An Event-based Compositional Reasoning Approach for Concurrent Reactive Systems

Yongwang Zhao
School of Computer Science and Engineering, Beihang University, China
zhaoyongwang@gmail.com, zhaoyw@buaa.edu.cn

July 18, 2018

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*

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*

type-synonym (*'l*,*'s*) *event'* = *'l* × (*'s guard* × *'s prog*)

definition *guard* :: (*'l*,*'s*) *event'* ⇒ *'s guard* **where**
 guard ev ≡ *fst* (*snd ev*)

definition *body* :: (*'l*,*'s*) *event'* ⇒ *'s prog* **where**
 body ev ≡ *snd* (*snd ev*)

datatype (*'l*,*'k*,*'s*) *event* =
  AnonyEvent (*'s prog*) *option*
 | *BasicEvent* (*'l*,*'s*) *event'*

datatype (*'l*,*'k*,*'s*) *esys* =
  EvtSeq (*'l*,*'k*,*'s*) *event* (*'l*,*'k*,*'s*) *esys*
 | *EvtSys* (*'l*,*'k*,*'s*) *event set*

type-synonym (*'l*,*'k*,*'s*) *paresys* = *'k* ⇒ (*'l*,*'k*,*'s*) *esys*

2 Some Lemmas of Abstract Syntax

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

primrec *is-anonyevt* :: (*'l*,*'k*,*'s*) *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*: $EvtSeq\ e\ es \neq es$
  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 $('l,'k,'s)$ *act* = *Cmd cmd*
  | *EvtEnt* $('l,'k,'s)$ *event*

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

definition *get-actk* :: $('l,'k,'s)$ *act* $\Rightarrow$ $'k \Rightarrow$ $('l,'k,'s)$ *actk* (-♯- [*91*,*91*] *90*)
  where *get-actk a k* $\equiv$ $(\!|Act{=}a,\ K{=}k|\!)$

type-synonym $('l,'k,'s)\ x = 'k \Rightarrow ('l,'k,'s)$ *event*

type-synonym $'s$ *pconf* = $(('s\ prog)\ option) \times 's$

definition *getspc-p* :: $'s$ *pconf* $\Rightarrow$ $('s\ prog)$ *option* **where**
  getspc-p conf $\equiv$ *fst conf*

definition *gets-p* :: $'s$ *pconf* $\Rightarrow$ $'s$ **where**
  gets-p conf $\equiv$ *snd conf*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Semantics of Programs

inductive-set
ptran :: ($'s$ *pconf* $\times$ $'s$ *pconf*) *set*
and *ptran'* :: $'s$ *pconf* $\Rightarrow$ $'s$ *pconf* $\Rightarrow$ *bool* (- $-c\rightarrow$ - [81,81] 80)
and *ptrans* :: $'s$ *pconf* $\Rightarrow$ $'s$ *pconf* $\Rightarrow$ *bool* (- $-c{*}\rightarrow$ - [81,81] 80)
where
$P\ -c\rightarrow\ Q \equiv (P,Q) \in$ *ptran*
| $P\ -c{*}\rightarrow\ Q \equiv (P,Q) \in$*ptran*ˆ∗

| *Basic*: (*Some* (*Basic f*), *s*) $-c\rightarrow$ (*None*, *f s*)
| *Seq1*: (*Some P0*, *s*) $-c\rightarrow$ (*None*, *t*) $\Longrightarrow$ (*Some* (*Seq P0 P1*), *s*) $-c\rightarrow$ (*Some P1*, *t*)
| *Seq2*: (*Some P0*, *s*) $-c\rightarrow$ (*Some P2*, *t*) $\Longrightarrow$ (*Some*(*Seq P0 P1*), *s*) $-c\rightarrow$ (*Some*(*Seq P2 P1*), *t*)
| *CondT*: $s \in b$ $\Longrightarrow$ (*Some*(*Cond b P1 P2*), *s*) $-c\rightarrow$ (*Some P1*, *s*)
| *CondF*: $s \notin b$ $\Longrightarrow$ (*Some*(*Cond b P1 P2*), *s*) $-c\rightarrow$ (*Some P2*, *s*)
| *WhileF*: $s \notin b$ $\Longrightarrow$ (*Some*(*While b P*), *s*) $-c\rightarrow$ (*None*, *s*)
| *WhileT*: $s \in b$ $\Longrightarrow$ (*Some*(*While b P*), *s*) $-c\rightarrow$ (*Some*(*Seq P* (*While b P*)), *s*)
| *Await*: $[\![s \in b;$ (*Some P*, *s*) $-c{*}\rightarrow$ (*None*, *t*)$]\!]$ $\Longrightarrow$ (*Some*(*Await b P*), *s*) $-c\rightarrow$ (*None*, *t*)
| *Nondt*: $(s,t) \in r$ $\Longrightarrow$ (*Some*(*Nondt r*), *s*) $-c\rightarrow$ (*None*, *t*)

monos *rtrancl-mono*

3.3 Semantics of Events

inductive-set
etran :: $(('l,'k,'s)$ *econf* $\times$ $('l,'k,'s)$ *actk* $\times$ $('l,'k,'s)$ *econf*) *set*
and *etran'* :: $('l,'k,'s)$ *econf* $\Rightarrow$ $('l,'k,'s)$ *actk* $\Rightarrow$ $('l,'k,'s)$ *econf* $\Rightarrow$ *bool* (- $-et-$-$\rightarrow$ - [81,81,81] 80)
where
$P\ -et-t\rightarrow\ Q \equiv (P,t,Q) \in$ *etran*
| *AnonyEvent*: (*P*, *s*) $-c\rightarrow$ (*Q*, *t*) $\Longrightarrow$ (*AnonyEvent P*, *s*, *x*) $-et-$(*Cmd CMP*)♯*k*$\rightarrow$ (*AnonyEvent Q*, *t*, *x*)
| *EventEntry*: $[\![P =$ *body e*; $s \in$ *guard e*; $x' = x(k{:=}$ *BasicEvent e*$)]\!]$
$\Longrightarrow$ (*BasicEvent e*, *s*, *x*) $-et-$(*EvtEnt* (*BasicEvent e*))♯*k*$\rightarrow$ ((*AnonyEvent* (*Some P*)), *s*, x')

3.4 Semantics of Event Systems

inductive-set
estran :: $(('l,'k,'s)$ *esconf* $\times$ $('l,'k,'s)$ *actk* $\times$ $('l,'k,'s)$ *esconf*) *set*
and *estran'* :: $('l,'k,'s)$ *esconf* $\Rightarrow$ $('l,'k,'s)$ *actk* $\Rightarrow$ $('l,'k,'s)$ *esconf* $\Rightarrow$ *bool*
(- $-es-$-$\rightarrow$ - [81,81] 80)
where
$P\ -es-t\rightarrow\ Q \equiv (P,t,Q) \in$ *estran*

| *EvtOccur*: $[\![evt\in evts;\ (evt,\ (s,\ x))\ -et-(EvtEnt\ evt)\sharp k\rightarrow\ (e,\ (s,\ x'))\ ]\!]$
$\Longrightarrow (EvtSys\ evts,\ (s,\ x))\ -es-(EvtEnt\ evt)\sharp k\rightarrow\ (EvtSeq\ e\ (EvtSys\ evts),\ (s,\ x'))$
| *EvtSeq1*: $[\![(e,\ s,\ x)\ -et-act\sharp k\rightarrow\ (e',\ s',\ x');\ e' \neq AnonyEvent\ None]\!]$
$\Longrightarrow (EvtSeq\ e\ es,\ s,\ x)\ -es-act\sharp k\rightarrow\ (EvtSeq\ e'\ es,\ s',\ x')$
| *EvtSeq2*: $[\![(e,\ s,\ x)\ -et-act\sharp k\rightarrow\ (e',\ s',\ x');\ e' = AnonyEvent\ None]\!]$
$\Longrightarrow (EvtSeq\ e\ es,\ s,\ x)\ -es-act\sharp k\rightarrow\ (es,\ s',\ x')$

3.5 Semantics of Parallel Event Systems

inductive-set
pestran :: (('l,'k,'s) *pesconf* × ('l,'k,'s) *actk* × ('l,'k,'s) *pesconf*) *set*
and *pestran'* :: ('l,'k,'s) *pesconf* ⇒ ('l,'k,'s) *actk*
⇒ ('l,'k,'s) *pesconf* ⇒ *bool* (- −*pes*−-→ - [*70*,*70*] *60*)
where
$P\ -pes-t\rightarrow\ Q \equiv (P,t,Q) \in pestran$
| *ParES*: $(pes(k),\ (s,\ x))\ -es-(a\sharp k)\rightarrow\ (es',\ (s',\ x')) \Longrightarrow (pes,\ (s,\ x))\ -pes-(a\sharp k)\rightarrow\ (pes(k{:=}es'),\ (s',\ x'))$

3.6 Lemmas

3.6.1 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 *seq-not-eq1*: $Seq\ c1\ c2{\neq}c1$
by (*induct c1*) *auto*

lemma *seq-not-eq2*: $Seq\ c1\ c2{\neq}c2$
by (*induct c2*) *auto*

lemma *if-not-eq1*: $Cond\ b\ c1\ c2\ {\neq}c1$
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*: $(P,s)\ -c\rightarrow\ (Q,t)$

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) $-c\rightarrow$ (P,t)
apply *clarify*
apply(*drule prog-not-eq-in-ctran-aux*)
apply *simp*
done

3.6.2 Events

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

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

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

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

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

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

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

lemma *noevtent-notran*: $ec1 = (BasicEvent\ e,\ s,\ x) \Longrightarrow \neg\ (\exists\, k.\ ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2)$
$\Longrightarrow \neg\ (ec1\ -et-t\rightarrow\ ec2)$
proof −
assume *p0*: $ec1 = (BasicEvent\ e,\ s,\ x)$ **and**
p1: $\neg\ (\exists\, k.\ ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2)$
then show $\neg\ (ec1\ -et-t\rightarrow\ ec2)$
proof −
{
assume *a0*: $ec1\ -et-t\rightarrow\ ec2$

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** *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*:(*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*]: ¬ (*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*. (*P*,*s*,*x*) −*et*−*et*→ (*P*,*t*,*y*)) **by** *simp*

3.6.3 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*:
⟦(*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*:
⟦(*EvtSeq e1 es*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*); *es2* ≠ *es*⟧ ⟹ ∃ *e*. *es2* = *EvtSeq e es* ∧ *is-anonyevt e*
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis event.exhaust is-anonyevt.simps*(*1*) *no-tran2basic0*)
by *simp*

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

lemma *evtseq-tran-exist-etran*:
(*EvtSeq e1 es*, *s1*, *x1*) −*es*−*et*→ (*EvtSeq e2 es*, *t1*, *y1*) ⟹ ∃ *t*. (*e1*, *s1*, *x1*) −*et*−*t*→ (*e2*, *t1*, *y1*)
apply(*rule estran.cases*)
apply(*simp*)+
apply *blast*
by (*metis add.right-neutral add-Suc-right esys.inject*(*1*) *esys.size*(*3*) *lessI not-less-eq trans-less-add2*)

lemma *evtseq-tran-0-exist-etran*:

$(EvtSeq\ e1\ es,\ s1,\ x1) -es-et\rightarrow (es,\ t1,\ y1) \Longrightarrow \exists\, t.\ (e1,\ s1,\ x1) -et-t\rightarrow (AnonyEvent\ (None),\ t1,\ y1)$
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*:
$[\![(es1,\ s1,\ x1) -es-et\rightarrow (es2,\ s2,\ x2);\ \exists\, e.\ es1 = EvtSeq\ e\ esys]\!] \Longrightarrow \neg(\exists\, e.\ es2 = EvtSeq\ (BasicEvent\ e)\ esys)$
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*:
$(EvtSeq\ e1\ es,\ s1,\ x1) -es-et\rightarrow (es2,\ t1,\ y1) \Longrightarrow es2 = es \vee (\exists\, e.\ es2 = EvtSeq\ e\ es)$
by (*meson evtseq-tran-evtseq*)

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

lemma *evtseq-no-evtent*:
$[\![(EvtSeq\ e1\ es,\ s1,\ x1) -es-t\sharp k\rightarrow (es2,\ s2,\ x2); is\text{-}anonyevt\ e1]\!] \Longrightarrow \neg(\exists\, e.\ t = EvtEnt\ e)$
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*:
$[\![esc1 -es-t\sharp k\rightarrow esc2;\ getspc\text{-}es\ esc1 = EvtSeq\ e\ esys;\ is\text{-}anonyevt\ e]\!] \Longrightarrow \neg(\exists\, e.\ t = EvtEnt\ e)$
proof −
  assume *p0*: $esc1 -es-t\sharp k\rightarrow esc2$
    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 e esys s1 x1 t k es2 s2 x2*]
    by *simp*
qed

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

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

lemma *evtent-is-basicevt*: $(es,\ s,\ x) -es-EvtEnt\ e\sharp k\rightarrow (es',\ s',\ x') \Longrightarrow \exists\, e'.\ e = BasicEvent\ e'$
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: ⟦(EvtSeq e es,s1,x1) −es−EvtEnt e1♯k→ (esc2,s2,x2)⟧
    ⟹ e = e1 ∧ (∃ 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 ent-spec1 esys.inject(1) evtent-is-basicevt get-actk-def)

lemma evtent-is-basicevt-inevtseq2: ⟦esc1 −es−EvtEnt e1♯k→ esc2; getspc-es esc1 = EvtSeq e es⟧
    ⟹ e = e1 ∧ (∃ e'. e = BasicEvent e')
  proof −
    assume p0: esc1 −es−EvtEnt e1♯k→ 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 e es s1 x1 e1 k es2 s2 x2] getspc-es-def[of esc1] by auto
  qed

lemma evtsysent-evtent0: (EvtSys es, s, x) −es−t→ (EvtSeq ev (EvtSys es), s1,x1) ⟹
        s = s1 ∧ (∃ evt e. evt ∈ es ∧ evt = BasicEvent e ∧ Act t = EvtEnt (BasicEvent e) ∧
            (BasicEvent e, s, x) −et−t→ (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 esys.inject(1) esys.inject(2) select-convs(1))

lemma evtsysent-evtent: (EvtSys es, s, x) −es−(EvtEnt (BasicEvent e))♯k→ (EvtSeq ev (EvtSys es), s1,x1) ⟹
        s = s1 ∧ BasicEvent e ∈ es ∧ (BasicEvent e, s, x) −et−(EvtEnt (BasicEvent e))♯k→ (ev, s1, x1)
  apply(rule estran.cases)
```

apply(*simp*)+
apply (*metis ent-spec1*)
apply(*simp*)+
done

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

lemma *evtsysent-evtent3*: $[\![esc1\ -es-(EvtEnt\ ev)\sharp k\rightarrow\ esc2;\ getspc\text{-}es\ esc1 = EvtSys\ es]\!] \Longrightarrow$
$(ev{\in}es)$
proof −
assume *p0*: $esc1\ -es-(EvtEnt\ ev)\sharp k\rightarrow\ esc2$
and *p1*: $getspc\text{-}es\ esc1 = EvtSys\ es$
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 es s1 x1 ev k es2 s2 x2*] **by** *simp*
qed

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

lemma *act-in-es-notchgstate*: $[\![(es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!] \Longrightarrow x = x'$
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*:
$[\![(es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!]$
$\Longrightarrow \exists\, e\ e'\ es1.\ es = EvtSeq\ e\ es1 \wedge e = AnonyEvent\ e'$
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*:
$[\![(es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')]\!]$
$\Longrightarrow \neg(\exists\, ess.\ es = EvtSys\ ess)$
apply(*rule estran.cases*)
apply (*simp add: get-actk-def*)+
done

lemma *cmd-enable-impl-notesys2*:
  ⟦*esc1* −*es*−(*Cmd c*)♯*k*→ *esc2*⟧
    ⟹ ¬(∃ *ess*. *getspc-es esc1* = *EvtSys ess*)
  proof −
    assume *p0*: *esc1* −*es*−(*Cmd c*)♯*k*→ *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 es1 s1 x1 c k es2 s2 x2*] *getspc-es-def*[*of esc1*]
      by *simp*
  qed

lemma *cmd-enable-impl-anonyevt2*:
  ⟦*esc1* −*es*−(*Cmd c*)♯*k*→ *esc2*⟧
    ⟹ ∃ *e e′ es1*. *getspc-es esc1* = *EvtSeq e es1* ∧ *e* = *AnonyEvent e′*
  proof −
    assume *p0*: *esc1* −*es*−(*Cmd c*)♯*k*→ *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 es1 s1 x1 c k es2 s2 x2*] *getspc-es-def*[*of esc1*]
      by *simp*
  qed

lemma *entevt-notchgstate*: ⟦(*es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*es′*, *s′*, *x′*)⟧ ⟹ *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*: ⟦(*es*, *s*, *x*) −*es*−(*EvtEnt e*)♯*k*→ (*es′*, *s′*, *x′*)⟧ ⟹ (∀ *k′*. *k′*≠*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 *entevt-ines-notchg-otherx2*: ⟦*esc1* −*es*−(*EvtEnt e*)♯*k*→ *esc2*⟧
      ⟹ (∀ *k′*. *k′*≠*k* ⟶ (*getx-es esc1*) *k′* = (*getx-es esc2*) *k′*)
  proof −
    assume *p0*: *esc1* −*es*−(*EvtEnt e*)♯*k*→ *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 ∀ *k′*. *k′*≠*k* ⟶ *x1 k′* = *x2 k′*

using *entevt-ines-notchg-otherx*[*of 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*: $\llbracket(es,\ s,\ x)\ -es-(Cmd\ c)\sharp k\rightarrow\ (es',\ s',\ x')\rrbracket \Longrightarrow (\forall k.\ x'\ k = x\ k)$
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*: $\llbracket esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2\rrbracket \Longrightarrow (\forall k.\ (getx\text{-}es\ esc2)\ k = (getx\text{-}es\ esc1)\ k)$
proof −
assume *p0*: $esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2$
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 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*: $\llbracket(es,\ s,\ x)\ -es-(EvtEnt\ e)\sharp k\rightarrow\ (es',\ s',\ x')\rrbracket \Longrightarrow x'\ k = e$
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*: $\llbracket esc1\ -es-(EvtEnt\ e)\sharp k\rightarrow\ esc2\rrbracket \Longrightarrow (getx\text{-}es\ esc2)\ k = e$
proof −
assume *p0*: $esc1\ -es-(EvtEnt\ e)\sharp k\rightarrow\ esc2$
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*: $\llbracket(es,\ s,\ x)\ -es-t\rightarrow\ (es',\ s',\ x')\rrbracket$
$\Longrightarrow (\exists e\ k.\ (es,\ s,\ x)\ -es-EvtEnt\ e\sharp k\rightarrow\ (es',\ s',\ x')) \lor (\exists c\ k.\ (es,\ s,\ x)\ -es-Cmd\ c\sharp k\rightarrow\ (es',\ s',\ x'))$
apply(*rule estran.cases*)
apply (*simp add*: *get-actk-def*)+
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*
apply(*rule etran.cases*)

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

lemma *estran-impl-evtentorcmd'*: $[\![(es,\ s,\ x)\ -es-t\sharp k\rightarrow\ (es',\ s',\ x')]\!]$
$\Longrightarrow (\exists\, e.\ (es,\ s,\ x)\ -es-EvtEnt\ e\sharp k\rightarrow\ (es',\ s',\ x')) \vee (\exists\, c.\ (es,\ s,\ x)\ -es-Cmd\ c\sharp k\rightarrow\ (es',\ s',\ x'))$
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*: $[\![esc1\ -es-t\rightarrow\ esc2]\!]$
$\Longrightarrow (\exists\, e\ k.\ esc1\ -es-EvtEnt\ e\sharp k\rightarrow\ esc2) \vee (\exists\, c\ k.\ esc1\ -es-Cmd\ c\sharp k\rightarrow\ esc2)$
proof −
assume *p0*: $esc1\ -es-t\rightarrow\ esc2$
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 es1 s1 x1 t es2 s2 x2*] **by** *simp*
qed

lemma *estran-impl-evtentorcmd2'*: $[\![esc1\ -es-t\sharp k\rightarrow\ esc2]\!]$
$\Longrightarrow (\exists\, e.\ esc1\ -es-EvtEnt\ e\sharp k\rightarrow\ esc2) \vee (\exists\, c.\ esc1\ -es-Cmd\ c\sharp k\rightarrow\ esc2)$
proof −
assume *p0*: $esc1\ -es-t\sharp k\rightarrow\ esc2$
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 es1 s1 x1 t k es2 s2 x2*] **by** *simp*
qed

3.6.4 Parallel Event Systems

lemma *pesconf-trip*: $[\![gets\ c = s;\ getspc\ c = spc;\ getx\ c = x]\!] \Longrightarrow c = (spc,s,x)$
by (*metis gets-def getspc-def getx-def prod.collapse*)

lemma *pestran-estran*: $[\![(pes,\ s,\ x)\ -pes-(a\sharp k)\rightarrow\ (pes',\ s',\ x')]\!] \Longrightarrow$
$\exists\, es'.\ ((pes\ k,\ s,\ x)\ -es-(a\sharp k)\rightarrow\ (es',\ s',\ x')) \wedge pes' = pes(k{:=}es')$
apply(*rule pestran.cases*)
apply(*simp*)
apply(*simp add*:*get-actk-def*)
by *auto*

lemma *act-in-pes-notchgstate*: $[\![(pes,\ s,\ x)\ -pes-(Cmd\ c)\sharp k\rightarrow\ (pes',\ s',\ x')]\!] \Longrightarrow x = x'$

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*: $[\![(pes,\ s,\ x)\ -pes-(EvtEnt\ e)\sharp k\rightarrow\ (pes',\ s',\ x')]\!] \Longrightarrow s = s'$
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*: $[\![esc1\ -pes-(EvtEnt\ e)\sharp k\rightarrow\ esc2]\!] \Longrightarrow gets\ esc1 = gets\ esc2$
using *evtent-in-pes-notchgstate* **by** (*metis pesconf-trip*)

end

4 Computations of PiCore Language

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

4.1 Environment transitions

inductive-set
petran :: $('s\ pconf \times 's\ pconf)\ set$
and *petran'* :: $'s\ pconf \Rightarrow 's\ pconf \Rightarrow bool$ (- $-pe\rightarrow$ - [81,81] 80)
where
$P\ -pe\rightarrow\ Q \equiv (P,Q) \in petran$
| *EnvP*: $(P,\ s)\ -pe\rightarrow\ (P,\ t)$

lemma *petranE*: $p\ -pe\rightarrow\ p' \Longrightarrow (\bigwedge P\ s\ t.\ p = (P,\ s) \Longrightarrow p' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
by (*induct p*, *induct p'*, *erule petran.cases*, *blast*)

inductive-set
eetran :: $(('l,'k,'s)\ econf \times ('l,'k,'s)\ econf)\ set$
and *eetran'* :: $('l,'k,'s)\ econf \Rightarrow ('l,'k,'s)\ econf \Rightarrow bool$ (- $-ee\rightarrow$ - [81,81] 80)
where
$P\ -ee\rightarrow\ Q \equiv (P,Q) \in eetran$
| *EnvE*: $(P,\ s,\ x)\ -ee\rightarrow\ (P,\ t,\ y)$

lemma *eetranE*: $p\ -ee\rightarrow\ p' \Longrightarrow (\bigwedge P\ s\ t.\ p = (P,\ s) \Longrightarrow p' = (P,\ t) \Longrightarrow Q) \Longrightarrow Q$
by (*induct p*, *induct p'*, *erule eetran.cases*, *blast*)

inductive-set
esetran :: $(('l,'k,'s)\ esconf \times ('l,'k,'s)\ esconf)\ set$
and *esetran'* :: $('l,'k,'s)\ esconf \Rightarrow ('l,'k,'s)\ esconf \Rightarrow bool$ (- $-ese\rightarrow$ - [81,81] 80)
where
$P\ -ese\rightarrow\ Q \equiv (P,Q) \in esetran$
| *EnvES*: $(P,\ s,\ x)\ -ese\rightarrow\ (P,\ t,\ y)$

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

inductive-set
  pesetran :: (('l,'k,'s) *pesconf* × ('l,'k,'s) *pesconf*) *set*
  and *pesetran'* :: ('l,'k,'s) *pesconf* ⇒ ('l,'k,'s) *pesconf* ⇒ *bool* (- −*pese*→ - [*81*,*81*] *80*)
where
  $P\ {-pese\rightarrow}\ Q \equiv (P,Q) \in pesetran$
| *EnvPES*: $(P,\ s,\ x)\ {-pese\rightarrow}\ (P,\ t,\ y)$

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

4.2 Sequential computations

4.2.1 Sequential computations of programs

type-synonym *'s pconfs* = *'s pconf list*

inductive-set *cpts-p* :: *'s pconfs set*
where
  CptsPOne: [(*P*,*s*)] ∈ *cpts-p*
| *CptsPEnv*: (*P*, *t*)#*xs* ∈ *cpts-p* ⟹ (*P*,*s*)#(*P*,*t*)#*xs* ∈ *cpts-p*
| *CptsPComp*: ⟦(*P*,*s*) −*c*→ (*Q*,*t*); (*Q*, *t*)#*xs* ∈ *cpts-p*⟧ ⟹ (*P*,*s*)#(*Q*,*t*)#*xs* ∈ *cpts-p*

definition *cpts-of-p* :: (*'s prog*) *option* ⇒ *'s* ⇒ (*'s pconfs*) *set* **where**
  cpts-of-p P s ≡ {*l*. *l*!*0*=(*P*,*s*) ∧ *l* ∈ *cpts-p*}

4.2.2 Sequential computations of events

type-synonym ('l,'k,'s) *econfs* = ('l,'k,'s) *econf list*

inductive-set *cpts-ev* :: ('l,'k,'s) *econfs set*
where
  CptsEvOne: [(*e*,*s*,*x*)] ∈ *cpts-ev*
| *CptsEvEnv*: (*e*, *t*, *x*)#*xs* ∈ *cpts-ev* ⟹ (*e*, *s*, *y*)#(*e*, *t*, *x*)#*xs* ∈ *cpts-ev*
| *CptsEvComp*: ⟦(*e1*,*s*,*x*) −*et*−*ct*→ (*e2*,*t*,*y*); (*e2*,*t*,*y*)#*xs* ∈ *cpts-ev*⟧ ⟹ (*e1*,*s*,*x*)#(*e2*,*t*,*y*)#*xs* ∈ *cpts-ev*

definition *cpts-of-ev* :: ('l,'k,'s) *event* ⇒ *'s* ⇒ ('l,'k,'s) *x* ⇒ ('l,'k,'s) *econfs set* **where**
  cpts-of-ev ev s x ≡ {*l*. *l*!*0*=(*ev*,(*s*,*x*)) ∧ *l* ∈ *cpts-ev*}

4.2.3 Sequential computations of event systems

type-synonym ('l,'k,'s) *esconfs* = ('l,'k,'s) *esconf list*

inductive-set *cpts-es* :: ('l,'k,'s) *esconfs set*
where
  CptsEsOne: [(*es*,*s*,*x*)] ∈ *cpts-es*
| *CptsEsEnv*: (*es*, *t*, *x*)#*xs* ∈ *cpts-es* ⟹ (*es*, *s*, *y*)#(*es*, *t*, *x*)#*xs* ∈ *cpts-es*
| *CptsEsComp*: ⟦(*es1*,*s*,*x*) −*es*−*ct*→ (*es2*,*t*,*y*); (*es2*,*t*,*y*)#*xs* ∈ *cpts-es*⟧ ⟹ (*es1*,*s*,*x*)#(*es2*,*t*,*y*)#*xs* ∈ *cpts-es*

definition *cpts-of-es* :: ('l,'k,'s) *esys* ⇒ *'s* ⇒ ('l,'k,'s) *x* ⇒ ('l,'k,'s) *esconfs set* **where**
  cpts-of-es es s x ≡ {*l*. *l*!*0*=(*es*,*s*,*x*) ∧ *l* ∈ *cpts-es*}

4.2.4 Sequential computations of par event systems

type-synonym ('l,'k,'s) *pesconfs* = ('l,'k,'s) *pesconf list*

inductive-set *cpts-pes* :: ('l,'k,'s) *pesconfs set*

where
CptsPesOne: $[(pes,s,x)] \in$ *cpts-pes*
| *CptsPesEnv*: $(pes,\ t,\ x)\#xs \in$ *cpts-pes* $\Longrightarrow (pes,\ s,\ y)\#(pes,\ t,\ x)\#xs \in$ *cpts-pes*
| *CptsPesComp*: $[\![(pes1,s,x)\ -pes-ct\rightarrow\ (pes2,t,y);\ (pes2,t,y)\#xs \in cpts\text{-}pes]\!] \Longrightarrow (pes1,s,x)\#(pes2,t,y)\#xs \in cpts\text{-}pes$

definition *cpts-of-pes* :: $('l,'k,'s)$ *paresys* $\Rightarrow$ $'s \Rightarrow ('l,'k,'s)\ x \Rightarrow ('l,'k,'s)$ *pesconfs set* **where**
cpts-of-pes pes s x $\equiv \{l.\ l!0=(pes,s,x) \wedge l \in cpts\text{-}pes\}$

4.3 Modular definition of program computations

definition *lift* :: $'s$ *prog* $\Rightarrow$ $'s$ *pconf* $\Rightarrow$ $'s$ *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* :: $('s\ pconfs)\ set$
where
CptPModOne: $[(P,\ s)] \in$ *cpt-p-mod*
| *CptPModEnv*: $(P,\ t)\#xs \in cpt\text{-}p\text{-}mod \Longrightarrow (P,\ s)\#(P,\ t)\#xs \in cpt\text{-}p\text{-}mod$
| *CptPModNone*: $[\![(Some\ P,\ s)\ -c\rightarrow\ (None,\ t);\ (None,\ t)\#xs \in cpt\text{-}p\text{-}mod\ ]\!] \Longrightarrow (Some\ P,s)\#(None,\ t)\#xs \in cpt\text{-}p\text{-}mod$

| *CptPModCondT*: $[\![(Some\ P0,\ s)\#ys \in cpt\text{-}p\text{-}mod;\ s \in b\ ]\!] \Longrightarrow (Some(Cond\ b\ P0\ P1),\ s)\#(Some\ P0,\ s)\#ys \in cpt\text{-}p\text{-}mod$
| *CptPModCondF*: $[\![(Some\ P1,\ s)\#ys \in cpt\text{-}p\text{-}mod;\ s \notin b\ ]\!] \Longrightarrow (Some(Cond\ b\ P0\ P1),\ s)\#(Some\ P1,\ s)\#ys \in cpt\text{-}p\text{-}mod$

| *CptPModSeq1*: $[\![(Some\ P0,\ s)\#xs \in cpt\text{-}p\text{-}mod;\ zs{=}map\ (lift\ P1)\ xs\ ]\!]$
$\Longrightarrow (Some(Seq\ P0\ P1),\ s)\#zs \in cpt\text{-}p\text{-}mod$
| *CptPModSeq2*:
$[\![(Some\ P0,\ s)\#xs \in cpt\text{-}p\text{-}mod;\ fst(last\ ((Some\ P0,\ s)\#xs)) = None;$
$(Some\ P1,\ snd(last\ ((Some\ P0,\ s)\#xs)))\#ys \in cpt\text{-}p\text{-}mod;$
$zs{=}(map\ (lift\ P1)\ xs)@ys\ ]\!] \Longrightarrow (Some(Seq\ P0\ P1),\ s)\#zs \in cpt\text{-}p\text{-}mod$

| *CptPModWhile1*:
$[\![\ (Some\ P,\ s)\#xs \in cpt\text{-}p\text{-}mod;\ s \in b;\ zs{=}map\ (lift\ (While\ b\ P))\ xs\ ]\!]$
$\Longrightarrow (Some(While\ b\ P),\ s)\#(Some(Seq\ P\ (While\ b\ P)),\ s)\#zs \in cpt\text{-}p\text{-}mod$
| *CptPModWhile2*:
$[\![\ (Some\ P,\ s)\#xs \in cpt\text{-}p\text{-}mod;\ fst(last\ ((Some\ P,\ s)\#xs)){=}None;\ s \in b;$
$zs{=}(map\ (lift\ (While\ b\ P))\ xs)@ys;$
$(Some(While\ b\ P),\ snd(last\ ((Some\ P,\ s)\#xs)))\#ys \in cpt\text{-}p\text{-}mod]\!]$
$\Longrightarrow (Some(While\ b\ P),\ s)\#(Some(Seq\ P\ (While\ b\ P)),\ s)\#zs \in cpt\text{-}p\text{-}mod$

4.4 Lemmas

4.4.1 Programs

lemma *tl-in-cptn*: $[\![\ a\#xs \in cpts\text{-}p;\ xs{\neq}[]\ ]\!] \Longrightarrow xs{\in}cpts\text{-}p$
by (*force elim*: *cpts-p.cases*)

lemma *tl-zero*[*rule-format*]:
$P\ (ys!Suc\ j) \longrightarrow Suc\ j{<}length\ ys \longrightarrow ys{\neq}[] \longrightarrow P\ (tl(ys)!j)$
by (*induct ys*) *simp-all*

4.4.2 Events

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

lemma *eetran-eqconf*: $(e1,\ s1,\ x1)\ -ee\rightarrow\ (e2,\ s2,\ x2) \Longrightarrow e1 = e2$
apply(*rule eetran.cases*)
apply(*simp*)+
done

