod)
  apply(erule-tac x=sa in allE)
  apply(drule-tac c=(Some P, sa) # xs in subsetD)
   apply (simp add:assume-p-def getspc-p-def gets-p-def del:list.map last.simps)
   apply clarify
    apply(erule-tac x=Suc ia in allE,simp add:nth-append snd-lift del:list.map
last.simps, erule mp)
   apply(case-tac fst(((Some P, sa) # xs) ! ia))
    apply(erule petranE,simp add:lift-def)
    apply(rule EnvP)
   apply(erule petranE,simp add:lift-def)
   apply(rule EnvP)
  apply (simp add:commit-p-def getspc-p-def gets-p-def del:list.map)
  apply clarify
  apply(erule allE,erule impE,assumption)
  apply(erule mp)
  apply(case-tac ((Some P, sa) # xs) ! i)
  apply(case-tac xs!i)
  apply(simp add:lift-def)
  apply(case-tac fst(xs!i))
   apply force
 apply force

apply(subgoal-tac i−length xs <length ys)
 prefer 2
 apply arith
apply(erule-tac x=i−length xs in allE,clarify)
apply(case-tac i=length xs)
 apply (simp add:nth-append snd-lift del:list.map last.simps)
 apply(simp add:last-length del:last.simps)
 apply(erule mp)
 apply(case-tac last((Some P, sa) # xs))
 apply(simp add:lift-def del:last.simps)

apply(case-tac i−length xs)
 apply arith
apply(simp add:nth-append del:list.map last.simps)
apply(rotate-tac −3)
apply(subgoal-tac i− Suc (length xs)=nat)
 prefer 2
 apply arith
apply simp

apply clarify
apply(case-tac ys)
 apply(simp add:Cons-lift del:list.map last.simps)
 apply(subgoal-tac (map (lift (While b P)) ((Some P, sa) # xs))≠[])
  apply(drule last-conv-nth)
  apply (simp del:list.map)
```

  apply(*simp only*:*last-lift-not-None*)
 apply *simp*
apply(*subgoal-tac* ((*Some* (*Seq P* (*While b P*)), *sa*) # *map* (*lift* (*While b P*)) *xs* @ *ys*)≠[])
 apply(*drule last-conv-nth*)
 apply (*simp del*:*list.map last.simps*)
 apply(*simp add*:*nth-append del*:*last.simps*)
 apply(*rename-tac a list*)
 apply(*subgoal-tac* ((*Some* (*While b P*), *snd* (*last* ((*Some P*, *sa*) # *xs*))) # *a* # *list*)≠[])
  apply(*drule last-conv-nth*)
  apply (*simp del*:*list.map last.simps*)
 apply *simp*
apply *simp*
done

lemma *While-sound*:
  ⟦*stable pre rely*; *pre* ∩ − *b* ⊆ *post*; *stable post rely*;
    $\models_I$ *Some P* sat_p [*pre* ∩ *b*, *rely*, *guar*, *pre*]; ∀ *s*. (*s*,*s*)∈*guar*⟧
  ⟹ $\models_I$ *Some* (*While b P*) sat_p [*pre*, *rely*, *guar*, *post*]
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*erule-tac xs*=*tl x* **in** *While-sound-aux*)
 apply(*simp add*:*prog-validity-def*)
 apply *force*
 apply *simp-all*
apply(*simp add*:*cpts-iff-cpt-p-mod cpts-of-p-def*)
apply(*simp add*:*cpts-of-p-def*)
apply *clarify*
apply(*rule nth-equalityI*)
 apply *simp-all*
 apply(*case-tac x*,*simp*+)
apply(*case-tac i*,*simp*+)
apply(*case-tac x*,*simp*+)
done

15.2.6 Soundness of the Rule of Consequence

lemma *Conseq-sound*:
  ⟦*pre* ⊆ *pre*′; *rely* ⊆ *rely*′; *guar*′ ⊆ *guar*; *post*′ ⊆ *post*;
  $\models_I$ *P* sat_p [*pre*′, *rely*′, *guar*′, *post*′]⟧
  ⟹ $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
apply(*simp add*:*prog-validity-def assume-p-def commit-p-def*)
apply *clarify*
apply(*erule-tac x*=*s* **in** *allE*)
apply(*drule-tac c*=*x* **in** *subsetD*)
 apply *force*
apply *force*
done

15.2.7 Soundness of the Nondt rule

lemma *unique-ctran-Nondt* [*rule-format*]:
$\forall s\ i.\ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Nondt\ r),\ s) \longrightarrow$
$Suc\ i{<}length\ x \longrightarrow x!i\ {-impc\rightarrow}\ x!Suc\ i \longrightarrow$
$(\forall j.\ Suc\ j{<}length\ x \longrightarrow i{\neq}j \longrightarrow x!j\ {-imppe\rightarrow}\ x!Suc\ j)$
apply(*induct x*,*simp*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
 apply(*case-tac i*,*simp+*)
 apply *clarify*
 apply(*case-tac j*,*simp*)
  apply(*rule EnvP*)
 apply *simp*
apply *clarify*
apply *simp*
apply(*case-tac i*)
 apply(*case-tac j*,*simp*,*simp*)
 apply(*erule ptran.cases*,*simp-all*)
 apply(*force elim*: *not-ctran-None*)
apply(*ind-cases* $((Some\ (Nondt\ r),\ sa),\ Q,\ t) \in ptran$ **for** *sa Q t*)
apply *simp*
apply(*drule-tac i=nat* **in** *not-ctran-None*,*simp*)
apply(*erule petranE*,*simp*)
done

lemma *exists-ctran-Nondt-None* [*rule-format*]:
$\forall s\ i.\ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Nondt\ r),\ s)$
$\longrightarrow i{<}length\ x \longrightarrow fst(x!i){=}None \longrightarrow (\exists j{<}i.\ x!j\ {-impc\rightarrow}\ x!Suc\ j)$
apply(*induct x*,*simp*)
apply *simp*
apply *clarify*
apply(*erule cpts-p.cases*,*simp*)
 apply(*case-tac i*,*simp*,*simp*)
 apply(*erule-tac x=nat* **in** *allE*,*simp*)
 apply *clarify*
 apply(*rule-tac x=Suc j* **in** *exI*,*simp*,*simp*)
apply *clarify*
apply(*case-tac i*,*simp*,*simp*)
apply(*rule-tac x=0* **in** *exI*,*simp*)
done

lemma *Nondt-sound*:
$[\![pre \subseteq \{s.\ (\forall t.\ (s,t) \in r \longrightarrow t \in post) \wedge (\exists t.\ (s,t) \in r)\};\ \{(s,t).\ s \in pre \wedge$
$(s,t){\in}r\} \subseteq guar;$
$stable\ pre\ rely;\ stable\ post\ rely]\!]$
$\Longrightarrow \models_I Some\ (Nondt\ r)\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*unfold prog-validity-def*)
apply(*clarify*)

```
apply(simp add:commit-p-def)
apply(simp add:getspc-p-def gets-p-def)
apply(rule conjI)
  apply clarify
  apply(simp add:cpts-of-p-def assume-p-def gets-p-def)
  apply clarify
  apply(frule-tac j=0 and k=i and p=pre in stability)
      apply simp-all
    apply simp
  apply(erule-tac i=i and r=r in unique-ctran-Nondt,simp-all)
 apply(case-tac x!i)
 apply clarify
 apply(drule-tac s=Some (Nondt r) in sym,simp)
 apply(thin-tac ∀j. H j for H)
 apply(force elim:ptran.cases)
apply(simp add:cpts-of-p-def)
apply clarify

apply(frule-tac i=length x - 1 and r=r in exists-ctran-Nondt-None,simp+)
  apply(case-tac x,simp+)
  apply(rule last-fst-esp,simp add:last-length)
 apply (case-tac x,simp+)
apply(simp add:assume-p-def gets-p-def)
apply clarify
apply(frule-tac j=0 and k=j and p=pre in stability)
      apply simp-all
  apply(erule-tac x=i in allE,simp)
 apply(erule-tac i=j and r=r in unique-ctran-Nondt,simp-all)
apply(case-tac x!j)
apply clarify
apply simp
apply(drule-tac s=Some (Nondt r) in sym,simp)
apply(case-tac x!Suc j,simp)
apply(rule ptran.cases,simp)
apply(simp-all)
apply(drule-tac c=sa in subsetD,simp)
apply clarify
apply(frule-tac j=Suc j and k=length x - 1 and p=post in stability,simp-all)
 apply(case-tac x,simp+)
 apply(erule-tac x=i in allE)
apply(erule-tac i=j and r=r in unique-ctran-Nondt, simp-all)
  apply arith+
apply(case-tac x)
apply(simp add:last-length)+
done
```

15.2.8 Soundness of the Rule of Unprecond

lemma *Unprecond-sound*:

assumes *p0*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
and *p1*: $\models_I$ *P* sat_p [*pre'*, *rely*, *guar*, *post*]
shows $\models_I$ *P* sat_p [*pre* ∪ *pre'*, *rely*, *guar*, *post*]
proof −
{
fix *s c*
assume *c* ∈ *cpts-of-p P s* ∩ *assume-p*(*pre* ∪ *pre'*, *rely*)
hence *a1*: *c* ∈ *cpts-of-p P s* **and**
a2: *c* ∈ *assume-p*(*pre* ∪ *pre'*, *rely*) **by** *auto*
hence *c* ∈ *assume-p*(*pre*, *rely*) ∨ *c* ∈ *assume-p*(*pre'*, *rely*)
by (*metis* (*no-types*, *lifting*) *CollectD CollectI Un-iff assume-p-def prod.simps*(*2*))
hence *c* ∈ *commit-p*(*guar*, *post*)
proof
assume *c* ∈ *assume-p* (*pre*, *rely*)
with *p0 a1* **show** *c* ∈ *commit-p* (*guar*, *post*)
unfolding *prog-validity-def* **by** *auto*
next
assume *c* ∈ *assume-p* (*pre'*, *rely*)
with *p1 a1* **show** *c* ∈ *commit-p* (*guar*, *post*)
unfolding *prog-validity-def* **by** *auto*
qed
}
then show *?thesis* **unfolding** *prog-validity-def* **by** *auto*
qed

15.2.9 Soundness of the Rule of Intpostcond

lemma *Intpostcond-sound*:
assumes *p0*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
and *p1*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post'*]
shows $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post* ∩ *post'*]
proof −
{
fix *s c*
assume *a0*: *c* ∈ *cpts-of-p P s* ∩ *assume-p*(*pre*, *rely*)
with *p0* **have** *c* ∈ *commit-p*(*guar*, *post*) **unfolding** *prog-validity-def* **by** *auto*
moreover
from *a0 p1* **have** *c* ∈ *commit-p*(*guar*, *post'*) **unfolding** *prog-validity-def* **by** *auto*
ultimately have *c* ∈ *commit-p*(*guar*, *post* ∩ *post'*)
by (*simp add*: *commit-p-def*)
}
then show *?thesis* **unfolding** *prog-validity-def* **by** *auto*
qed

15.2.10 Soundness of the Rule of Allprecond

lemma *Allprecond-sound*:
assumes *p1*: ∀ *v*∈*U*. $\models_I$ *P* sat_p [{*v*}, *rely*, *guar*, *post*]
shows $\models_I$ *P* sat_p [*U*, *rely*, *guar*, *post*]
proof −

{
  fix *s c*
  assume *a0*: $c \in$ *cpts-of-p P s* $\cap$ *assume-p*(U, *rely*)
  then obtain x **where** *a1*: $x \in U \wedge$ *gets-p* ($c!0$) $= x$
    by (*metis* (*no-types*, *lifting*) *CollectD IntD2 assume-p-def prod.simps*(*2*))

  with *p1* **have** $\models_I$ *P* sat_p [{x}, *rely*, *guar*, *post*] **by** *simp*
   hence *a2*: $\forall s$. *cpts-of-p P s* $\cap$ *assume-p*({x}, *rely*) $\subseteq$ *commit-p*(*guar*, *post*)
unfolding *prog-validity-def* **by** *simp*

  from *a0* **have** $c \in$ *assume-p*(U, *rely*) **by** *simp*
  hence *gets-p* ($c!0$) $\in U \wedge (\forall i$. *Suc i<length c* $\longrightarrow$
           $c!i$ *−imppe→* $c!$(*Suc i*) $\longrightarrow$ (*gets-p* ($c!i$), *gets-p* ($c!$*Suc i*)) $\in$ *rely*) **by**
(*simp add*:*assume-p-def*)
  with *a1* **have** *gets-p* ($c!0$) $\in$ {x} $\wedge (\forall i$. *Suc i<length c* $\longrightarrow$
           $c!i$ *−imppe→* $c!$(*Suc i*) $\longrightarrow$ (*gets-p* ($c!i$), *gets-p* ($c!$*Suc i*)) $\in$ *rely*) **by**
simp

  hence $c \in$ *assume-p*({x}, *rely*) **by** (*simp add*:*assume-p-def*)
  with *a0 a2* **have** $c \in$ *commit-p*(*guar*, *post*) **by** *auto*
}
then show *?thesis* **using** *prog-validity-def* **by** *blast*
qed

15.2.11 Soundness of the Rule of Emptyprecond

lemma *Emptyprecond-sound*: $\models_I$ *P* sat_p [{}, *rely*, *guar*, *post*]
unfolding *prog-validity-def* **by**(*simp add*:*assume-p-def*)

15.2.12 Soundness of None rule

lemma *none-all-none*: $c!0$=(*None*,*s*) $\wedge c \in$ *cpts-p* $\Longrightarrow \forall i$<*length c*. *fst* ($c$! i) = *None*
proof(*induct c arbitrary*:*s*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a c*)
  assume *p1*: $\bigwedge s$. c ! 0 = (*None*, s) $\wedge c \in$ *cpts-p* $\Longrightarrow \forall i$<*length c*. *fst* ($c$! i) = *None*
    and *p2*: (a # c) ! 0 = (*None*, s) $\wedge a$ # $c \in$ *cpts-p*
  hence *a0*: a = (*None*,*s*) **by** *simp*
  thus *?case*
    proof(*cases* c = [])
      case *True*
      with *a0* **show** *?thesis* **by** *auto*
    next
      case *False*
      assume *b0*: $c \neq$ []
      with *p2* **have** *c-cpts*: $c \in$ *cpts-p* **using** *tl-in-cptn* **by** *fast*

from *b0* **obtain** c' **and** *b* **where** bc': $c = b \# c'$
  using *list.exhaust* **by** *blast*
from *a0* **have** $\neg$ *a* $-impc\rightarrow$ *b* **by**(*force elim*: *ptran.cases*)
with *p2* **have** *a* $-imppe\rightarrow$ *b* **using** bc' *etran-or-ctran2-disjI3*[*of a#c 0*] **by** *auto*
hence *fst b = None* **using** *petran.cases*
  by (*metis a0 prod.collapse*)
with *p1* bc' *c-cpts* **have** $\forall$ *i<length c. fst* (*c* ! *i*) = *None*
  by (*metis nth-Cons-0 prod.collapse*)
with *a0* **show** *?thesis*
  by (*simp add*: *nth-Cons'*)
qed

qed

lemma *None-sound-h*: $\forall x.\ x \in pre \longrightarrow (\forall y.\ (x, y) \in rely \longrightarrow y \in pre) \Longrightarrow$
$pre \subseteq post \Longrightarrow$
$snd\ (c\ !\ 0) \in pre \Longrightarrow$
$c \neq [] \Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (snd\ (c\ !\ i),\ snd\ (c\ !\ Suc\ i)) \in rely$
$\Longrightarrow i < length\ c \Longrightarrow snd\ (c\ !\ i) \in pre$
apply(*induct i*) **by** *auto*

lemma *None-sound*:
$\llbracket$ *stable pre rely*; *pre* $\subseteq$ *post* $\rrbracket$
$\Longrightarrow \models_I$ *None sat*$_p$ [*pre*, *rely*, *guar*, *post*]
proof –
assume *p0*: *stable pre rely*
  and *p2*: *pre* $\subseteq$ *post*
{
  fix *s c*
  assume *a0*: $c \in$ *cpts-of-p None s* $\cap$ *assume-p*(*pre*, *rely*)
  hence *c1*: *c*!*0*=(*None*,*s*) $\wedge$ $c \in$ *cpts-p* **by** (*simp add*:*cpts-of-p-def*)
  from *a0* **have** *c2*: *gets-p* (*c*!*0*) $\in$ *pre* $\wedge$ ($\forall$ *i*. *Suc i<length c* $\longrightarrow$