```
lemma eetran-eqconf1: ec1 −ee→ ec2 ⟹ getspc-e ec1 = getspc-e ec2
  proof −
    assume a0: 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 ⟹ (e1, s1, x1) −ee→ (e2, s2, x2)
  by (simp add: eetran.intros)

lemma eqconf-eetran: getspc-e ec1 = getspc-e ec2 ⟹ 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; Suc 0 < length el⟧ ⟹ drop (Suc 0) el ∈ cpts-ev
  apply(rule cpts-ev.cases)
  apply(simp)+
  done

lemma cpts-ev-subi: ⟦el ∈ cpts-ev; Suc i < length el⟧ ⟹ drop (Suc i) el ∈ cpts-ev
  proof −
    assume p0:el ∈ cpts-ev and p1:Suc i < length el
    have ∀ el i. el ∈ cpts-ev ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-ev
      proof −
      {
        fix el i
        have el ∈ cpts-ev ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-ev
          proof(induct i)
            case 0 show ?case by (simp add: cpts-ev-sub0)
          next
            case (Suc j)
            assume b0: el ∈ cpts-ev ∧ Suc j < length el ⟶ drop (Suc j) el ∈ cpts-ev
            show ?case
              proof
                assume c0: el ∈ cpts-ev ∧ Suc (Suc j) < length el
                with b0 have c1: drop (Suc j) el ∈ cpts-ev
                  by (simp add: c0 Suc-lessD)
                then show drop (Suc (Suc j)) el ∈ cpts-ev
                  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 ∈ cpts-ev; Suc 0 < length el; ¬ (∃ t. el ! 0 −et−t→ el ! 1)⟧
                ⟹ getspc-e (el ! 0) = getspc-e (el ! 1)
  apply(simp)
  apply(rule cpts-ev.cases)
  apply(simp)+
```

```
  apply(simp add:getspc-e-def)+
  done

lemma notran-confeqi: ⟦el ∈ cpts-ev; Suc i < length el; ¬ (∃t. el ! i −et−t→ el ! Suc i)⟧
                    ⟹ getspc-e (el ! i) = getspc-e (el ! (Suc i))
  proof −
    assume p0: el ∈ cpts-ev and
           p1: Suc i < length el and
           p2: ¬ (∃t. el ! i −et−t→ el ! Suc i)
    have ∀el i. el ∈ cpts-ev ∧  Suc i < length el ∧ ¬ (∃t. el ! i −et−t→ el ! Suc i)
              ⟶ getspc-e (el ! i) = getspc-e (el ! (Suc i))
      proof −
      {
        fix el i
        assume a0: el ∈ cpts-ev ∧  Suc i < length el ∧ ¬ (∃t. el ! i −et−t→ el ! Suc i)
        then have getspc-e (el ! i) = getspc-e (el ! (Suc i))
          proof(induct i)
            case 0 show ?case by (simp add: 0.prems notran-confeq0)
          next
            case (Suc j)
            let ?subel = drop (Suc j) el
            assume b0: el ∈ cpts-ev ∧ Suc (Suc j) < length el ∧ ¬ (∃t. el ! Suc j −et−t→ el ! Suc (Suc j))
            then have b1: ?subel ∈ cpts-ev by (simp add: Suc-lessD b0 cpts-ev-subi)
            from b0 have b2: Suc 0 < length ?subel by auto
            from b0 have b3: ¬ (∃t. ?subel ! 0 −et−t→ ?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 ∈ cpts-ev; length el > 0; el ! (length el − 1) −et−t→ ec⟧ ⟹
                        el @ [ec] ∈ cpts-ev
  proof −
    assume p0: el ∈ cpts-ev
      and  p1: length el > 0
      and  p2: el ! (length el − 1) −et−t→ ec

    have ∀el ec t. el ∈ cpts-ev ∧ length el > 0 ∧ el ! (length el − 1) −et−t→ ec ⟶ el @ [ec] ∈ cpts-ev
      proof −
      {
        fix el ec t
        assume a0: el ∈ cpts-ev
          and  a1: length el > 0
          and  a2: el ! (length el − 1) −et−t→ ec
        from a0 a1 a2 have el @ [ec] ∈ cpts-ev
          proof(induct el)
            case (CptsEvOne e s x)
            assume b0: [(e, s, x)] ! (length [(e, s, x)] − 1) −et−t→ ec
            then have (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
        and b1: 0 < length ((e, s1, x) # xs) ⟹
                ((e, s1, x) # xs) ! (length ((e, s1, x) # xs) − 1) −et−t→ ec
                ⟹ ((e, s1, x) # xs) @ [ec] ∈ cpts-ev
        and b2: 0 < length ((e, s2, y) # (e, s1, x) # xs)
        and b3: ((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 (e, s1, x)−et−t→ ec by simp
          with b1 c0 have ((e, s1, x) # xs) @ [ec] ∈ cpts-ev by simp
          then show ?thesis by (simp add: cpts-ev.CptsEvEnv)
        next
          assume c0: xs ≠ []
          with b3 have last xs −et−t→ ec by (simp add: last-conv-nth)
          with b1 c0 have ((e, s1, x) # xs) @ [ec] ∈ cpts-ev 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: (e1, s1, x1) −et−et→ (e2, t1, y1)
        and b1: (e2, t1, y1) # xs1 ∈ cpts-ev
        and b2: 0 < length ((e2, t1, y1) # xs1) ⟹
          ((e2, t1, y1) # xs1) ! (length ((e2, t1, y1) # xs1) − 1) −et−t→ ec
            ⟹ ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-ev
        and b3: 0 < length ((e1, s1, x1) # (e2, t1, y1) # xs1)
        and b4: ((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 (e2, t1, y1)−et−t→ ec by simp
          with b2 c0 have ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-ev by simp
          with b0 show ?thesis using cpts-ev.CptsEvComp by fastforce
        next
          assume c0: xs1 ≠ []
          with b4 have last xs1 −et−t→ ec by (simp add: last-conv-nth)
          with b2 c0 have ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-ev 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] ∈ cpts-ev using p0 p1 p2 by blast
qed

lemma cpts-ev-same: ⟦length el > 0; ∀ i. i < length el ⟶ getspc-e (el!i) = es⟧ ⟹ el ∈ cpts-ev
  proof −
    assume p0: length el > 0
      and p1: ∀ i. i < length el ⟶ getspc-e (el!i) = es
    have ∀ el es. length el > 0 ∧ (∀ i. i < length el ⟶ getspc-e (el!i) = es) ⟶ el ∈ cpts-ev
      proof −
      {
        fix el es
        assume a0: length el > 0
          and a1: ∀ i. i < length el ⟶ getspc-e (el!i) = es
```

then have *el* ∈ *cpts-ev*
  proof(*induct el*)
    case *Nil* **show** *?case* **using** *Nil.prems*(*1*) **by** *auto*
  next
    case (*Cons a as*)
    assume *b0*: $0 < length\ as \Longrightarrow \forall i<length\ as.\ getspc\text{-}e\ (as\ !\ i) = es \Longrightarrow as \in cpts\text{-}ev$
      and *b1*: $0 < length\ (a\ \#\ as)$
      and *b2*: $\forall i<length\ (a\ \#\ as).\ getspc\text{-}e\ ((a\ \#\ as)\ !\ i) = es$
    then show *?case*
      proof(*cases as* = [])
        assume *c0*: *as* = []
        then show *?thesis* **by** (*metis cpts-ev.CptsEvOne old.prod.exhaust*)
      next
        assume *c0*: ¬(*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** $\forall i<length\ as.\ getspc\text{-}e\ (as\ !\ i) = es \Longrightarrow as \in cpts\text{-}ev$ **by** *simp*
        with *b2* **have** *as* ∈ *cpts-ev* **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

4.4.3 Event systems

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

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

lemma *esetran-eqconf1*: *esc1* −*ese*→ *esc2* ⟹ *getspc-es esc1* = *getspc-es esc2*
  proof −
    assume *a0*: *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* ⟹ (*es1*, *s1*, *x1*) −*ese*→ (*es2*, *s2*, *x2*)
  by (*simp add*: *esetran.intros*)

lemma *eqconf-esetran*: *getspc-es esc1* = *getspc-es esc2* ⟹ *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: (es1, s1, x1) −ese→ (es2, s2, x2) by (simp add:eqconf-esetran1)
    from a3 a1 a2 show ?thesis by simp
  qed

lemma exist-estran: ⟦(es1, s1, x1) # (es, s, x) # esl ∈ cpts-es; es1 ≠ es⟧ ⟹ (∃ est. (es1, s1, x1) −es−est→ (es, s,
x))
  apply(rule cpts-es.cases)
  apply(simp)+
  by auto

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

lemma cpts-es-dropi: ⟦el ∈ cpts-es; Suc i < length el⟧ ⟹ drop (Suc i) el ∈ cpts-es
  proof −
    assume p0:el ∈ cpts-es and p1:Suc i < length el
    have ∀ el i. el ∈ cpts-es ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-es
      proof −
      {
        fix el i
        have el ∈ cpts-es ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-es
          proof(induct i)
            case 0 show ?case by (simp add: cpts-es-drop0)
          next
            case (Suc j)
            assume b0: el ∈ cpts-es ∧ Suc j < length el ⟶ drop (Suc j) el ∈ cpts-es
            show ?case
              proof
                assume c0: el ∈ cpts-es ∧ Suc (Suc j) < length el
                with b0 have c1: drop (Suc j) el ∈ cpts-es
                  by (simp add: c0 Suc-lessD)
                then show drop (Suc (Suc j)) el ∈ cpts-es
                  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 ∈ cpts-es; i < length el⟧ ⟹ drop i el ∈ cpts-es
  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; i < length el; el1 = take (Suc i) el; j < length el1⟧
                        ⟹ drop (length el1 − Suc j) el1 ∈ cpts-es
  proof −
    assume p0: el ∈ cpts-es
      and p1: i < length el
      and p2: el1 = take (Suc i) el
```

```
  and  p3: j < length el1
have ∀ i j. el ∈ cpts-es ∧ i < length el ∧ el1 = take (Suc i) el ∧ j < length el1
    ⟶ drop (length el1 − Suc j) el1 ∈ cpts-es
  proof −
  {
    fix i j
    assume a0: el ∈ cpts-es
      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
      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 ⟹ i < length el ⟹ el1 = take (Suc i) el
                ⟹ jj < length el1 ⟹ drop (length el1 − Suc jj) el1 ∈ cpts-es
          and  b1: el ∈ cpts-es
          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
          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
          by (metis a7 append-is-Nil-conv b5 cpts-es-dropi2 drop-all not-less)
        from b1 b4 have a11: ?ellng ∈ cpts-es
          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
          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
              and  c1: (es1, t1, x1) # xs1 = drop (length el1 − Suc (Suc jj)) el ⟹
                    drop (length el1 − Suc (Suc jj)) el1 ∈ cpts-es
              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
            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: (es1, s1, x1) −es−et→ (es2, t1, y1)
            and  c1: (es2, t1, y1) # xs1 ∈ cpts-es
            and  c2: (es2, t1, y1) # xs1 = drop (length el1 − Suc (Suc jj)) el
                   ⟹ drop (length el1 − Suc (Suc jj)) el1 ∈ cpts-es
            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: ∃ 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
   using p0 p1 p2 p3 by blast
qed


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

lemma cpts-es-seg: ⟦el ∈ cpts-es; m ≤ length el; n ≤ length el; m < n⟧
                 ⟹ take (n − m) (drop m el) ∈ cpts-es
  proof −
    assume p0: el ∈ cpts-es
      and  p1: m ≤ length el
      and  p2: n ≤ length el
      and  p3: m < n
    then have drop m el ∈ cpts-es
      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; m ≤ length el; n ≤ length el; take (n − m) (drop m el) ≠ []⟧
                 ⟹ take (n − m) (drop m el) ∈ cpts-es
  proof −
    assume p0: el ∈ cpts-es
      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⟧ ⟹ el ∈ cpts-es
  proof −
    assume p0: length el > 0
      and  p1: ∀ i. i < length el ⟶ getspc-es (el!i) = es
    have ∀ el es. length el > 0 ∧ (∀ i. i < length el ⟶ getspc-es (el!i) = es) ⟶ el ∈ cpts-es
      proof −
      {
        fix el es
        assume a0: length el > 0
          and  a1: ∀ i. i < length el ⟶ getspc-es (el!i) = es
        then have el ∈ cpts-es
          proof(induct el)
            case Nil show ?case using Nil.prems(1) by auto
          next
            case (Cons a as)
            assume b0: 0 < length as ⟹ ∀ i<length as. getspc-es (as ! i) = es ⟹ as ∈ cpts-es
              and  b1: 0 < length (a # as)
              and  b2: ∀ 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: ¬(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 ∀ i<length as. getspc-es (as ! i) = es ⟹ as ∈ cpts-es by simp
                with b2 have as ∈ cpts-es 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:
    (¬ (∃ j. j > 0 ∧ Suc j < length esl ∧ getspc-es (esl ! j) = EvtSys es ∧ getspc-es (esl ! Suc j) ≠ EvtSys es))
      = (∀ j. j > 0 ∧ Suc j < length esl ⟶ getspc-es (esl ! j) = EvtSys es ⟶ getspc-es (esl ! Suc j) = EvtSys es)
      by blast

lemma evtseq-next-in-cpts:
  esl∈cpts-es ⟹ ∀ i. Suc i < length esl ∧ getspc-es (esl!i) = EvtSeq e esys
                    ⟶ getspc-es (esl!Suc i) = esys ∨ (∃ e. getspc-es (esl!Suc i) = EvtSeq e esys)
  proof −
    assume p0: esl∈cpts-es
    then show ?thesis
```

```
    proof -
    {
      fix i
      assume a0: Suc i < length esl
        and a1: getspc-es (esl!i) = EvtSeq e esys
      let ?esl1 = drop i esl
      from p0 a0 have a2: ?esl1∈cpts-es 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)
        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
          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 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: (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) = EvtSeq e esys
          then have b3: es1' = EvtSeq e esys
            by (metis Pair-inject a3 nth-Cons-0)
          from b0 b3 have es2' = esys ∨ (∃ 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 ⟹ ∀ 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
    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 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: (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) = EvtSeq e esys ∧ is-anonyevt e
        then have b3: es1' = EvtSeq e esys
          by (metis Pair-inject a3 nth-Cons-0)
        from b0 b3 have es2' = esys ∨ (∃ e. es2' = EvtSeq e esys ∧ 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
      ∨ (∃ e. getspc-es (esl!Suc i) = EvtSeq e esys ∧ is-anonyevt e)
      using a0 by fastforce
  }
  then show ?thesis by auto
  qed
qed

lemma evtsys-next-in-cpts:
  esl∈cpts-es ⟹ ∀ 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))
  proof −
    assume p0: esl∈cpts-es
    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 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))
        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: (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 ⟹ ∀ 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
    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 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*: (*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*∈*cpts-es*; *length esl* > *0*; *getspc-es* (*esl*!*0*) = *EvtSys es* ⟧ ⟹
    ∀ *i*. *i* < *length esl* ⟶ *getspc-es* (*esl*!*i*) = *EvtSys es* ∨ (∃ *e*. *getspc-es* (*esl*!*i*) = *EvtSeq e* (*EvtSys es*))
proof −
  assume *p0*: *esl*∈*cpts-es*
    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*))
        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))
          and  b1: Suc j < length esl
        then have getspc-es (esl ! j) = EvtSys es ∨ (∃ 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: ∃ e. getspc-es (esl ! j) = EvtSeq e (EvtSys es)
            with p0 b1 show ?thesis using evtseq-next-in-cpts by auto
          qed
      qed
    }
    then show ?thesis by auto
    qed
  qed

lemma evtsys-all-es-in-cpts-anony:
  ⟦esl∈cpts-es; length esl > 0; getspc-es (esl!0) = EvtSys es ⟧ ⟹
      ∀ i. i < length esl ⟶ getspc-es (esl!i) = EvtSys es
          ∨ (∃ e. getspc-es (esl!i) = EvtSeq e (EvtSys es) ∧ is-anonyevt e)
  proof −
    assume p0: esl∈cpts-es
      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 auto
              qed
          qed
      }
      then show ?thesis by auto
      qed
  qed
```

```
lemma not-anonyevt-none-in-evtseq:
  ⟦esl∈cpts-es; esl = (EvtSeq e es,s1,x1)#(es,s2,x2)#xs ⟧ ⟹ e ≠ AnonyEvent None
 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)
 apply(rule ptran.cases)
 apply(simp)+
 done

lemma not-anonyevt-none-in-evtseq1:
  ⟦esl∈cpts-es; length esl > 1; getspc-es (esl!0) = EvtSeq e es;
   getspc-es (esl!1) = es ⟧ ⟹ e ≠ AnonyEvent None
 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:
  ⟦esl∈cpts-es; esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs⟧ ⟹
      ∃ t. (EvtSys es, s, x) −es−t→ (EvtSeq ev (EvtSys es), s1,x1)
 apply(rule cpts-es.cases)
 apply(simp)+
 apply blast
 by blast

lemma fst-esys-snd-eseq-exist-evtent2:
  ⟦esl∈cpts-es; esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs⟧ ⟹
      ∃ e k. (EvtSys es, s, x) −es−(EvtEnt (BasicEvent e))♯k→ (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:
 ⟦esl∈cpts-es; length esl ≥ 2 ∧ getspc-es (esl!0) = EvtSys es ∧ getspc-es (esl!1) ≠ EvtSys es⟧
  ⟹ ∃ s x ev s1 x1 xs. esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs
 proof −
  assume a0: length esl ≥ 2 ∧ getspc-es (esl!0) = EvtSys es ∧ getspc-es (esl!1) ≠ EvtSys es
   and  c1: esl∈cpts-es
  from a0 have b0: getspc-es (esl!0) = EvtSys es ∧ getspc-es (esl!1) ≠ EvtSys es
   by (metis (no-types, lifting))

  from a0 have b1: 2 ≤ length esl by fastforce
  moreover from b0 b1 have ∃ s x. esl!0 = (EvtSys es, s, x) using getspc-es-def
   by (metis eq-fst-iff)
  moreover have ∃ ev s1 x1. esl!1 = (EvtSeq ev (EvtSys es), s1,x1) using getspc-es-def
   proof −
    from c1 a0 b0 have ∃ 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; length esl ≥ 2; ¬ (∃ e k. esl ! 0 −es−EvtEnt e♯k→ esl ! 1)⟧
    ⟹ esl ! 0 −ese→ esl ! 1 ∨ (∃ c k. 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 ⟹ ∀ 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
    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 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: (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 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* ⟹ ∀ *i*. *Suc i* < *length esl* ⟶ *esl* ! *i* −*ese*→ *esl* ! *Suc i* ∨ (∃ *et*. *esl* ! *i* −*es*−*et*→ *esl* ! *Suc i*)
proof −
assume *p0*: *esl*∈*cpts-es*
{
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*
using *cpts-es-seg*[*of esl i Suc* (*Suc i*)] **by** *simp*
then have *?esl1* ∈ *cpts-es* **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*
by (*metis Cons-nth-drop-Suc Suc-lessD a0 numeral-2-eq-2 take-0 take-Suc-Cons*)
then have (*esc1*, *s1*, *x1*) −*ese*→ (*esc2*, *s2*, *x2*) ∨ (∃ *et*. (*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** *esl* ! *i* −*ese*→ *esl* ! *Suc i* ∨ (∃ *et*. *esl* ! *i* −*es*−*et*→ *esl* ! *Suc i*) **by** *simp*
}
then show *?thesis* **by** *auto*
qed

lemma *incpts-es-eseq-not-evtent*:
⟦*esl*∈*cpts-es*; *Suc i* < *length esl*; ∃ *e esys*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*⟧
⟹ ¬(∃ *e k*. *t* = *EvtEnt e* ∧ *esl*!*i* −*es*−*t*♯*k*→ *esl*!*Suc i*)
proof −
assume *p0*: *esl*∈*cpts-es*
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* **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* ∧ *?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*: (*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*:(*P*,*s*,*x*) −*es*−*est*→ (*Q*,*t*,*y*) ⟹ *P* ≠ *Q*
apply(*erule estran.cases*)
apply (*simp add*: *evt-not-eq-in-tran-aux*)
apply (*simp add*: *evt-not-eq-in-tran-aux*)
by (*metis add.right-neutral add-Suc-right esys.size*(*3*) *lessI less-irrefl trans-less-add2*)

lemma *evtsys-not-eq-in-tran-aux1*:*esc1* −*es*−*est*→ *esc2* ⟹ *getspc-es esc1* ≠ *getspc-es esc2*
proof −
assume *p0*: *esc1* −*es*−*est*→ *esc2*
obtain *es1* **and** *s1* **and** *x1* **and** *es2* **and** *s2* **and** *x2* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*) ∧ *esc2* = (*es2*,*s2*,*x2*)
by (*metis prod.collapse*)
with *p0* **have** *es1* ≠ *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*]: ¬ (*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*]: ¬(∃ *est*. (*P*,*s*,*x*) −*es*−*est*→ (*P*,*t*,*y*)) **by** *simp*

lemma *es-tran-not-etran2*: (*P*,*s*,*x*) −*es*−*pt*→ (*Q*,*t*,*y*) ⟹ ¬((*P*,*s*,*x*) −*ese*→(*Q*,*t*,*y*))
by (*metis esetran.cases evtsys-not-eq-in-tran-aux*)

lemma *es-tran-not-etran1*: *esc1* −*es*−*pt*→ *esc2* ⟹ ¬(*esc1* −*ese*→*esc2*)
using *esetran-eqconf1 evtsys-not-eq-in-tran-aux1* **by** *blast*

4.4.4 Parallel event systems

lemma *cpts-pes-not-empty* [*simp*]:[] ∉ *cpts-pes*

apply(*force elim*:*cpts-pes.cases*)
done

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

lemma *pesetran-eqconf1*: *esc1* −*pese*→ *esc2* ⟹ *getspc esc1* = *getspc esc2*
  proof −
    assume *a0*: *esc1* −*pese*→ *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* ⟹ (*es1*, *s1*, *x1*) −*pese*→ (*es2*, *s2*, *x2*)
  by (*simp add*: *pesetran.intros*)

lemma *eqconf-pesetran*: *getspc esc1* = *getspc esc2* ⟹ *esc1* −*pese*→ *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*: (*es1*, *s1*, *x1*) −*pese*→ (*es2*, *s2*, *x2*) **by** (*simp add*:*eqconf-pesetran1*)
    from *a3 a1 a2* **show** *?thesis* **by** *simp*
  qed

lemma *pestran-cpts-pes*: ⟦*C1* −*pes*−*ct*→ *C2*; *C2*#*xs* ∈ *cpts-pes*⟧ ⟹ *C1*#*C2*#*xs* ∈ *cpts-pes*
  proof −
    assume *p0*: *C1* −*pes*−*ct*→ *C2*
      and *p1*: *C2*#*xs* ∈ *cpts-pes*
    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* ∈ *cpts-pes*; (*el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (*el* ! (*length el* − *1*) −*pese*→ *ec*)⟧ ⟹
      el @ [*ec*] ∈ *cpts-pes*
  proof −
    assume *p0*: *el* ∈ *cpts-pes*
      and *p2*: (*el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (*el* ! (*length el* − *1*) −*pese*→ *ec*)
    from *p0* **have** *p1*: *el* ≠ [] **by** *auto*
    have ∀ *el ec t*. *el* ∈ *cpts-pes* ∧ ((*el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (*el* ! (*length el* − *1*) −*pese*→ *ec*))
      ⟶ *el* @ [*ec*] ∈ *cpts-pes*
      proof −
      {
        fix *el ec t*
        assume *a0*: *el* ∈ *cpts-pes*
          and *a2*: (*el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (*el* ! (*length el* − *1*) −*pese*→ *ec*)

```
then have a1: length el > 0 by auto
from a0 a1 a2 have el @ [ec] ∈ cpts-pes
  proof(induct el)
    case (CptsPesOne e s x)
    assume b0: ([(e, s, x)] ! (length [(e, s, x)] − 1) −pes−t→ ec)
                ∨ [(e, s, x)] ! (length [(e, s, x)] − 1) −pese→ ec
    then have ((e, s, x) −pes−t→ ec) ∨ ((e, s, x) −pese→ ec) by simp
    then show ?case
      proof
        assume (e, s, x) −pes−t→ ec
        then show ?thesis by (metis append-Cons append-Nil
            cpts-pes.CptsPesComp cpts-pes.CptsPesOne surj-pair)
      next
        assume (e, s, x) −pese→ 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 ∈ cpts-pes
      and b1: 0 < length ((e, s1, x) # xs) ⟹
                (((e, s1, x) # xs) ! (length ((e, s1, x) # xs) − 1) −pes−t→ ec) ∨
                (((e, s1, x) # xs) ! (length ((e, s1, x) # xs) − 1) −pese→ ec) ⟹
                ((e, s1, x) # xs) @ [ec] ∈ cpts-pes
      and b2: 0 < length ((e, s2, y) # (e, s1, x) # xs)
      and b3: (((e, s2, y) # (e, s1, x) # xs) ! (length ((e, s2, y) # (e, s1, x) # xs) − 1) −pes−t→ ec) ∨
              (((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 ((e, s1, x) −pes−t→ ec) ∨ ((e, s1, x) −pese→ ec) by simp
        with b1 c0 have ((e, s1, x) # xs) @ [ec] ∈ cpts-pes by simp
        then show ?thesis by (simp add: cpts-pes.CptsPesEnv)
      next
        assume c0: xs ≠ []
        with b3 have (last xs −pes−t→ ec) ∨ (last xs −pese→ ec) by (simp add: last-conv-nth)
        with b1 c0 have ((e, s1, x) # xs) @ [ec] ∈ cpts-pes 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: (e1, s1, x1) −pes−et→ (e2, t1, y1)
      and b1: (e2, t1, y1) # xs1 ∈ cpts-pes
      and b2: 0 < length ((e2, t1, y1) # xs1) ⟹
              (((e2, t1, y1) # xs1) ! (length ((e2, t1, y1) # xs1) − 1) −pes−t→ ec) ∨
              (((e2, t1, y1) # xs1) ! (length ((e2, t1, y1) # xs1) − 1) −pese→ ec) ⟹
              ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-pes
      and b3: 0 < length ((e1, s1, x1) # (e2, t1, y1) # xs1)
     and b4: (((e1, s1, x1) # (e2, t1, y1) # xs1) ! (length ((e1, s1, x1) # (e2, t1, y1) # xs1) − 1) −pes−t→
ec) ∨
              ((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 ((e2, t1, y1) −pes−t→ ec) ∨ ((e2, t1, y1) −pese→ ec) by simp
        with b2 c0 have ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-pes by simp
        with b0 show ?thesis using cpts-pes.CptsPesComp by fastforce
```

```
          next
            assume c0: xs1 ≠ []
            with b4 have (last xs1 −pes−t→ ec) ∨ (last xs1 −pese→ ec) by (simp add: last-conv-nth)
            with b2 c0 have ((e2, t1, y1) # xs1) @ [ec] ∈ cpts-pes 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] ∈ cpts-pes using p0 p1 p2 by blast
  qed

lemma pes-not-eq-in-tran-aux:(P,s,x) −pes−est→ (Q,t,y) ⟹ P ≠ Q
  apply(erule pestran.cases)
  by (metis evtsys-not-eq-in-tran-aux fun-upd-apply)

lemma pes-not-eq-in-tran [simp]: ¬ (P,s,x) −pes−est→ (P,t,y)
  apply clarify
  apply(drule pes-not-eq-in-tran-aux)
  apply simp
  done

lemma pes-tran-not-etran1: pes1 −pes−t→ pes2 ⟹ ¬(pes1 −pese→pes2)
  by (metis pes-not-eq-in-tran pesetranE surj-pair)

lemma pes-tran-not-etran2: (P,s,x) −pes−pt→ (Q,t,y) ⟹ ¬((P,s,x) −pese→(Q,t,y))
  by (simp add: pes-tran-not-etran1)

lemma incpts-pes-impl-evnorcomptran:
  esl∈cpts-pes ⟹ ∀ i. Suc i < length esl ⟶ esl ! i −pese→ esl ! Suc i ∨ (∃ et. esl ! i −pes−et→ esl ! Suc i)
  proof −
    assume p0: esl∈cpts-pes
    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 ∈ cpts-pes
          and  a1: ∀ i. Suc i < length ((pes, t, x) # xs) ⟶
                     ((pes, t, x) # xs) ! i −pese→ ((pes, t, x) # xs) ! Suc i ∨
                     (∃ et. ((pes, t, x) # xs) ! i −pes−et→ ((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 ((pes, s, y) # (pes, t, x) # xs) ! i −pese→ ((pes, s, y) # (pes, t, x) # xs) ! Suc i ∨
                  (∃ et. ((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: (pes1, s, x) −pes−ct→ (pes2, t, y)
        and  a1: (pes2, t, y) # xs ∈ cpts-pes
        and  a2: ∀ i. Suc i < length ((pes2, t, y) # xs) ⟶
                  ((pes2, t, y) # xs) ! i −pese→ ((pes2, t, y) # xs) ! Suc i ∨
                  (∃ et. ((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 ((pes1, s, x) # (pes2, t, y) # xs) ! i −pese→ ((pes1, s, x) # (pes2, t, y) # xs) ! Suc i ∨
               (∃ et. ((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; Suc 0 < length el]] ⟹ drop (Suc 0) el ∈ cpts-pes
  apply(rule cpts-pes.cases)
  apply(simp)+
  done

lemma cpts-pes-dropi: [[el ∈ cpts-pes; Suc i < length el]] ⟹ drop (Suc i) el ∈ cpts-pes
  proof −
    assume p0:el ∈ cpts-pes and p1:Suc i < length el
    have ∀ el i. el ∈ cpts-pes ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-pes
      proof −
      {
        fix el i
        have el ∈ cpts-pes ∧ Suc i < length el ⟶ drop (Suc i) el ∈ cpts-pes
          proof(induct i)
            case 0 show ?case by (simp add: cpts-pes-drop0)
          next
            case (Suc j)
            assume b0: el ∈ cpts-pes ∧ Suc j < length el ⟶ drop (Suc j) el ∈ cpts-pes
            show ?case
              proof
                assume c0: el ∈ cpts-pes ∧ Suc (Suc j) < length el
                with b0 have c1: drop (Suc j) el ∈ cpts-pes
                  by (simp add: c0 Suc-lessD)
                then show drop (Suc (Suc j)) el ∈ cpts-pes
                  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; i < length el; el1 = take (Suc i) el; j < length el1⟧
                    ⟹ drop (length el1 − Suc j) el1 ∈ cpts-pes
  proof −
    assume p0: el ∈ cpts-pes
      and  p1: i < length el
      and  p2: el1 = take (Suc i) el
      and  p3: j < length el1
    have ∀ i j. el ∈ cpts-pes ∧ i < length el ∧ el1 = take (Suc i) el ∧ j < length el1
          ⟶ drop (length el1 − Suc j) el1 ∈ cpts-pes
      proof −
      {
        fix i j
        assume a0: el ∈ cpts-pes
          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
          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 ⟹ i < length el ⟹ el1 = take (Suc i) el
                      ⟹ jj < length el1 ⟹ drop (length el1 − Suc jj) el1 ∈ cpts-pes
              and  b1: el ∈ cpts-pes
              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
              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
              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
              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 ∈ cpts-pes
          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 ∈ cpts-pes
              and c1: (es1, t1, x1) # xs1 = drop (length el1 − Suc (Suc jj)) el ⟹
                    drop (length el1 − Suc (Suc jj)) el1 ∈ cpts-pes
              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-pes
              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-pes.CptsPesEnv by fastforce
          next
            case (CptsPesComp es1 s1 x1 et es2 t1 y1 xs1)
            assume c0: (es1, s1, x1) −pes−et→ (es2, t1, y1)
              and c1: (es2, t1, y1) # xs1 ∈ cpts-pes
              and c2: (es2, t1, y1) # xs1 = drop (length el1 − Suc (Suc jj)) el
                   ⟹ drop (length el1 − Suc (Suc jj)) el1 ∈ cpts-pes
              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: ∃ 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 ∈ cpts-pes
    using p0 p1 p2 p3 by blast
qed

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

lemma cpts-pes-seg: ⟦el ∈ cpts-pes; m ≤ length el; n ≤ length el; m < n⟧
          ⟹ take (n − m) (drop m el) ∈ cpts-pes
  proof −
    assume p0: el ∈ cpts-pes
      and p1: m ≤ length el
      and p2: n ≤ length el
      and p3: m < n
```

  then have *drop m el* $\in$ *cpts-pes*
    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*; $m \leq$ *length el*; $n \leq$ *length el*; *take* $(n - m)$ (*drop m el*) $\neq [\,]]\!]$
    $\Longrightarrow$ *take* $(n - m)$ (*drop m el*) $\in$ *cpts-pes*
 proof −
  assume *p0*: *el* $\in$ *cpts-pes*
   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

4.5 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* $\in$ *cpt-p-mod* $\Longrightarrow$
 ($\forall$ *s P Q zs*. *list*=(*Some* (*Seq P Q*), *s*)*#zs* $\longrightarrow$
  ($\exists$ *xs*. (*Some P*, *s*)*#xs* $\in$ *cpt-p-mod* $\wedge$ (*zs*=(*map* (*lift Q*) *xs*) $\vee$
  (*fst*(((*Some P*, *s*)*#xs*)!*length xs*)=*None* $\wedge$
  ($\exists$ *ys*. (*Some Q*, *snd*(((*Some P*, *s*)*#xs*)!*length xs*))*#ys* $\in$ *cpt-p-mod*
  $\wedge$ *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*
— basic
apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*rule Basic*,*simp*)
apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
— Seq1
apply(*rule-tac xs*=[(*None*,*ta*)] **in** *CptPModSeq2*)
apply(*erule CptPModNone*)
apply(*rule CptPModOne*)
apply *simp*
apply *simp*
apply(*simp add*:*lift-def*)
— Seq2
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*)
— Cond
apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*force elim*: *CptPModCondT*)
apply(*force elim*: *CptPModCondF*)
— While
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*)

— await
apply *clarify*
apply(*erule ptran.cases*,*simp-all*)
apply(*rule CptPModNone*,*erule Await*,*simp+*)
— nondt
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-p* $\Longrightarrow c \in$ *cpt-p-mod*
apply(*erule cpts-p.induct*)
  apply(*rule CptPModOne*)
 apply(*erule CptPModEnv*)
apply(*case-tac P*)
 apply *simp*
 apply(*erule ptran.cases*,*simp-all*)
apply(*force elim*:*cpts-onlyif-cpt-p-mod-aux*)
done

lemma *lift-is-cptn*: $c\in$*cpts-p* $\Longrightarrow$ *map* (*lift P*) *c* $\in$ *cpts-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-p* $\longrightarrow$ *a*#*c2*$\in$*cpts-p* $\longrightarrow$ (*b*#*c1*)!*length c1*=*a* $\longrightarrow$ *b*#*c1*@*c2*$\in$*cpts-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 ⟹ map (lift Q) xs ! i = lift Q  (xs! i)
  by (simp add:lift-def)

lemma snd-lift: i< length xs ⟹ snd(lift Q (xs ! i))= snd (xs ! i)
  by (cases xs!i) (simp add:lift-def)

lemma cpts-if-cpt-p-mod: c ∈ cpt-p-mod ⟹ c ∈ cpts-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*: $(c \in \textit{cpts-p}) = (c \in \textit{cpt-p-mod})$
apply(*rule iffI*)
apply(*erule cpts-onlyif-cpt-p-mod*)
apply(*erule cpts-if-cpt-p-mod*)
done

4.6 Compositionality of the Semantics

4.6.1 Definition of the conjoin operator

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

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

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

4.6.2 Lemmas of conjoin

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

proof
assume $b0$: $\exists\, t\ k.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\, k\ t.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (cs\ k!i\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$
then obtain t **and** $k1$ **where** $b1$: $(c!i\ -pes-(t\sharp k1)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\, k\ t.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (cs\ k!i\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$ **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 $b0$: $((c!i)\ -pese\rightarrow\ (c!Suc\ i)) \wedge (\forall\, k.\ (((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)))$
then show *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *simp*
qed

lemma *entevt-in-conjoin-cpts*:
$[\![c \propto cs;\ Suc\ i < length\ (cs\ k);\ getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es;$
$getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es\ ]\!]$
$\Longrightarrow (\exists\, e.\ ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow\ ((cs\ k)!\ Suc\ i))$
proof $-$
assume $p0$: $c \propto cs$
and $p1$: $Suc\ i < length\ (cs\ k)$
and $p2$: $getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es$
and $p3$: $getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es$
then have $((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$
$\vee (\exists\, e.\ ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow\ ((cs\ k)!\ Suc\ i))$
$\vee (\exists\, c.\ ((cs\ k)!i)\ -es-(Cmd\ c\sharp k)\rightarrow\ ((cs\ k)!\ Suc\ i))$
using *acts-in-conjoin-cpts* **by** *fastforce*
then show *?thesis*
proof
assume $((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$
with $p2\ p3$ **show** *?thesis* **by** (*simp add*: *esetran-eqconf1*)
next
assume $(\exists\, e.\ cs\ k\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ cs\ k\ !\ Suc\ i)$
$\vee (\exists\, c.\ cs\ k\ !\ i\ -es-Cmd\ c\sharp k\rightarrow\ cs\ k\ !\ Suc\ i)$
then show *?thesis*
proof
assume $\exists\, e.\ cs\ k\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ cs\ k\ !\ Suc\ i$
then show *?thesis* **by** *simp*
next
assume $\exists\, c.\ cs\ k\ !\ i\ -es-Cmd\ c\sharp k\rightarrow\ cs\ k\ !\ Suc\ i$
with $p2\ p3$ **show** *?thesis*
by (*meson cmd-enable-impl-anonyevt2 esys-not-eseq*)
qed
qed
qed

lemma *notentevt-in-conjoin-cpts*:
$[\![c \propto cs;\ Suc\ i < length\ (cs\ k);\ \neg(getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es \wedge getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es);$
$\forall\, i < length\ (cs\ k).\ getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSys\ es$

$\lor (\exists e.\ \textit{is-anonyevt } e \land \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSeq } e\ (\textit{EvtSys } es))]\!]$

$\Longrightarrow \neg(\exists e.\ ((cs\ k)!i) -es-(\textit{EvtEnt } e\sharp k)\rightarrow ((cs\ k)!\ \textit{Suc } i))$

proof −

assume *p0*: $c \propto cs$

and *p1*: $\textit{Suc } i < \textit{length } (cs\ k)$

and *p2*: $\neg(\textit{getspc-es } ((cs\ k)!i) = \textit{EvtSys } es \land \textit{getspc-es } ((cs\ k)!\textit{Suc } i) \neq \textit{EvtSys } es)$

and *p3*: $\forall i < \textit{length } (cs\ k).\ \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSys } es$

$\lor (\exists e.\ \textit{is-anonyevt } e \land \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSeq } e\ (\textit{EvtSys } es))$

from *p2* **have** $\textit{getspc-es } ((cs\ k)!i) \neq \textit{EvtSys } es \lor \textit{getspc-es } ((cs\ k)!\textit{Suc } i) = \textit{EvtSys } es$ **by** *simp*

with *p3* **have** $(\exists e.\ \textit{is-anonyevt } e \land \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSeq } e\ (\textit{EvtSys } es))$

$\lor\ \textit{getspc-es } ((cs\ k)!\textit{Suc } i) = \textit{EvtSys } es$

using *Suc-lessD p1* **by** *blast*

then show *?thesis*

proof

assume $\exists e.\ \textit{is-anonyevt } e \land \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSeq } e\ (\textit{EvtSys } es)$

then obtain *e1* **where** $\textit{is-anonyevt } e1 \land \textit{getspc-es } ((cs\ k)\ !\ i) = \textit{EvtSeq } e1\ (\textit{EvtSys } es)$ **by** *auto*

then show *?thesis* **using** *evtent-is-basicevt-inevtseq2* **by** *fastforce*

next

assume $\textit{getspc-es } ((cs\ k)!\textit{Suc } i) = \textit{EvtSys } es$

then show *?thesis* **by** (*metis Suc-lessD evtseq-no-evtent2 evtsys-not-eq-in-tran-aux1 p1 p3*)

qed

qed

lemma *take-n-conjoin*: $[\![c \propto cs;\ n \leq \textit{length } c;\ c1 = \textit{take } n\ c;\ cs1 = (\lambda k.\ \textit{take } n\ (cs\ k))]\!]$

$\Longrightarrow c1 \propto cs1$

proof −

assume *p0*: $c \propto cs$

and *p1*: $n \leq \textit{length } c$

and *p2*: $c1 = \textit{take } n\ c$

and *p3*: $cs1 = (\lambda k.\ \textit{take } n\ (cs\ k))$

have *a0*: *same-length c1 cs1* **by** (*metis conjoin-def length-take p0 p2 p3 same-length-def*)

then have *a1*: $\forall k.\ \textit{length } (cs1\ k) = \textit{length } c1$ **by** (*simp add:same-length-def*)

have *same-state c1 cs1*

proof −

{

fix *k j*

assume *b0*: $j < \textit{length } c1$

from *p1 p3 a1* **have** *b1*: $cs1\ k = \textit{take } n\ (cs\ k)$ **by** *simp*

from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < \textit{length } c$

$\longrightarrow \textit{gets } (c!j) = \textit{gets-es } ((cs\ k)!j) \land \textit{getx } (c!j) = \textit{getx-es } ((cs\ k)!j)$

by (*simp add:conjoin-def same-state-def*)

from *p2 b1 b0* **have** $\textit{gets } (c\ !\ j) = \textit{gets } (c1\ !\ j) \land \textit{gets-es } ((cs\ k)!j) = \textit{gets-es } ((cs1\ k)!j)$

$\land\ \textit{getx } (c!j) = \textit{getx } (c1!j)$

by (*simp add: nth-append*)

with *p1 p2 b1 b2*[*of j k*] *b0* **have** $\textit{gets } (c1!j) = \textit{gets-es } ((cs1\ k)!j) \land \textit{getx } (c1!j) = \textit{getx-es } ((cs1\ k)!j)$

by *simp*

}

then show *?thesis* **by** (*simp add:same-state-def*)

qed

moreover

have *same-spec c1 cs1*

proof −

{

fix *k j*

assume *b0*: $j < \textit{length } c1$

from *p1 p3 a1* **have** *b1*: $cs1\ k = \textit{take } n\ (cs\ k)$ **by** *simp*

from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < \textit{length } c$