    c!*i* $-imppe\rightarrow$ *c*!(*Suc i*) $\longrightarrow$ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) $\in$ *rely*)
    by (*simp add*:*assume-p-def*)
  from *c1* **have** *c-ne-empty*: $c \neq []$
    by *auto*
  from *c1* **have** *c-all-none*: $\forall$ *i<length c. fst* (*c* ! *i*) = *None* **using** *none-all-none* **by** *fast*

  {
    fix *i*
    assume *suci*: *Suc i<length c*
      and *cc*: *c*!*i* $-impc\rightarrow$ *c*!(*Suc i*)
    from *suci c-all-none* **have** *c*!*i* $-imppe\rightarrow$ *c*!(*Suc i*)
      by (*metis Suc-lessD petran.intros prod.collapse*)
    with *cc* **have**(*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) $\in$ *guar*
      using *c1 etran-or-ctran2-disjI1 suci* **by** *auto*

}
moreover
{
assume *last-none*: *getspc-p* (*last c*) = *None*
from *c2 c-all-none* **have** $\forall$ *i*. *Suc i*<*length c* $\longrightarrow$ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) $\in$ *rely*
by (*metis Suc-lessD petran.intros prod.collapse*)
with *p0 p2 c2 c-ne-empty* **have** $\forall$ *i*. *i* < *length c* $\longrightarrow$ *snd* (*c* ! *i*) $\in$ *pre*
apply(*simp add*:*gets-p-def stable-def*) **apply** *clarify* **using** *None-sound-h* **by** *blast*
with *p2 c-ne-empty* **have** *gets-p* (*last c*) $\in$ *post* **apply**(*simp add*:*gets-p-def*)
using *One-nat-def c-ne-empty last-conv-nth* **by** *force*

}
ultimately have *c* $\in$ *commit-p*(*guar*, *post*) **by** (*simp add*:*commit-p-def*)
}
thus $\models_I$ *None* sat_p [*pre*, *rely*, *guar*, *post*] **using** *prog-validity-def* **by** *blast*
qed

15.2.13 Soundness of the system for programs

theorem *rgsound-p*:
$\vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*] $\Longrightarrow$ $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-p.induct*)
apply(*force elim*:*Basic-sound*)
apply(*force elim*:*Seq-sound*)
apply(*force elim*:*Cond-sound*)
apply(*force elim*:*While-sound*)
apply(*force elim*:*Await-sound*)
apply(*force elim*:*Nondt-sound*)
apply(*force elim*:*None-sound*)
apply(*erule Conseq-sound*,*simp*+)
apply(*erule Unprecond-sound*,*simp*+)
apply(*erule Intpostcond-sound*,*simp*+)
using *Allprecond-sound* **apply** *force*
using *Emptyprecond-sound* **apply** *force*
done

end

16 Integrating the SIMP language into Picore

theory *picore-SIMP*
imports *SIMP/SIMP-sound ../picore/PiCore-RG-Invariant*
begin

definition *ptranI* :: *'Env* $\Rightarrow$ (*'s imp-pconf* $\times$ *'s imp-pconf*) *set*
where *ptranI* $\Gamma \equiv$ *ptran*

definition *petranI* :: *'Env* ⇒ *'s imp-pconf* ⇒ *'s imp-pconf* ⇒ *bool*
where *petranI* Γ ≡ *petran'*

definition *cpts-pI* :: *'Env* ⇒ *'s imp-pconfs set*
where *cpts-pI* Γ ≡ *cpts-p*

definition *cpts-of-pI* :: *'Env* ⇒ (*'s prog*) *option* ⇒ *'s* ⇒ (*'s imp-pconfs*) *set* **where**
cpts-of-pI Γ ≡ *cpts-of-p*

definition *prog-validityI* :: *'Env* ⇒ *'s prog option* ⇒ *'s set* ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *bool*
where *prog-validityI* Γ *P* ≡ *prog-validity P*

definition *assume-pI* :: *'Env* ⇒ (*'s set* × (*'s* × *'s*) *set*) ⇒ (*'s imp-pconfs*) *set*
where *assume-pI* Γ ≡ *assume-p*

definition *commit-pI* :: *'Env* ⇒ ((*'s* × *'s*) *set* × *'s set*) ⇒ (*'s imp-pconfs*) *set*
where *commit-pI* Γ ≡ *commit-p*

definition *rghoare-pI* :: *'Env* ⇒ [*'s prog option*, *'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
(- $\vdash_I$ - sat_p [-, -, -, -] [*60,0,0,0,0*] *45*)
where *rghoare-pI* Γ ≡ *rghoare-p*

lemma *cpts-pI-ne-empty*: [] ∉ *cpts-pI* Γ
by (*simp add*: *cpts-pI-def*)

lemma *petran-simpsI*:
petranI Γ (*a*, *b*) (*c*, *d*) ⟹ *a* = *c*
by(*simp add*:*petranI-def petran.simps*)

lemma *none-no-tranI'*: ((*Q*, *s*),(*P*,*t*)) ∈ *ptranI* Γ ⟹ *Q* ≠ *None*
apply (*simp add*:*ptranI-def*) **apply**(*rule ptran.cases*)
by *simp+*

lemma *none-no-tranI*: ((*None*, *s*),(*P*,*t*)) ∉ *ptranI* Γ
using *none-no-tranI'*
by *fast*

lemma *ptran-neqI*: ((*P*, *s*),(*P*,*t*)) ∉ *ptranI* Γ
by (*simp add*:*ptranI-def*)

interpretation *event ptranI petranI None*
using *petran-simpsI none-no-tranI ptran-neqI*
event.intro[*of petranI None ptranI*] **by** *blast*

thm *ptran'-def*

lemma *cpts-p-simpsI*:

$((\exists P\ s.\ aa = [(P, s)]) \lor$
$(\exists P\ t\ xs\ s.\ aa = (P, s) \# (P, t) \# xs \land (P, t) \# xs \in \textit{cpts-pI}\ \Gamma) \lor$
$(\exists P\ s\ Q\ t\ xs.\ aa = (P, s) \# (Q, t) \# xs \land \textit{ptran}'\ \Gamma\ (P, s)\ (Q, t) \land (Q, t) \#$
$xs \in \textit{cpts-pI}\ \Gamma))$
$\Longrightarrow (aa \in \textit{cpts-pI}\ \Gamma)$
apply(*simp add*:*cpts-pI-def ptranI-def ptran′-def*) **using** *cpts-p.simps*[*of aa*] **by** *blast*

lemma *cpts-of-p-defI*: $l!0=(P,s) \land l \in \textit{cpts-pI}\ \Gamma \Longrightarrow l \in \textit{cpts-of-pI}\ \Gamma\ P\ s$
by(*simp add*:*cpts-pI-def cpts-of-pI-def cpts-of-p-def*)

interpretation *event-comp ptranI petranI None cpts-pI cpts-of-pI*
using *cpts-pI-ne-empty cpts-p-simpsI cpts-of-p-defI petran-simpsI none-no-tranI ptran-neqI*
event-comp.intro[*of ptranI petranI None cpts-pI cpts-of-pI*] *event.intro*[*of petranI None ptranI*]
event-comp-axioms.intro[*of cpts-pI ptranI cpts-of-pI*]
apply(*simp add*:*ptranI-def ptran′-def*) **by** *blast*

lemma *prog-validity-defI*: $\textit{prog-validityI}\ \Gamma\ P\ \textit{pre rely guar post} \Longrightarrow$
$\forall s.\ \textit{cpts-of-pI}\ \Gamma\ P\ s \cap \textit{assume-pI}\ \Gamma\ (\textit{pre}, \textit{rely}) \subseteq \textit{commit-pI}\ \Gamma\ (\textit{guar}, \textit{post})$
by (*simp add*:*prog-validityI-def cpts-of-pI-def assume-pI-def commit-pI-def prog-validity-def*)

lemma *assume-p-defI*: $\textit{gets-p}\ (c!0) \in \textit{pre} \land (\forall i.\ \textit{Suc}\ i<\textit{length}\ c \longrightarrow$
$\textit{petranI}\ \Gamma\ (c!i)\ (c!(\textit{Suc}\ i)) \longrightarrow (\textit{gets-p}\ (c!i), \textit{gets-p}\ (c!\textit{Suc}\ i)) \in \textit{rely})$
$\Longrightarrow c \in \textit{assume-pI}\ \Gamma\ (\textit{pre}, \textit{rely})$
by(*simp add*:*assume-pI-def petranI-def assume-p-def SIMP-validity.gets-p-def PiCore-Semantics.gets-p-def*)

lemma *commit-p-defI*: $c \in \textit{commit-pI}\ \Gamma\ (\textit{guar}, \textit{post}) \Longrightarrow (\forall i.\ \textit{Suc}\ i<\textit{length}\ c \longrightarrow$

$(c!i,c!(\textit{Suc}\ i)) \in \textit{ptranI}\ \Gamma\ \longrightarrow (\textit{gets-p}\ (c!i), \textit{gets-p}\ (c!\textit{Suc}\ i)) \in \textit{guar})$
$\land$
$(\textit{getspc-p}\ (\textit{last}\ c) = \textit{None} \longrightarrow \textit{gets-p}\ (\textit{last}\ c) \in \textit{post})$
by(*simp add*:*commit-pI-def ptranI-def commit-p-def PiCore-Semantics.getspc-p-def*

SIMP-validity.getspc-p-def SIMP-validity.gets-p-def PiCore-Semantics.gets-p-def)

lemma *rgsound-pI*: $\textit{rghoare-pI}\ \Gamma\ P\ \textit{pre rely guar post} \longrightarrow \textit{prog-validityI}\ \Gamma\ P\ \textit{pre rely guar post}$
apply(*simp add*:*rghoare-pI-def prog-validityI-def*) **using** *rgsound-p* **by** *blast*

interpretation *event-hoare ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI rghoare-pI*
using *cpts-pI-ne-empty cpts-p-simpsI cpts-of-p-defI petran-simpsI none-no-tranI ptran-neqI*
prog-validity-defI assume-p-defI commit-p-defI rgsound-pI
event-comp-axioms.intro[*of cpts-pI ptranI cpts-of-pI*]

event-comp.intro[*of ptranI petranI None cpts-pI cpts-of-pI*] *event.intro*[*of petranI None ptranI*]
event-validity-axioms.intro[*of prog-validityI cpts-of-pI assume-pI commit-pI petranI ptranI None*]
event-validity.intro[*of ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI*]
event-hoare.intro[*of ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI rghoare-pI*]
event-hoare-axioms.intro[*of rghoare-pI prog-validityI*]
apply(*simp add*:*ptranI-def ptran′-def*) **by** *blast*

thm *invariant-theorem*

end

17 Concrete Syntax of PiCore-SIMP

theory *picore-SIMP-Syntax*
imports *picore-SIMP*

begin

syntax
-quote :: ′*b* ⇒ (′*s* ⇒ ′*b*) ((-) [*0*] *1000*)
-antiquote :: (′*s* ⇒ ′*b*) ⇒ ′*b* (´- [*1000*] *1000*)
-Assert :: ′*s* ⇒ ′*s set* ((⦃-⦄) [*0*] *1000*)

translations
⦃*b*⦄ ⇀ *CONST Collect b*

parse-translation ‹
let
fun quote-tr [*t*] = *Syntax-Trans.quote-tr* @{*syntax-const -antiquote*} *t*
| *quote-tr ts* = *raise TERM* (*quote-tr*, *ts*);
in [(@{*syntax-const -quote*}, *K quote-tr*)] *end*
›

definition *Skip* :: ′*s prog* (*SKIP*)
where *SKIP* ≡ *Basic id*

abbreviation *Wrap-prog* :: ′*s prog* ⇒ ′*s prog option* (*W*(-) *0*)
where *Wrap-prog p* ≡ *Some p*

notation *Seq* ((-;;/ -) [*60*,*61*] *60*)

syntax
rghoare-p :: *'Env* ⇒ *'prog* ⇒ [*'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
  (- ⊢ - sat_p [-, -, -, -] [*60,60,0,0,0,0*] *45*)
rghoare-e :: *'Env* ⇒ (*'l,'k,'s,'prog*) *event* ⇒ [*'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
  (- ⊢ - sat_e [-, -, -, -] [*60,60,0,0,0,0*] *45*)
Evt-sat-RG:: *'Env* ⇒ (*'l,'k,'s,'prog*) *event* ⇒ *'s rgformula* ⇒ *bool* ((- ⊢-) [*60,60,60*] *61*)
rghoare-es :: *'Env* ⇒ (*'l,'k,'s,'prog*) *rgformula-ess* ⇒ [*'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
  (- ⊢ - sat_s [-, -, -, -] [*60,60,0,0,0,0*] *45*)
rghoare-pes :: *'Env* ⇒ (*'l,'k,'s,'prog*) *rgformula-par* ⇒ [*'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
  (- ⊢ - *SAT* [-, -, -, -] [*60,60,0,0,0,0*] *45*)
Evt-sat-RG:: *'Env* ⇒ (*'l,'k,'s,'prog*) *event* ⇒ *'s rgformula* ⇒ *bool* ((- ⊢-) [*60,60,60*] *61*)
Esys-sat-RG :: *'Env* ⇒ (*'l,'k,'s,'prog*) *rgformula-ess* ⇒ *'s rgformula* ⇒ *bool* ((- $\vdash_{es}$-) [*60,60,60*] *61*)

syntax
-Assign :: *idt* ⇒ *'b* ⇒ *'s prog* ((´- :=/ -) [*70*, *65*] *61*)
-Cond :: *'s bexp* ⇒ *'s prog* ⇒ *'s prog* ⇒ *'s prog* ((*0IF* -/ *THEN* -/ *ELSE* -/*FI*) [*0*, *0*, *0*] *61*)
-Cond2 :: *'s bexp* ⇒ *'s prog* ⇒ *'s prog* ((*0IF* - *THEN* - *FI*) [*0,0*] *62*)
-While :: *'s bexp* ⇒ *'s prog* ⇒ *'s prog* ((*0WHILE* - /*DO* - /*OD*) [*0*, *0*] *61*)
-Await :: *'s bexp* ⇒ *'s prog* ⇒ *'s prog* ((*0AWAIT* - /*THEN* /- /*END*) [*0,0*] *61*)
-Atom :: *'s prog* ⇒ *'s prog* ((*0ATOMIC* - *END*) *61*)
-Wait :: *'s bexp* ⇒ *'s prog* ((*0WAIT* - *END*) *61*)
-For :: *'s prog* ⇒ *'s bexp* ⇒ *'s prog* ⇒ *'s prog* ⇒ *'s prog* ((*0FOR* -;/ -;/ -/ *DO* -/ *ROF*))
-Event :: [*'a*, *'a*, *'a*] ⇒ (*'l,'k,'s,'s prog option*) *event* ((*EVENT* - *WHEN* - *THEN* - *END*) [*0,0,0*] *61*)
-Event2 :: [*'a*, *'a*] ⇒ (*'l,'k,'s,'s prog option*) *event* ((*EVENT* - *THEN* - *END*) [*0,0*] *61*)
-Event-a :: [*'a*, *'a*, *'a*] ⇒ (*'l,'k,'s,'s prog option*) *event* (($EVENT_A$ - *WHEN* - *THEN* - *END*) [*0,0,0*] *61*)

translations
´x := a ⇀ *CONST Basic ´*(*-update-name x* (λ-. *a*))
IF b THEN c1 ELSE c2 FI ⇀ *CONST Cond* ⦃*b*⦄ *c1 c2*
IF b THEN c FI ⇌ *IF b THEN c ELSE SKIP FI*
WHILE b DO c OD ⇀ *CONST While* ⦃*b*⦄ *c*
AWAIT b THEN c END ⇌ *CONST Await* ⦃*b*⦄ *c*

ATOMIC c END ⇌ *AWAIT CONST True THEN c END*
WAIT b END ⇌ *AWAIT b THEN SKIP END*
FOR a; *b*; *c DO p ROF* ⇀ *a*;; *WHILE b DO p*;;*c OD*

EVENT l WHEN g THEN bd END ⇀ *CONST BasicEvent* (*l*,({|*g*|}, *W*(*bd*)))
EVENT l THEN bd END ⇌ *EVENT l WHEN CONST True THEN bd END*
$EVENT_A$ *l WHEN g THEN bd END* ⇌ *EVENT l THEN* (*AWAIT g THEN bd END*) *END*

Translations for variables before and after a transition:

syntax
-before :: *id* ⇒ *'a* (ᵒ-)
-after :: *id* ⇒ *'a* (ᵃ-)

translations
ᵒ*x* ⇌ *x* ´*CONST fst*
ᵃ*x* ⇌ *x* ´*CONST snd*

print-translation ‹

```
let
  fun quote-tr' f (t :: ts) =
        Term.list-comb (f $ Syntax-Trans.quote-tr' @{syntax-const -antiquote} t, ts)
    | quote-tr' - - = raise Match;

  val assert-tr' = quote-tr' (Syntax.const @{syntax-const -Assert});

  fun bexp-tr' name ((Const (@{const-syntax Collect}, -) $ t) :: ts) =
        quote-tr' (Syntax.const name) (t :: ts)
    | bexp-tr' - - = raise Match;

  fun assign-tr' (Abs (x, -, f $ k $ Bound 0) :: ts) =
      quote-tr' (Syntax.const @{syntax-const -Assign} $ Syntax-Trans.update-name-tr' f)
          (Abs (x, dummyT, Syntax-Trans.const-abs-tr' k) :: ts)
    | assign-tr' - = raise Match;
in
 [(@{const-syntax Collect}, K assert-tr'),
  (@{const-syntax Basic}, K assign-tr'),
  (@{const-syntax Cond}, K (bexp-tr' @{syntax-const -Cond})),
  (@{const-syntax While}, K (bexp-tr' @{syntax-const -While}))]
end
```
›

lemma *colltrue-eq-univ*[*simp*]: {|*True*|} = *UNIV* **by** *auto*

end