```
        ⟶ (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!j)
      ∧ 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 c1 cs1
  proof −
  {
    fix j
    assume b0: Suc j < length c1
    with p0 p2 have ((∃ t k. (c!j −pes−(t♯k)→ c!Suc j)) ∧
                (∀ k t. (c!j −pes−(t♯k)→ c!Suc j) ⟶ (cs k!j −es−(t♯k)→ cs k! Suc j) ∧
                    (∀ k'. k' ≠ k ⟶ (cs k'!j −ese→ cs k'! Suc j))))
                ∨
                (((c!j) −pese→ (c!Suc j)) ∧ (∀ k. (((cs k)!j) −ese→ ((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 ∀ k. cs1 k!j = cs k!j
      by (simp add: Suc-lessD)
    moreover
    from p1 p2 p3 a1 b0 have ∀ k. cs1 k!Suc j = cs k!Suc j
      by (simp add: Suc-lessD)
    ultimately
    have ((∃ t k. (c1!j −pes−(t♯k)→ c1!Suc j)) ∧
              (∀ k t. (c1!j −pes−(t♯k)→ c1!Suc j) ⟶ (cs1 k!j −es−(t♯k)→ cs1 k! Suc j) ∧
                  (∀ k'. k' ≠ k ⟶ (cs1 k'!j −ese→ cs1 k'! Suc j))))
              ∨
              (((c1!j) −pese→ (c1!Suc j)) ∧ (∀ k. (((cs1 k)!j) −ese→ ((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: ⟦c ∝ cs; n ≤ length c; c1 = drop n c; cs1 = (λk. drop n (cs k))⟧
  ⟹ c1 ∝ cs1
proof −
  assume p0: c ∝ cs
    and p1: n ≤ length c
    and p2: c1 = drop n c
    and p3: cs1 = (λk. 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: ∀ 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*]: $\forall k\ j.\ j < length\ c$
$\longrightarrow gets\ (c!j) = gets\text{-}es\ ((cs\ k)!j) \wedge getx\ (c!j) = getx\text{-}es\ ((cs\ k)!j)$
by (*simp add:conjoin-def same-state-def*)
from *p2 b1 b0* **have** $gets\ (c\ !\ (n + j)) = gets\ (c1\ !\ j) \wedge gets\text{-}es\ ((cs\ k)!(n + j)) = gets\text{-}es\ ((cs1\ k)!j)$
$\wedge\ getx\ (c!(n + j)) = getx\ (c1!j)$
proof −
have *f1*: $n + j \leq length\ c$
using *b0 p2* **by** *auto*
then have $n + j \leq length\ (cs\ k)$
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** $gets\ (c1!j) = gets\text{-}es\ ((cs1\ k)!j) \wedge getx\ (c1!j) = getx\text{-}es\ ((cs1\ k)!j)$
by (*metis* (*no-types*, *lifting*) *a1 add.commute length-drop less-diff-conv less-or-eq-imp-le nth-drop*)
}
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*]: $\forall k\ j.\ j < length\ c$
$\longrightarrow (getspc\ (c!j))\ k = getspc\text{-}es\ ((cs\ k)\ !\ j)$
by (*simp add:conjoin-def same-spec-def*)
from *p2 b1 b0* **have** $getspc\ (c1!j) = getspc\ (c!(n{+}j))$
$\wedge\ getspc\text{-}es\ ((cs\ k)\ !\ (n{+}j)) = getspc\text{-}es\ ((cs1\ k)\ !\ j)$
proof −
have *f1*: $n + j \leq length\ c$
using *b0 p2* **by** *auto*
then have $n + j \leq length\ (cs\ k)$
by (*metis* (*no-types*) *conjoin-def p0 same-length-def*)
then show *?thesis*
using *f1* **by** (*simp add*: *b1 p2*)
qed
then have $(getspc\ (c1!j))\ k = getspc\text{-}es\ ((cs1\ k)\ !\ j)$
using *b0 b2 p2* **by** *auto*
}
then show *?thesis* **by** (*simp add:same-spec-def*)
qed
moreover
have *compat-tran c1 cs1*
proof −
{
fix *j*
assume *b0*: $Suc\ j < length\ c1$
with *p0 p2* **have** $((\exists t\ k.\ (c!(n{+}j)\ {-pes-}(t\sharp k){\rightarrow}\ c!Suc\ (n{+}j))) \wedge$
$(\forall k\ t.\ (c!(n{+}j)\ {-pes-}(t\sharp k){\rightarrow}\ c!Suc\ (n{+}j)) \longrightarrow (cs\ k!(n{+}j)\ {-es-}(t\sharp k){\rightarrow}\ cs\ k!\ Suc\ (n{+}j)) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!(n{+}j)\ {-ese}{\rightarrow}\ cs\ k'!\ Suc\ (n{+}j)))))$
$\vee$
$(((c!(n{+}j))\ {-pese}{\rightarrow}\ (c!Suc\ (n{+}j))) \wedge (\forall k.\ (((cs\ k)!(n{+}j))\ {-ese}{\rightarrow}\ ((cs\ k)!\ Suc\ (n{+}j)))))$
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 ∀ k. cs1 k!j = cs k!(n+j)
          by (metis (no-types, lifting) Suc-lessD add.commute length-drop
              less-diff-conv less-or-eq-imp-le nth-drop)
        moreover
        from p1 p2 p3 a1 b0 have ∀ k. cs1 k!Suc j = cs k!Suc (n+j)
          by (smt add.commute add-Suc-right length-drop less-diff-conv less-or-eq-imp-le nth-drop)
        ultimately
        have ((∃ t k. (c1!j −pes−(t♯k)→ c1!Suc j)) ∧
                 (∀ k t. (c1!j −pes−(t♯k)→ c1!Suc j) ⟶ (cs1 k!j −es−(t♯k)→ cs1 k! Suc j) ∧
                        (∀ k'. k' ≠ k ⟶ (cs1 k'!j −ese→ cs1 k'! Suc j))))
                 ∨
                 (((c1!j) −pese→ (c1!Suc j)) ∧ (∀ k. (((cs1 k)!j) −ese→ ((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 ∈ cpts-pes]] ⟹
      ∀ cs k. c ∝ cs ⟶ (cs k ∈ cpts-es)
  proof −
    assume p0: c ∈ cpts-pes
    {
      fix k
      from p0 have ∀ cs. c ∈ cpts-pes ∧ c ∝ cs ⟶ (cs k ∈ cpts-es)
        proof(induct c)
          case (CptsPesOne pes s x)

          {
            fix cs
            assume a0: [(pes, s, x)] ∝ 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 ∧ 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 ∈ cpts-es by (metis One-nat-def cpts-es-def
                cpts-esp.CptsEsOne length-0-conv length-Suc-conv mem-Collect-eq nth-Cons-0)
          }
          then show ?case by auto
        next
          case (CptsPesEnv pes t x xs s y)
          assume a0: (pes, t, x) # xs ∈ cpts-pes
            and  a1[rule-format]: ∀ cs. (pes, t, x) # xs ∈ cpts-pes ∧ (pes, t, x) # xs ∝ cs ⟶ cs k ∈ cpts-es
          {
            fix cs
            assume b0: (pes, s, y) # (pes, t, x) # xs ∈ cpts-pes
              and  b1: (pes, s, y) # (pes, t, x) # xs ∝ cs
            let ?esl = (pes, t, x) # xs
            let ?esllon = (pes, s, y) # (pes, t, x) # xs
            let ?cs = (λk. drop 1 (cs k))
            from b1 have ?esl ∝ ?cs using drop-n-conjoin[of ?esllon cs 1 ?esl ?cs] by auto
            with a0 a1[of ?cs] have b2: ?cs k ∈ cpts-es by simp
```

from *b1* **have** *b3*: *cs k ! 0 = (pes k, s, y)*
using *conjoin-def[of ?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 ?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 ?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* **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*: *(pes1, s, y) −pes−ct→ (pes2, t, x)*
and *a1*: *(pes2, t, x) # xs ∈ cpts-pes*
and *a2[rule-format]*: *∀ cs. (pes2, t, x) # xs ∈ cpts-pes ∧ (pes2, t, x) # xs ∝ cs ⟶ cs k ∈ cpts-es*
{
fix *cs*
assume *b0*: *(pes1, s, y) # (pes2, t, x) # xs ∈ cpts-pes*
and *b1*: *(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 = (λk. drop 1 (cs k))*
from *b1* **have** *?esl ∝ ?cs* **using** *drop-n-conjoin[of ?esllon cs 1 ?esl ?cs]* **by** *auto*
with *a1 a2[of ?cs]* **have** *b2*: *?cs k ∈ cpts-es* **by** *simp*
from *b1* **have** *b3*: *cs k ! 0 = (pes1 k, s, y)*
using *conjoin-def[of ?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 ?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 ?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 ?esllon cs* **by** *(simp add:conjoin-def)*
then have *((∃ t k. (?esllon!0 −pes−(t♯k)→ ?esllon!Suc 0)) ∧*
(∀ k t. (?esllon!0 −pes−(t♯k)→ ?esllon!Suc 0) ⟶ (cs k!0 −es−(t♯k)→ cs k! Suc 0) ∧
(∀ k′. k′ ≠ k ⟶ (cs k′!0 −ese→ cs k′! Suc 0))))
∨
(((?esllon!0) −pese→ (?esllon!Suc 0)) ∧ (∀ k. (((cs k)!0) −ese→ ((cs k)! Suc 0))))
using *compat-tran-def[of ?esllon cs]* **by** *fastforce*

then have *cs k* ∈ *cpts-es*
proof
assume *c0*: (∃ *t k*. (*?esllon*!*0* −*pes*−(*t*♯*k*)→ *?esllon*!*Suc 0*)) ∧
(∀ *k t*. (*?esllon*!*0* −*pes*−(*t*♯*k*)→ *?esllon*!*Suc 0*) ⟶ (*cs k*!*0* −*es*−(*t*♯*k*)→ *cs k*! *Suc 0*) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (*cs k*′!*0* −*ese*→ *cs k*′! *Suc 0*)))
then obtain *t1* **and** *k1* **where** *c1*: (*?esllon*!*0* −*pes*−(*t1*♯*k1*)→ *?esllon*!*Suc 0*) **by** *auto*
with *c0* **have** *c2*: (*cs k1*!*0* −*es*−(*t1*♯*k1*)→ *cs k1*! *Suc 0*) ∧
(∀ *k*′. *k*′ ≠ *k1* ⟶ (*cs k*′!*0* −*ese*→ *cs k*′! *Suc 0*)) **by** *auto*
show *?thesis*
proof(*cases k* = *k1*)
assume *d0*: *k* = *k1*
with *c2* **have** (*cs k*!*0* −*es*−(*t1*♯*k*)→ *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* ≠ *k1*
with *c2* **have** *cs k*!*0* −*ese*→ *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*: ((*?esllon*!*0*) −*pese*→ (*?esllon*!*Suc 0*)) ∧ (∀ *k*. (((*cs k*)!*0*) −*ese*→ ((*cs k*)! *Suc 0*)))
then have ((*cs k*)!*0*) −*ese*→ ((*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* ∈ *cpts-of-pes pes s x*; *c* ∝ *cs*⟧
⟹ *cs k* ∈ *cpts-of-es* (*pes k*) *s x*
proof −
assume *p0*: *c* ∈ *cpts-of-pes pes s x*
and *p1*: *c* ∝ *cs*
from *p0* **have** *a1*: *c*∈*cpts-pes* ∧ *c*!*0* = (*pes*,*s*,*x*) **by** (*simp add*:*cpts-of-pes-def*)
from *a1 p1* **have** *cs k* ∈ *cpts-es* **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

4.6.3 Semantics is Compositional

lemma *conjoin-cs-imp-cpt*: ⟦∃ *k p*. *pes k* = *p*; (∃ *cs*. (∀ *k*. (*cs k*) ∈ *cpts-of-es* (*pes k*) *s x*) ∧ *c* ∝ *cs*)⟧
⟹ *c* ∈ *cpts-of-pes pes s x*
proof −
assume *p0*: ∃ *cs*. (∀ *k*. (*cs k*) ∈ *cpts-of-es* (*pes k*) *s x*) ∧ *c* ∝ *cs*
and *p1*: ∃ *k p*. *pes k* = *p*
then obtain *cs* **where** (∀ *k*. (*cs k*) ∈ *cpts-of-es* (*pes k*) *s x*) ∧ *c* ∝ *cs* **by** *auto*
then have *a0*: (∀ *k*. (*cs k*)!*0*=(*pes k*,*s*,*x*) ∧ (*cs k*) ∈ *cpts-es*) ∧ *c* ∝ *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) ∧ (cs k) ∈ cpts-es by auto
then have (cs k) ≠ []  by auto
moreover
from a0 have same-length c cs by (simp add:conjoin-def)
ultimately have a3: c ≠ [] 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: ∀ k. (getspc (c!0)) k = getspc-es ((cs k) ! 0) by (simp add:same-spec-def)
    with a0 have ∀ 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-es ((cs k)!0) ∧ getx (c!0) = getx-es ((cs k)!0)
      by (simp add:same-state-def)
    with a2 have gets (c!0) = s ∧ 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 ∀ i. i > 0 ∧ i ≤ length c ⟶ take i c ∈ cpts-pes
  proof -
  {
    fix i
    assume b0: i > 0 ∧ i ≤ length c
    then have take i c ∈ cpts-pes
      proof(induct i)
        case 0 show ?case using 0.prems by auto
      next
        case (Suc j)
        assume c0: 0 < j ∧ j ≤ length c ⟹ take j c ∈ cpts-pes
          and  c1: 0 < Suc j ∧ Suc j ≤ 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 ≠ 0
            from a0 have d1: compat-tran c cs by (simp add:conjoin-def)
            then have d2: ∀ j. Suc j < length c ⟶
                       (∃ t k. (c!j −pes−(t♯k)→ c!Suc j) ∧
                       (∀ k t. (c!j −pes−(t♯k)→ c!Suc j) ⟶ (cs k!j −es−(t♯k)→ cs k! Suc j) ∧
                              (∀ k'. k' ≠ k ⟶ (cs k'!j −ese→ cs k'! Suc j))))
                       ∨
                       (((c!j) −pese→ (c!Suc j)) ∧ (∀ k. (((cs k)!j) −ese→ ((cs k)! Suc j))))
              by (simp add:compat-tran-def)

            from d0 have d3: j − 1 ≥ 0 by simp
            from c1 have d6: Suc (j − 1) < length c using d0 by auto

            with d3 have d4: (∃ t k. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ∧
                       (∀ k t. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ⟶ (cs k!(j−1) −es−(t♯k)→ cs k! Suc (j−1)) ∧
                              (∀ k'. k' ≠ k ⟶ (cs k'!(j−1) −ese→ cs k'! Suc (j−1)))))
                       ∨
                       (((c!(j−1)) −pese→ (c!Suc (j−1))) ∧ (∀ k. (((cs k)!(j−1)) −ese→ ((cs k)!Suc (j−1)))))
              using d2 by auto
```

from *c0 c1 d0* **have** *d5*: *take j c* ∈ *cpts-pes* **by** *auto*
from *d4* **show** *?case*
proof
assume $(\exists\, t\ k.\ (c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1))\ \wedge$
$(\forall\, k\ t.\ (c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1))\longrightarrow(cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (j-1))\ \wedge$
$(\forall\, k'.\ k'\neq k\longrightarrow(cs\ k'!(j-1)\ -ese\rightarrow\ cs\ k'!\ Suc\ (j-1)))))$
then obtain *t* **and** *k* **where** *e0*: $((c!(j-1))\ -pes-(t\sharp k)\rightarrow\ (c!Suc\ (j-1)))$ **by** *auto*
then have $((take\ j\ c)\ !\ (length\ (take\ j\ c)-1))\ -pes-(t\sharp k)\rightarrow\ (c!Suc\ (j-1))$
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** $(take\ j\ c)\ @\ [c!Suc\ (j-1)]\in cpts\text{-}pes$ **using** *cpts-pes-onemore* **by** *blast*
then show *?thesis* **using** *d0 d6 take-Suc-conv-app-nth* **by** *fastforce*
next
assume $((c!(j-1))\ -pese\rightarrow\ (c!Suc\ (j-1)))\wedge(\forall\, k.\ (((cs\ k)!(j-1))\ -ese\rightarrow\ ((cs\ k)!Suc\ (j-1))))$
then have $((take\ j\ c)\ !\ (length\ (take\ j\ c)-1))\ -pese\rightarrow\ (c!Suc\ (j-1))$
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** $(take\ j\ c)\ @\ [c!Suc\ (j-1)]\in cpts\text{-}pes$ **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*∈*cpts-pes* **by** *auto*
from *g0 g1* **show** *?thesis* **by** (*simp add*:*cpts-of-pes-def*)
qed

lemma *comp-tran-env*: $[\![(\forall\, k.\ cs\ k\in cpts\text{-}of\text{-}es\ (pes\ k)\ t1\ x1);\ c=(pes,\ t1,\ x1)\ \#\ xs;\ c\in cpts\text{-}pes;$
$c\propto cs;\ c'=(pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs]\!]\Longrightarrow$
$compat\text{-}tran\ c'\ (\lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k)$
proof −
let $?cs'=\lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k$
assume *p0*: $\forall\, k.\ cs\ k\in cpts\text{-}of\text{-}es\ (pes\ k)\ t1\ x1$
and *p1*: $c\in cpts\text{-}pes$
and *p2*: $c\propto cs$
and *p3*: $c'=(pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs$
and *p4*: $c=(pes,\ t1,\ x1)\ \#\ xs$
from *p0* **have** *b3*: $\forall\, k.\ cs\ k\in cpts\text{-}es\wedge(cs\ k)!0=(pes\ k,t1,x1)$ **by** (*simp add*:*cpts-of-es-def*)
show *compat-tran c′ ?cs′*
proof −
{
fix *j*
assume *dd0*: $Suc\ j<length\ c'$
have $(\exists\, t\ k.\ ((c'!j)\ -pes-(t\sharp k)\rightarrow\ (c'!Suc\ j))\ \wedge$
$(\forall\, k\ t.\ (c'!j\ -pes-(t\sharp k)\rightarrow\ c'!Suc\ j)\longrightarrow(?cs'\ k!j\ -es-(t\sharp k)\rightarrow\ ?cs'\ k!\ Suc\ j)\ \wedge$
$(\forall\, k'.\ k'\neq k\longrightarrow(?cs'\ k'!j\ -ese\rightarrow\ ?cs'\ k'!\ Suc\ j))))$
$\vee$
$(((c'!j)\ -pese\rightarrow\ (c'!Suc\ j))\wedge(\forall\, k.\ (((?cs'\ k)!j)\ -ese\rightarrow\ ((?cs'\ k)!\ Suc\ j))))$
proof(*cases* $j=0$)
assume *d0*: $j=0$
from *p3* **have** $((c'!0)\ -pese\rightarrow\ (c'!1))$
by (*simp add*: *pesetran.intros*)
moreover
have $\forall\, k.\ (((?cs'\ k)!0)\ -ese\rightarrow\ ((?cs'\ k)!1))$

```
          by (simp add: b3 esetran.intros)
        ultimately show ?thesis using d0 by simp
      next
        assume d0: j ≠ 0
        then have d0-1: j > 0 by simp
        from p2 have compat-tran c cs by (simp add:conjoin-def)
        then have d1: ∀ j. Suc j < length c ⟶
                        (∃ t k. (c!j −pes−(t♯k)→ c!Suc j) ∧
                        (∀ k t. (c!j −pes−(t♯k)→ c!Suc j) ⟶ (cs k!j −es−(t♯k)→ cs k! Suc j) ∧
                                (∀ k'. k' ≠ k ⟶ (cs k'!j −ese→ cs k'! Suc j))))
                        ∨
                        (((c!j) −pese→ (c!Suc j)) ∧ (∀ k. (((cs k)!j) −ese→ ((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: (∃ t k. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ∧
                        (∀ k t. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ⟶ (cs k!(j−1) −es−(t♯k)→ cs k! Suc (j−1)) ∧
                                (∀ k'. k' ≠ k ⟶ (cs k'!(j−1) −ese→ cs k'! Suc (j−1)))))
                        ∨
                        (((c!(j−1)) −pese→ (c!Suc (j−1))) ∧ (∀ k. (((cs k)!(j−1)) −ese→ ((cs k)!Suc (j−1)))))
          by auto
        from p3 p4 d0 dd0 have d4: c'!j = c!(j−1) ∧ c'!Suc j = c!Suc (j−1) by simp
        have d5: (∀ k. (?cs' k) ! j= (cs k)! (j−1)) ∧ (∀ 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: ⟦(∀ k. cs k ∈ cpts-of-es (pes2 k) t1 x1); c = (pes2, t1, x1) # xs; c∈cpts-pes;
                    c ∝ cs; c' = (pes1, s1, y1) # (pes2, t1, x1) # xs; (pes1, s1, y1) −pes−ct→ (pes2, t1, x1)⟧
                    ⟹ compat-tran c' (λk. (pes1 k, s1, y1) # cs k)
  proof −
    let ?cs' = λk. (pes1 k, s1, y1) # cs k
    assume p0: ∀ k. cs k ∈ cpts-of-es (pes2 k) t1 x1
      and p1: c ∈ cpts-pes
      and p2: c ∝ cs
      and p3: c' = (pes1, s1, y1) # (pes2, t1, x1) # xs
      and p4: c = (pes2, t1, x1) # xs
      and p5: (pes1, s1, y1) −pes−ct→ (pes2, t1, x1)
    from p0 have b3: ∀ k. cs k ∈ cpts-es ∧ (cs k)!0 = (pes2 k,t1,x1) by (simp add:cpts-of-es-def)
    show compat-tran c' ?cs'
      proof −
      {
        fix j
        assume dd0: Suc j < length c'
        have (∃ t k. ((c'!j) −pes−(t♯k)→ (c'!Suc j)) ∧
                (∀ k t. (c'!j −pes−(t♯k)→ c'!Suc j) ⟶ (?cs' k!j −es−(t♯k)→ ?cs' k! Suc j) ∧
                        (∀ k'. k' ≠ k ⟶ (?cs' k'!j −ese→ ?cs' k'! Suc j))))
                ∨
                (((c'!j) −pese→ (c'!Suc j)) ∧ (∀ k. (((?cs' k)!j) −ese→ ((?cs' k)! Suc j))))
          proof(cases j = 0)
            assume d0: j = 0
            from p5 obtain k and aa where c0: ct = (aa♯k) using get-actk-def by (metis cases)
            with p5 have ∃ es'. ((pes1 k, s1, y1) −es−(aa♯k)→ (es', t1, x1)) ∧ pes2 = pes1(k:=es')
```

```
          using pestran-estran by auto
        then obtain es′ where c1: ((pes1 k, s1, y1) −es−(aa♯k)→ (es′, t1, x1)) ∧ pes2 = pes1(k:=es′)
          by auto
        from b3 have c2: cs k ∈cpts-es ∧ (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: (c′!0) −pes−(aa♯k)→ (c′!Suc 0) by auto
        moreover
        have ∀ k1 t1. (c′!0 −pes−(t1♯k1)→ c′!Suc 0) ⟶ (?cs′ k1!0 −es−(t1♯k1)→ ?cs′ k1! Suc 0) ∧
                         (∀ k′. k′ ≠ k1 ⟶ (?cs′ k′!0 −ese→ ?cs′ k′! Suc 0))
          proof −
          {
            fix k1 t1
            assume d0: c′!0 −pes−(t1♯k1)→ c′!Suc 0
            with p3 have ?cs′ k1!0 −es−(t1♯k1)→ ?cs′ k1! Suc 0
              using b3 fun-upd-apply nth-Cons-0 nth-Cons-Suc pestran-estran by fastforce
            moreover
            from d0 have ∀ k′. k′ ≠ k1 ⟶ (?cs′ k′!0 −ese→ ?cs′ k′! Suc 0)
              using b3 esetran.intros fun-upd-apply nth-Cons-0 nth-Cons-Suc p3 pestran-estran by fastforce
            ultimately have (c′!0 −pes−(t1♯k1)→ c′!Suc 0) ⟶ (?cs′ k1!0 −es−(t1♯k1)→ ?cs′ k1! Suc 0) ∧
                         (∀ k′. k′ ≠ k1 ⟶ (?cs′ k′!0 −ese→ ?cs′ k′! Suc 0)) by simp
          }
          then show ?thesis by auto
          qed
        ultimately show ?thesis using d0 by auto
      next
        assume d0: j ≠ 0
        then have d0-1: j > 0 by simp
        from p2 have compat-tran c cs by (simp add:conjoin-def)
        then have d1: ∀ j. Suc j < length c ⟶
                    (∃ t k. (c!j −pes−(t♯k)→ c!Suc j) ∧
                    (∀ k t. (c!j −pes−(t♯k)→ c!Suc j) ⟶ (cs k!j −es−(t♯k)→ cs k! Suc j) ∧
                            (∀ k′. k′ ≠ k ⟶ (cs k′!j −ese→ cs k′! Suc j))))
                    ∨
                    (((c!j) −pese→ (c!Suc j)) ∧ (∀ k. (((cs k)!j) −ese→ ((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: (∃ t k. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ∧
                    (∀ k t. (c!(j−1) −pes−(t♯k)→ c!Suc (j−1)) ⟶ (cs k!(j−1) −es−(t♯k)→ cs k! Suc (j−1)) ∧
                            (∀ k′. k′ ≠ k ⟶ (cs k′!(j−1) −ese→ cs k′! Suc (j−1)))))
                    ∨
                    (((c!(j−1)) −pese→ (c!Suc (j−1))) ∧ (∀ k. (((cs k)!(j−1)) −ese→ ((cs k)!Suc (j−1)))))
          by blast
        from p3 p4 d0 dd0 have d4: c′!j = c!(j−1) ∧ c′!Suc j = c!Suc (j−1) by simp
        have d5: (∀ k. (?cs′ k) ! j= (cs k)! (j−1)) ∧ (∀ 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:
  ∀ c. c ∈ cpts-pes ⟶
```

```
          (∃ cs. (∀ k. (cs k) ∈ cpts-of-es ((getspc (c!0)) k) (gets (c!0)) (getx (c!0))) ∧ c ∝ cs)
proof −
{
  fix c
  assume p0: c ∈ cpts-pes
  then have ∃ cs. (∀ k. (cs k) ∈ cpts-of-es ((getspc (c!0)) k) (gets (c!0)) (getx (c!0))) ∧ c ∝ cs
    proof(induct c)
      case (CptsPesOne pes1 s1 x1)
      let ?cs = λk. [(pes1 k, s1,x1)]
      let ?c = [(pes1, s1, x1)]
      have ∀ k. ?cs k ∈ cpts-of-es (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 ∈ cpts-es by (simp add: cpts-es.CptsEsOne)
          ultimately have ?cs k ∈ cpts-of-es (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 ?c ∝ ?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 ?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∈ cpts-pes
        and  b1: ∃ cs. (∀ k. cs k ∈ cpts-of-es (getspc (?c ! 0) k) (gets (?c ! 0))
                  (getx (?c ! 0))) ∧ ?c ∝ cs
      then obtain cs where b2: (∀ k. cs k ∈ cpts-of-es (pes1 k) t1 x1) ∧ ?c ∝ cs
        using getspc-def gets-def getx-def by (metis fst-conv nth-Cons-0 snd-conv)
      then have b3: ∀ k. cs k ∈ cpts-es ∧ (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' = λk. (pes1 k,s1,y1)#(cs k)
      have g0: ∀ k. ?cs' k ∈ cpts-of-es (getspc (?c' ! 0) k)  (gets (?c' ! 0)) (getx (?c' ! 0))
        proof −
        {
          fix k
          from b3 have c0: cs k ∈cpts-es ∧ (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 ∈ cpts-es by (simp add: cpts-es.CptsEsEnv)
          then have ?cs' k ∈ cpts-of-es (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: ?c ∝ cs by simp
  from b1 have g1: ?c′ ∝ ?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 c0 conjoin-def
                  less-SucO 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 ?c′ ?cs′
        using comp-tran-env [of cs 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
    and  b1: ∃ cs. (∀ k. cs k ∈ cpts-of-es (getspc (?c ! 0) k) (gets (?c ! 0))
              (getx (?c ! 0))) ∧ ?c ∝ cs
    and  b00: (pes1, s1, y1) −pes−ct→ (pes2, t1, x1)
```

```
then obtain cs where b2: (∀k. cs k ∈ cpts-of-es (pes2 k) t1 x1) ∧ ?c ∝ cs
  using getspc-def gets-def getx-def by (metis fst-conv nth-Cons-0 snd-conv)
then have b3: ∀k. cs k ∈ cpts-es ∧ (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' = λk. (pes1 k,s1,y1)#(cs k)
have g0: ∀k. ?cs' k ∈ cpts-of-es (getspc (?c' ! 0) k)  (gets (?c' ! 0)) (getx (?c' ! 0))
  proof −
  {
    fix k
    obtain ka and aa where c0: ct = (aa♯ka) using get-actk-def by (metis cases)
    with b00 have ∃es'. ((pes1 ka, s1, y1) −es−(aa♯ka)→ (es', t1, x1)) ∧ pes2 = pes1(ka:=es')
      using pestran-estran by auto
    then obtain es' where c1: ((pes1 ka, s1, y1) −es−(aa♯ka)→ (es', t1, x1)) ∧ pes2 = pes1(ka:=es')
      by auto
    from b3 have c2: cs k ∈cpts-es ∧ (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 ∈ cpts-of-es (getspc (?c' ! 0) k)  (gets (?c' ! 0)) (getx (?c' ! 0))
      proof(cases k = ka)
        assume d0: k = ka
        with c1 have (pes1 k, s1, y1) −es−(aa♯k)→ (pes2 k, t1, x1) by auto
        with c2 c3 d0 have ?cs' k ∈ cpts-es
          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 ≠ ka
        with c1 have pes1 k = pes2 k by simp
        with c2 c3 have d1: ?cs' k ∈ cpts-es
          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: ?c ∝ cs by simp
from b1 have g1: ?c' ∝ ?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 ?c' ?cs'
            using comp-tran-pestran [of cs 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 ∈ cpts-of-pes pes s x
            ⟹ ∃ cs. (∀ k. (cs k) ∈ cpts-of-es (pes k) s x) ∧ c ∝ cs
  proof -
    assume p0: c ∈ cpts-of-pes pes s x
    then have c!0=(pes,s,x) ∧ c ∈ cpts-pes 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 pes s x = {c. ∃ cs. (∀ k. (cs k) ∈ cpts-of-es (pes k) s x) ∧ c ∝ cs}
  proof -
    have ∀ c. c∈cpts-of-pes pes s x ⟶ (∃ cs. (∀ k. (cs k) ∈ cpts-of-es (pes k) s x) ∧ c ∝ cs)
      proof -
      {
        fix c
        assume a0: c∈cpts-of-pes pes s x
        then have ∃ cs. (∀ k. (cs k) ∈ cpts-of-es (pes k) s x) ∧ c ∝ 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 $\forall c.\ (\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \wedge c \propto cs) \longrightarrow c{\in}cpts\text{-}of\text{-}pes\ pes\ s\ x$
  proof −
  {
    fix *c*
    assume *a0*: $\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \wedge c \propto cs$
    then have $c{\in}cpts\text{-}of\text{-}pes\ pes\ s\ x$
      using *conjoin-cs-imp-cpt* **by** *fastforce*
  }
  then show *?thesis* **by** *auto*
  qed
ultimately show *?thesis* **by** *auto*
qed

end

5 Validity of Correctness Formulas

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

5.1 Definitions Correctness Formulas

definition *assume-p* :: $('s\ set \times ('s \times 's)\ set) \Rightarrow ('s\ pconfs)\ set$ **where**
$assume\text{-}p \equiv \lambda(pre,\ rely).\ \{c.\ gets\text{-}p\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-pe\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in rely)\}$

definition *commit-p* :: $(('s \times 's)\ set \times 's\ set) \Rightarrow ('s\ pconfs)\ set$ **where**
$commit\text{-}p \equiv \lambda(guar,\ post).\ \{c.\ (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-c\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar) \wedge$
$(getspc\text{-}p\ (last\ c) = None \longrightarrow gets\text{-}p\ (last\ c) \in post)\}$

definition *prog-validity* :: $'s\ prog \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
$(\models\ \text{-}\ sat_p\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$ **where**
$\models P\ sat_p\ [pre,\ rely,\ guar,\ post] \equiv$
$\forall s.\ cpts\text{-}of\text{-}p\ (Some\ P)\ s \cap assume\text{-}p(pre,\ rely) \subseteq commit\text{-}p(guar,\ post)$

definition *assume-e* :: $('s\ set \times ('s \times 's)\ set) \Rightarrow (('l,'k,'s)\ econfs)\ set$ **where**
$assume\text{-}e \equiv \lambda(pre,\ rely).\ \{c.\ gets\text{-}e\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-ee\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}e\ (c!i),\ gets\text{-}e\ (c!Suc\ i)) \in rely)\}$

definition *commit-e* :: $(('s \times 's)\ set \times 's\ set) \Rightarrow (('l,'k,'s)\ econfs)\ set$ **where**
$commit\text{-}e \equiv \lambda(guar,\ post).\ \{c.\ (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$(\exists t.\ c!i\ {-et-t\rightarrow}\ c!(Suc\ i)) \longrightarrow (gets\text{-}e\ (c!i),\ gets\text{-}e\ (c!Suc\ i)) \in guar) \wedge$
$(getspc\text{-}e\ (last\ c) = AnonyEvent\ (None) \longrightarrow gets\text{-}e\ (last\ c) \in post)\}$

definition *evt-validity* :: $('l,'k,'s)\ event \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
$(\models\ \text{-}\ sat_e\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$ **where**
$\models Evt\ sat_e\ [pre,\ rely,\ guar,\ post] \equiv$
$\forall s\ x.\ (cpts\text{-}of\text{-}ev\ Evt\ s\ x) \cap assume\text{-}e(pre,\ rely) \subseteq commit\text{-}e(guar,\ post)$

definition *assume-es* :: $('s\ set \times ('s \times 's)\ set) \Rightarrow (('l,'k,'s)\ esconfs)\ set$ **where**
$assume\text{-}es \equiv \lambda(pre,\ rely).\ \{c.\ gets\text{-}es\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-ese\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}es\ (c!i),\ gets\text{-}es\ (c!Suc\ i)) \in rely)\}$

definition *commit-es* :: $(('s \times 's)\ set \times 's\ set) \Rightarrow (('l,'k,'s)\ esconfs)\ set$ **where**

commit-es ≡ λ(*guar*, *post*). {*c*. (∀ *i*. *Suc i*<*length c* ⟶
(∃ *t*. *c*!*i* −*es*−*t*→ *c*!(*Suc i*)) ⟶ (*gets-es* (*c*!*i*), *gets-es* (*c*!*Suc i*)) ∈ *guar*) }

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

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

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

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

5.2 Lemmas of Correctness Formulas

lemma *assume-es-one-more*:
⟦*esl*∈*cpts-es*; *m* > *0*; *m* < *length esl*; *take m esl*∈*assume-es*(*pre*, *rely*); ¬(*esl*!(*m*−*1*) −*ese*→*esl*!*m*)⟧
⟹ *take* (*Suc m*) *esl* ∈ *assume-es*(*pre*, *rely*)
proof −
assume *p0*: *esl*∈*cpts-es*
and *p1*: *m* > *0*
and *p2*: *m* < *length esl*
and *p3*: *take m esl*∈*assume-es*(*pre*, *rely*)
and *p4*: ¬(*esl*!(*m*−*1*) −*ese*→*esl*!*m*)
let *?esl1* = *take* (*Suc m*) *esl*
let *?esl* = *take m esl*
have *gets-es* (*?esl1*!*0*) ∈ *pre* ∧ (∀ *i*. *Suc i*<*length ?esl1* ⟶
?esl1!*i* −*ese*→ *?esl1*!(*Suc i*) ⟶ (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *rely*)
proof
from *p1 p2 p3* **show** *gets-es* (*?esl1*!*0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
next
show ∀ *i*. *Suc i*<*length ?esl1* ⟶
?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*: *?esl1*!*i* −*ese*→ *?esl1*!(*Suc i*)
have (*gets-es* (*?esl1*!*i*), *gets-es* (*?esl1*!*Suc i*)) ∈ *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** ∀ *i*. *Suc i*<*length ?esl* ⟶
?esl!*i* −*ese*→ *?esl*!(*Suc i*) ⟶
(*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *rely*
by (*simp add*:*assume-es-def*)
with *b0* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *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 ¬(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 ≤ length esl; esl∈assume-es(pre, rely)⟧
      ⟹ take m esl ∈ assume-es(pre, rely)
  proof −
    assume p1: m > 0
      and p2: m ≤ length esl
      and p3: esl∈assume-es(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 ⟶
          ?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: ?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(pre, rely); gets-es (esl!m) ∈ pre1⟧
      ⟹ drop m esl ∈ assume-es(pre1, rely)
  proof −
    assume p1: m < length esl
      and p3: esl∈assume-es(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 ⟶
          ?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: ?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 ≤ length esl; esl∈commit-es(guar, post)⟧
      ⟹ take m esl ∈ commit-es(guar, post)
  proof −
    assume p1: m > 0
      and  p2: m ≤ length esl
      and  p3: esl∈commit-es(guar, post)
    let ?esl1 = take m esl
    have ∀ i. Suc i<length ?esl1 ⟶
          (∃ t. ?esl1!i −es−t→ ?esl1!(Suc i)) ⟶ (gets-es (?esl1!i), gets-es (?esl1!Suc i)) ∈ guar
      proof −
      {
        fix i
        assume a0: Suc i<length ?esl1
          and  a1: (∃ t. ?esl1!i −es−t→ ?esl1!(Suc i))
        with p3 have (gets-es (esl!i), gets-es (esl!Suc i)) ∈ guar by (simp add:commit-es-def)
        with p1 p2 a0 have (gets-es (?esl1!i), gets-es (?esl1!Suc i)) ∈ 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∈commit-es(guar, post)⟧
      ⟹ drop m esl ∈ commit-es(guar, post)
  proof −
    assume p1: m < length esl
      and  p3: esl∈commit-es(guar, post)
    let ?esl1 = drop m esl
    have ∀ i. Suc i<length ?esl1 ⟶
          (∃ t. ?esl1!i −es−t→ ?esl1!(Suc i)) ⟶ (gets-es (?esl1!i), gets-es (?esl1!Suc i)) ∈ guar
      proof −
      {
        fix i
        assume a0: Suc i<length ?esl1
          and  a1: (∃ t. ?esl1!i −es−t→ ?esl1!(Suc i))
        with p3 have (gets-es (esl!(m+i)), gets-es (esl!Suc (m+i))) ∈ guar by (simp add:commit-es-def)
        with p1 a0 have (gets-es (?esl1!i), gets-es (?esl1!Suc i)) ∈ 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-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 ⟶
```

$c!i$ *−pe→* $c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in rely1)$
by (*simp add:assume-p-def*)
show *?thesis*
proof(*simp add:assume-p-def,rule conjI*)
from *p0 a0* **show** $gets\text{-}p\ (c\ !\ 0) \in pre$ **by** *auto*
next
from *p1 a0* **show** $\forall i.\ Suc\ i < length\ c \longrightarrow c\ !\ i\ {-pe\rightarrow}\ c\ !\ Suc\ i$
$\longrightarrow (gets\text{-}p\ (c\ !\ i),\ gets\text{-}p\ (c\ !\ Suc\ i)) \in rely$
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*: $(\forall i.\ Suc\ i < length\ c \longrightarrow$
$c!i\ {-c\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar1) \land$
$(getspc\text{-}p\ (last\ c) = None \longrightarrow gets\text{-}p\ (last\ c) \in post1)$
by (*simp add:commit-p-def*)
show *?thesis*
proof(*simp add:commit-p-def*)
from *p0 p1 a0* **show** $(\forall i.\ Suc\ i < length\ c \longrightarrow$
$c!i\ {-c\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar) \land$
$(getspc\text{-}p\ (last\ c) = None \longrightarrow gets\text{-}p\ (last\ c) \in post)$
by *auto*
qed
qed

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

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

proof(*simp add:commit-es-def*)
  from *p0 a0* **show** $\forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ c\ !\ i\ -es-t\rightarrow\ c\ !\ Suc\ i)$
  $\longrightarrow (gets\text{-}es\ (c\ !\ i),\ gets\text{-}es\ (c\ !\ Suc\ i)) \in guar$
    by *auto*
  qed
qed

lemma *assume-pes-imp*: $[\![pre1\subseteq pre;\ rely1\subseteq rely;\ c\in assume\text{-}pes(pre1,rely1)]\!] \Longrightarrow c\in assume\text{-}pes(pre,rely)$
proof −
  assume *p0*: $pre1\subseteq pre$
    and *p1*: $rely1\subseteq rely$
    and *p3*: $c\in assume\text{-}pes(pre1,rely1)$
  then have *a0*: $gets\ (c!0) \in pre1 \wedge (\forall i.\ Suc\ i<length\ c \longrightarrow$
    $c!i\ -pese\rightarrow\ c!(Suc\ i) \longrightarrow (gets\ (c!i),\ gets\ (c!Suc\ i)) \in rely1)$
  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** $\forall i.\ Suc\ i < length\ c \longrightarrow c\ !\ i\ -pese\rightarrow\ c\ !\ Suc\ i$
    $\longrightarrow (gets\ (c\ !\ i),\ gets\ (c\ !\ Suc\ i)) \in rely$
      by *auto*
  qed
qed

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

lemma *assume-pes-take-n*:
$[\![m > 0;\ m \le length\ esl;\ esl\in assume\text{-}pes(pre,\ rely)]\!]$
  $\Longrightarrow take\ m\ esl \in assume\text{-}pes(pre,\ rely)$
proof −
  assume *p1*: $m > 0$
    and *p2*: $m \le length\ esl$
    and *p3*: $esl\in assume\text{-}pes(pre,\ rely)$
  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 $\forall i.\ Suc\ i<length\ ?esl1 \longrightarrow$
    $?esl1!i\ -pese\rightarrow\ ?esl1!(Suc\ i) \longrightarrow (gets\ (?esl1!i),\ gets\ (?esl1!Suc\ i)) \in rely$
  proof −
  {
    fix *i*
    assume *a0*: $Suc\ i<length\ ?esl1$

and *a1*: *?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*(*pre*, *rely*); *gets* (*esl!m*) ∈ *pre1*⟧
⟹ *drop m esl* ∈ *assume-pes*(*pre1*, *rely*)
proof −
assume *p1*: *m* < *length esl*
and *p3*: *esl*∈*assume-pes*(*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* ⟶
?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*: *?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 — theory Validity

6 The Proof System of PiCore

theory *PiCore-Hoare*
imports *PiCore-Validity*
begin

6.1 Proof System for Programs

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

definition *stable* :: *'a set* ⇒ (*'a* × *'a*) *set* ⇒ *bool* **where**
stable ≡ λ*f g*. (∀ *x y*. *x* ∈ *f* ⟶ (*x*, *y*) ∈ *g* ⟶ *y* ∈ *f*)

inductive *rghoare-p* :: [*'s prog*, *'s set*, (*'s* × *'s*) *set*, (*'s* × *'s*) *set*, *'s set*] ⇒ *bool*
(⊢ - sat_p [-, -, -, -] [*60,0,0,0,0*] *45*)
where
Basic: ⟦ *pre* ⊆ {*s*. *f s* ∈ *post*}; {(*s*,*t*). *s* ∈ *pre* ∧ (*t*=*f s*)} ⊆ *guar*;
stable pre rely; *stable post rely* ⟧
⟹ ⊢ *Basic f* sat_p [*pre*, *rely*, *guar*, *post*]

| *Seq*: ⟦ ⊢ *P* sat_p [*pre*, *rely*, *guar*, *mid*]; ⊢ *Q* sat_p [*mid*, *rely*, *guar*, *post*] ⟧

$\Longrightarrow \vdash$ *Seq P Q* sat_p [*pre*, *rely*, *guar*, *post*]

| *Cond*: ⟦ *stable pre rely*; $\vdash$ *P1* sat_p [*pre* ∩ *b*, *rely*, *guar*, *post*];
$\vdash$ *P2* sat_p [*pre* ∩ −*b*, *rely*, *guar*, *post*]; ∀ *s*. (*s*,*s*)∈*guar* ⟧
$\Longrightarrow \vdash$ *Cond b P1 P2* sat_p [*pre*, *rely*, *guar*, *post*]

| *While*: ⟦ *stable pre rely*; (*pre* ∩ −*b*) ⊆ *post*; *stable post rely*;
$\vdash$ *P* sat_p [*pre* ∩ *b*, *rely*, *guar*, *pre*]; ∀ *s*. (*s*,*s*)∈*guar* ⟧
$\Longrightarrow \vdash$ *While b P* sat_p [*pre*, *rely*, *guar*, *post*]

| *Await*: ⟦ *stable pre rely*; *stable post rely*;
∀ *V*. $\vdash$ *P* sat_p [*pre* ∩ *b* ∩ {*V*}, {(*s*, *t*). *s* = *t*},
UNIV, {*s*. (*V*, *s*) ∈ *guar*} ∩ *post*] ⟧
$\Longrightarrow \vdash$ *Await b P* sat_p [*pre*, *rely*, *guar*, *post*]

| *Nondt*: ⟦ *pre* ⊆ {*s*. (∀ *t*. (*s*,*t*) ∈ *r* ⟶ *t* ∈ *post*) ∧ (∃ *t*. (*s*,*t*) ∈ *r*)}; {(*s*,*t*). *s* ∈ *pre* ∧ (*s*,*t*)∈*r*} ⊆ *guar*;
stable pre rely; *stable post rely* ⟧
$\Longrightarrow \vdash$ *Nondt r* sat_p [*pre*, *rely*, *guar*, *post*]

| *Conseq*: ⟦ *pre* ⊆ *pre'*; *rely* ⊆ *rely'*; *guar'* ⊆ *guar*; *post'* ⊆ *post*;
$\vdash$ *P* sat_p [*pre'*, *rely'*, *guar'*, *post'*] ⟧
$\Longrightarrow \vdash$ *P* sat_p [*pre*, *rely*, *guar*, *post*]

| *Unprecond*: ⟦ $\vdash$ *P* sat_p [*pre*, *rely*, *guar*, *post*]; $\vdash$ *P* sat_p [*pre'*, *rely*, *guar*, *post*] ⟧
$\Longrightarrow \vdash$ *P* sat_p [*pre* ∪ *pre'*, *rely*, *guar*, *post*]

| *Intpostcond*: ⟦ $\vdash$ *P* sat_p [*pre*, *rely*, *guar*, *post*]; $\vdash$ *P* sat_p [*pre*, *rely*, *guar*, *post'*] ⟧
$\Longrightarrow \vdash$ *P* sat_p [*pre*, *rely*, *guar*, *post* ∩ *post'*]

| *Allprecond*: ∀ *v*∈*U*. $\vdash$ *P* sat_p [{*v*}, *rely*, *guar*, *post*]
$\Longrightarrow \vdash$ *P* sat_p [*U*, *rely*, *guar*, *post*]

| *Emptyprecond*: $\vdash$ *P* sat_p [{}, *rely*, *guar*, *post*]

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

lemma *Seq2*: ⟦ $\vdash$ *P* sat_p [*pre*, *rely*, *guar*, *mida*]; *mida* ⊆ *midb*; $\vdash$ *Q* sat_p [*midb*, *rely*, *guar*, *post*] ⟧
$\Longrightarrow \vdash$ *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 Q*]
by *blast*

6.2 Rely-guarantee Condition

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

definition *getrgformula* ::
's set ⇒ (*'s* × *'s*) *set* ⇒ (*'s* × *'s*) *set* ⇒ *'s set* ⇒ *'s rgformula* (*RG*[-,-,-,-] [*91*,*91*,*91*,*91*] *90*)
where *getrgformula pre r g pst* ≡ ⦇*pre-rgf* = *pre*, *rely-rgf* = *r*, *guar-rgf* = *g*, *post-rgf* = *pst*⦈

definition Pre_f :: *'s rgformula* ⇒ *'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 $({'l},{'k},{'s})\ rgformula\text{-}e = ({'l},{'k},{'s})\ event \times {'s}\ rgformula$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.3 Proof System for Events

inductive *rghoare-e* :: $[('l,'k,'s)\ event,\ 's\ set,\ ('s \times 's)\ set,\ ('s \times 's)\ set,\ 's\ set] \Rightarrow bool$
$(\vdash\ \text{-}\ sat_e\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$
where
$AnonyEvt$: $\vdash P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow \vdash AnonyEvent\ (Some\ P)\ sat_e\ [pre,\ rely,\ guar,\ post]$

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

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

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

6.4 Proof System for Event Systems

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

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

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

6.5 Proof System for Parallel Event Systems

inductive *rghoare-pes* :: $[('l,'k,'s)\ rgformula\text{-}par,\ 's\ set,\ ('s \times 's)\ set,\ ('s \times 's)\ set,\ 's\ set] \Rightarrow bool$
$(\vdash\ \text{-}\ SAT\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,0,0,0,0]\ 45)$
where
$ParallelESys$: $[\![\forall k.\ \vdash fst\ (pesf\ k)\ sat_s\ [Pre_{es}\ (pesf\ k),\ Rely_{es}\ (pesf\ k),\ Guar_{es}\ (pesf\ k),\ Post_{es}\ (pesf\ k)];$
$\forall k.\ pre \subseteq Pre_{es}\ (pesf\ k);$
$\forall k.\ rely \subseteq Rely_{es}\ (pesf\ k);$
$\forall k\ j.\ j \neq k \longrightarrow\ Guar_{es}\ (pesf\ j) \subseteq Rely_{es}\ (pesf\ k);$
$\forall k.\ Guar_{es}\ (pesf\ k) \subseteq guar;$
$\forall k.\ Post_{es}\ (pesf\ k) \subseteq post]\!]$
$\Longrightarrow \vdash pesf\ SAT\ [pre,\ rely,\ guar,\ post]$

$|\ ParallelESys\text{-}conseq$: $[\![\ pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\vdash pesf\ SAT\ [pre',\ rely',\ guar',\ post']\ ]\!]$
$\Longrightarrow \vdash pesf\ SAT\ [pre,\ rely,\ guar,\ post]$