18 Formal Specification and Reasoning of messaging system

theory *dmbuscase*

imports *dmbus HOL.Real*
../SIMP/picore-SIMP-Syntax
../SIMP/picore-SIMP

begin

18.1 Config

record *'a buffer = data :: 'a option*
datatype *Module = DGPS | Locator | Planner | Chassis | Interactive | Monitor*
definition *write-buffer d* ≡ (|*data = d*|)
definition *read-buffer-data b* ≡ *data b*

typedecl *Btype*

type-synonym *speed = real*
type-synonym *angle = real*

type-synonym *posLng = real*
type-synonym *posLat = real*
type-synonym *dgpsLng = real*
type-synonym *dgpsLat = real*
type-synonym *loactorLng = real*
type-synonym *locatorLat = real*
datatype *msg-type = SINGLE-POINT dgpsLng dgpsLat | PSEU-DIFF dgpsLng dgpsLat*
| RTK-FIX dgpsLng dgpsLat | RTK-FLOAT dgpsLng dgpsLat
|HIGH-PREC loactorLng locatorLat | LOW-PREC loactorLng locatorLat | LOSS
| Manual | Auto
| Destination posLng posLat | Operate-Steer | Operate-Autopilot
| CtrlCMD real real
| Paths (real × real) list
| Order real real
| AV-status angle speed

datatype *Buffer = dgps-buf | locator-buf | AV-status-buf | planner-buf |control-mode-buf |order-buf*
| interactive-buf|path-buf
|ex1-buf | ex2-buf |ex3-buf | ex4-buf

type-synonym *Mset = Module set*
datatype *Parameter = Locator-data | DGPS-data | Interactive-data | Planner-data | CtrlOrder*
| Control-mode|Path | BufP Buffer | MSet Mset

datatype *EL = System-InitE| LocatorE | PlannerE | ChassisE | DGPSE | InteractiveE |CtrlModeE*
| MonitorE | OrderCtrlE |PATHE|CreateE|RemoveE | Buffer-InitE

type-synonym *EventLabel* = *EL* × (*Parameter list* × *Module*)
definition *get-evt-label* :: *EL* ⇒ *Parameter list* ⇒ *Module* ⇒ *EventLabel* (- - @ - [*0*,*0*,*0*] *20*)
where *get-evt-label el ps k* ≡ (*el*,(*ps*,*k*))

definition *allbuf* ≡ { *dgps-buf* , *locator-buf* , *AV-status-buf* , *planner-buf* ,*control-mode-buf*

,*order-buf* ,*interactive-buf*,*path-buf* ,*ex1-buf* , *ex2-buf* ,*ex3-buf* , *ex4-buf*}

datatype *BufMode* = *IDLE* | *USED*

consts *status* :: *Module* ⇒ *bool*

18.2 State

record *State* = *buf-writer* :: *Buffer* ⇒ *Module option*
buf-readers :: *Buffer* ⇒ *Module set*
bufset :: *Buffer set*
buf-msg :: *Buffer* ⇒ *msg-type buffer*

axiomatization *sysconf* :: *Module* ⇒(*Buffer*,*Buffer*) *Config*
where *sysconfstb* : ∀ *m1 m2 b* .*b* ∈ *writer* (*sysconf m1*) ⟶ *b* ∉ *writer* (*sysconf m1*)
and *writerstb'* : ∀ *s sys* .{*b*. *buf-writer s b* = *Some*(*sys*)} ⊆ *writer* (*sysconf sys*)
and *readerstb'* :∀ *s sys* .{*b*. *sys* ∈ *buf-readers s b*} ⊆ *readers* (*sysconf sys*)

definition *bufstatus*:: *State* ⇒ *Module* ⇒ *Buffer* ⇒ *bool*
where *bufstatus s m b* ≡ (*if b* ∉ (*bufset s*) ∧ *b*∈ *writer* (*sysconf m*) *then True else False*)

fun *buf-writer'*::*Buffer* ⇒ *Module option*
where *buf-writer' dgps-buf* = *Some DGPS* |
buf-writer' locator-buf = *Some Locator* |
buf-writer' AV-status-buf = *Some Chassis*|
buf-writer' planner-buf = *Some Planner* |
buf-writer' control-mode-buf = *Some Chassis* |
buf-writer' order-buf = *Some Chassis*|
buf-writer' interactive-buf = *Some Interactive* |
buf-writer' path-buf = *Some Planner* |
buf-writer' ex1-buf = *None* |
buf-writer' ex2-buf = *None* |
buf-writer' ex3-buf = *None* |
buf-writer' ex4-buf = *None*

fun *buf-readers'*::*Buffer* ⇒ *Module set*
where *buf-readers' dgps-buf* = {*Locator*} |
buf-readers' locator-buf = {*Planner*} |

buf-readers′ AV-status-buf = {*Planner*}|
buf-readers′ planner-buf = {*Chassis*} |
buf-readers′ control-mode-buf = {*Chassis*} |
buf-readers′ order-buf = {*Planner*}|
buf-readers′ interactive-buf = {*Planner*, *Chassis*} |
buf-readers′ path-buf = {*Planner*} |
buf-readers′ ex1-buf = {} |
buf-readers′ ex2-buf = {} |
buf-readers′ ex3-buf = {} |
buf-readers′ ex4-buf = {}

definition *local-vars* ≡ λ*s m* .*True*

interpretation *dmsg-bus ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI commit-pI*
rghoare-pI sysconf buf-writer buf-readers buf-msg local-vars bufset
apply(*simp add*:*allbuf-def event-hoare-axioms dmsg-bus-def*)
using *dmsg-bus-axioms-def sysconfstb writerstb′ readerstb′*
by *metis*

18.3 functional specification

definition *createNode* :: *State* ⇒*Module* ⇒*Buffer*⇒(*EventLabel*,*Module*,*State*,*State prog option*)*event*
where *createNode s k b* ≡
EVENT CreateE [*BufP b*] @ *k*
WHEN
bufstatus s k b ∧ *status k*
THEN
´*buf-writer* := ´*buf-writer*(*b* := *Some k*);;
´*buf-readers* := ´*buf-readers*(*b* := (*get-conf-rd-set b*));;
´*bufset* := (*insert b* ´*bufset*) ;;
´*buf-msg* :=´*buf-msg*(*b* :=(|*data* = *Some*(*SOME x*:: *msg-type*. *True*)|))
END

definition *removeNode* :: *State*⇒*Module*⇒*Buffer* ⇒(*EventLabel*,*Module*,*State*,*State prog option*) *event*
where *removeNode s k b* ≡
EVENT RemoveE [*BufP b*] @ *k*
WHEN
bufstatus s k b ∧ (´*buf-writer b* = *Some k*)
THEN
´*buf-writer* := ´*buf-writer*(*b* :=*None*);;
´*buf-readers* := ´*buf-readers*(*b* := {});;
´*bufset* := ´*bufset* − {*b*}
END

definition *Buffer-Init* :: *Module* ⇒*Buffer*⇒*Mset*⇒(*EventLabel*,*Module*,*State*,*State prog option*) *event*
  where *Buffer-Init k b ms* ≡
    EVENT Buffer-InitE [*BufP b*,*MSet ms*] @ *k*
    THEN
    ´*buf-writer* := (λ*b*. *Some k*);;
    ´*buf-readers* := (λ*b* .*ms*);;
    ´*bufset* := *insert b* ´*bufset* ;;
    ´*buf-msg* := (λ*b* .⦇*data* = *Some*(*SOME x*:: *msg-type*. *True*)⦈)
    END

definition *System-Init* :: (*State* × (*Module*⇒*Buffer* ⇒*Mset*⇒
                                   (*EventLabel*, *Module*, *State*, *State prog option*)*event*))
  where *System-Init* ≡ (
  ⦇*buf-writer* = (*buf-writer′*),
  buf-readers = (*buf-readers′*),
  bufset = { *dgps-buf* , *locator-buf* , *AV-status-buf* , *planner-buf* ,*control-mode-buf* ,*order-buf* ,*interactive-buf*,*path-buf*},
  buf-msg = (λ*b* .⦇*data* = *None*⦈)
 ⦈,
  (λ*k b ms*. *Buffer-Init k b ms*)
)

axiomatization *confs*:: *State*
  where *msg-type-no-eq* :∀ *b1 b2 s*. *buf-writer s b1* ≠ *buf-writer s b2*⟶ *buf-msg s b1* ≠ *buf-msg s b2*
   and *dgps-buf-type* :∀ *s b* . *b* = *dgps-buf* ⟶(∀ *x y* . *data* ((*buf-msg s*) *b*) = *Some*(*SINGLE-POINT x y*)∨
       data ((*buf-msg s*) *b*) = *Some*(*PSEU-DIFF x y*)∨ *data* ((*buf-msg s*) *b*) = *Some*(*RTK-FIX x y*) ∨
      data ((*buf-msg s*) *b*) = *Some*(*RTK-FLOAT x y*))
  and *locator-buf-type* : ∀ *s b*. *b* = *locator-buf* ⟶(∀ *x y*. *data* ((*buf-msg s*) *b*) = *Some*(*HIGH-PREC x y*)
      ∨ *data* ((*buf-msg s*) *b*) = *Some*(*LOW-PREC x y*)∨ *data* ((*buf-msg s*) *b*) = *Some LOSS*)
  and *interactive-buf-type* : ∀ *s b* . *b* = *interactive-buf* ⟶
   (∀ *x y* . *data* ((*buf-msg s*) *b*) = *Some*(*Destination x y*) ∨ *data* ((*buf-msg s*) *b*) = *Some*(*Operate-Steer*)
    ∨ *data* ((*buf-msg s*) *b*) = *Some Operate-Autopilot*)
  and *control-mode-buf-type* :∀ *s b* . *b* = *control-mode-buf* ⟶(*data* ((*buf-msg s*) *b*) = *Some*(*Manual*)
    ∨ *data* ((*buf-msg s*) *b*) = *Some Auto*)
and *planner-buf-type* : ∀ *s b* . *b* = *planner-buf* ⟶(∀ *x y*. *data* ((*buf-msg s*) *b*) = *Some*(*CtrlCMD x y*))
and *order-buf-type* : ∀ *s b* . *b* = *order-buf* ⟶(∀ *x y* . *data* ((*buf-msg s*) *b*) = *Some*(*Order x y*))
and *av-status-buf-type* :∀ *s b* . *b* = *AV-status-buf* ⟶

$(\forall x\ y\ .\ data\ ((buf\text{-}msg\ s)\ b) = Some(AV\text{-}status\ x\ y)\ )$
and *path-type* $:\forall\ s\ b\ .\ b = path\text{-}buf\ \longrightarrow(\ \forall\ x\ .\ data\ ((buf\text{-}msg\ s)\ b) = Some(Paths\ x\ )\ )$

definition *Module-exec* ::*Module* ⇒ *Buffer* ⇒*EL*⇒(*EventLabel*, *Module*, *State*,*State prog option*)*event*
where *Module-exec k b el*≡
EVENT el [*BufP b*] @ *k*
THEN
´*buf-msg* := ´*buf-msg*(*b* := (|*data* = *Some*(*SOME x*::*msg-type*. *True*)|))
END

definition *DGPS-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *DGPS-exec* ≡ *Module-exec DGPS dgps-buf DGPSE*

definition *Interactive-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*

where *Interactive-exec* ≡ *Module-exec Interactive interactive-buf InteractiveE*

definition *path-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *path-exec* ≡ *Module-exec Planner path-buf PATHE*

definition *monitor-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *monitor-exec* ≡ *Module-exec Chassis AV-status-buf MonitorE*

definition *orderctrl-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *orderctrl-exec* ≡ *Module-exec Chassis order-buf OrderCtrlE*

definition *locator-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *locator-exec* ≡
EVENT LocatorE [] @ *Locator*
THEN
IF data (´*buf-msg dgps-buf*) = *None THEN*
´*buf-msg* := ´*buf-msg*(*locator-buf* := (|*data* = *Some LOSS*|))
ELSE
´*buf-msg* := ´*buf-msg*(*locator-buf* := (|*data* =
Some (*case the* (*data* (´*buf-msg dgps-buf*)) *of RTK-FIX x y* ⇒
(*HIGH-PREC x y*)|
PSEU-DIFF x y ⇒ (*LOW-PREC x*
y) |
SINGLE-POINT x y ⇒ *LOSS* |
RTK-FLOAT x y ⇒ (*LOW-PREC*
x y))
|))
FI
END

definition *planner-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *planner-exec* $\equiv$
EVENT PlannerE [] @ *Planner*
THEN
IF (*data* (´*buf-msg locator-buf*) = *None*) *THEN*
´*buf-msg* := ´*buf-msg* (*planner-buf* :=(| *data* = *Some*(*CtrlCMD 0 0*)|))
ELSE
´*buf-msg* :=´*buf-msg* (*planner-buf* :=
(| *data* = *Some* (*CtrlCMD* (*SOME x*::*real*. *True*) (*SOME x*::*real*. *True*))|))

FI
END

definition *ctrlmode-exec* :: (*EventLabel*, *Module*, *State*,*State prog option*) *event*
where *ctrlmode-exec* $\equiv$
EVENT CtrlModeE [] @ *Chassis*
THEN
IF (*data* (´*buf-msg control-mode-buf*)) = *Some Manual THEN*
IF (*data* (´*buf-msg interactive-buf*)) = *Some Operate-Autopilot*
THEN
´*buf-msg* := ´*buf-msg*(*control-mode-buf*:=(| *data* = *Some Auto*|))
FI
ELSE
IF (*data* (´*buf-msg interactive-buf*)) = *Some Operate-Steer*
THEN
´*buf-msg* := ´*buf-msg* (*control-mode-buf*:=(| *data* = *Some Auto*|))
FI
FI
END

18.4 Rely-guarantee condition of events

abbreviation *dgps-post* $\equiv$ {| ´*buf-msg dgps-buf* = (| *data* = *Some*(*SOME x*::*msg-type*. *True*)|) |}
abbreviation *interactive-post* $\equiv$ {|´*buf-msg interactive-buf* = (| *data* = *Some*(*SOME x*:: *msg-type*. *True*)|)|}
abbreviation *locator-post* $\equiv$ {| *data* (´*buf-msg locator-buf*) = *Some*(*SOME x*:: *msg-type*. *True*) $\vee$
data (´*buf-msg locator-buf*) = *Some*(*LOSS*) |}

abbreviation *path-post* $\equiv$ {|´*buf-msg path-buf* = (| *data* = *Some*(*SOME x*:: *msg-type*. *True*)|) |}
abbreviation *planner-post* $\equiv$ {| $\exists$ *x y* . ´*buf-msg planner-buf* = (| *data* = *Some*(*CtrlCMD x y*)|)|}
abbreviation *ctrlmode-post* $\equiv${|((*data* (´*buf-msg control-mode-buf*) = *Some Auto*) $\vee$
(*data* (´*buf-msg control-mode-buf*) = *Some Manual*))|}

abbreviation *monitor-post* $\equiv\{\!|$ $(\exists\, x\ y.\ data\ (\acute{}buf\text{-}msg\ AV\text{-}status\text{-}buf) = Some\ (AV\text{-}status\ x\ y\ ))$ $|\!\}$
abbreviation *orderctrl-post* $\equiv\{\!|$ $(\exists\, x\, y.\ data\ (\acute{}buf\text{-}msg\ order\text{-}buf) = Some\ (Order\ x\ y)\ )$ $|\!\}$
abbreviation *chassis-post* $\equiv$ (*ctrlmode-post* $\cup$ *monitor-post* $\cup$ *orderctrl-post*)

definition *DGPS-exec-RGCond* :: *State* $\Rightarrow$(*State*) *rgformula*
  where *DGPS-exec-RGCond s* $\equiv$
    RG[$\{\!|$ *True* $|\!\}$,
    (*assm-bufs-stb-sys s DGPS* $\cup$ *Id*),
    (*guar-bufs-stb-sys s DGPS* $\cup$ *Id*),
    dgps-post]

definition *Interactive-exec-RGCond* :: *State* $\Rightarrow$(*State*) *rgformula*
  where *Interactive-exec-RGCond s* $\equiv$
    RG[$\{\!|$ *True* $|\!\}$,
    (*assm-bufs-stb-sys s Interactive* $\cup$ *Id*),
    (*guar-bufs-stb-sys s Interactive* $\cup$ *Id*),
    interactive-post]

definition *Locator-exec-RGCond* :: *State* $\Rightarrow$(*State*) *rgformula*
  where *Locator-exec-RGCond s* $\equiv$
    RG[$\{\!|$ *True* $|\!\}$,
    (*assm-bufs-stb-sys s Locator* $\cup$ *Id*),
    (*guar-bufs-stb-sys s Locator* $\cup$ *Id*),
    locator-post]

definition *Planner-exec-RGCond* :: *State* $\Rightarrow$(*State*) *rgformula*
  where *Planner-exec-RGCond s*$\equiv$
    RG[$\{\!|$ *True* $|\!\}$,
    (*assm-bufs-stb-sys s Planner* $\cup$ *Id*),
    (*guar-bufs-stb-sys s Planner* $\cup$ *Id*),
    planner-post
]

definition *Chassis-exec-RGCond* :: *State* $\Rightarrow$(*State*) *rgformula*
  where *Chassis-exec-RGCond s* $\equiv$