7 Soundness

7.1 Some previous lemmas

7.1.1 program

lemma *tl-of-assum-in-assum*:
  $(P,\ s)\ \#\ (P,\ t)\ \#\ xs \in$ *assume-p* $(pre,\ rely) \Longrightarrow$ *stable pre rely*
  $\Longrightarrow (P,\ t)\ \#\ xs \in$ *assume-p* $(pre,\ rely)$
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*:
  $(P,\ t)\ \#\ xs \in$ *commit-p*$(guar,\ post) \Longrightarrow (P,\ s)\ \#\ (P,\ t)\ \#\ xs \in$ *commit-p*$(guar,\ post)$
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*:
  $[\![(s,\ s) \in guar;\ (Q,\ s)\ \#\ xs \in$ *commit-p*$(guar,\ post)]\!]$
  $\Longrightarrow (P,\ s)\ \#\ (Q,\ s)\ \#\ xs \in$ *commit-p*$(guar,\ post)$
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*!]:
  $\forall j.\ c \in$ *cpts-p* $\longrightarrow$ *take* $(Suc\ j)\ c \in$ *cpts-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*: $[\![x \# xs \in cpts\text{-}p;\ xs \neq []]\!] \Longrightarrow xs \in cpts\text{-}p$
apply(*subgoal-tac 1* $<$ *length* (*x* # *xs*))
 apply(*drule dropcptn-is-cptn*,*simp*+)
done

lemma *not-ctran-None* [*rule-format*]:
 $\forall s.\ (None,\ s)\#xs \in cpts\text{-}p \longrightarrow (\forall i{<}length\ xs.\ ((None,\ s)\#xs)!i\ {-pe\rightarrow}\ xs!i)$
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*]:$[] \notin cpts\text{-}p$
apply(*force elim*:*cpts-p.cases*)
done

lemma *etran-or-ctran* [*rule-format*]:
 $\forall m\ i.\ x{\in}cpts\text{-}p \longrightarrow m \leq length\ x$
  $\longrightarrow (\forall i.\ Suc\ i < m \longrightarrow \neg\ x!i\ {-c\rightarrow}\ x!Suc\ i) \longrightarrow Suc\ i < m$
  $\longrightarrow x!i\ {-pe\rightarrow}\ x!Suc\ i$
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* $(\forall i.\ Suc\ i < nata \longrightarrow (((P,\ t)\ \#\ xs)\ !\ i,\ xs\ !\ i) \notin ptran)$)
  apply *force*
 apply *clarify*
 apply(*erule-tac x*=*Suc ia* **in** *allE*,*simp*)
apply(*erule-tac x*=*0* **and** $P{=}\lambda j.\ H\ j \longrightarrow (J\ j) \notin ptran$ **for** *H J* **in** *allE*,*simp*)
done

lemma *etran-or-ctran2* [*rule-format*]:
 $\forall i.\ Suc\ i{<}length\ x \longrightarrow x{\in}cpts\text{-}p \longrightarrow (x!i\ {-c\rightarrow}\ x!Suc\ i \longrightarrow \neg\ x!i\ {-pe\rightarrow}\ x!Suc\ i)$
 $\vee\ (x!i\ {-pe\rightarrow}\ x!Suc\ i \longrightarrow \neg\ x!i\ {-c\rightarrow}\ x!Suc\ i)$
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*:
 $[\![\ x{\in}cpts\text{-}p;\ Suc\ i{<}length\ x;\ x!i\ {-c\rightarrow}\ x!Suc\ i]\!] \Longrightarrow \neg\ x!i\ {-pe\rightarrow}\ x!Suc\ i$

by(*drule etran-or-ctran2*,*simp-all*)

lemma *etran-or-ctran2-disjI2*:
 ⟦ $x \in cpts\text{-}p$; *Suc i*<*length x*; $x!i$ −*pe*→ $x!Suc\ i$⟧ ⟹ ¬ $x!i$ −*c*→ $x!Suc\ i$
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* −*c*→ *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*) ⟶ (∃ *m*. *P m* ∧ (∀ *i*<*m*. ¬ *P i*))
apply(*rule nat-less-induct*)
apply *clarify*
apply(*case-tac* ∀ *m*. *m*<*n* ⟶ ¬ *P m*)
apply *auto*
done

lemma *stability* [*rule-format*]:
 ∀ *j k*. *x* ∈ *cpts-p* ⟶ *stable p rely* ⟶ *j*≤*k* ⟶ *k*<*length x* ⟶ *snd*(*x*!*j*)∈*p* ⟶
 (∀ *i*. (*Suc i*)<*length x* ⟶
  (*x*!*i* −*pe*→ *x*!(*Suc i*)) ⟶ (*snd*(*x*!*i*), *snd*(*x*!(*Suc i*))) ∈ *rely*) ⟶
 (∀ *i*. *j*≤*i* ∧ *i*<*k* ⟶ *x*!*i* −*pe*→ *x*!*Suc i*) ⟶ *snd*(*x*!*k*)∈*p* ∧ *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*=λ*j*. (*0*≤*j*) ⟶ (*J j*) **for** *J* **in** *allE*,*simp*)
  apply(*subgoal-tac t*∈*p*)
   apply(*subgoal-tac* (∀ *i*. *i* < *length xs* ⟶ ((*P*, *t*) # *xs*) ! *i* −*pe*→ *xs* ! *i* ⟶ (*snd* (((*P*, *t*) # *xs*) ! *i*), *snd* (*xs* ! *i*)) ∈ *rely*))
    apply *clarify*
    apply(*erule-tac x*=*Suc i* **and** *P*=λ*j*. (*H j*) ⟶ (*J j*)∈*petran* **for** *H J* **in** *allE*,*simp*)
   apply *clarify*
   apply(*erule-tac x*=*Suc i* **and** *P*=λ*j*. (*H j*) ⟶ (*J j*) ⟶ (*T j*)∈*rely* **for** *H J T* **in** *allE*,*simp*)
  apply(*erule-tac x*=*0* **and** *P*=λ*j*. (*H j*) ⟶ (*J j*)∈*petran* ⟶ *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*=λ*j*. (*s*≤*j*) ⟶ (*J j*) **for** *s J* **in** *allE*,*simp*)
 apply(*subgoal-tac* (∀ *i*. *i* < *length xs* ⟶ ((*P*, *t*) # *xs*) ! *i* −*pe*→ *xs* ! *i* ⟶ (*snd* (((*P*, *t*) # *xs*) ! *i*), *snd* (*xs* ! *i*)) ∈ *rely*))

```
  apply clarify
  apply(erule-tac x=Suc i and P=λj. (H j) ⟶ (J j)∈petran for H J in allE,simp)
 apply clarify
 apply(erule-tac x=Suc i and P=λj. (H j) ⟶ (J j) ⟶ (T j)∈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=λj. (H j) ⟶ (J j)∈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=λj. (s≤j) ⟶ (J j) for s J in allE,simp)
apply(subgoal-tac (∀ i. i < length xs ⟶ ((Q, t) # xs) ! i −pe→ xs ! i ⟶ (snd (((Q, t) # xs) ! i), snd (xs ! i)) ∈
rely))
 apply clarify
 apply(erule-tac x=Suc i and P=λj. (H j) ⟶ (J j)∈petran for H J in allE,simp)
apply clarify
apply(erule-tac x=Suc i and P=λj. (H j) ⟶ (J j) ⟶ (T j)∈rely for H J T in allE,simp)
done
```

7.1.2 event

```
lemma assume-e-imp: ⟦pre1⊆pre; rely1⊆rely; c∈assume-e(pre1,rely1)⟧ ⟹ c∈assume-e(pre,rely)
  proof −
    assume p0: pre1⊆pre
      and  p1: rely1⊆rely
      and  p3: c∈assume-e(pre1,rely1)
    then have a0: gets-e (c!0) ∈ pre1 ∧ (∀ i. Suc i<length c ⟶
              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 ⟶ 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(guar1,post1)⟧ ⟹ c∈commit-e(guar,post)
  proof −
    assume p0: guar1⊆guar
      and  p1: post1⊆post
      and  p3: c∈commit-e(guar1,post1)
    then have a0: (∀ i. Suc i<length c ⟶
              (∃ t. c!i −et−t→ c!(Suc i)) ⟶ (gets-e (c!i), gets-e (c!Suc i)) ∈ guar1) ∧
              (getspc-e (last c) = AnonyEvent (None) ⟶ 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. c ! i −et−t→ c ! Suc i)
                       ⟶ (gets-e (c ! i), gets-e (c ! Suc i)) ∈ guar) ∧
              (getspc-e (last c) = AnonyEvent (None) ⟶ gets-e (last c) ∈ post)
          by auto
      qed
  qed
```

7.1.3 event system

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

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

lemma *concat-i-lm*[*rule-format*]: $\forall ls\ l.\ concat\ ls = l \wedge (\forall i<length\ ls.\ ls!i \neq [])\longrightarrow (\forall i.\ Suc\ i < length\ ls \longrightarrow$
    $(\exists m\ n.\ m \leq length\ l \wedge n \leq length\ l \wedge m \leq n \wedge ls!i@[(ls!Suc\ i)!0] = take\ (n - m)\ (drop\ m\ l)))$
  proof –
  {
    fix *ls*
    have $\forall l.\ concat\ ls = l \wedge (\forall i<length\ ls.\ ls!i \neq [])\longrightarrow (\forall i.\ Suc\ i < length\ ls \longrightarrow$
        $(\exists m\ n.\ m \leq length\ l \wedge n \leq length\ l \wedge m \leq n \wedge ls!i@[(ls!Suc\ i)!0] = take\ (n - m)\ (drop\ m\ l)))$
    proof(*induct ls*)
      case *Nil* **show** *?case* **by** *simp*
    next
      case (*Cons x xs*)
      assume *a0*: $\forall l.\ concat\ xs = l \wedge (\forall i<length\ xs.\ xs\ !\ i \neq []) \longrightarrow$
          $(\forall i.\ Suc\ i < length\ xs \longrightarrow (\exists m\ n.\ m \leq length\ l \wedge n \leq length\ l \wedge$
            $m \leq n \wedge xs\ !\ i\ @\ [xs\ !\ Suc\ i\ !\ 0] = take\ (n - m)\ (drop\ m\ l)))$
      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 ∀ i. Suc i < length (x # xs) ⟶ (∃ m n. m ≤ length l ∧ n ≤ length l ∧
                m ≤ n ∧ (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 ∃ m n. m ≤ length l ∧ n ≤ length l ∧ m ≤ n ∧
                  (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 ≠ [] 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 ≤ length l using calculation by auto
                moreover
                from b0 d1 have length x + 1 ≤ 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 ≠ 0
                moreover
                from b1 have d1: ∀ i<length xs. xs ! i ≠ [] by auto
                moreover
                from c0 have Suc (i − 1) < length xs using d0 by auto
                ultimately have ∃ m n. m ≤ length ?l′ ∧ n ≤ length ?l′ ∧
                        m ≤ n ∧ 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 ≤ length ?l′ ∧ n ≤ length ?l′ ∧
                        m ≤ n ∧ 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′ ≤ length l by auto
                moreover
                from b0 d2 have ?n′ ≤ length l by auto
                moreover
                from d2 have ?m′ ≤ ?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′ ≤ length l ∧ ?n′ ≤ length l ∧ ?m′ ≤ ?n′ ∧
                      (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: ∀ ls l. concat ls = l ∧ length ls > 0 ⟶
                      (∃ m . m ≤ length l ∧ last ls = drop m l)
  proof
    fix ls
    show ∀ l. concat ls = l ∧ length ls > 0 ⟶
                      (∃ m . m ≤ length l ∧ last ls = drop m l)
      proof(induct ls)
        case Nil show ?case by simp
      next
        case (Cons x xs)
        assume a0: ∀ l. concat xs = l ∧ 0 < length xs ⟶ (∃ m≤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 ∃ m≤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*: $m \leq$ *length l* $\Longrightarrow \exists k\, j.\ k <$ *length lt* $\wedge\ j \leq$ *length* (*lt* ! *k*) $\wedge$
  drop m l = drop j (*lt* ! *k*) @ *concat* (*drop* (*Suc k*) *lt*)
  and *b1*: *Suc m* $\leq$ *length l*
then have $\exists k\, j.\ k <$ *length lt* $\wedge\ j \leq$ *length* (*lt* ! *k*) $\wedge$
  drop m l = drop j (*lt* ! *k*) @ *concat* (*drop* (*Suc k*) *lt*)
  by *auto*
then obtain *k* **and** *j* **where** *b2*: $k <$ *length lt* $\wedge\ j \leq$ *length* (*lt* ! *k*) $\wedge$
  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* $\neq$ [] **by** *simp*
    with *c1* **have** *c2*: *drop* (*Suc k*) *lt* $\neq$ [] **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* $\neq$ *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*: $\forall i.$ *Suc i<length l* $\longrightarrow$ *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 = take m l* $\longrightarrow$ ($\forall i.$ *Suc i<length subl* $\longrightarrow$ *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* $\longrightarrow$ *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 = take m l* $\longrightarrow$ ($\forall i.$ *Suc i<length subl* $\longrightarrow$ *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*: ∀ *subl. n* ≤ *length l* ∧ *subl* = *take n l* ⟶
(∀ *i. Suc i* < *length subl* ⟶ *subl* ! *i* −*ese*→ *subl* ! *Suc i* ⟶
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) ∈ *rely*)
show *?case*
proof −
{
fix *subl*
assume *b0*: *Suc n* ≤ *length l*
and *b1*: *subl* = *take* (*Suc n*) *l*
with *a0* **have** ∀ *i. Suc i* < *length subl* ⟶ *subl* ! *i* −*ese*→ *subl* ! *Suc i* ⟶
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) ∈ *rely*
using *p0* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *rely-drop-rely*: ∀ *i. Suc i*<*length l* ⟶ *l*!*i* −*ese*→ *l*!(*Suc i*)
⟶ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) ∈ *rely* ⟹
∀ *m subl. m* ≤ *length l* ∧ *subl* = *drop m l* ⟶ (∀ *i. Suc i*<*length subl* ⟶ *subl*!*i* −*ese*→ *subl*!(*Suc i*)
⟶ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) ∈ *rely*)
proof −
assume *p0*: ∀ *i. Suc i*<*length l* ⟶ *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* = *drop m l* ⟶ (∀ *i. Suc i*<*length subl* ⟶ *subl*!*i* −*ese*→ *subl*!(*Suc i*)
⟶ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) ∈ *rely*)
proof(*induct m*)
case *0* **show** *?case* **by** (*simp add*: *p0*)
next
case (*Suc n*)
assume *a0*: ∀ *subl. n* ≤ *length l* ∧ *subl* = *drop n l* ⟶
(∀ *i. Suc i* < *length subl* ⟶ *subl* ! *i* −*ese*→ *subl* ! *Suc i* ⟶
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) ∈ *rely*)
show *?case*
proof −
{
fix *subl*
assume *b0*: *Suc n* ≤ *length l*
and *b1*: *subl* = *drop* (*Suc n*) *l*
with *a0* **have** ∀ *i. Suc i* < *length subl* ⟶ *subl* ! *i* −*ese*→ *subl* ! *Suc i* ⟶
(*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) ∈ *rely*
using *p0* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*

qed
qed

lemma *rely-takedrop-rely*: ⟦∀ *i*. *Suc i*<*length l* ⟶ *l*!*i* −*ese*→ *l*!(*Suc i*)
⟶ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) ∈ *rely*;
∃ *m n*. *m* ≤ *length l* ∧ *n* ≤ *length l* ∧ *m* ≤ *n* ∧ *subl* = *take* (*n* − *m*) (*drop m l*)⟧ ⟹
∀ *i*. *Suc i*<*length subl* ⟶ *subl*!*i* −*ese*→ *subl*!(*Suc i*)
⟶ (*gets-es* (*subl*!*i*), *gets-es* (*subl*!*Suc i*)) ∈ *rely*
proof −
assume *p1*: ∀ *i*. *Suc i*<*length l* ⟶ *l*!*i* −*ese*→ *l*!(*Suc i*)
⟶ (*gets-es* (*l*!*i*), *gets-es* (*l*!*Suc i*)) ∈ *rely*
and *p3*: ∃ *m n*. *m* ≤ *length l* ∧ *n* ≤ *length l* ∧ *m* ≤ *n* ∧ *subl* = *take* (*n* − *m*) (*drop m l*)

from *p3* **obtain** *m* **and** *n* **where** *a0*: *m* ≤ *length l* ∧ *n* ≤ *length l* ∧ *m* ≤ *n* ∧ *subl* = *take* (*n* − *m*) (*drop m l*)
by *auto*
let *?subl1* = *drop m l*
have *a1*: ∀ *i*. *Suc i*<*length ?subl1* ⟶ *?subl1*!*i* −*ese*→ *?subl1*!(*Suc i*)
⟶ (*gets-es* (*?subl1*!*i*), *gets-es* (*?subl1*!*Suc i*)) ∈ *rely*
using *a0 p1 rely-drop-rely* **by** *blast*
show *?thesis* **by** (*simp add*: *a1 a0*)
qed

lemma *pre-trans*: ⟦*esl* ∈ *assume-es*(*pre*, *rely*); ∀ *i*<*length esl*. *getspc-es* (*esl*!*i*) = *es*; *stable pre rely*⟧
⟹ ∀ *i*<*length esl*. *gets-es* (*esl*!*i*) ∈ *pre*
proof −
assume *p0*: *esl* ∈ *assume-es*(*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 *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*(*pre*, *rely*); *n* < *length esl*;
∀ *j*. *j* ≤ *n* ⟶ *getspc-es* (*esl* ! *j*) = *es*; *stable pre rely*⟧

```
    ⟹ drop n esl ∈ assume-es(pre, rely)
  proof −
    assume p0: esl ∈ assume-es(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(pre, rely)
          using assume-es-take-n[of Suc n esl 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 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 ⟶
              ?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: ?esl!i −ese→ ?esl!(Suc i)
            from p0 have ∀ i. Suc i<length esl ⟶
                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
```

7.1.4 parallel event system

7.2 State trace equivalence

7.2.1 trace equivalence of program and anonymous event

```
definition lift-progs :: ('s pconfs) ⇒ ('l,'k,'s) x ⇒ ('l,'k,'s) econfs
  where lift-progs pcfs x ≡ map (λc. (AnonyEvent (fst c), snd c, x)) pcfs


lemma equiv-prog-lift0 : p∈cpts-p ⟹ lift-progs p x ∈ cpts-of-ev (AnonyEvent (getspc-p (p!0))) (gets-p (p!0)) x
  proof−
    assume a0: p∈cpts-p
    have ∀ p s x. p∈cpts-p ⟶ lift-progs p x ∈ cpts-of-ev (AnonyEvent (getspc-p (p!0))) (gets-p (p!0)) x
      proof −
      {
```

```
      fix p s x
      assume b0: p∈cpts-p
      then have lift-progs p x ∈ cpts-of-ev (AnonyEvent (getspc-p (p!0))) (gets-p (p!0)) x
        proof(induct p)
          case (CptsPOne P' s')
          have c0:lift-progs [(P', s')] x ! 0 = ((AnonyEvent (getspc-p ([(P', s')]!0))), (gets-p ([(P', s')]!0)), x)
            by (simp add: lift-progs-def getspc-p-def gets-p-def)
          have c1:lift-progs [(P', s')] x ∈ cpts-ev
            by (simp add: cpts-ev.CptsEvOne lift-progs-def)
          with c0 show ?case by (simp add: cpts-of-ev-def)
        next
          case (CptsPEnv P' t' xs' s')
          assume c0: (P', t') # xs' ∈ cpts-p and
                 c1: lift-progs ((P', t') # xs') x ∈ cpts-of-ev (AnonyEvent (getspc-p (((P', t') # xs') ! 0))) (gets-p (((P',
t') # xs') ! 0)) x
          have c2: lift-progs ((P', s') # (P', t') # xs') x ! 0 =
              ((AnonyEvent (getspc-p (((P', s') # (P', t') # xs') ! 0))), (gets-p (((P', s') # (P', t') # xs') ! 0)), x)
              by (simp add: lift-progs-def getspc-p-def gets-p-def)
          have c3: lift-progs ((P', s') # (P', t') # xs') x = (AnonyEvent P', s', x) # lift-progs ((P', t') # xs') x
            by (simp add: lift-progs-def)
          from c1 have c5: lift-progs ((P', t') # xs') x ∈ cpts-ev
            by (simp add: cpts-of-ev-def)
          with c3 have c4: lift-progs ((P', s') # (P', t') # xs') x ∈ cpts-ev
            by (simp add: cpts-ev.CptsEvEnv lift-progs-def)
          with c2 show ?case using cpts-of-ev-def by fastforce
        next
          case (CptsPComp P' s' Q' t' xs')
          assume c0: (P', s') −c→ (Q', t') and
                 c1: (Q', t') # xs' ∈ cpts-p and
                 c2: lift-progs ((Q', t') # xs') x ∈ cpts-of-ev (AnonyEvent (getspc-p (((Q', t') # xs') ! 0))) (gets-p (((Q',
t') # xs') ! 0)) x
          have c3: lift-progs ((P', s') # (Q', t') # xs') x ! 0 =
                 ((AnonyEvent (getspc-p (((P', s') # (Q', t') # xs') ! 0))), (gets-p (((P', s') # (Q', t') # xs') ! 0)), x)
             by (simp add: lift-progs-def getspc-p-def gets-p-def)
          have c4: lift-progs ((P', s') # (Q', t') # xs') x = (AnonyEvent P', s', x) # lift-progs ((Q', t') # xs') x
            by (simp add: lift-progs-def)
          from c2 have c5: lift-progs ((Q', t') # xs') x ∈ cpts-ev
            by (simp add: cpts-of-ev-def)
          from c0 have c6: (AnonyEvent P', s', x) −et−(Cmd CMP)♯k→ (AnonyEvent Q', t', x)
            by (simp add: etran.AnonyEvent)
          with c6 c5 c4 have c7: lift-progs ((P', s') # (Q', t') # xs') x ∈ cpts-ev
            by (simp add: cpts-ev.CptsEvComp lift-progs-def)

          with c3 show ?case using cpts-of-ev-def by fastforce
        qed
    }
    then show ?thesis by auto
    qed

  with a0 show ?thesis by auto
qed


lemma equiv-prog-lift : p∈cpts-of-p P s ⟹ lift-progs p x ∈ cpts-of-ev (AnonyEvent P) s x
proof −
  assume a0: p∈cpts-of-p P s
  then have a1: p∈cpts-p by (simp add: cpts-of-p-def)
  from a0 have a2: p!0=(P,s) by (simp add: cpts-of-p-def)
```

```
    with a1 show ?thesis using equiv-prog-lift0 getspc-p-def gets-p-def
      by (metis fst-conv snd-conv)
  qed

primrec lower-anonyevt0 :: ('l,'k,'s) event ⇒ 's ⇒ 's pconf
  where AnonyEv: lower-anonyevt0 (AnonyEvent p) s = (p, s) |
        BasicEv: lower-anonyevt0 (BasicEvent p) s = (None, s)

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

definition lower-evts :: ('l,'k,'s) econfs ⇒ ('s 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 equiv-lower-evts0 : ⟦∃P. getspc-e (es ! 0) = AnonyEvent P; es ∈ cpts-ev⟧ ⟹ lower-evts es ∈cpts-p
proof−
    assume a0: es ∈ cpts-ev and a1: ∃P. getspc-e (es ! 0) = AnonyEvent P
    have ∀es P. getspc-e (es ! 0) = AnonyEvent P ∧ es ∈ cpts-ev ⟶ lower-evts es ∈cpts-p
      proof −
      {
        fix es
        assume b0: ∃P. getspc-e (es ! 0) = AnonyEvent P and
               b1: es ∈ cpts-ev
        from b1 b0 have lower-evts es ∈cpts-p
          proof(induct es)
            case (CptsEvOne e' s' x')
            assume c0: ∃P. getspc-e ([(e', s', x')] ! 0) = AnonyEvent P
            then obtain P where getspc-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-anonyevt1 (e', s', x') = (P, s')
              by (simp add: gets-e-def getspc-e-def lower-anonyevt1-def)
            then have c2: lower-evts [(e', s', x')] = [(P, s')]
              by (simp add: lower-evts-def)
            then show ?case by (simp add: cpts-of-p-def cpts-p.CptsPOne)
          next
            case (CptsEvEnv e' t' x' xs' s' y')
            assume c0:  (e', t', x') # xs' ∈ cpts-ev and
                   c1: ∃P. getspc-e (((e', t', x') # xs') ! 0) = AnonyEvent P ⟹ lower-evts ((e', t', x') # xs') ∈ cpts-p
and
                   c2: ∃P. getspc-e (((e', s', y') # (e', t', x') # xs') ! 0) = AnonyEvent P
            let ?ob = lower-evts ((e', s', y') # (e', t', x') # xs')
            from c2 obtain P where c-:getspc-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-evts ((e', t', x') # xs') ∈ cpts-p by (simp add:getspc-e-def)
            from c5 have c7: ?ob = (P, s') # lower-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-evts ((e', t', x') # xs') = (P, t') # lower-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: cpts-p.CptsPEnv)
          next
            case (CptsEvComp e1 s1 x1 et e2 t1 y1 xs1)
```

```
          assume c0: (e1, s1, x1) −et−et→ (e2, t1, y1) and
                 c1: (e2, t1, y1) # xs1 ∈ cpts-ev and
                 c2: ∃ P. getspc-e (((e2, t1, y1) # xs1) ! 0) = AnonyEvent P
                      ⟹ lower-evts ((e2, t1, y1) # xs1) ∈ cpts-p and
                 c3: ∃ 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 ∃ 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) ∈ cpts-p 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 ∈ cpts-p 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: (AnonyEvent P, s1, x1) −et−et→ (AnonyEvent Q, t1, y1) by simp
          then have (P, s1) −c→ (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 ∈ cpts-p
            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-evts : es ∈ cpts-of-ev (AnonyEvent P) s x ⟹ lower-evts es ∈cpts-of-p P s
  proof −
    assume a0: es ∈ cpts-of-ev (AnonyEvent P) s x
    then have a1: es!0=(AnonyEvent P,(s,x)) ∧ es ∈ cpts-ev 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 ∈cpts-p 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 add:cpts-of-p-def)
  qed
```