    RG[$\{\!|$*True*$|\!\}$,
    (*assm-bufs-stb-sys s Chassis* $\cup$*Id*),
    (*guar-bufs-stb-sys s Chassis* $\cup$ *Id*),
    chassis-post]

definition *createNode-RGCond* :: (*State*) *rgformula*
  where *createNode-RGCond* $\equiv$
    RG[$\{\!|$*True*$|\!\}$,$\{\!|$*True*$|\!\}$,
$\{\!|\ {}^{\mathrm{o}}buf\text{-}msg = {}^{\mathrm{a}}buf\text{-}msg|\!\}$,$\{\!|$*True*$|\!\}$]

definition *removeNode-RGCond* :: (*State*) *rgformula*
  where *removeNode-RGCond* ≡
    RG[⦃*True*⦄,⦃*True*⦄,
⦃$^{\circ}$*buf-msg* = ^{a}*buf-msg*⦄,⦃*True*⦄]

type-synonym *sevent* = (*EL* × *Parameter list* × *Module*, *Module*, *State*, *State prog option*) *event*
type-synonym *srgfformula-e* = *State rgformula*
type-synonym *event-rgf* = (*EL* × *Parameter list* × *Module*, *Module*, *State*, *State prog option*) *event*
× *State rgformula*
type-synonym *post* = *State set*

definition *RGCond* :: *State* ⇒*Module* ⇒*post* ⇒(*State*) *rgformula* (*RGCond*[-,-,-] [*97*,*97*,*97*] *96*)
  where *RGCond s m pst* ≡
    RG[⦃*True*⦄,
    (*assm-bufs-stb-sys s m* ∪*Id*),
    (*guar-bufs-stb-sys s m* ∪ *Id*),
    pst]

definition *RGF* :: *sevent* ⇒ *srgfformula-e* ⇒ *event-rgf* (*RGF*[-,-] [*95*,*95*] *94*)
  where *RGF evt rgf* = (*evt*,*rgf*)

definition *create-RGF* :: *State* ⇒*Module* ⇒ *Buffer*
⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *create-RGF s k b* ≡ (*createNode s k b*,*createNode-RGCond*)

definition *remove-RGF* :: *State* ⇒*Module* ⇒ *Buffer*
⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *remove-RGF s k b* ≡ (*removeNode s k b*,*removeNode-RGCond*)

definition *DGPS-exec-RGF′* :: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *DGPS-exec-RGF′ s* ≡ (*DGPS-exec*,*DGPS-exec-RGCond s*)

definition *Interactive-exec-RGF′*:: *State* ⇒(*EventLabel*,*Module*,*State*,*State prog option*) *rgformula-e*
  where *Interactive-exec-RGF′ s* ≡ (*Interactive-exec*,*Interactive-exec-RGCond s*)

definition *Locator-exec-RGF′* :: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *Locator-exec-RGF′ s* ≡ (*locator-exec*,*Locator-exec-RGCond s*)

definition *Planner-exec-RGF′* :: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog*

option) *rgformula-e*
  where *Planner-exec-RGF′ s* ≡ (*planner-exec*,*Planner-exec-RGCond s*)

definition *Ctrlmode-RGF′* :: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *Ctrlmode-RGF′ s* ≡ (*ctrlmode-exec*,*Chassis-exec-RGCond s*)

definition *Monitor-RGF′* :: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *Monitor-RGF′ s*≡ (*monitor-exec*,*Chassis-exec-RGCond s*)

definition *Orderctrl-exec-RGF′* ::*State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-e*
  where *Orderctrl-exec-RGF′ s* ≡ (*orderctrl-exec*,*Chassis-exec-RGCond s*)

definition *EvtSys-on-RGF* ::
  State ⇒*Module* ⇒*event-rgf set* ⇒ *post*⇒
  (*EventLabel*,*Module*, *State*,*State prog option*) *rgformula-es* (*EvtSysRGF*[-,-,-,-] [*93*,*93*,*93*] *92*)
  where *EvtSys-on-RGF s m evt-rg pst* ≡
      (*rgf-EvtSys* (*evt-rg*),
    RG[⦃ *True* ⦄,
    (*assm-bufs-stb-sys s m* ∪ *Id*),
    (*guar-bufs-stb-sys s m* ∪ *Id*),
    pst]
)

definition *EvtSys-on-Chassis-RGF*::*State* ⇒(*EventLabel*,*Module*, *State*,*State prog option*) *rgformula-es*
  where *EvtSys-on-Chassis-RGF s* ≡
      (*rgf-EvtSys* (
  {*RGF*[*ctrlmode-exec*,*RGCond*[*s*,*Chassis*,*chassis-post*]]}
  ∪ {*RGF*[*monitor-exec*,*RGCond*[*s*,*Chassis*,*chassis-post*]]}
  ∪ {*RGF*[*orderctrl-exec*,*RGCond*[*s*,*Chassis*,*chassis-post*]]}
  ),
      RG[⦃*True*⦄,
    (*assm-bufs-stb-sys s Chassis* ∪*Id*),
    (*guar-bufs-stb-sys s Chassis* ∪ *Id*),
    chassis-post]
)

definition *EvtSys-on-Monitor-RGF* :: *State*⇒ *Module* ⇒
          (*EventLabel*,*Module*, *State*,*State prog option*) *rgformula-es*
  where *EvtSys-on-Monitor-RGF s k* ≡
      (*rgf-EvtSys* (
($\bigcup$ *b*.{*create-RGF s k b*}) ∪ ($\bigcup$ *b*.{*remove-RGF s k b*})

),
$RG[\{\!| True |\!\},\{\!| True |\!\},$
$\{\!|{}^{\mathrm{o}}buf\text{-}msg = {}^{\mathrm{a}}buf\text{-}msg|\!\},\{\!| True |\!\}]$
)

definition *L4-Spec*:: *State* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-par*
where *L4-Spec* ≡ λ*s k*.
case k of DGPS ⇒ *EvtSysRGF*[*s*,*DGPS*,{*RGF*[*DGPS-exec*,*RGCond*[*s*,*DGPS*,*dgps-post*]]},*dgps-post*]
| *Interactive* ⇒ *EvtSysRGF*[*s*,*Interactive*,
{*RGF*[*Interactive-exec*,*RGCond*[*s*,*Interactive*,*interactive-post*]]},*interactive-post*]
| *Locator* ⇒ *EvtSysRGF*[*s*,*Locator*,
{*RGF*[*locator-exec*,*RGCond*[*s*,*Locator*,*locator-post*]]},*locator-post*]
| *Planner* ⇒*EvtSysRGF*[*s*,*Planner*,
{*RGF*[*planner-exec*,*RGCond*[*s*,*Planner*,*planner-post*]]},*planner-post*]
| *Chassis* ⇒ *EvtSys-on-Chassis-RGF s*
| *Monitor* ⇒ *EvtSys-on-Monitor-RGF s k*

definition *L4-Spec′*:: *State* ⇒*Module* ⇒(*EventLabel*, *Module*, *State*,*State prog option*) *rgformula-es*
where *L4-Spec′* ≡ λ*s k* .
case k of DGPS ⇒ *EvtSysRGF*[*s*,*DGPS*,{*RGF*[*DGPS-exec*,*RGCond*[*s*,*DGPS*,*dgps-post*]]},*dgps-post*]

| *Interactive* ⇒ *EvtSysRGF*[*s*,*Interactive*,{*RGF*[*Interactive-exec*,
RGCond[*s*,*Interactive*,*interactive-post*]]},*interactive-post*]
| *Locator* ⇒ *EvtSysRGF*[*s*,*Locator*,{*RGF*[*locator-exec*,*RGCond*[*s*,*Locator*,*locator-post*]]},*locator-post*]

| *Planner* ⇒*EvtSysRGF*[*s*,*Planner*,{*RGF*[*planner-exec*,*RGCond*[*s*,*Planner*,*planner-post*]]},*planner-post*]

| *Chassis* ⇒ *EvtSys-on-Chassis-RGF s*
| *Monitor* ⇒*EvtSys-on-Monitor-RGF s k*

consts *s0*::*State set*
definition *s0-witness*::*State set*
where *s0-witness* ≡ {*fst* (*System-Init*)}

specification (*s0*)
s0-init: *s0* ≡ {*fst* (*System-Init*)}
by *simp*

18.5 some lemma

definition *Moduleset* ≡ {*DGPS* , *Locator* , *Planner* , *Chassis* , *Interactive* }
definition *buf-init* ≡ { *dgps-buf* , *locator-buf* , *AV-status-buf* ,
planner-buf ,*control-mode-buf* ,*order-buf* ,*interactive-buf*,*path-buf*}

lemma *buf-set-init* :∃ *s* .*s* = *s0* ⟶*buf-set s* =

{ *dgps-buf* , *locator-buf* , *AV-status-buf* , *planner-buf* ,*control-mode-buf* ,*order-buf*
,
interactive-buf,*path-buf*}
by *auto*

lemma *buf-no* :$\forall$ *s b* .$\exists$ *m* .*the* (*buf-writer s b*) $\neq$ *m* $\longrightarrow$ *b* $\notin$ *get-wrt-bufs s m*
by *auto*

lemma *buf-writer-same* :$\exists$ *b1 b2 s* . *b1*$\in$ *bufset s* $\wedge$ *b2* $\in$ *bufset s* $\wedge$ *b1* $\neq$ *b2*
$\longrightarrow$ *buf-writer s b1* = *buf-writer s b2*
by *auto*

lemma *msg-type-eq* : $\forall$ *s t* .$\exists$ *b1 b2* .*b1* $\neq$ *b2* $\longrightarrow$ *buf-msg s b1* = *buf-msg t b2*
by *blast*

lemma *msg-type-noteq* : $\forall$ *s t* .$\exists$ *b1 b2* .*b1* $\neq$ *b2* $\longrightarrow$ *buf-msg s b1* $\neq$ *buf-msg t b2*
by *blast*

lemma *msgeq* :$\forall$ *s t* .$\exists$ *b m* .*buf-writer s b* = *m* $\longrightarrow$ *buf-msg s b* = *buf-msg t b*
by (*metis option.distinct*(*1*))

lemma *msgneq* :$\forall$ *s t* .$\exists$ *b m* .*buf-writer s b* $\neq$ *m* $\longrightarrow$ *buf-msg s b* = *buf-msg t b*
by *auto*

18.6 Functional correctness by rely guarantee proof

18.6.1 event

lemma *Create-satRG*: ($\forall$ *s k b* .Γ (*createNode s k b*) $\vdash$ *createNode-RGCond*)
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*: *createNode-def createNode-RGCond-def bufstatus-def*)
apply *auto*[*1*]
using *sysconfstb* **apply** *auto*[*1*]
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f-*def* $Post_f$-*def guard-def* $Rely_f$-*def* $Guar_f$-*def getrgformula-def*)
apply (*simp add*: *Emptyprecond*)
apply(*simp add*: *SIMP-hoare.stable-def* Pre_f-*def getrgformula-def* $Rely_f$-*def*)
apply (*simp add*: *PiCore-Hoare.stable-def*)
apply(*simp add*:*stable-def* Pre_f-*def* $Rely_f$-*def getrgformula-def* $Guar_f$-*def guar-bufs-stb-sys-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply (*simp add*: *Emptyprecond guard-def*)
apply (*simp add*: *PiCore-Hoare.stable-def* Pre_f-*def getrgformula-def* $Rely_f$-*def*)
apply(*simp add*:*stable-def* Pre_f-*def* $Rely_f$-*def getrgformula-def* $Guar_f$-*def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply (*simp add*: *Emptyprecond guard-def*)
apply (*simp add*: *PiCore-Hoare.stable-def* Pre_f-*def getrgformula-def* $Rely_f$-*def*)
apply(*simp add*:*stable-def* Pre_f-*def* $Rely_f$-*def getrgformula-def* $Guar_f$-*def*)

done

lemma *Remove-satRG*: ($\forall s\ k\ b$.Γ (*removeNode s k b*) ⊢ *removeNode-RGCond*)
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*: *removeNode-def removeNode-RGCond-def bufstatus-def*)
apply *auto*[*1*]
using *sysconfstb* **apply** *auto*[*1*]
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply (*simp add*: *Emptyprecond*)
apply(*simp add*: *SIMP-hoare.stable-def* Pre_f*-def getrgformula-def* $Rely_f$*-def*)
apply (*simp add*: *PiCore-Hoare.stable-def*)
apply(*simp add*:*stable-def* Pre_f*-def* $Rely_f$*-def getrgformula-def* $Guar_f$*-def guar-bufs-stb-sys-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply (*simp add*: *Emptyprecond guard-def*)
apply (*simp add*: *PiCore-Hoare.stable-def* Pre_f*-def getrgformula-def* $Rely_f$*-def*)
apply(*simp add*:*stable-def* Pre_f*-def* $Rely_f$*-def getrgformula-def* $Guar_f$*-def*)
done

lemma *dgps-2*: $\bigwedge s\ x\ y.$
buf-msg x dgps-buf = (|*data* = *Some* (*SOME x. True*)|) ⟹
$\forall b'$. *buf-writer s b'* = *Some DGPS* ∧ *x* = *s* ∧ *buf-writer y* = *buf-writer x* ∧
buf-readers y = *buf-readers x* ⟶*buf-msg s b'* = *buf-msg y b'* ⟹
buf-msg y dgps-buf = (|*data* = *Some* (*SOME x. True*)|)
proof −
fix *s* :: *'a State-scheme* **and** *x* :: *'a State-scheme* **and** *y* :: *'b State-scheme*
have *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,*bufset* = *buf-init*,
buf-msg = (λ*b*. (|*data* = *None*|)) |) *interactive-buf* ≠
buf-writer (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,*bufset* = *buf-init*,
buf-msg = (λ*b*. (|*data* = *None*|)) |) *dgps-buf*
by (*metis* (*no-types*) *Module.simps*(*7*) *State.select-convs*(*1*)
buf-writer'.simps(*1*) *buf-writer'.simps*(*7*) *option.inject*)
then show *buf-msg y dgps-buf* = (|*data* = *Some* (*SOME x. True*)|)
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *DGPS-satRG*: (∀ *s* .Γ *DGPS-exec* ⊢ *DGPS-exec-RGCond s*)
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*: *DGPS-exec-def DGPS-exec-RGCond-def*)
apply *auto*[*1*]
apply(*simp add*:*Module-exec-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)

apply(*rule Basic*)
apply *simp*
apply *auto*
apply(*simp add*:*guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *dgps-2* **apply** *force*
apply(*simp add*:*SIMP-hoare.stable-def*)+
apply(*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*
using *dgps-2* **apply** *blast*
apply(*simp add*: *SIMP-hoare.stable-def* Pre_f*-def getrgformula-def* $Rely_f$*-def*)
apply (*simp add*: *PiCore-Hoare.stable-def*)
apply(*simp add*:*stable-def* Pre_f*-def* $Rely_f$*-def getrgformula-def* $Guar_f$*-def guar-bufs-stb-sys-def*)
done

lemma *Interactive-satRG*: (∀ *s*. Γ *Interactive-exec* ⊢ *Interactive-exec-RGCond s*)
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*: *Interactive-exec-def Interactive-exec-RGCond-def Module-exec-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Basic*)
apply(*simp add*:*SIMP-hoare.stable-def*)+ **apply** *auto*[*1*]
apply(*simp add*:*guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*metis* (*no-types*) *State.select-convs*(*4*) *dgps-2*)
apply(*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*SIMP-hoare.stable-def*)+ **apply** *auto*
apply(*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply (*metis* (*no-types*) *State.select-convs*(*4*) *dgps-2*)
apply(*simp add*: *SIMP-hoare.stable-def* Pre_f*-def getrgformula-def* $Rely_f$*-def*)
apply (*simp add*: *PiCore-Hoare.stable-def*)
apply(*simp add*:*stable-def* Pre_f*-def* $Rely_f$*-def getrgformula-def* $Guar_f$*-def guar-bufs-stb-sys-def*)
done

lemma *locator-1*:⋀*s*. *s* ≠ *s*⦇*buf-msg* := (*buf-msg s*)(*locator-buf* := ⦇*data* = *Some LOSS*⦈)⦈ ⟹
data (*buf-msg s dgps-buf*) = *None* ⟹ *buf-writer s locator-buf* ≠
Some Locator ⟹ *buf-msg s locator-buf* = ⦇*data* = *Some LOSS*⦈
by (*metis* (*mono-tags*, *lifting*) *State.select-convs*(*4*) *control-mode-buf-type fun-upd-same option.simps*(*3*))

lemma *locator-3*:⋀*s x y*.