7.2.2 trace between of basic and anonymous events

```
lemma evtent-in-cpts1: el ∈ cpts-ev ∧ el ! 0 = (BasicEvent ev, s, x) ⟹
      Suc i < length el ∧ el ! i −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc i) ⟹
```

```
    (∀j. Suc j ≤ i ⟶ getspc-e (el ! j) = BasicEvent ev ∧ el ! j −ee→ el ! (Suc j))
  proof −
    assume p0: el ∈ cpts-ev ∧ el ! 0 = (BasicEvent ev, s, x)
    assume p1: Suc i < length el ∧ el ! i −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc i)
    from p0 have p01: el ∈ cpts-ev 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 ∀j. Suc j ≤ i ⟶ getspc-e (el ! j) = BasicEvent ev ∧ el ! j −ee→ el ! (Suc j)
      proof −
      {
        fix j
        assume a0: Suc j ≤ i
        have ∀k. k < i ⟶ getspc-e (el ! (i −k −1)) = BasicEvent ev ∧ el ! (i −k −1)−ee→ el ! (i − k)
          proof −
          {
            fix k
            assume k < i
            then have getspc-e (el ! (i −k −1)) = BasicEvent ev ∧ el ! (i −k −1)−ee→ el ! (i − k)
              proof(induct k)
                case 0
                from p3 have b0: ¬(∃t ec1. 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
                                  ∧ 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: 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. 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 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 ∧ el ! 0 = (BasicEvent ev, s, x) ⟹
    Suc i < length el ∧ el ! i −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc i) ⟹
    (gets-e (el ! i) ∈ guard ev ∧ drop (Suc i) el ∈
```

```
        cpts-of-ev (AnonyEvent (Some (body ev))) (gets-e (el ! (Suc i))) ((getx-e (el ! i)) (k := BasicEvent ev)) )
  proof −
    assume p0: el ∈ cpts-ev ∧ el ! 0 = (BasicEvent ev, s, x)
    assume p1: Suc i < length el ∧ 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 (Some (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 (Some (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 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 (AnonyEvent (Some (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 ⟹ el ! 0 = (BasicEvent ev, s, x) ⟹
      (¬ (∃i k. Suc i < length el ∧ el ! i −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc i)) ) ⟹
      (∀j. Suc j < length el ⟶ getspc-e (el ! j) = BasicEvent ev
                              ∧ el ! j −ee→ el ! (Suc j)
                              ∧ getspc-e (el ! (Suc j)) = BasicEvent ev)
  proof −
    assume p0: el ∈ cpts-ev and
           p1: el ! 0 = (BasicEvent ev, s, x) and
           p2: ¬ (∃i k. Suc i < length el ∧ 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 ∧ 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. el ! 0 −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc 0)) by simp
            with p0 p1 have ¬ (∃t. 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 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 ∧ 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 ∧ 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. el ! (Suc m) −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc (Suc m) )) by simp
        with a4 have a5: ¬ (∃ t. 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 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) esys ⇒ 's ⇒ ('l,'k,'s) x ⇒ ('l,'k,'s) econf
  where EvtSeqrm: rm-evtsys0 (EvtSeq e es) s x= (e, s, x) |
        EvtSysrm: rm-evtsys0 (EvtSys es) s x= (AnonyEvent None, s, x)

definition rm-evtsys1 :: ('l,'k,'s) esconf ⇒ ('l,'k,'s) econf
  where rm-evtsys1 esc ≡ rm-evtsys0 (getspc-es esc) (gets-es esc) (getx-es esc)

definition rm-evtsys :: ('l,'k,'s) esconfs ⇒ ('l,'k,'s) econfs
  where rm-evtsys escfs ≡ map rm-evtsys1 escfs

definition e-eqv-einevtseq :: ('l,'k,'s) esconfs ⇒ ('l,'k,'s) econfs ⇒ ('l,'k,'s) esys ⇒ bool
  where e-eqv-einevtseq esl el es ≡ 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) ∧
                                   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⟧ ⟹ 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: ∀ i. Suc i ≤ length el ⟶ gets-e (el ! i) = gets-es (esl ! i) ∧
                                         getx-e (el ! i) = getx-es (esl ! i) ∧
                                         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 ∀ i. Suc i ≤ length ?el' ⟶ gets-e (?el' ! i) = gets-es (?esl' ! i) ∧
                                   getx-e (?el' ! i) = getx-es (?esl' ! i) ∧
                                   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) esconfs ⇒ ('l,'k,'s) 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 None, 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)
$\wedge$ *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) *esconfs* $\Rightarrow$ ('l,'k,'s) *econfs*
$\Rightarrow$ ('l,'k,'s) *event set* $\Rightarrow$ ('l,'s) *event'* $\Rightarrow$ *bool*
where *e-sim-es esl el es e* $\equiv$ *length esl = length el* $\wedge$ *getspc-es (esl!0) = EvtSys es* $\wedge$
getspc-e (el!0) = BasicEvent e $\wedge$
($\forall$ *i*. *i* < *length el* $\longrightarrow$ *gets-e (el ! i) = gets-es (esl ! i)* $\wedge$
getx-e (el ! i) = getx-es (esl ! i)) $\wedge$
($\forall$ *i*. *i* > *0* $\wedge$ *i* < *length el* $\longrightarrow$
(*getspc-es (esl!i) = EvtSys es* $\wedge$ *getspc-e (el!i) = AnonyEvent None*)
$\vee$ (*getspc-es (esl!i) = EvtSeq (getspc-e (el!i)) (EvtSys es)*)
)

7.3 Soundness of Programs

7.3.1 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* −*c*→ *x*!*Suc i* $\longrightarrow$
($\forall$ *j*. *Suc j*<*length x* $\longrightarrow$ *i*$\neq$*j* $\longrightarrow$ *x*!*j* −*pe*→ *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) ∈ 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]:
  ∀ s i. x ∈ cpts-p ⟶ x ! 0 = (Some (Basic f), s)
  ⟶ i<length x ⟶ fst(x!i)=None ⟶ (∃ j<i. x!j −c→ 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 Basic-sound:
   ⟦pre ⊆ {s. f s ∈ post}; {(s, t). s ∈ pre ∧ t = f s} ⊆ guar;
  stable pre rely; stable post rely⟧
  ⟹ ⊨ Basic f 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(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 ∀ 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

7.3.2 Soundness of the Await rule

lemma *unique-ctran-Await* [*rule-format*]:
$\forall s\ i.\ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s) \longrightarrow$
$Suc\ i{<}length\ x \longrightarrow x!i\ {-c\rightarrow}\ x!Suc\ i \longrightarrow$
$(\forall j.\ Suc\ j{<}length\ x \longrightarrow i{\neq}j \longrightarrow x!j\ {-pe\rightarrow}\ x!Suc\ j)$
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*]:
$\forall s\ i.\ \ x \in cpts\text{-}p \longrightarrow x\ !\ 0 = (Some\ (Await\ b\ c),\ s)$
$\longrightarrow i{<}length\ x \longrightarrow fst(x!i){=}None \longrightarrow (\exists j{<}i.\ x!j\ {-c\rightarrow}\ x!Suc\ j)$
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*:
  $(P, s) -c{*}\rightarrow (R, t) \Longrightarrow \exists l \in$ *cpts-of-p P s*. $(last\ l)=(R, t)$
  $\wedge$ ($\forall$ *i*. *Suc i<length l* $\longrightarrow$ *l*!*i* $-c\rightarrow$ *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*;
  $\forall V. \vdash P\ sat_p$ [*pre* $\cap$ *b* $\cap$ {*s*. *s* = *V*}, {(*s*, *t*). *s* = *t*},
        UNIV, {*s*. (*V*, *s*) $\in$ *guar*} $\cap$ *post*] $\wedge$
  $\models P\ sat_p$ [*pre* $\cap$ *b* $\cap$ {*s*. *s* = *V*}, {(*s*, *t*). *s* = *t*},
        UNIV, {*s*. (*V*, *s*) $\in$ *guar*} $\cap$ *post*] ⟧
  $\Longrightarrow \models$ *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*$\in$ *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*)
— here starts the different part.
 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
```

7.3.3 Soundness of the Conditional rule

```
lemma Cond-sound:
  ⟦ stable pre rely; ⊨ P1 sat_p [pre ∩ b, rely, guar, post];
  ⊨ P2 sat_p [pre ∩ − b, rely, guar, post]; ∀ s. (s,s)∈guar⟧
  ⟹ ⊨ (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 −c→ 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=λi. 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=λj. 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=λj. 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
```

7.3.4 Soundness of the Sequential rule

inductive-cases *Seq-cases* [*elim*!]: (*Some* (*Seq P Q*), *s*) −*c*→ *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*) $\in$ *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* ($\forall$ *j*<*Suc nata*. *fst* (((*Some* (*Seq P2 Q*), *ta*) # *xs*) ! *j*) $\neq$ *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*))) $\neq$ *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*:
$[\![\models P\ sat_p\ [pre, rely, guar, mid]; \models Q\ sat_p\ [mid, rely, guar, post]]\!]$
$\Longrightarrow \models Seq\ P\ Q\ sat_p\ [pre, rely, guar, post]$
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* $P{=}\lambda j.\ H\ j \longrightarrow J\ j \longrightarrow I\ j$ **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* $P{=}\lambda j.\ H\ j \longrightarrow J\ j \longrightarrow I\ j$ **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*
— $\exists\, i{<}length\ x.\ fst\ (x\ !\ i) = Some\ Q$
apply(*erule exE*)
apply(*drule-tac n=i* **and** $P{=}\lambda i.\ i < length\ x \wedge fst\ (x\ !\ i) = Some\ Q$ **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:etran.cases 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

7.3.5 Soundness of the While rule

lemma *last-append*[*rule-format*]:
$\forall xs.\ ys \neq [] \longrightarrow ((xs@ys)!(length\ (xs@ys) - (Suc\ 0))) = (ys!(length\ ys - (Suc\ 0)))$
apply(*induct ys*)
apply *simp*
apply *clarify*
apply (*simp add*:*nth-append*)
done

lemma *assum-after-body*:
$[\![\models P\ sat_p\ [pre \cap b,\ rely,\ guar,\ pre];$
$(Some\ P,\ s)\ \#\ xs \in cpt\text{-}p\text{-}mod;\ fst\ (last\ ((Some\ P,\ s)\ \#\ xs)) = None;\ s \in b;$
$(Some\ (While\ b\ P),\ s)\ \#\ (Some\ (Seq\ P\ (While\ b\ P)),\ s)\ \#$
$map\ (lift\ (While\ b\ P))\ xs\ @\ ys \in assume\text{-}p\ (pre,\ rely)]\!]$
$\Longrightarrow (Some\ (While\ b\ P),\ snd\ (last\ ((Some\ P,\ s)\ \#\ xs)))\ \#\ ys \in assume\text{-}p\ (pre,\ rely)$
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*]:
$[\![pre \cap - b \subseteq post;\ \models P\ sat_p\ [pre \cap b,\ rely,\ guar,\ pre];\ \forall s.\ (s,\ s) \in guar;$

```
   stable pre rely;  stable post rely; x ∈ cpt-p-mod ⟧
  ⟹  ∀ s xs. x=(Some(While b P),s)#xs ⟶ x∈assume-p(pre, rely) ⟶ x ∈ commit-p (guar, post)
apply(erule cpt-p-mod.induct)
apply safe
apply (simp-all del:last.simps)
— 5 subgoals left
apply(simp add:commit-p-def getspc-p-def gets-p-def)
— 4 subgoals left
apply(rule etran-in-comm)
apply(erule mp)
apply(erule tl-of-assum-in-assum,simp)
— While-None
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)
— WhileOne
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) −c→ 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*
— last=None
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*
— WhileMore
apply(*thin-tac P* = *While b P* ⟶ *Q* **for** *Q*)
apply(*rule ctran-in-comm*,*simp del*:*last.simps*)
— metiendo la hipotesis antes de dividir la conclusion.
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*)+
— lo de antes.
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*) −*c*→ *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
— i ≥ length xs
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)
— i>length xs
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
— last=None
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;
    ⊨ P sat_p [pre ∩ b, rely, guar, pre]; ∀ s. (s,s)∈guar⟧
  ⟹ ⊨ 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 clarify
apply(case-tac i,simp+)
apply(case-tac x,simp+)
done
```

7.3.6 Soundness of the Rule of Consequence

```
lemma Conseq-sound:
  ⟦pre ⊆ pre'; rely ⊆ rely'; guar' ⊆ guar; post' ⊆ post;
  ⊨ P sat_p [pre', rely', guar', post']⟧
  ⟹ ⊨ 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
```

7.3.7 Soundness of the Nondt rule

```
lemma unique-ctran-Nondt [rule-format]:
  ∀ s i. x ∈ cpts-p ⟶ x ! 0 = (Some (Nondt r), s) ⟶
  Suc i<length x ⟶ x!i −c→ x!Suc i ⟶
  (∀ j. Suc j<length x ⟶ i≠j ⟶ x!j −pe→ 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) ∈ 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]:
  ∀ s i. x ∈ cpts-p ⟶ x ! 0 = (Some (Nondt r), s)
  ⟶ i<length x ⟶ fst(x!i)=None ⟶ (∃ j<i. x!j −c→ 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* ⊆ {*s*. (∀ *t*. (*s*,*t*) ∈ *r* ⟶ *t* ∈ *post*) ∧ (∃ *t*. (*s*,*t*) ∈ *r*)}; {(*s*,*t*). *s* ∈ *pre* ∧ (*s*,*t*)∈*r*} ⊆ *guar*;
        stable pre rely; *stable post rely*⟧
  ⟹ ⊨ *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
```

7.3.8 Soundness of the Rule of Unprecond

```
lemma Unprecond-sound:
  assumes p0:  ⊨ P sat_p [pre, rely, guar, post]
    and  p1:  ⊨ P sat_p [pre', rely, guar, post]
   shows  ⊨ P sat_p [pre ∪ pre', rely, guar, post]
proof −
{
  fix s c
  assume c ∈ cpts-of-p (Some P) s ∩ assume-p(pre ∪ pre', rely)
  hence a1: c ∈ cpts-of-p (Some 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
```

7.3.9 Soundness of the Rule of Intpostcond

```
lemma Intpostcond-sound:
  assumes p0:  ⊨ P sat_p [pre, rely, guar, post]
    and  p1:  ⊨ P sat_p [pre, rely, guar, post']
   shows  ⊨ P sat_p [pre, rely, guar, post ∩ post']
proof −
{
  fix s c
  assume a0: c ∈ cpts-of-p (Some 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
```

7.3.10 Soundness of the Rule of Allprecond

```
lemma Allprecond-sound:
  assumes p1: ∀ v∈U. ⊨ P sat_p [{v}, rely, guar, post]
    shows  ⊨ P sat_p [U, rely, guar, post]
```

proof −
{
fix *s c*
assume *a0*: *c* ∈ *cpts-of-p* (*Some P*) *s* ∩ *assume-p*(*U*, *rely*)
then obtain *x* **where** *a1*: *x*∈*U* ∧ *gets-p* (*c*!*0*) = *x*
by (*metis* (*no-types*, *lifting*) *CollectD IntD2 assume-p-def prod.simps*(*2*))

with *p1* **have** ⊨ *P* sat_p [{*x*}, *rely*, *guar*, *post*] **by** *simp*
hence *a2*: ∀ *s*. *cpts-of-p* (*Some P*) *s* ∩ *assume-p*({*x*}, *rely*) ⊆ *commit-p*(*guar*, *post*) **unfolding** *prog-validity-def* **by** *simp*

from *a0* **have** *c* ∈ *assume-p*(*U*, *rely*) **by** *simp*
hence *gets-p* (*c*!*0*) ∈ *U* ∧ (∀ *i*. *Suc i*<*length c* ⟶
c!*i* −*pe*→ *c*!(*Suc i*) ⟶ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) ∈ *rely*) **by** (*simp add*:*assume-p-def*)
with *a1* **have** *gets-p* (*c*!*0*) ∈ {*x*} ∧ (∀ *i*. *Suc i*<*length c* ⟶
c!*i* −*pe*→ *c*!(*Suc i*) ⟶ (*gets-p* (*c*!*i*), *gets-p* (*c*!*Suc i*)) ∈ *rely*) **by** *simp*

hence *c*∈*assume-p*({*x*}, *rely*) **by** (*simp add*:*assume-p-def*)
with *a0 a2* **have** *c* ∈ *commit-p*(*guar*, *post*) **by** *auto*
}
then show *?thesis* **using** *prog-validity-def* **by** *blast*
qed

7.3.11 Soundness of the Rule of Emptyprecond

lemma *Emptyprecond-sound*: ⊨ *P* sat_p [{}, *rely*, *guar*, *post*]
unfolding *prog-validity-def* **by**(*simp add*:*assume-p-def*)

7.3.12 Soundness of the system for programs

theorem *rgsound-p*:
⊢ *P* sat_p [*pre*, *rely*, *guar*, *post*] ⟹ ⊨ *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(*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

7.4 Soundness of Events

lemma *anony-cfgs0* : ⟦∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*; *es* ∈ *cpts-ev*⟧
⟹ ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof −
assume *a0*: *es* ∈ *cpts-ev* **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 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: (e1, s1, x1) −et−et→ (e2, t1, y1) and
             b1: (e2, t1, y1) # xs1 ∈ cpts-ev 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 (AnonyEvent P) s x ⟹ ∀i. (i < length es ⟶ (∃Q. getspc-e (es!i) = AnonyEvent
Q) )
  proof −
    assume a0: es ∈ cpts-of-ev (AnonyEvent P) s x
    then have a1: es!0=(AnonyEvent P,(s,x)) ∧ es ∈ cpts-ev 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: ⊨ P sat_p [pre, rely, guar, post] ⟹ ⊨ AnonyEvent (Some P) sat_e [pre, rely, guar, post]
  proof −
    assume a0: ⊨ P sat_p [pre, rely, guar, post]
    then have a1: ∀s. cpts-of-p (Some P) s ∩ assume-p (pre, rely) ⊆ commit-p (guar, post)
      unfolding prog-validity-def cpts-of-p-def by simp
    then have ∀s x. (cpts-of-ev (AnonyEvent (Some P)) s x) ∩ assume-e (pre, rely)
                   ⊆ commit-e (guar, post)
      proof −
      {
        fix s x
        have ∀el. el∈(cpts-of-ev (AnonyEvent (Some P)) s x) ∩ assume-e (pre, rely) ⟶ el∈ commit-e (guar, post)
          proof −
          {
            fix el
            assume b0: el∈(cpts-of-ev (AnonyEvent (Some P)) s x) ∩ assume-e (pre, rely)
            then obtain pl where b1: pl = lower-evts el by simp
            with b0 have b2: pl ∈ cpts-of-p (Some P) s using equiv-lower-evts by auto
```

```
from b0 have b3: el!0=(AnonyEvent (Some P),(s,x)) and b4: el ∈ cpts-ev
  by (simp add:cpts-of-ev-def)+
from b0 have b5: el ∈ assume-e (pre, rely) by simp
have b6: gets-p (pl!0) ∈ pre
  proof −
    from b5 have c0: gets-e (el!0) ∈ pre by (simp add:assume-e-def)
    from b2 b3 have c1: gets-p (pl!0) = gets-e (el!0) by (simp add:cpts-of-p-def gets-p-def gets-e-def)
    with c0 show ?thesis by simp
  qed

have b7: ∀ i. Suc i<length pl ⟶
   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: 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
      by (metis fst-conv petranE)
    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: 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 ⟶ 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 (pre, rely) by (simp add:assume-p-def)

with a1 b2 have b9: pl ∈ commit-p (guar, post) by auto
then have b10: (∀ i. Suc i<length el ⟶
```

```
(∃ t. 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. 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: pl!i −c→ pl!(Suc i)
   proof −
     from c1 obtain t where d0: 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: (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: (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 (None) ⟶ gets-e (last el) ∈ post)
  proof
    assume c0: getspc-e (last el) = AnonyEvent (None)
    from b1 have c1: last pl = lower-anonyevt1 (last el)
     by (metis (no-types, lifting) CollectD b2 cptn-not-empty cpts-of-p-def
         last-map length-greater-0-conv length-map lower-evts-def)
```

from *b9* **have** *c2*: *getspc-p* (*last pl*) = *None* ⟶ *gets-p* (*last pl*) ∈ *post* **by** (*simp add:commit-p-def*)
from *c0 c1* **have** *c3*: *getspc-p* (*last pl*) = *None*
by (*simp add*: *getspc-p-def lower-anonyevt1-def*)
with *c2* **have** *c4*: *gets-p* (*last pl*) ∈ *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*) ∈ *post* **by** *simp*
qed

with *b10* **have** *el* ∈ *commit-e* (*guar*, *post*) **by** (*simp add:commit-e-def*)

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

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

lemma *BasicEvt-sound*:
⟦ ⊨ (*body ev*) sat_p [*pre* ∩ (*guard ev*), *rely*, *guar*, *post*];
stable pre rely; ∀ *s*. (*s*, *s*)∈*guar*⟧
⟹ ⊨ ((*BasicEvent ev*)::(′*l*,′*k*,′*s*) *event*) sat_e [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: ⊨ (*body ev*) sat_p [*pre* ∩ (*guard ev*), *rely*, *guar*, *post*]
assume *p1*: ∀ *s*. (*s*, *s*)∈*guar*
assume *p2*: *stable pre rely*
have ∀ *s x*. (*cpts-of-ev* ((*BasicEvent ev*)::(′*l*,′*k*,′*s*) *event*) *s x*) ∩ *assume-e* (*pre*, *rely*)
⊆ *commit-e* (*guar*, *post*)
proof −
{
fix *s x*
have ∀ *el*. *el*∈(*cpts-of-ev* (*BasicEvent ev*) *s x*) ∩ *assume-e* (*pre*, *rely*) ⟶ *el*∈ *commit-e* (*guar*, *post*)
proof −
{
fix *el*
assume *b0*: *el*∈(*cpts-of-ev* (*BasicEvent ev*) *s x*) ∩ *assume-e* (*pre*, *rely*)
then have *b0-1*: *el*∈(*cpts-of-ev* (*BasicEvent ev*) *s x*) **and**
b0-2: *el* ∈ *assume-e* (*pre*, *rely*) **by** *auto*
from *b0-1* **have** *b1*: *el* ! *0* = (*BasicEvent ev*, (*s*, *x*)) **and**
b2: *el* ∈ *cpts-ev* **by** (*simp add:cpts-of-ev-def*)+
from *b0-2* **have** *b3*: *gets-e* (*el*!*0*) ∈ *pre* **and**
b4: (∀ *i*. *Suc i*<*length el* ⟶ *el*!*i* −*ee*→ *el*!(*Suc i*) ⟶
(*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely*) **by** (*simp add*: *assume-e-def*)+
have *el*∈ *commit-e* (*guar*, *post*)
proof(*cases* ∃ *i k*. *Suc i* < *length el* ∧ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*))
assume *c0*: ∃ *i k*. *Suc i* < *length el* ∧ *el* ! *i* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc i*)
then obtain *m* **and** *k* **where** *c1*: *Suc m* < *length el* ∧ *el* ! *m* −*et*−(*EvtEnt* (*BasicEvent ev*))♯*k*→ *el* ! (*Suc m*)
by *auto*
with *b1 b2* **have** *c2*: ∀ *j*. *Suc j* ≤ *m* ⟶ *getspc-e* (*el* ! *j*) = *BasicEvent ev* ∧ *el* ! *j* −*ee*→ *el* ! (*Suc j*)
by (*meson evtent-in-cpts1*)
from *b1 b2 c1* **have** *c4*: *gets-e* (*el* ! *m*) ∈ *guard ev* **and**
c6: *drop* (*Suc m*) *el* ∈ *cpts-of-ev* (*AnonyEvent* (*Some* (*body ev*))) (*gets-e* (*el* ! (*Suc m*))) ((*getx-e* (*el* ! *m*)) (*k* := *BasicEvent ev*))

```
      using evtent-in-cpts2[of el ev s x m k] by auto

from p0[rule-format] c4 have c7: ⊨ ((AnonyEvent (Some (body ev)))::('l,'k,'s) event)
            sat_e [pre ∩ (guard ev), rely, guar, post]
  by (simp add: AnonyEvt-sound)

from b4 c1 c2 have c8:∀ j. Suc j ≤ m ⟶ (gets-e (el ! j), gets-e (el ! (Suc j))) ∈ rely by auto
with p2 b3 have c9: ∀ j. j ≤ m ⟶ gets-e (el ! j) ∈ pre
  proof –
  {
    fix j
    assume d0: j ≤ m
    then have gets-e (el ! j) ∈ pre
      proof(induct j)
        case 0 show ?case by (simp add: b3)
      next
        case (Suc jj)
        assume e0: Suc jj ≤ m
        assume e1: jj ≤ m ⟹ gets-e (el ! jj) ∈ pre
        from e0 c8 have (gets-e (el ! jj), gets-e (el ! (Suc jj))) ∈ 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)) ∈ pre by auto
from c7 have c12: ∀ s x. (cpts-of-ev ((AnonyEvent (Some (body ev)))::('l,'k,'s) event) s x) ∩
    assume-e(pre ∩ (guard ev), rely) ⊆ commit-e(guar, post) by (simp add:evt-validity-def)


have drop (Suc m) el ∈ assume-e(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 ⟶
          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(guar, post)
  by (meson AnonyEvt-sound IntI contra-subsetD evt-validity-def p0)


have c14: ∀ i. Suc i < length el ⟶ (∃ t. 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. 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 el ! i −ee→ el ! (Suc i) by auto
```