data (*buf-msg x locator-buf*) = *Some LOSS* ⟹

$\forall b'.$ *buf-writer s* b' = *Some Locator* $\wedge$ $x = s$ $\wedge$ *buf-writer y* = *buf-writer x* $\wedge$
buf-readers y = *buf-readers x* $\longrightarrow$ *buf-msg s* b' = *buf-msg y* b' $\Longrightarrow$
data (*buf-msg y locator-buf*) $\neq$ *Some LOSS* $\Longrightarrow$
data (*buf-msg y locator-buf*) = *Some* (*SOME x. True*)

proof –
fix *s* :: *'a State-scheme* **and** *x* :: *'a State-scheme* **and** *y* :: *'b State-scheme*
have *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = ($\lambda b.$ (|*data* = *z*|)) |) *dgps-buf* $\neq$
buf-writer (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = ($\lambda b.$ (|*data* = *z*|)) |) *locator-buf*
by (*metis* (*no-types*) *Module.distinct*(*1*) *State.select-convs*(*1*) *buf-writer'.simps*(*1*)

buf-writer'.simps(*2*) *option.inject*)
then show *data* (*buf-msg y locator-buf*) = *Some* (*SOME x. True*)
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *locator-4*:$\bigwedge x\ y.$ *data* (*buf-msg x dgps-buf*) = *Some y* $\Longrightarrow$
(*case y of SINGLE-POINT x y* $\Rightarrow$ *LOSS* | *PSEU-DIFF x xa* $\Rightarrow$ *LOW-PREC*
x xa
| *RTK-FIX x xa* $\Rightarrow$ *HIGH-PREC x xa*
| *RTK-FLOAT x xa* $\Rightarrow$ *LOW-PREC x xa*) $\neq$
LOSS $\Longrightarrow$
(*case y of SINGLE-POINT x y* $\Rightarrow$ *LOSS* | *PSEU-DIFF x xa* $\Rightarrow$ *LOW-PREC*
x xa
| *RTK-FIX x xa* $\Rightarrow$ *HIGH-PREC x xa*
| *RTK-FLOAT x xa* $\Rightarrow$ *LOW-PREC x xa*) =
(*SOME x. True*)

proof –
fix *x* :: *'a State-scheme* **and** *y* :: *msg-type*
have *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = ($\lambda b.$ (|*data* = *z*|)) |)*dgps-buf* $\neq$
buf-writer (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = ($\lambda b.$ (|*data* = *z*|)) |)*locator-buf*
by (*metis Module.distinct*(*1*) *State.select-convs*(*1*)
buf-writer'.simps(*1*) *buf-writer'.simps*(*2*) *option.inject*)
then show (*case y of SINGLE-POINT x y* $\Rightarrow$ *LOSS* | *PSEU-DIFF x xa* $\Rightarrow$
LOW-PREC x xa |
RTK-FIX x xa $\Rightarrow$ *HIGH-PREC x xa* | *RTK-FLOAT x xa* $\Rightarrow$
LOW-PREC x xa) = (*SOME x. True*)
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *locator-5*:$\bigwedge s.$ {(*s*, *t*).
s $\in$ – {|*data* (*´buf-msg dgps-buf*) = *None*|} $\wedge$
t = *s*
(|*buf-msg* := (*buf-msg s*)
(*locator-buf* :=

$(\!|$ *data* =
Some
(*case the* (*data* (*buf-msg s dgps-buf*)) *of SINGLE-POINT x y* ⇒ *LOSS*
| *PSEU-DIFF x xa* ⇒ *LOW-PREC x xa*
| *RTK-FIX x xa* ⇒ *HIGH-PREC x xa* | *RTK-FLOAT x xa* ⇒ *LOW-PREC x xa*)$|\!)$)$|\!)$}
⊆ (*guar-bufs-stb-sys s Locator*)$^=$
proof −
fix *s* :: *'a State-scheme*
have *buf-writer* $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*
,*bufset* = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$*dgps-buf* ≠
buf-writer $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$ *locator-buf*
by (*metis Module.distinct*(*1*) *State.select-convs*(*1*)
buf-writer'.simps(*1*) *buf-writer'.simps*(*2*) *option.inject*)
then have *False*
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
then show {(*s*, *t*). *s* ∈ − {|*data* (´*buf-msg dgps-buf*) = *None*|} ∧
t = *s* $(\!|$*buf-msg* := (*buf-msg s*) (*locator-buf* := $(\!|$*data* = *Some* (*case the* (*data* (*buf-msg s dgps-buf*))
of SINGLE-POINT x y ⇒ *LOSS* |
PSEU-DIFF x xa ⇒ *LOW-PREC x xa* |
RTK-FIX x xa ⇒ *HIGH-PREC x xa* |
RTK-FLOAT x xa ⇒ *LOW-PREC x xa*)$|\!)$)$|\!)$} ⊆ (*guar-bufs-stb-sys s Locator*)$^=$
by *metis*
qed

lemma *locator-6*: ⋀*s x y ya*.
data (*buf-msg x dgps-buf*) = *Some y* ⟹
∀ *b'*. *buf-writer s b'* = *Some Locator* ∧ *x* = *s* ∧ *buf-writer ya* = *buf-writer x* ∧
buf-readers ya = *buf-readers x* ⟶ *buf-msg s b'* = *buf-msg ya b'* ⟹
∃ *y*. *data* (*buf-msg ya dgps-buf*) = *Some y*
proof −
fix *s* :: *'a State-scheme* **and** *x* :: *'a State-scheme* **and** *y* :: *msg-type* **and** *ya* :: *'b State-scheme*
have *f1*: *buf-writer* $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$*dgps-buf* ≠
buf-writer $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$*locator-buf*
by (*metis* (*no-types*) *Module.distinct*(*1*) *State.select-convs*(*1*)
buf-writer'.simps(*1*) *buf-writer'.simps*(*2*) *option.inject*)
have *buf-msg* $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$ *dgps-buf* =
buf-msg $(\!|$*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λ*b*. $(\!|$*data* = *z*$|\!)$) $|\!)$*locator-buf*
by (*metis* (*no-types*) *State.select-convs*(*4*))
then show ∃ *y*. *data* (*buf-msg ya dgps-buf*) = *Some y*

using *f1 msg-type-no-eq* **by** *blast*
qed

lemma *locator-7* : $\bigwedge s\ a.\ a \neq a(\!|buf\text{-}msg := (buf\text{-}msg\ a)(locator\text{-}buf := (\!|data = Some\ LOSS|\!))|\!) \Longrightarrow$
$data\ (buf\text{-}msg\ a\ dgps\text{-}buf) = None \Longrightarrow$
$locator\text{-}buf \in bufset\ a \Longrightarrow buf\text{-}writer\ s\ locator\text{-}buf \neq Some\ Locator \Longrightarrow$
$buf\text{-}msg\ a\ locator\text{-}buf = (\!|data = Some\ LOSS|\!)$
proof −
fix $s :: 'a\ State\text{-}scheme$ **and** $a :: 'b\ State\text{-}scheme$
assume *a1*: $data\ (buf\text{-}msg\ a\ dgps\text{-}buf) = None$
have $\forall s.\ data\ (buf\text{-}msg\ (s::'b\ State\text{-}scheme)\ locator\text{-}buf) = Some\ LOSS \vee$
$data\ (buf\text{-}msg\ s\ locator\text{-}buf) \neq None$
by (*metis* (*full-types*) *State.select-convs*(*4*) *buffer.select-convs*(*1*) *locator-3 option.distinct*(*1*))
then show $buf\text{-}msg\ a\ locator\text{-}buf = (\!|data = Some\ LOSS|\!)$
using *a1* **by** (*metis* (*no-types*) *State.select-convs*(*4*) *fun-upd-same option.distinct*(*1*))
qed

lemma *Locator-satRG*: $\forall s\ .\Gamma\ locator\text{-}exec \vdash Locator\text{-}exec\text{-}RGCond\ s$
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*: *locator-exec-def Locator-exec-RGCond-def*)
apply *auto*[*1*]
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Cond*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply (*simp add*:*get-wrt-bufs-def*)
apply(*rule Basic*)
apply *simp*
apply(*simp add*:*guar-bufs-stb-sys-def get-wrt-bufs-def bufs-stb-cpl-def*)
apply *auto*[*1*]
using *locator-7* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*
apply (*metis State.select-convs*(*4*) *control-mode-buf-type fun-upd-same option.simps*(*3*))
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*metis State.select-convs*(*4*) *buffer.select-convs*(*1*) *locator-3*)
using *locator-3* **apply** *blast*
apply(*rule Basic*)
apply *auto*[*1*]
using *locator-4* **apply** *auto*[*1*]

using *locator-5* **apply** *auto*[*1*]
apply(*simp add*: *SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *locator-6* **apply** *blast*
apply(*simp add*: *SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*metis State.select-convs*(*4*) *buffer.select-convs*(*1*) *locator-3*)
using *locator-3* **apply** *auto*[*1*]
apply(*simp add*:*PiCore-Hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:Pre_f*-def* $Rely_f$*-def guar-bufs-stb-sys-def*
bufs-stb-cpl-def get-wrt-bufs-def getrgformula-def)
apply(*simp add*:$Guar_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def*
get-wrt-bufs-def assm-bufs-stb-sys-def bufs-stb-def)
apply (*simp add*: *getrgformula-def*)
done

lemma *planner-1*: $\bigwedge s\ x\ y\ ya.$
data (*buf-msg x locator-buf*) = *Some y* $\Longrightarrow$
$\forall b'$. *buf-writer s b'* = *Some Planner* $\wedge$ *b'* $\in$ *bufset x*
$\longrightarrow$ *buf-msg x b'* = *buf-msg ya b'* $\Longrightarrow$ $\exists y$. *data* (*buf-msg ya locator-buf*) = *Some y*

proof −
fix *s* :: *'a State-scheme* **and** *x* :: *'b State-scheme* **and** *y* :: *msg-type* **and** *ya* :: *'c State-scheme*
have *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = {*dgps-buf*, *locator-buf*, *AV-status-buf*, *planner-buf*, *control-mode-buf*,

order-buf, *interactive-buf*, *path-buf*}, *buf-msg* = λb. (|*data* = *None*|)|) *locator-buf* = *Some Locator*
by (*metis State.select-convs*(*1*) *buf-writer'.simps*(*2*))
then show $\exists y$. *data* (*buf-msg ya locator-buf*) = *Some y*
by (*metis* (*no-types*) *Module.distinct*(*1*) *State.select-convs*(*1*) *State.select-convs*(*4*)

buf-writer'.simps(*1*) *msg-type-no-eq option.inject*)
qed

lemma *planner-2* : $\bigwedge s.\ \forall x.$ ($\exists$ *xa y*. *buf-msg x planner-buf* = (|*data* = *Some* (*CtrlCMD xa y*)|)) $\longrightarrow$
($\forall y$. (($\forall b'$. *buf-writer s b'* = *Some Planner* $\wedge$ *b'* $\in$ *bufset x* $\longrightarrow$ *buf-msg x b'* = *buf-msg y b'*) $\longrightarrow$
($\exists$ *x ya*. *buf-msg y planner-buf* = (|*data* = *Some* (*CtrlCMD x ya*)|)))
$\wedge$
(*x* = *y* $\longrightarrow$ ($\exists$ *x ya*. *buf-msg y planner-buf* = (|*data* = *Some* (*CtrlCMD x ya*)|))))
apply *auto*
proof −
fix *s* :: *'a State-scheme* **and** *x* :: *'b State-scheme* **and** *xa* :: *real* **and** *y* :: *real*
and *ya* :: *'b State-scheme*

have *buf-writer* $(\!|$ *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = {*dgps-buf*, *locator-buf*, *AV-status-buf*, *planner-buf*, *control-mode-buf*,
order-buf, *interactive-buf*, *path-buf*},
buf-msg = $\lambda b.$ $(\!|$ *data* = *None* $|\!)$ $|\!)$ *planner-buf* $\neq$ *buf-writer* $(\!|$ *buf-writer* = *buf-writer'*,
buf-readers = *buf-readers'*,
bufset = {*dgps-buf*, *locator-buf*, *AV-status-buf*, *planner-buf*, *control-mode-buf*,
order-buf, *interactive-buf*, *path-buf*},
buf-msg = $\lambda b.$ $(\!|$ *data* = *None* $|\!)$ $|\!)$ *locator-buf*
by *auto*
then show $\exists$ *x yaa. buf-msg ya planner-buf* = $(\!|$ *data* = *Some* (*CtrlCMD x yaa*) $|\!)$
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *Planner-satRG*: $\forall$ *s* .Γ *planner-exec* $\vdash$ *Planner-exec-RGCond s*
apply(*simp add*:*Evt-sat-RG-def*)
apply (*rule allI*)
apply(*simp add*: *planner-exec-def Planner-exec-RGCond-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def*
getrgformula-def)
apply(*rule Cond*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def*
get-wrt-bufs-def)
apply (*simp add*:*get-wrt-bufs-def*)
apply(*rule Basic*)
apply *simp*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def*
get-wrt-bufs-def)
apply *auto*[*1*]
apply (*metis State.select-convs*(*4*) *buffer.select-convs*(*1*) *locator-3 op-*
tion.discI)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def*
get-wrt-bufs-def)
defer
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def*
get-wrt-bufs-def)
using *planner-2* **apply** *blast*
apply(*rule Basic*)
apply *auto*
apply(*simp add*:*SIMP-hoare.stable-def guar-bufs-stb-sys-def bufs-stb-cpl-def*
get-wrt-bufs-def)
apply (*simp add*: *planner-buf-type*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*simp add*: *planner-1*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply (*simp add*: *planner-buf-type*)
apply(*simp add*:*PiCore-Hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def*

get-wrt-bufs-def)
  apply(*simp add*:Pre_f*-def* $Rely_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def getrgformula-def*)
 apply(*simp add*:$Guar_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*

assm-bufs-stb-sys-def bufs-stb-def)
 apply (*simp add*: *getrgformula-def*)
 by (*metis locator-buf-type option.distinct*(*1*))

lemma *chassis-1* :$\bigwedge s\ x\ y.$
data (*buf-msg x control-mode-buf*) = *Some Manual* $\Longrightarrow$
$\forall\, b'$. *buf-writer s b'* = *Some Chassis* $\wedge$ *x* = *s* $\wedge$ *buf-writer y* = *buf-writer x* $\wedge$
buf-readers y = *buf-readers x* $\longrightarrow$
buf-msg s b' = *buf-msg y b'* $\Longrightarrow$
data (*buf-msg y control-mode-buf*) = *Some Manual*

proof −
 fix *s* :: *'a State-scheme* **and** *x* :: *'a State-scheme* **and** *y* :: *'b State-scheme*
 have *buf-writer*(|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (|*data* = *z*|)) |) *dgps-buf* $\neq$
buf-writer (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (|*data* =*z*|)) |) *locator-buf*
 by (*metis Module.distinct*(*1*) *State.select-convs*(*1*) *buf-writer'.simps*(*1*) *buf-writer'.simps*(*2*) *option.inject*)
 then show *data* (*buf-msg y control-mode-buf*) = *Some Manual*
  by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *chassis-2* :$\bigwedge s\ a.$ (*a*, *a*(|*buf-msg* := (*buf-msg a*)(*control-mode-buf* := (|*data* = *Some Auto*|))|))
$\notin$ *guar-bufs-stb-sys s Chassis* $\Longrightarrow$
data (*buf-msg a control-mode-buf*) = *Some Manual* $\Longrightarrow$
data (*buf-msg a interactive-buf*) = *Some Operate-Autopilot* $\Longrightarrow$
a = *a*(|*buf-msg* := (*buf-msg a*)(*control-mode-buf* := (|*data* = *Some Auto*|))|)
proof −
 fix *s* :: *'a State-scheme* **and** *a* :: *'a State-scheme*
 have $\forall$ *f fa B fb u*. *buf-writer* (|*buf-writer* = *f*, *buf-readers* = *fa*,*bufset* = *B*,*buf-msg* = *fb*|) = *f*
  by (*metis State.select-convs*(*1*))
 then have *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (|*data* = *z*|)) |)*dgps-buf*
$\neq$ *buf-writer* (|*buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (|*data* = *z*|)) |)*locator-buf*
  by *fastforce*
 then show *a* = *a* (|*buf-msg* := (*buf-msg a*) (*control-mode-buf* := (|*data* = *Some Auto*|))|)
  by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *chassis-3* : $\bigwedge s\ x\ y.$
$data\ (buf\text{-}msg\ x\ control\text{-}mode\text{-}buf) = Some\ Manual \Longrightarrow$
$data\ (buf\text{-}msg\ x\ interactive\text{-}buf) = Some\ Operate\text{-}Autopilot \Longrightarrow$
$\forall\, b'.\ buf\text{-}writer\ s\ b' = Some\ Chassis \land x = s \land buf\text{-}writer\ y = buf\text{-}writer\ x \land$
$buf\text{-}readers\ y = buf\text{-}readers\ x \longrightarrow$
$buf\text{-}msg\ s\ b' = buf\text{-}msg\ y\ b' \Longrightarrow$
$data\ (buf\text{-}msg\ y\ interactive\text{-}buf) = Some\ Operate\text{-}Autopilot$
proof $-$
fix $s :: 'a\ State\text{-}scheme$ **and** $x :: 'a\ State\text{-}scheme$ **and** $y :: 'b\ State\text{-}scheme$
have $\forall f\, fa\, B\, fb\, u.\ buf\text{-}writer\ (\!|buf\text{-}writer = f,\ buf\text{-}readers = fa, bufset = B, buf\text{-}msg = fb|\!) = f$
by $(metis\ (no\text{-}types)\ State.select\text{-}convs(1)\ )$
then have $buf\text{-}writer\ (\!|buf\text{-}writer = buf\text{-}writer',\ buf\text{-}readers = buf\text{-}readers',$
$bufset = buf\text{-}init, buf\text{-}msg = (\lambda b.\ (\!|data = z|\!))\ |\!)\ dgps\text{-}buf \neq$
$buf\text{-}writer\ (\!|buf\text{-}writer = buf\text{-}writer',\ buf\text{-}readers = buf\text{-}readers',$
$bufset = buf\text{-}init, buf\text{-}msg = (\lambda b.\ (\!|data = z|\!))\ |\!) locator\text{-}buf$
by *auto*
then show $data\ (buf\text{-}msg\ y\ interactive\text{-}buf) = Some\ Operate\text{-}Autopilot$
by $(metis\ (no\text{-}types)\ State.select\text{-}convs(4)\ \ msg\text{-}type\text{-}no\text{-}eq)$
qed

lemma *chassis-4* : $\bigwedge s\ a.\ (a,\ a(\!|buf\text{-}msg := (buf\text{-}msg\ a)(control\text{-}mode\text{-}buf := (\!|data = Some\ Auto|\!))|\!))$
$\notin guar\text{-}bufs\text{-}stb\text{-}sys\ s\ Chassis \Longrightarrow$
$data\ (buf\text{-}msg\ a\ control\text{-}mode\text{-}buf) \neq Some\ Manual \Longrightarrow$
$data\ (buf\text{-}msg\ a\ interactive\text{-}buf) = Some\ Operate\text{-}Steer \Longrightarrow$
$a = a(\!|buf\text{-}msg := (buf\text{-}msg\ a)(control\text{-}mode\text{-}buf := (\!|data = Some\ Auto|\!))|\!)$
proof $-$
fix $s :: 'a\ State\text{-}scheme$ **and** $a :: 'a\ State\text{-}scheme$
have $buf\text{-}writer\ (\!|buf\text{-}writer = buf\text{-}writer',\ buf\text{-}readers = buf\text{-}readers',$
$bufset = buf\text{-}init, buf\text{-}msg = (\lambda b.\ (\!|data = z|\!))\ |\!)\ dgps\text{-}buf \neq$
$buf\text{-}writer\ (\!|buf\text{-}writer = buf\text{-}writer',$
$buf\text{-}readers = buf\text{-}readers', bufset = buf\text{-}init, buf\text{-}msg = (\lambda b.\ (\!|data = z|\!))\ |\!)$
$locator\text{-}buf$
by $(metis\ Module.distinct(1)\ State.select\text{-}convs(1)\ \ buf\text{-}writer'.simps(1)$
$buf\text{-}writer'.simps(2)\ option.inject)$
then show $a = a\ (\!|buf\text{-}msg := (buf\text{-}msg\ a)\ (control\text{-}mode\text{-}buf := (\!|data = Some\ Auto|\!))|\!)$
by $(metis\ (no\text{-}types)\ State.select\text{-}convs(4)\ \ msg\text{-}type\text{-}no\text{-}eq)$
qed

lemma *chassis-5*:$\bigwedge s\ x\ y.$
$data\ (buf\text{-}msg\ x\ control\text{-}mode\text{-}buf) \neq Some\ Manual \Longrightarrow$
$data\ (buf\text{-}msg\ x\ interactive\text{-}buf) = Some\ Operate\text{-}Steer \Longrightarrow$
$\forall\, b'.\ buf\text{-}writer\ s\ b' = Some\ Chassis \land x = s \land buf\text{-}writer\ y = buf\text{-}writer\ x \land buf\text{-}readers\ y = buf\text{-}readers\ x \longrightarrow$
$buf\text{-}msg\ s\ b' = buf\text{-}msg\ y\ b' \Longrightarrow$
$data\ (buf\text{-}msg\ y\ interactive\text{-}buf) = Some\ Operate\text{-}Steer$
proof $-$

fix s :: $'a$ *State-scheme* **and** x :: $'a$ *State-scheme* **and** y :: $'b$ *State-scheme*
have *buf-writer* (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |)*dgps-buf* $\neq$
buf-writer (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |) *locator-buf*
by (*metis* (*no-types*) *Module.distinct*(*1*) *State.select-convs*(*1*)
buf-writer'.simps(*1*) *buf-writer'.simps*(*2*) *option.inject*)
then show *data* (*buf-msg y interactive-buf*) = *Some Operate-Steer*
by (*metis* (*no-types*) *State.select-convs*(*4*) *msg-type-no-eq*)
qed

lemma *chassis-6*: $\bigwedge s\ a.\ \exists\, b'.$ *buf-writer s b'* $\neq$ *Some Chassis* $\wedge$
$a = s \wedge$ *buf-msg s b'* $\neq$ (*if b'* = *AV-status-buf then* (| *data* = *Some* (*SOME x. True*) |) *else buf-msg s b'*)
$\Longrightarrow$ $a = a$(| *buf-msg* := (*buf-msg a*)(*AV-status-buf* := (| *data* = *Some* (*SOME x. True*) |)) |)
apply *auto*
by (*metis* (*mono-tags*, *lifting*) *State.ext-inject State.surjective*
av-status-buf-type buffer.equality fun-upd-eqD fun-upd-triv old.unit.exhaust)

lemma *chassis-7* :$\bigwedge s\ a.\ \exists\, b'.$ *buf-writer s b'* $\neq$ *Some Chassis* $\wedge$
$a = s \wedge$ *buf-msg s b'* $\neq$ (*if b'* = *order-buf then* (| *data* = *Some* (*SOME x. True*) |) *else buf-msg s b'*) $\Longrightarrow$
$a = a$(| *buf-msg* := (*buf-msg a*)(*order-buf* := (| *data* = *Some* (*SOME x. True*) |)) |)
proof –
fix s :: $'a$ *State-scheme* **and** a :: $'a$ *State-scheme*
have $\forall$ *f fa B fb u. buf-writer* (| *buf-writer* = *f*, *buf-readers* = *fa*,*bufset* = *B*,*buf-msg* = *fb* |) = *f*
by (*metis State.select-convs*(*1*))
then have *f1*: *buf-writer* (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |)*dgps-buf* $\neq$
buf-writer (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |)*locator-buf*
by *simp*
have $\forall$ *f fa B fb u. buf-writer* (| *buf-writer* = *f*, *buf-readers* = *fa*,*bufset* = *B*,*buf-msg* = *fb* |) = *f*
by *simp*
have *buf-writer* (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |)*dgps-buf* $\neq$
buf-writer (| *buf-writer* = *buf-writer'*, *buf-readers* = *buf-readers'*,
bufset = *buf-init*,*buf-msg* = (λb. (| *data* = z |)) |)*locator-buf*
by *fastforce*
then show $a = a$ (| *buf-msg* := (*buf-msg a*) (*order-buf* := (| *data* = *Some* (*SOME x. True*) |)) |)
using *f1 msg-type-no-eq*
by *fastforce*
qed

18.6.2 event system proof

lemma *Chassis-satRG*: ∀ *s* .Γ *ctrlmode-exec* ⊢ *Chassis-exec-RGCond s* ∧
Γ *monitor-exec* ⊢ *Chassis-exec-RGCond s* ∧
Γ *orderctrl-exec* ⊢ *Chassis-exec-RGCond s*
apply(*simp add*:*Evt-sat-RG-def*)
apply *auto*[*1*]
apply(*simp add*: *ctrlmode-exec-def Chassis-exec-RGCond-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Cond*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply (*simp add*:*get-wrt-bufs-def*)
apply(*rule Cond*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*
using *chassis-1* **apply** *blast*
apply(*rule Basic*)
apply *auto*
apply (*simp add*: *chassis-2*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *chassis-1* **apply** *blast*
using *chassis-3* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*simp add*: *order-buf-type*)+
apply(*simp add* :*Skip-def*)
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*guar-bufs-stb-sys-def get-wrt-bufs-def bufs-stb-cpl-def*)
apply *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *chassis-1* **apply** *blast*
using *chassis-1 chassis-3* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*rule Cond*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *chassis-1* **apply** *blast*
apply(*rule Basic*)
apply *auto*
apply (*simp add*: *chassis-4*)

apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*
using *chassis-1* **apply** *blast*
using *chassis-5* **apply** *auto*[*1*]
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *control-mode-buf-type* **apply** *blast*
apply (*simp add*: *order-buf-type*)+
apply(*simp add* :*Skip-def*)
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*guar-bufs-stb-sys-def get-wrt-bufs-def bufs-stb-cpl-def*)
apply *auto*[*1*]
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *chassis-1* **apply** *blast*
using *chassis-1 chassis-5* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*PiCore-Hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:Pre_f*-def* $Rely_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def*
get-wrt-bufs-def getrgformula-def)
apply(*simp add*:$Guar_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*

assm-bufs-stb-sys-def bufs-stb-def)
apply (*simp add*: *getrgformula-def*)

apply(*simp add*: *monitor-exec-def Chassis-exec-RGCond-def Module-exec-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f*-def* $Post_f$*-def guard-def* $Rely_f$*-def* $Guar_f$*-def*
getrgformula-def)
apply(*rule Basic*)
apply *auto*
apply (*simp add*: *order-buf-type*)
apply(*simp add*:*guar-bufs-stb-sys-def get-wrt-bufs-def bufs-stb-cpl-def*)
using *dgps-2* **apply** *force*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
apply (*simp add*: *order-buf-type*)
using *control-mode-buf-type* **apply** *blast*
using *control-mode-buf-type* **apply** *blast*
using *control-mode-buf-type* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*PiCore-Hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:Pre_f*-def* $Rely_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def*
get-wrt-bufs-def getrgformula-def)
apply(*simp add*:$Guar_f$*-def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*
assm-bufs-stb-sys-def bufs-stb-def)

apply (*simp add*: *getrgformula-def*)

apply(*simp add*: *orderctrl-exec-def Chassis-exec-RGCond-def Module-exec-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*rghoare-pI-def*)
apply(*simp add*:*body-def* Pre_f-*def* $Post_f$-*def guard-def* $Rely_f$-*def* $Guar_f$-*def*
getrgformula-def)
apply(*rule Basic*)
apply *auto*
using *control-mode-buf-type* **apply** *auto*[*1*]
apply(*simp add*:*guar-bufs-stb-sys-def get-wrt-bufs-def bufs-stb-cpl-def*)
using *dgps-2* **apply** *force*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply *auto*[*1*]
using *control-mode-buf-type* **apply** *blast*
using *control-mode-buf-type* **apply** *blast*
using *control-mode-buf-type* **apply** *blast*
using *control-mode-buf-type* **apply** *blast*
apply(*simp add*:*SIMP-hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:*PiCore-Hoare.stable-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*)
apply(*simp add*:Pre_f-*def* $Rely_f$-*def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*
getrgformula-def)
apply(*simp add*:$Guar_f$-*def guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*
assm-bufs-stb-sys-def bufs-stb-def)
apply (*simp add*: *getrgformula-def*)
done

lemma *EvtSys-on-Monitor-SatRG*:
$\forall s\ k.\ \Gamma \vdash fst\ (EvtSys\text{-}on\text{-}Monitor\text{-}RGF\ s\ k\ )\ sat_s$
$[Pre_f\ (snd\ (EvtSys\text{-}on\text{-}Monitor\text{-}RGF\ s\ k\ )),$
$Rely_f\ (snd\ (EvtSys\text{-}on\text{-}Monitor\text{-}RGF\ s\ k\ )),$
$Guar_f\ (snd\ (EvtSys\text{-}on\text{-}Monitor\text{-}RGF\ s\ k\ )),$
$Post_f\ (snd\ (EvtSys\text{-}on\text{-}Monitor\text{-}RGF\ s\ k\ ))]$
apply(*simp add*:*EvtSys-on-Monitor-RGF-def* Pre_f-*def* $Rely_f$-*def* $Guar_f$-*def* $Post_f$-*def*
getrgformula-def)
apply *auto*[*1*]
apply(*rule EvtSys-h*)
apply *clarify*
apply(*case-tac* $(a,b) \in (\bigcup b.\ \{create\text{-}RGF\ s\ k\ b\})$)
using *Create-satRG create-RGF-def Evt-sat-RG-def* E_e-*def* Pre_e-*def* $Rely_e$-*def*
$Guar_e$-*def* $Post_e$-*def*
$Guar_f$-*def* $Post_f$-*def* Pre_f-*def* $Rely_f$-*def snd-conv fst-conv*
apply (*metis* (*no-types*, *lifting*) *UN-E singletonD*)
apply(*case-tac* $(a,b) \in (\bigcup b.\ \{remove\text{-}RGF\ s\ k\ b\})$)
using *Remove-satRG remove-RGF-def Evt-sat-RG-def* E_e-*def* Pre_e-*def* $Rely_e$-*def*
$Guar_e$-*def* $Post_e$-*def*
$Guar_f$-*def* $Post_f$-*def* Pre_f-*def* $Rely_f$-*def snd-conv fst-conv*
apply (*metis* (*no-types*, *lifting*) *UN-E singletonD*)

apply *simp*
apply *clarify*
apply(*case-tac* $(a,b) \in (\bigcup b.\ \{$*create-RGF s k b*$\})$)
apply(*simp add*: *create-RGF-def* E_e*-def* Pre_e*-def createNode-RGCond-def getrgformula-def*)

apply(*case-tac* $(a,b) \in (\bigcup b.\ \{$*remove-RGF s k b*$\})$)
apply(*simp add*: *remove-RGF-def* E_e*-def* Pre_e*-def removeNode-RGCond-def getrgformula-def*)
apply *fastforce*
unfolding *Ball-def* **apply**(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup b.\ \{$*create-RGF s k b*$\})$)
apply(*simp add*: *create-RGF-def* $Rely_e$*-def createNode-RGCond-def getrgformula-def*)

apply (*erule exE*) **apply** *auto*[*1*]
apply(*case-tac* $x \in (\bigcup b.\ \{$*remove-RGF s k b*$\})$)
apply(*simp add*: *remove-RGF-def* $Rely_e$*-def removeNode-RGCond-def getrgformula-def*)
apply (*erule exE*)
apply *simp*
apply *blast*

apply(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup b.\ \{$*create-RGF s k b*$\})$)
apply(*simp add*: *create-RGF-def* $Guar_e$*-def createNode-RGCond-def getrgformula-def*)

apply (*erule exE*)
apply (*simp add*: *getrgformula-def removeNode-RGCond-def create-RGF-def*)
apply(*case-tac* $x \in (\bigcup b.\ \{$*remove-RGF s k b*$\})$)
apply(*simp add*: *remove-RGF-def* $Guar_e$*-def removeNode-RGCond-def getrgformula-def*)
apply *auto*[*1*]
apply(*simp add*:*createNode-def removeNode-def*)
apply (*simp add*: *createNode-RGCond-def create-RGF-def getrgformula-def*)
apply(*simp add*: $Guar_e$*-def create-RGF-def remove-RGF-def*)
apply *auto*[*1*]
apply(*simp add*:*createNode-def removeNode-def*)
apply (*simp add*:*create-RGF-def createNode-RGCond-def remove-RGF-def removeNode-RGCond-def* Pre_e*-def* $Post_e$*-def getrgformula-def*)+
apply(*simp add*:*PiCore-Hoare.stable-def*)
by *simp*

definition *EvtSys-on-DGPS-RGF s* ≡
EvtSysRGF[*s*,*DGPS*,{*RGF*[*DGPS-exec*,*RGCond*[*s*,*DGPS*,*dgps-post*]]},*dgps-post*]

lemma *EvtSys-on-DGPS-SatRG*:
$\forall s\ .\ \Gamma \vdash$ *fst* (*EvtSys-on-DGPS-RGF s*) sat_s
[Pre_f (*snd* (*EvtSys-on-DGPS-RGF s*)),

Rely$_f$ (*snd* (*EvtSys-on-DGPS-RGF s*)),
Guar$_f$ (*snd* (*EvtSys-on-DGPS-RGF s*)),
Post$_f$ (*snd* (*EvtSys-on-DGPS-RGF s*))]

apply(*simp add*:*EvtSys-on-DGPS-RGF-def Pre*$_f$*-def Rely*$_f$*-def*
Guar$_f$*-def Post*$_f$*-def getrgformula-def EvtSys-on-RGF-def RGF-def RGCond-def*)
apply *auto*[*1*]
apply(*rule EvtSys-h*)
apply *clarify*
apply(*simp add*:*E*$_e$*-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*
guar-bufs-stb-sys-def bufs-stb-cpl-def)
using *DGPS-satRG*
apply (*simp add*: *Evt-sat-RG-def Guar*$_e$*-def Guar*$_f$*-def Post*$_e$*-def Post*$_f$*-def*
Pre$_e$*-def Pre*$_f$*-def*
Rely$_e$*-def Rely*$_f$*-def*)
apply(*simp add*:*Pre*$_e$*-def DGPS-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*
guar-bufs-stb-sys-def bufs-stb-cpl-def)
apply *fastforce*
apply(*simp add*:*Rely*$_e$*-def DGPS-exec-RGCond-def getrgformula-def Pre*$_e$*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def*
guar-bufs-stb-sys-def bufs-stb-cpl-def)
apply(*simp add*:*Guar*$_e$*-def DGPS-exec-RGCond-def getrgformula-def Re-*
ly$_e$*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def*

bufs-stb-cpl-def)
apply(*simp add*:*Post*$_e$*-def DGPS-exec-RGCond-def getrgformula-def Guar*$_e$*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def*
bufs-stb-cpl-def)
apply(*simp add*:*Post*$_e$*-def Pre*$_e$*-def DGPS-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def*
bufs-stb-cpl-def)
apply(*simp add*: *Post*$_e$*-def Pre*$_e$*-def DGPS-exec-RGCond-def getrgformula-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def*
bufs-stb-cpl-def)
apply(*simp add*:*PiCore-Hoare.stable-def*) **by** *simp*

definition *EvtSys-on-Interactive-RGF s* ≡
EvtSysRGF[*s*,*Interactive*,{*RGF*[*Interactive-exec*
,*RGCond*[*s*,*Interactive*,*interactive-post*]]},*interactive-post*]

lemma *EvtSys-on-Interactive-SatRG*:
∀ *s* . Γ ⊢ *fst* (*EvtSys-on-Interactive-RGF s*) *sat*$_s$
[*Pre*$_f$ (*snd* (*EvtSys-on-Interactive-RGF s*)),
Rely$_f$ (*snd* (*EvtSys-on-Interactive-RGF s*)),

$Guar_f$ (snd (EvtSys-on-Interactive-RGF s)),
$Post_f$ (snd (EvtSys-on-Interactive-RGF s))]
apply(simp add:EvtSys-on-Interactive-RGF-def Pre_f-def $Rely_f$-def
$Guar_f$-def $Post_f$-def getrgformula-def EvtSys-on-RGF-def RGF-def
RGCond-def)
apply auto[1]
apply(rule EvtSys-h)
apply clarify
apply(simp add:E_e-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def
guar-bufs-stb-sys-def bufs-stb-cpl-def)
using Interactive-satRG
apply (simp add: Evt-sat-RG-def $Guar_e$-def $Guar_f$-def $Post_e$-def $Post_f$-def
Pre_e-def Pre_f-def $Rely_e$-def $Rely_f$-def)
apply(simp add:Interactive-exec-def Interactive-exec-RGCond-def)
apply(simp add:Pre_e-def Interactive-exec-RGCond-def getrgformula-def)
apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
using State.select-convs(1) State.select-convs(4) **apply** fastforce
apply(simp add:$Rely_e$-def Interactive-exec-RGCond-def getrgformula-def
Pre_e-def)
apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
apply(simp add:$Guar_e$-def Interactive-exec-RGCond-def getrgformula-def
$Rely_e$-def)
apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
apply(simp add:$Post_e$-def Interactive-exec-RGCond-def getrgformula-def
$Guar_e$-def)
apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
apply(simp add:$Post_e$-def Pre_e-def Interactive-exec-RGCond-def getrgformula-def)
apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
apply(simp add: $Post_e$-def Pre_e-def DGPS-exec-RGCond-def getrgformula-def)

apply (simp add:assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def
bufs-stb-cpl-def)
apply(simp add:PiCore-Hoare.stable-def) **by** simp

definition EvtSys-on-Locator-RGF s ≡ EvtSysRGF[s,Locator,{RGF[locator-exec,RGCond[s,Locator,locator-

lemma EvtSys-on-Locator-SatRG:
$\forall s$. $\Gamma \vdash$ fst (EvtSys-on-Locator-RGF s) sat_s
[Pre_f (snd (EvtSys-on-Locator-RGF s)),
$Rely_f$ (snd (EvtSys-on-Locator-RGF s)),
$Guar_f$ (snd (EvtSys-on-Locator-RGF s)),
$Post_f$ (snd (EvtSys-on-Locator-RGF s))]

apply(*simp add*:*EvtSys-on-Locator-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*

getrgformula-def EvtSys-on-RGF-def RGF-def RGCond-def)
apply *auto*[*1*]
apply(*rule EvtSys-h*)
apply *clarify*
apply(*simp add*:E_e*-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
using *Locator-satRG*
apply (*simp add*: *Evt-sat-RG-def* $Guar_e$*-def* $Guar_f$*-def* $Post_e$*-def* $Post_f$*-def* Pre_e*-def* Pre_f*-def* $Rely_e$*-def* $Rely_f$*-def*)
apply(*simp add*:Pre_e*-def Locator-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply *fastforce*
apply(*simp add*:$Rely_e$*-def Locator-exec-RGCond-def getrgformula-def* Pre_e*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Guar_e$*-def Locator-exec-RGCond-def getrgformula-def* $Rely_e$*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def Locator-exec-RGCond-def getrgformula-def* $Guar_e$*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Locator-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*: $Post_e$*-def* Pre_e*-def Locator-exec-RGCond-def getrgformula-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:*PiCore-Hoare.stable-def*) **apply** *simp* **done**

definition *EvtSys-on-Planner-RGF s* ≡ *EvtSysRGF*[*s*,*Planner*,{*RGF*[*planner-exec*,*RGCond*[*s*,*Planner*,*plann*

lemma *EvtSys-on-Planner-SatRG*:
$\forall s$. $\Gamma \vdash$ *fst* (*EvtSys-on-Planner-RGF s*) sat_s
[Pre_f (*snd* (*EvtSys-on-Planner-RGF s*)),
$Rely_f$ (*snd* (*EvtSys-on-Planner-RGF s*)),
$Guar_f$ (*snd* (*EvtSys-on-Planner-RGF s*)),
$Post_f$ (*snd* (*EvtSys-on-Planner-RGF s*))]

apply(*simp add*:*EvtSys-on-Planner-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def*

getrgformula-def EvtSys-on-RGF-def RGF-def RGCond-def)
apply *auto*[*1*]
apply(*rule EvtSys-h*)
apply *clarify*
apply(*simp add*:E_e*-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
using *Planner-satRG*
apply (*simp add*: *Evt-sat-RG-def* $Guar_e$*-def* $Guar_f$*-def* $Post_e$*-def* $Post_f$*-def* Pre_e*-def* Pre_f*-def* $Rely_e$*-def* $Rely_f$*-def*)
apply(*simp add*:Pre_e*-def Planner-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply *fastforce*
apply(*simp add*:$Rely_e$*-def Planner-exec-RGCond-def getrgformula-def* Pre_e*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Guar_e$*-def Planner-exec-RGCond-def getrgformula-def* $Rely_e$*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def Planner-exec-RGCond-def getrgformula-def* $Guar_e$*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Planner-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*: $Post_e$*-def* Pre_e*-def Locator-exec-RGCond-def getrgformula-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:*PiCore-Hoare.stable-def*) **apply** *simp*
done

lemma *EvtSys-on-Chassis-SatRG*:
$\forall s . \Gamma \vdash$ *fst* (*EvtSys-on-Chassis-RGF s*) sat_s
[Pre_f (*snd* (*EvtSys-on-Chassis-RGF s*)),
$Rely_f$ (*snd* (*EvtSys-on-Chassis-RGF s*)),
$Guar_f$ (*snd* (*EvtSys-on-Chassis-RGF s*)),
$Post_f$ (*snd* (*EvtSys-on-Chassis-RGF s*))]

apply(*simp add*:*EvtSys-on-Chassis-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def getrgformula-def EvtSys-on-RGF-def*
RGF-def RGCond-def)
apply *auto*[*1*]
apply(*rule EvtSys-h*)
apply *clarify*

apply(*simp add*:E_e*-def assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
using *Chassis-satRG*
apply (*simp add*: *Evt-sat-RG-def* $Guar_e$*-def* $Guar_f$*-def* $Post_e$*-def* $Post_f$*-def* Pre_e*-def* Pre_f*-def* $Rely_e$*-def* $Rely_f$*-def*)
apply(*simp add*:Pre_e*-def Chassis-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply *fastforce*
apply(*simp add*:$Rely_e$*-def Chassis-exec-RGCond-def getrgformula-def* Pre_e*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Guar_e$*-def Chassis-exec-RGCond-def getrgformula-def* $Rely_e$*-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def Chassis-exec-RGCond-def getrgformula-def* $Guar_e$*-def*)

apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:$Post_e$*-def* Pre_e*-def Chassis-exec-RGCond-def getrgformula-def*)
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*: $Post_e$*-def* Pre_e*-def Chassis-exec-RGCond-def getrgformula-def*)
apply *fastforce*
apply (*simp add*:*assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def guar-bufs-stb-sys-def bufs-stb-cpl-def*)
apply(*simp add*:*PiCore-Hoare.stable-def*) **apply** *simp*
done

lemma *esys-sat*: $\forall s . \Gamma \vdash$ *fst* (*L4-Spec s k*)
sat_s [Pre_{es} (*L4-Spec s k*),
$Rely_{es}$ (*L4-Spec s k*),
$Guar_{es}$ (*L4-Spec s k*),
$Post_{es}$ (*L4-Spec s k*)]
apply *auto*[*1*]
apply(*induct k*)
apply(*simp add*:*L4-Spec-def*)
apply(*simp add*: Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*)
using *EvtSys-on-DGPS-SatRG*
apply(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def EvtSys-on-DGPS-RGF-def*)

apply *fastforce*
apply(*simp add*:*L4-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*)
using *EvtSys-on-Locator-SatRG*
apply(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def EvtSys-on-Locator-RGF-def*)

apply *fast*
apply(*simp add*:*L4-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*)
using *EvtSys-on-Planner-SatRG*
apply(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def EvtSys-on-Planner-RGF-def*)
apply *fast*
apply(*simp add*:*L4-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*)
using *EvtSys-on-Chassis-SatRG* **apply**(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def*
$Post_f$*-def*) **apply** *fast*
apply(*simp add*:*L4-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*
)
using *EvtSys-on-Interactive-SatRG*
apply(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def EvtSys-on-Interactive-RGF-def*)
apply *fast*
apply(*simp add*:*L4-Spec-def* Pre_{es}*-def* $Rely_{es}$*-def* $Guar_{es}$*-def* $Post_{es}$*-def getrgformula-def*
)
using *EvtSys-on-Monitor-SatRG*
apply(*simp add*:Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def EvtSys-on-Monitor-RGF-def*)
by *fast*

definition *sys-guar s* ≡ *guar-bufs-stb-sys s DGPS*∪ *guar-bufs-stb-sys s Interactive*∪
guar-bufs-stb-sys s Locator∪ *guar-bufs-stb-sys s Planner*∪
guar-bufs-stb-sys s Chassis∪ *Id* ∪ ⦃$^{\mathrm{o}}$*buf-msg* = $^{\mathrm{a}}$*buf-msg*⦄

lemma *esys-guar-in-sys*: ∀ *s*. $Guar_{es}$ (*L4-Spec s k*) ⊆ *sys-guar s*
apply (*rule allI*)
apply(*induct k*)
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def*
EvtSys-on-DGPS-RGF-def
RGF-def RGCond-def EvtSys-on-RGF-def)
apply *auto*[*1*]
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def EvtSys-on-Locator-RGF-def*

RGF-def RGCond-def EvtSys-on-RGF-def)
apply *fast*
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def EvtSys-on-Planner-RGF-def*

RGF-def RGCond-def EvtSys-on-RGF-def)
apply *fast*
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def EvtSys-on-Chassis-RGF-def*

RGF-def RGCond-def EvtSys-on-RGF-def)
apply *fast*
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def EvtSys-on-Interactive-RGF-def*

RGF-def RGCond-def EvtSys-on-RGF-def)
apply *fast*
apply(*simp add*:$Guar_{es}$*-def L4-Spec-def getrgformula-def sys-guar-def EvtSys-on-RGF-def*
RGF-def

RGCond-def EvtSys-on-Monitor-RGF-def)
done

18.7 Invariant and ParSystem proof

definition *buf-writer-inv* :: *State* ⇒ *bool*
where *buf-writer-inv s* ≡ ∀ *b*. *b*∈ *bufset s* ⟶*buf-writer s b* ≠ *None*

definition *buf-readers-inv* :: *State* ⇒ *bool*
where *buf-readers-inv s* ≡ ∀ *b*. *b*∈ *bufset s*⟶*buf-readers s b* ≠ {}

definition *buf-msg-inv* :: *State* ⇒ *bool*
where *buf-msg-inv s* ≡ ∀ *b*. *b*∈ *bufset s*⟶*buf-msg s b* = (|*data* = *Some*(*SOME x*:: *msg-type*. *True*) |)

definition *bufset-inv* :: *State* ⇒ *bool*
where*bufset-inv s* ≡ *bufset s* ≠ {}

definition *invariant s* ≡ *buf-writer-inv s* ∧ *buf-readers-inv s* ∧ *buf-msg-inv s* ∧ *bufset-inv s*

interpretation *dmsg-bus-rg ptranI petranI None cpts-pI cpts-of-pI prog-validityI assume-pI*
commit-pI rghoare-pI sysconf buf-writer buf-readers buf-msg local-vars
bufset invariant L4-Spec′ s0

apply(*simp add*:*dmsg-bus-rg-def*)
apply *auto*[*1*]
apply (*simp add*: *dmsg-bus-axioms rghoare-pI-def*)
using *dmsg-bus-rg.axioms buf-writer′.simps*
apply(*simp add*:*L4-Spec′-def local-vars-def s0-def System-Init-def*)
by (*smt Module.distinct*(*1*) *State.select-convs*(*1*) *State.select-convs*(*4*)
buf-writer′.simps(*1*) *buf-writer′.simps*(*2*) *msg-type-no-eq option.inject*)

lemma *stb-guar*: *stable* (*Collect invariant*) (*G*)
using *inv-stb-G*
apply(*simp add*:*SIMP-hoare.stable-def*)
by (*simp add*: *PiCore-Hoare.stable-def*)

lemma *stb-pred-rel*: *stable* (*Collect P′*) *RG* ⟹ (*s*, *r*) ∈ *RG* ⟹ *P′ s* ⟹ *P′ r*
by(*simp add*:*stable-def*)

lemma *fun-g-stb-inv*: (*s*,*r*)∈ *G*⟹ *invariant s* ⟹ *invariant r*
using *stb-guar stb-pred-rel*[*of invariant G s r*]
by *simp*

lemma *functional-correctness′*: Γ ⊢ *L4-Spec′ s SAT*
[{|*True*|},
(*assm-bufs-stb-sys s DGPS* ∩ *assm-bufs-stb-sys s Interactive* ∩ *assm-bufs-stb-sys*

s Locator
$\cap$ *assm-bufs-stb-sys s Planner* $\cap$ *assm-bufs-stb-sys s Chassis*
$\cap$ $\{\!|^{\circ}$*buf-writer* $=$ ^{a}*buf-writer* $\wedge$ $^{\circ}$*buf-readers* $=$ ^{a}*buf-readers* $\wedge$ $^{\circ}$*bufset* $=$ ^{a}*bufset*$|\!\}$$)\cup$
Id ,
sys-guar s,
dgps-post$\cup$ *interactive-post* $\cup$ *locator-post* $\cup$ *planner-post* $\cup$ *chassis-post*]
apply (*rule ParallelESys*)
apply(*simp add*:*L4-Spec'-def*)
using *EvtSys-on-DGPS-SatRG EvtSys-on-Interactive-SatRG EvtSys-on-Locator-SatRG*
EvtSys-on-Planner-SatRG EvtSys-on-Chassis-SatRG wholesys-sat-RG
apply (*simp add*: *Guar*$_{es}$*-def Guar*$_f$*-def Post*$_{es}$*-def Post*$_f$*-def Pre*$_{es}$*-def*
Pre$_f$*-def*
Rely$_{es}$*-def Rely*$_f$*-def*)
apply (*smt Guar*$_{es}$*-def L4-Spec'-def Post*$_{es}$*-def Pre*$_{es}$*-def Rely*$_{es}$*-def syssat*)
apply(*simp add*:*L4-Spec'-def EvtSys-on-DGPS-RGF-def EvtSys-on-Locator-RGF-def*
EvtSys-on-Planner-RGF-def
EvtSys-on-Chassis-RGF-def EvtSys-on-Interactive-RGF-def Pre$_{es}$*-def getrgformula-def*

RGF-def RGCond-def EvtSys-on-RGF-def)
apply *auto*[*1*]
apply(*case-tac k* = *DGPS*)
apply (*simp add*:*EvtSys-on-DGPS-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
apply(*case-tac k* = *Locator*)
apply (*simp add*:*EvtSys-on-Locator-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
apply(*case-tac k* = *Interactive*)
apply (*simp add*:*EvtSys-on-Interactive-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
apply(*case-tac k* = *Planner*)
apply (*simp add*:*EvtSys-on-Planner-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
apply(*case-tac k* = *Chassis*)
apply (*simp add*:*EvtSys-on-Chassis-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
apply(*case-tac k* = *Monitor*)
apply (*simp add*:*EvtSys-on-Monitor-RGF-def getrgformula-def EvtSys-on-RGF-def*
RGF-def)
using *Module.exhaust*
apply *blast*
apply(*rule allI*)
apply(*simp add*:*L4-Spec'-def EvtSys-on-DGPS-RGF-def EvtSys-on-Interactive-RGF-def*
EvtSys-on-Locator-RGF-def
EvtSys-on-Planner-RGF-def EvtSys-on-Chassis-RGF-def Rely$_{es}$*-def getrgformula-def*
RGF-def
RGCond-def EvtSys-on-RGF-def)
apply(*case-tac k* = *DGPS*)
apply (*simp add*:*EvtSys-on-DGPS-RGF-def getrgformula-def EvtSys-on-RGF-def*)
apply *auto*[*1*]

apply(*case-tac* k = *Locator*)
apply (*simp add*:*EvtSys-on-Locator-RGF-def getrgformula-def EvtSys-on-RGF-def*)**apply** *auto*[*1*]
apply(*case-tac* k = *Interactive*)
apply (*simp add*:*EvtSys-on-Interactive-RGF-def getrgformula-def EvtSys-on-RGF-def*)**apply** *auto*[*1*]
apply(*case-tac* k = *Planner*)
apply (*simp add*:*EvtSys-on-Planner-RGF-def getrgformula-def EvtSys-on-RGF-def*)**apply** *auto*[*1*]
apply(*case-tac* k = *Chassis*)
apply (*simp add*:*EvtSys-on-Chassis-RGF-def getrgformula-def EvtSys-on-RGF-def*)**apply** *auto*[*1*]
apply(*case-tac* k = *Monitor*)
apply (*simp add*:*EvtSys-on-Monitor-RGF-def getrgformula-def EvtSys-on-RGF-def RGF-def*)
apply (*simp add*: *Collect-mono-iff Id-fstsnd-eq*)
using *Module.exhaust* **apply** *blast*

apply(*simp add*:*L4-Spec*′*-def EvtSys-on-DGPS-RGF-def EvtSys-on-Interactive-RGF-def EvtSys-on-Planner-RGF-def EvtSys-on-Chassis-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def*
EvtSys-on-Locator-RGF-def getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def)
apply *auto*[*1*]
apply(*case-tac* j = *DGPS*)
apply(*case-tac* k = *DGPS*)
apply *simp*
apply(*case-tac* k = *Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac* k = *Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac* k = *Planner*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac* k = *Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac* k = *Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*

EvtSys-on-Monitor-RGF-def)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j = Locator*)
apply(*case-tac k = Locator*)
using *guar-in-rely-bufs* **apply** *auto*[*1*]
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *blast*
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
)
apply *blast*
apply(*case-tac k = Planner*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
)
apply *blast*
apply(*case-tac k = Chassis*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
EvtSys-on-Chassis-RGF-def)
apply *blast*
apply(*case-tac k = Monitor*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
EvtSys-on-Monitor-RGF-def)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j = Interactive*)
apply(*case-tac k = Interactive*)
apply *simp*
apply(*case-tac k = Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Planner*)

apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k* = *Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j* = *Planner*)
apply(*case-tac k* = *Planner*)
apply *simp*
apply(*case-tac k* = *Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k* = *DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k* = *Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j* = *Chassis*)
apply(*case-tac k* = *Chassis*)
apply *simp*
apply(*case-tac k* = *Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)

using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Planner*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k = Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j = Monitor*)
apply(*case-tac k = Monitor*)
apply *simp*
apply(*case-tac k = Locator*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*
assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def)
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
apply *auto*[*1*]
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
apply *auto*[*1*]
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k = Planner*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *guar-in-rely-bufs*

apply *auto[1]*
apply (*smt msg-type.inject(8) option.inject planner-buf-type*)
apply(*case-tac k = Chassis*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*
EvtSys-on-Chassis-RGF-def)
apply (*smt msg-type.inject(8) option.inject planner-buf-type*)
using *Module.exhaust* **apply** *blast*

apply(*simp add:L4-Spec′-def EvtSys-on-DGPS-RGF-def EvtSys-on-Interactive-RGF-def EvtSys-on-Locator-RGF-def*
EvtSys-on-Planner-RGF-def EvtSys-on-Chassis-RGF-def $Guar_{es}$*-def getrgformula-def*
RGF-def RGCond-def EvtSys-on-RGF-def)
apply(*rule allI*)
apply(*case-tac k = DGPS*)
apply(*simp add:EvtSys-on-DGPS-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)
apply *blast*
apply(*case-tac k = Locator*)
apply(*simp add:EvtSys-on-Locator-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Interactive*)
apply(*simp add:EvtSys-on-Interactive-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Planner*)
apply(*simp add:EvtSys-on-Planner-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Chassis*)
apply(*simp add:EvtSys-on-Chassis-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Monitor*)
apply(*simp add:EvtSys-on-Chassis-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-Monitor-RGF-def sys-guar-def*)
using *Module.exhaust* **apply** *blast*

apply(*simp add:L4-Spec′-def* $Post_{es}$*-def getrgformula-def sys-guar-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply(*rule allI*)
apply(*case-tac k = DGPS*)
apply(*simp add:EvtSys-on-DGPS-RGF-def RGF-def RGCond-def EvtSys-on-RGF-def getrgformula-def*)

apply *auto*[*1*]
apply(*case-tac* k = *Locator*)
apply(*simp add*:*EvtSys-on-Locator-RGF-def RGF-def RGCond-def EvtSys-on-RGF-def getrgformula-def*)
apply *auto*[*1*]
apply(*case-tac* k = *Planner*)
apply(*simp add*:*EvtSys-on-Planner-RGF-def RGF-def RGCond-def EvtSys-on-RGF-def getrgformula-def*)
apply *auto*[*1*]
apply(*case-tac* k = *Chassis*)
apply(*simp add*:*EvtSys-on-Chassis-RGF-def RGF-def RGCond-def EvtSys-on-RGF-def getrgformula-def*)
apply *auto*[*1*]
apply(*case-tac* k = *Interactive*)
apply(*simp add*:*EvtSys-on-Interactive-RGF-def RGF-def RGCond-def EvtSys-on-RGF-def getrgformula-def*)
apply(*case-tac* k = *Monitor*)
apply(*simp add*:*EvtSys-on-Interactive-RGF-def RGF-def RGCond-def EvtSys-on-Monitor-RGF-def getrgformula-def*)
apply *auto*[*1*]
using *Module.exhaust*
apply (*simp add*: *order-buf-type*)
using *Module.exhaust* **by** *blast*

lemma *esys-sat'*: $\forall k.\ \Gamma \vdash$ *fst* (*L4-Spec'* s k)
sat$_s$ [*Pre*$_{es}$ (*L4-Spec'* s k),
Rely$_{es}$ (*L4-Spec'* s k),
Guar$_{es}$ (*L4-Spec'* s k),
Post$_{es}$ (*L4-Spec'* s k)]
apply *auto*[*1*]
using *syssat*
by *metis*

lemma *functional-correctness*: $\forall s.\ \Gamma \vdash$ *L4-Spec'* s *SAT* [*s0*, {}, *sys-guar* s, {|*True*|}]
apply(*rule allI*)
apply (*rule ParallelESys*)
apply(*simp add*:*L4-Spec'-def*)
apply(*rule allI*)
using *wholesys-sat-RG*
apply (*simp add*: *Guar*$_{es}$*-def Guar*$_f$*-def Post*$_{es}$*-def Post*$_f$*-def Pre*$_{es}$*-def Pre*$_f$*-def Rely*$_{es}$*-def Rely*$_f$*-def esys-sat'*)
apply (*smt Guar*$_{es}$*-def L4-Spec-def Post*$_{es}$*-def Pre*$_{es}$*-def Rely*$_{es}$*-def esys-sat*)
apply(*simp add*:*L4-Spec'-def Pre*$_{es}$*-def getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def s0-def System-Init-def*)
apply *auto*[*1*]
apply (*smt Module.distinct*(*1*) *State.select-convs*(*1*) *State.select-convs*(*4*) *buf-writer'.simps*(*1*)
buf-writer'.simps(*2*) *msg-type-no-eq option.inject*)

apply *simp*
apply(*simp add:L4-Spec′-def EvtSys-on-DGPS-RGF-def EvtSys-on-Interactive-RGF-def EvtSys-on-Locator-RGF-def*
EvtSys-on-Planner-RGF-def EvtSys-on-Chassis-RGF-def $Guar_{es}$*-def* $Rely_{es}$*-def*
getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def)
apply *auto*[*1*]
apply(*case-tac j = DGPS*)
apply(*case-tac k = DGPS*)
apply *simp*
apply(*case-tac k = Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Planner*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k = Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j = Locator*)
apply(*case-tac k = Locator*)
using *guar-in-rely-bufs* **apply** *auto*[*1*]
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *blast*
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
)

apply *blast*
apply(*case-tac k* = *Planner*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
)
apply *blast*
apply(*case-tac k* = *Chassis*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
EvtSys-on-Chassis-RGF-def)
apply *blast*
apply(*case-tac k* = *Monitor*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
EvtSys-on-Monitor-RGF-def)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j* = *Interactive*)
apply(*case-tac k* = *Interactive*)
apply *simp*
apply(*case-tac k* = *Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k* = *DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Planner*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k* = *Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k* = *Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*
EvtSys-on-Monitor-RGF-def)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j* = *Planner*)
apply(*case-tac k* = *Planner*)

apply *simp*
apply(*case-tac k = Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Chassis*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k = Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* apply *blast*

apply(*case-tac j = Chassis*)
apply(*case-tac k = Chassis*)
apply *simp*
apply(*case-tac k = Locator*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
using *guar-in-rely-bufs*
apply *blast*
apply(*case-tac k = DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)
apply *auto*[*1*]
apply(*case-tac k = Planner*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Chassis-RGF-def*)

using *guar-in-rely-bufs*
apply *auto*[*1*]
apply(*case-tac k* = *Monitor*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *Module.exhaust* **apply** *blast*

apply(*case-tac j* = *Monitor*)
apply(*case-tac k* = *Monitor*)
apply *simp*
apply(*case-tac k* = *Locator*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*
assm-bufs-stb-sys-def bufs-stb-def get-wrt-bufs-def)
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k* = *DGPS*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
apply *auto*[*1*]
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k* = *Interactive*)
apply *auto*[*1*]
using *guar-in-rely-bufs*
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
apply *auto*[*1*]
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k* = *Planner*)
apply *auto*[*1*]
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*)
using *guar-in-rely-bufs*
apply *auto*[*1*]
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
apply(*case-tac k* = *Chassis*)
apply(*simp add*: *getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def EvtSys-on-Monitor-RGF-def*
EvtSys-on-Chassis-RGF-def)
apply (*smt msg-type.inject*(*8*) *option.inject planner-buf-type*)
using *Module.exhaust* **apply** *blast*

apply(*simp add*:*L4-Spec'-def EvtSys-on-DGPS-RGF-def EvtSys-on-Interactive-RGF-def EvtSys-on-Locator-RGF-def*
EvtSys-on-Planner-RGF-def EvtSys-on-Chassis-RGF-def $Guar_{es}$*-def getrgformula-def*
RGF-def RGCond-def EvtSys-on-RGF-def)
apply(*rule allI*)
apply(*case-tac k* = *DGPS*)

apply(*simp add:EvtSys-on-DGPS-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)
apply *blast*
apply(*case-tac k = Locator*)
apply(*simp add:EvtSys-on-Locator-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Interactive*)
apply(*simp add:EvtSys-on-Interactive-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Planner*)
apply(*simp add:EvtSys-on-Planner-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Chassis*)
apply(*simp add:EvtSys-on-Chassis-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-RGF-def sys-guar-def*)**apply** *blast*
apply(*case-tac k = Monitor*)
apply(*simp add:EvtSys-on-Chassis-RGF-def getrgformula-def*)
apply(*simp add: getrgformula-def RGF-def RGCond-def EvtSys-on-Monitor-RGF-def sys-guar-def*)
using *Module.exhaust* **apply** *blast*
by *simp*

theorem *invariant-presv-pares* Γ *invariant* (*paresys-spec* (*L4-Spec′ s*)) *s0* {}
apply(*rule invariant-theorem*[**where** *G=sys-guar s* **and** *pst = UNIV*])
using *functional-correctness inv*
apply *force*
apply(*simp add:PiCore-Hoare.stable-def*)
apply(*simp add:sys-guar-def*)
apply (*simp add: PiCore-Hoare.stable-def invariant-def*)
apply(*rule allI*)
apply(*simp add:buf-writer-inv-def buf-readers-inv-def buf-msg-inv-def*)
apply(*simp add:guar-bufs-stb-sys-def bufs-stb-cpl-def get-wrt-bufs-def*)
defer
apply(*simp add:invariant-def s0-def System-Init-def*)
apply *auto*[*1*]
apply (*simp add: buf-writer-inv-def*)
apply (*simp add: buf-readers-inv-def*)
apply (*metis State.select-convs*(*4*) *buf-msg-inv-def buffer.equality old.unit.exhaust path-type*)
apply (*simp add: bufset-inv-def*)
by (*metis Buffer.distinct*(*1*) *Module.distinct*(*1*) *State.select-convs*(*1*) *State.select-convs*(*4*)

buf-writer′.simps(*1*) *buf-writer′.simps*(*2*) *fun-upd-apply msg-type-no-eq option.inject*)

theorem *invariant-presv-pares* Γ *invariant* (*paresys-spec* (*L4-Spec′ s*)) *s0 R*
  using *inv*
  by *fast*
end