```
          then have ¬(∃t. 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 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: ∀i. Suc i < length (drop (Suc m) el) ⟶
                      (∃t. (drop (Suc m) el) ! i −et−t→ (drop (Suc m) el) ! Suc i) ⟶
                      (gets-e ((drop (Suc m) el) ! i), gets-e ((drop (Suc m) el) ! Suc i)) ∈ 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))) ∈ guar
                proof −
                  from d0 f1 have g0: Suc (i − Suc m) < length (drop (Suc m) el) by auto
                   from d1 f1 have (∃t. drop (Suc m) el ! (i − Suc m) −et−t→ drop (Suc m) el ! Suc (i −
Suc m))
                    using d0 by auto
                  with g0 f2 show ?thesis by simp
                qed
              then show ?thesis
                by (metis (no-types, lifting) Suc-lessD add-Suc-right
                  add-diff-inverse-nat d0 f1 less-imp-le-nat not-less-eq nth-drop)
            qed
        qed
      }
      then show ?thesis by auto
      qed


    from c13 have c15: getspc-e (last el) = AnonyEvent None ⟶ gets-e (last el) ∈ 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: ¬ (∃i k. Suc i < length el ∧ el ! i −et−(EvtEnt (BasicEvent ev))♯k→ el ! (Suc i) )
    with b1 b2 have c1: ∀j. Suc j < length el ⟶ getspc-e (el ! j) = BasicEvent ev
              ∧ el ! j −ee→ el ! (Suc j)
              ∧ getspc-e (el ! (Suc j)) = BasicEvent ev
      using no-evtent-in-cpts by simp
    then have c2: (∀i. Suc i<length el ⟶ (∃t. el!i −et−t→ el!(Suc i))
            ⟶ (gets-e (el!i), gets-e (el!Suc i)) ∈ guar)
      proof −
      {
        fix i
        assume Suc i<length el
```

and *d0*: $\exists t.\ el!i\ -et-t\rightarrow\ el!(Suc\ i)$
with *c1* **have** $el\ !\ i\ -ee\rightarrow\ el\ !\ Suc\ i$ **by** *auto*
then have $\neg\ (\exists t.\ el!i\ -et-t\rightarrow\ el!(Suc\ i))$
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** $el \neq []$ **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 last.simps last-length nth-Cons-Suc*)
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* (*None*) $\longrightarrow$ *gets-e* (*last el*) $\in$ *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*:
$[\![pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\models ev\ sat_e\ [pre',\ rely',\ guar',\ post']]\!]$
$\Longrightarrow \models ev\ sat_e\ [pre,\ rely,\ guar,\ post]$
proof –
assume *p0*: $pre \subseteq pre'$
and *p1*: $rely \subseteq rely'$
and *p2*: $guar' \subseteq guar$
and *p3*: $post' \subseteq post$
and *p4*: $\models ev\ sat_e\ [pre',\ rely',\ guar',\ post']$
from *p4* **have** *p5*: $\forall s\ x.\ (cpts\text{-}of\text{-}ev\ ev\ s\ x) \cap assume\text{-}e(pre',\ rely') \subseteq commit\text{-}e(guar',\ post')$
by (*simp add*: *evt-validity-def*)
have $\forall s\ x.\ (cpts\text{-}of\text{-}ev\ ev\ s\ x) \cap assume\text{-}e(pre,\ rely) \subseteq commit\text{-}e(guar,\ post)$
proof –
{
fix *c s x*

assume *a0*: c∈(cpts-of-ev ev s x) ∩ assume-e(pre, rely)
then have c∈(cpts-of-ev ev s x) ∧ c∈assume-e(pre, rely) **by** *simp*
with *p0 p1* **have** c∈(cpts-of-ev ev s x) ∧ c∈assume-e(pre′, rely′)
using *assume-e-imp*[*of pre pre′ rely rely′ c*] **by** *simp*
with *p5* **have** c∈commit-e(guar′, post′) **by** *auto*
with *p2 p3* **have** c∈commit-e(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*:
⊢ Evt sat_e [pre, rely, guar, post] ⟹ ⊨ 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

7.5 Soundness of Event Systems

lemma *evtseq-nfin-samelower*: ⟦esl ∈ cpts-of-es (EvtSeq e es) s x; ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ es⟧
⟹ (∃ el. (el ∈ cpts-of-ev e s x ∧ length esl = length el ∧ e-eqv-einevtseq esl el es))
proof −
assume *p0*: esl ∈ cpts-of-es (EvtSeq e es) s x
and *p1*: ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ es
from *p0* **have** *p01*: esl ! 0 = (EvtSeq e es, s, x) ∧ esl ∈ cpts-es **by** (*simp add*: *cpts-of-es-def*)
then have *p01-1*: esl ! 0 = (EvtSeq e es, s, x) **by** *simp*
then have *p2*: ∃ e. getspc-es (esl ! 0) = EvtSeq e es **by** (*simp add*:*getspc-es-def*)
from *p01* **have** *p01-2*: esl ∈ cpts-es **by** *simp*
let ?el = rm-evtsys esl
have *a1*: length esl = length ?el **by** (*simp add*: *rm-evtsys-def*)
moreover have ?el ∈ cpts-of-ev e s x
proof −
from *p01-2 p1 p2* **have** *b1*: ?el ∈ cpts-ev
proof(*induct esl*)
case (CptsEsOne es1 s1 x1)
assume *c0*: ∃ 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 ∈ cpts-es
and *c1*: ∀ i. Suc i ≤ length ((es1, t1, x1) # xs1) ⟶ getspc-es (((es1, t1, x1) # xs1) ! i) ≠ es
⟹∃ e. getspc-es (((es1, t1, x1) # xs1) ! 0) = EvtSeq e es
⟹ rm-evtsys ((es1, t1, x1) # xs1) ∈ cpts-ev
and *c11*: ∀ i. Suc i ≤ length ((es1, s1, y1) # (es1, t1, x1) # xs1)
⟶ getspc-es (((es1, s1, y1) # (es1, t1, x1) # xs1) ! i) ≠ es
and *c2*: ∃ 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** ∀ i. Suc i ≤ length ((es1, t1, x1) # xs1) ⟶ getspc-es (((es1, t1, x1) # xs1) ! i) ≠ es

by *auto*
with *c1 c4* **have** *c5*: *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) ∈ *cpts-ev* **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*: (*es1*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-es*
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*
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*. (*e1*,*s1*,*x1*) −*et*−*t*→ (*e2*,*t1*,*y1*) **by** (*simp add*: *evtseq-tran-exist-etran*)
then obtain *t* **where** *c71*: (*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* **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 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 ∈ cpts-es
    and  a1: ∀ i. Suc i ≤ length ((es1, t1, x1) # xs1) ⟶ getspc-es (((es1, t1, x1) # xs1) ! i) ≠ es ⟹
           ∃ e. getspc-es (((es1, t1, x1) # xs1) ! 0) = EvtSeq e es ⟹
           e-eqv-einevtseq ((es1, t1, x1) # xs1) (rm-evtsys ((es1, t1, x1) # xs1)) es
    and  a2: ∀ i. Suc i ≤ length ((es1, s1, y1) # (es1, t1, x1) # xs1)
             ⟶ getspc-es (((es1, s1, y1) # (es1, t1, x1) # xs1) ! i) ≠ es
    and  a3: ∃ e. getspc-es (((es1, s1, y1) # (es1, t1, x1) # xs1) ! 0) = EvtSeq e es
  from a2 have a4: ∀ i. Suc i ≤ length ((es1, t1, x1) # xs1) ⟶ getspc-es (((es1, t1, x1) # xs1) ! i) ≠ 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 ∃ 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: (es1, s1, x1) −es−et→ (es2, t1, y1)
    and  a1: (es2, t1, y1) # xs1 ∈ cpts-es
```

    and a2: ∀ 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 ⟹
        e-eqv-einevtseq ((es2, t1, y1) # xs1) (rm-evtsys ((es2, t1, y1) # xs1)) es
    and a3: ∀ i. Suc i ≤ length ((es1, s1, x1) # (es2, t1, y1) # xs1)
        ⟶ getspc-es (((es1, s1, x1) # (es2, t1, y1) # xs1) ! i) ≠ es
    and a4: ∃ e. getspc-es (((es1, s1, x1) # (es2, t1, y1) # xs1) ! 0) = EvtSeq e es
  from a3 **have** a5: ∀ i. Suc i ≤ length ((es2, t1, y1) # xs1) ⟶ getspc-es (((es2, t1, y1) # xs1) ! i) ≠ 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) ≠ es **by** auto
  then have a7: es2 ≠ es **by** (simp add:getspc-es-def)
  with a0 a6 **have** ∃ e2. es2 = EvtSeq e2 es **by** (meson evtseq-tran-evtsys)
  then obtain e2 **where** a8: es2 = EvtSeq e2 es **by** auto
  then have a9: ∃ 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 ∈ cpts-of-ev e s x ∧ length esl = length ?el ∧ e-eqv-einevtseq esl ?el es **by** auto
then show ?thesis **by** auto
qed

lemma evtseq-fst-finish:
⟦esl ∈ cpts-es; getspc-es (esl ! 0) = EvtSeq e es; Suc m ≤ length esl;
  ∃ i. i ≤ m ∧ getspc-es (esl ! i) = es⟧ ⟹
  ∃ i. (i ≤ m ∧ getspc-es (esl ! i) = es) ∧ (∀ j. j < i ⟶ getspc-es (esl ! j) ≠ es)
proof −
  assume p0: esl ∈ cpts-es
    and p1: getspc-es (esl ! 0) = EvtSeq e es
    and p2: Suc m ≤ length esl
    and p3: ∃ i. i ≤ m ∧ getspc-es (esl ! i) = es
  have ∀ m. esl ∈ cpts-es ∧ getspc-es (esl ! 0) = EvtSeq e es ∧ Suc m ≤ length esl ∧
      (∃ i. i ≤ m ∧ getspc-es (esl ! i) = es) ⟶
    (∃ i. (i ≤ m ∧ getspc-es (esl ! i) = es) ∧ (∀ j. j < i ⟶ getspc-es (esl ! j) ≠ es))
  proof −
  {
    fix m
    assume a0: esl ∈ cpts-es
      and a1: getspc-es (esl ! 0) = EvtSeq e es
      and a2: Suc m ≤ length esl
      and a3: (∃ i. i ≤ m ∧ getspc-es (esl ! i) = es)
    then have ∃ i. (i ≤ m ∧ getspc-es (esl ! i) = es) ∧ (∀ j. j < i ⟶ getspc-es (esl ! j) ≠ es)
      proof(induct m)
        case 0 **show** ?case **using** 0.prems(4) **by** auto
      next
        case (Suc n)
        assume b0: esl ∈ cpts-es ⟹
            getspc-es (esl ! 0) = EvtSeq e es ⟹
            Suc n ≤ length esl ⟹
            ∃ i≤n. getspc-es (esl ! i) = es ⟹
            ∃ i. (i ≤ n ∧ getspc-es (esl ! i) = es) ∧ (∀ j. j < i ⟶ getspc-es (esl ! j) ≠ es)
          and b1: esl ∈ cpts-es

and *b2*: *getspc-es* (*esl* ! *0*) = *EvtSeq e es*
and *b3*: *Suc* (*Suc n*) ≤ *length esl*
and *b4*: ∃ *i*≤*Suc n*. *getspc-es* (*esl* ! *i*) = *es*
show *?case*
proof(*cases* ∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) = *es*)
assume *c0*: ∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) = *es*
with *b0 b1 b2 b3* **have** ∃ *i*. (*i* ≤ *n* ∧ *getspc-es* (*esl* ! *i*) = *es*) ∧ (∀ *j*. *j* < *i* ⟶ *getspc-es* (*esl* ! *j*) ≠ *es*)
using *Suc-leD* **by** *blast*
then show *?case* **using** *le-Suc-eq* **by** *blast*
next
assume *c0*: ¬ (∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) = *es*)
with *b4* **have** *getspc-es* (*esl* ! (*Suc n*)) = *es* **using** *le-SucE* **by** *auto*
moreover from *c0* **have** ∀ *j*. *j* < *Suc n* ⟶ *getspc-es* (*esl* ! *j*) ≠ *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* :
⟦ ⊨ *e* sat_e [*pre*, *rely1*, *guar1*, *post1*]; ⊨ *es* sat_s [*pre2*, *rely2*, *guar2*, *post*];
rely ⊆ *rely1*; *rely* ⊆ *rely2*; *guar1* ⊆ *guar*; *guar2* ⊆ *guar*; *post1* ⊆ *pre2*⟧
⟹ ⊨ *EvtSeq e es* sat_s [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: ⊨ *e* sat_e [*pre*, *rely1*, *guar1*, *post1*]
and *p1*: ⊨ *es* sat_s [*pre2*, *rely2*, *guar2*, *post*]
and *p2*: *rely* ⊆ *rely1*
and *p3*: *rely* ⊆ *rely2*
and *p4*: *guar1* ⊆ *guar*
and *p5*: *guar2* ⊆ *guar*
and *p6*: *post1* ⊆ *pre2*
then have ∀ *s x*. (*cpts-of-es* (*EvtSeq e es*) *s x*) ∩ *assume-es*(*pre*, *rely*) ⊆ *commit-es*(*guar*, *post*)
proof −
{
fix *s x*
have ∀ *esl*. *esl*∈(*cpts-of-es* (*EvtSeq e es*) *s x*) ∩ *assume-es* (*pre*, *rely*) ⟶ *esl*∈ *commit-es* (*guar*, *post*)
proof −
{
fix *esl*
assume *a0*: *esl* ∈ (*cpts-of-es* (*EvtSeq e es*) *s x*) ∩ *assume-es* (*pre*, *rely*)
then have *a01*: *esl* ∈ *cpts-of-es* (*EvtSeq e es*) *s x* **by** *simp*
from *a0* **have** *a02*: *esl* ∈ *assume-es* (*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* **by** (*simp add*: *cpts-of-es-def*)

have *esl*∈ *commit-es* (*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 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 e s x* ∧ *length esl* = *length el* ∧ *e-eqv-einevtseq esl el es*
by *auto*
have *el* ∈ *assume-e* (*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 ⟶ 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: 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 esl ! i −ese→ esl ! Suc i by (simp add: eqconf-esetran)
        with a02 b1 c0 have (gets-es (esl!i), gets-es (esl!Suc i)) ∈ 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)) ∈ 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
with p0 b1 have el ∈ commit-e(guar1, post1)
  by (meson IntI contra-subsetD evt-validity-def)
then have ∀ i. Suc i<length el ⟶ (∃ t. 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 b2: ∀ i. Suc i<length el ⟶ (∃ t. el!i −et−t→ el!(Suc i))
      ⟶ (gets-e (el!i), gets-e (el!Suc i)) ∈ guar by auto
show ?thesis
  proof(simp add:commit-es-def)
    show ∀ i. Suc i < length esl ⟶ (∃ t. 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. esl ! i −es−t→ 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) ≠ 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) ≠ getspc-e (el ! Suc i) by simp
        then have ∃t. (el ! i) −et−t→ (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)) ∈ 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)) ∈ guar by simp
      }
      then show ?thesis by auto
      qed
    qed
  next
    assume b0: ¬ (∀i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ es)
    from a01-1 have b00: getspc-es (esl ! 0) = EvtSeq e es by (simp add:getspc-es-def)
    from b0 have ∃m. Suc m ≤ length esl ∧ getspc-es (esl ! m) = es by auto
    then obtain m where b1: Suc m ≤ length esl ∧ getspc-es (esl ! m) = es by auto
    then have ∃i. i ≤ m ∧ getspc-es (esl ! i) = es by auto
   with a01-1 a01-2 b00 b1 have b2: ∃i. (i ≤ m ∧ getspc-es (esl ! i) = es) ∧ (∀j. j < i ⟶ getspc-es (esl !
j) ≠ es)
      using evtseq-fst-finish by blast
    then obtain n where b3: (n ≤ m ∧ getspc-es (esl ! n) = es) ∧ (∀j. j < n ⟶ getspc-es (esl ! j) ≠ es)
      by auto
    with b00 have b41: n ≠ 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 ∈ cpts-es
      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 ∈ cpts-es
      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 (EvtSeq e es) s x by (simp add:cpts-of-es-def)
    ultimately have b12: ∃el. (el ∈ cpts-of-ev 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 e s x ∧ length esl0 = length el ∧ e-eqv-einevtseq esl0 el es
      by auto
    then have b12-2: el ∈ cpts-ev by (simp add:cpts-of-ev-def)

    from a02 have b13: gets-es (esl!0) ∈ pre ∧ (∀i. Suc i<length esl ⟶
                    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 (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 ⟶ 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* (*pre*, *rely1*)
  by (*simp add*: *assume-es-def subset-iff*)

have *b16*: *el* ∈ *assume-e* (*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* ⟶ *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*: *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*: *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* ⟶ *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*(*guar1*, *post1*)
  using *b12-1 b16 evt-validity-def p0* **by** *fastforce*
then have *b18*: ∀ *i*. *Suc i*<*length el* ⟶ (∃ *t*. *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*. *el*!*i* −*et*−*t*→ *el*!(*Suc i*))
    ⟶ (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *guar* **by** *auto*

from *b11* **have** ∃ *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* ∈ *cpts-of-es es sn xn* **by** (*simp add*:*cpts-of-es-def*)

have ∀ *i*. *Suc i*<*length esl* ⟶ (∃ *t*. *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*. *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*)
have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *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*) ≠ *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*) ≠ *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *Suc-lessD b5 d0*)
then have ∃ *t*. (*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*)) ∈ *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*)) ∈ *guar* **by** *simp*

then show *?thesis* **by** (*simp add*: *Suc-lessD b5 d0*)
next
assume *d0*: ¬ (*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* **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 d2 d4 have d6: ∃ est. esl ! (n−1) −es−est→ esl ! n
  by (metis (no-types, lifting) One-nat-def Suc-le-lessD Suc-pred a01-2
    b3 b4 b6 b9 cpts-es-not-empty d1-1 diff-less esetran.cases
    incpts-es-impl-evnorcomptran le-numeral-extra(4) length-drop
    length-greater-0-conv zero-less-diff)
with d2 have d7: ∃ t. (getspc-e (el ! (n−1)), sn1, xn1) −et−t→(AnonyEvent (None),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 (None),sn, xn)] ∈ cpts-ev
  using cpts-ev-onemore by fastforce
let ?el1 = el @ [(AnonyEvent (None),sn, xn)]

from d8 have d9: ?el1 ∈ cpts-of-ev 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 (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 ⟶  ?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: ?el1!i −ee→ ?el1!(Suc i)
        from b16 have e2: ∀ i. Suc i<length el ⟶  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 None, 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 ∈ commit-e(guar1, post1)
  using evt-validity-def p0 by fastforce
```

```
have d11: getspc-e (last ?el1) = AnonyEvent (None) by (simp add:getspc-e-def)
with d10 have d12: gets-e (last ?el1) ∈ post1 by (simp add: commit-e-def)

show ?thesis
  proof(cases Suc i = n)
    assume g0: Suc i = n
    from d10 have (∀ i. Suc i<length ?el1 ⟶ (∃ t. ?el1!i −et−t→ ?el1!(Suc i))
        ⟶ (gets-e (?el1!i), gets-e (?el1!Suc i)) ∈ guar1) by (simp add: commit-e-def)
    with d7 have g1: (gets-e (?el1!i), gets-e (?el1!Suc i)) ∈ 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 None, 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) ∈ guar1 by (simp add:gets-e-def)
    with p4 have (sn1,sn) ∈ guar by auto
    with d4 d2 have (gets-es (esl ! (n − 1)), gets-es (esl ! Suc (n − 1))) ∈ guar
      by (simp add: gets-es-def b4)
    then show ?thesis using g0 by auto
  next
    assume Suc i ≠ 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 ∈ cpts-of-es es sn xn by (simp add:cpts-of-es-def)

    have esl1 ∈ assume-es (pre2, rely2)
      proof(simp add:assume-es-def, rule conjI)
        from d12 have sn ∈ post1 by (simp add:gets-e-def)
        with g2 p6 show gets-es (esl1 ! 0) ∈ pre2
          using gets-es-def by (metis fst-conv rev-subsetD snd-conv)
        show ∀ i. Suc i < length esl1 ⟶ esl1 ! i −ese→ esl1 ! Suc i
          ⟶ (gets-es (esl1 ! i), gets-es (esl1 ! Suc i)) ∈ rely2
          proof −
          {
            fix i
            assume h0: Suc i < length esl1
              and  h1: esl1 ! i −ese→ 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: esl ! (n + i) −ese→ 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))) ∈ rely
              by (simp add:assume-es-def)
            with h2 h3 have (gets-es (esl1 ! i), gets-es (esl1 ! Suc i)) ∈ rely by simp

            then have (gets-es (esl1 ! i), gets-es (esl1 ! Suc i)) ∈ rely2
              using p3 by auto
          }
          then show ?thesis by auto
          qed

      qed
    with p1 g3 have g4: esl1 ∈ commit-es (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 ∀ i. Suc i<length esl1 ⟶ (∃ t. esl1!i −es−t→ esl1!(Suc i))
                      ⟶ (gets-es (esl1!i), gets-es (esl1!Suc i)) ∈ guar2 by (simp add:commit-es-def)
                  with g7 have (gets-es (esl1!(i − n)), gets-es (esl1!(Suc i − n))) ∈ guar2
                    using Suc-diff-le c1 g1 g5 g6 by auto
                  with g5 g6 have (gets-es (esl ! i), gets-es (esl ! Suc i)) ∈ 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) esconfs ⇒('l,'k,'s) event set ⇒
                            (('l,'k,'s) esconfs) list ⇒ (('l,'k,'s) 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*≤*length esl* ∧ *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* ≤ *length lst*⟧ ⟹
∃ *k*. *k* ≤ *length esl* ∧ *k* = *length* (*concat* (*take i lst*)) ∧ *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*. *k* = *length* (*concat* (*take ii lst*))
∧ *take k esl* = *concat* (*take ii lst*)
and *a1*: *esl* = *concat lst*
and *a2*: *Suc ii* ≤ *length lst*
then have ∃ *k*≤*length esl*. *k* = *length* (*concat* (*take ii lst*))
∧ *take k esl* = *concat* (*take ii lst*)
using *Suc-leD* **by** *blast*
then obtain *k* **where** *a3*: *k*≤*length esl* ∧ *k* = *length* (*concat* (*take ii lst*))
∧ *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*]:
∀ *esl lst*. *esl* = *concat lst* ∧ (∀ *i*<*length lst*. *length* (*lst*!*i*) > *0*)⟶
(∀ *i*. *Suc i* < *length esl*
⟶ (∃ *k j*. *Suc k* < *length lst* ∧ *Suc j* < *length* (*lst*!*k*@[*lst*!(*Suc k*)!*0*])
∧ *esl*!*i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*j* ∧ *esl*!*Suc i* = (*lst*!*k*@[*lst*!(*Suc k*)!*0*])!*Suc j*
∨ *Suc k* = *length lst* ∧ *Suc j* < *length* (*lst*!*k*) ∧ *esl*!*i* = *lst*!*k*!*j* ∧ *esl*!*Suc i* = *lst*!*k*!*Suc j*))
proof −
{
fix *lst*
have ∀ *esl*. *esl* = *concat lst* ∧ (∀ *i*<*length lst*. *length* (*lst*!*i*) > *0*)⟶

```
(∀i. Suc i < length esl
  ⟶ (∃k j. Suc k < length lst ∧ Suc j < length (lst!k@[lst!(Suc k)!0])
          ∧ esl!i = (lst!k@[lst!(Suc k)!0])!j ∧ esl!Suc i = (lst!k@[lst!(Suc k)!0])!Suc j
      ∨ Suc k = length lst ∧ Suc j < length (lst!k) ∧ esl!i = lst!k!j ∧ esl!Suc i = lst!k!Suc j))
proof(induct lst)
  case Nil then show ?case by simp
next
  case (Cons l lt)
  assume a0: ∀esl. esl = concat lt ∧ (∀i<length lt. 0 < length (lt ! i)) ⟶
  (∀i. Suc i < length esl ⟶
     (∃k j. Suc k < length lt ∧
          Suc j < length (lt ! k @ [lt ! Suc k ! 0]) ∧
          esl ! i = (lt ! k @ [lt ! Suc k ! 0]) ! j ∧ esl ! Suc i = (lt ! k @ [lt ! Suc k ! 0]) ! Suc j ∨
          Suc k = length lt ∧ Suc j < length (lt ! k) ∧ esl ! i = lt ! k ! j ∧ esl ! Suc i = lt ! k ! Suc j))
  {
    fix esl
    assume b0: esl = concat (l # lt)
      and  b1: ∀i<length (l # lt). 0 < length ((l # lt) ! i)

    {
      fix i
      assume c0: Suc i < length esl
      then have ∃k j. Suc k < length (l # lt) ∧
          Suc j < length ((l # lt) ! k @ [(l # lt) ! Suc k ! 0]) ∧
          esl ! i = ((l # lt) ! k @ [(l # lt) ! Suc k ! 0]) ! j ∧
          esl ! Suc i = ((l # lt) ! k @ [(l # lt) ! Suc k ! 0]) ! Suc j ∨
          Suc k = length (l # lt) ∧
          Suc j < length ((l # lt) ! k) ∧ esl ! i = (l # lt) ! k ! j ∧ 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 # []) ∧
            Suc i < length ((l # []) ! 0) ∧ esl ! i = (l # []) ! 0 ! i ∧ 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*: $i \neq 0$
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** $\exists k.\ k \le length\ esl \wedge k = length\ (concat\ (take\ i\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ i\ lst)$
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** (*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*:
$[\![esl = concat\ lst;\ Suc\ i < length\ lst;\ length\ (lst!Suc\ i) > 1]\!] \Longrightarrow$
$\exists k\ j.\ k = length\ (concat\ (take\ i\ lst)) \wedge j = Suc\ (length\ (concat\ (take\ (Suc\ i)\ lst))) \wedge$
$k \le length\ esl \wedge j < length\ esl \wedge k < j \wedge drop\ k\ (take\ j\ esl) = lst!i\ @\ [lst!Suc\ i!0]$
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*:

```
    ∀ esl rlst es. esl∈cpts-es ∧ (rlst::(('l,'k,'s) esconfs) list) ≠ []
                ⟶ concat (parse-es-cpts-i2 esl es rlst) = concat rlst @ esl
  proof –
  {
    fix esl
    have ∀ rlst es. esl∈cpts-es ∧ (rlst::(('l,'k,'s) esconfs) list) ≠ [] ⟶ 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: ∀ rlst es. esl1 ∈ cpts-es ∧ rlst ≠ [] ⟶ concat (parse-es-cpts-i2 esl1 es rlst) = concat rlst @ esl1
        then show ?case
          proof –
          {
            fix rlst es
            assume b0: esc # esl1 ∈ cpts-es ∧ (rlst::(('l,'k,'s) esconfs) list) ≠ []
            have concat (parse-es-cpts-i2 (esc # esl1) es rlst) = concat rlst @ (esc # esl1)
              proof(cases getspc-es esc = EvtSys es ∧ length esl1 > 0 ∧ getspc-es (esl1!0) ≠ EvtSys es)
                assume c0: getspc-es esc = EvtSys es ∧ length esl1 > 0 ∧ getspc-es (esl1!0) ≠ 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]] ≠ [] by simp
                from b0 c0 have esl1 ∈ cpts-es 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: ¬(getspc-es esc = EvtSys es ∧ length esl1 > 0 ∧ getspc-es (esl1!0) ≠ 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 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 ⟹ 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 ∧ (l2::(('l,'k,'s) 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 ∧ (l2::(('l,'k,'s) esconfs) list) ≠ []
              ⟶ 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: ∀ l1 l2 es. esl1 ∈ cpts-es ∧ (l2::(('l,'k,'s) esconfs) list) ≠ []
                    ⟶ 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 ∈ cpts-es
              and  b1: (l2::(('l,'k,'s) esconfs) list) ≠ []
            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 append1-eq-conv append-assoc append-butlast-last-id
                      append-is-Nil-conv length-butlast list-update-length)
                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 ∈ cpts-es using cpts-es-dropi by force
                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
                    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 ∧ (l2::(('l,'k,'s) 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 ∧ 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 ∧ 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 ∧ 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
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* **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** *auto*
      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* ∧ *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* ∧ *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 ∧ 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
∧ getspc-es (elst ! i ! 1) ≠ EvtSys es)
then show ?case
proof −
{
fix elst rlst es l
assume b0: esc # esl1 ∈ cpts-es
and b1: rlst ≠ []
and b2: elst = parse-es-cpts-i2 (esc # esl1) es rlst
have ∀ 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(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 ∈ cpts-es **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 (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** auto
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** $e3$: $?elst2 = parse\text{-}es\text{-}cpts\text{-}i2\ esl2\ es\ [[esc]@[ec1]]$ **by** *simp*
from $c1\ c2\ e0$ **have** $esl2{\in}cpts\text{-}es$ **using** *cpts-es-dropi* **by** *force*
with $e3$ **have** $e4$: $\exists\, i{\le}length\ (?elst2!0).\ take\ i\ (?elst2!0) = [esc]@[ec1]$
using *parse-es-cpts-i2-fst* **by** *blast*
with $d0\ d01\ e1\ e2\ e3$ **show** *?thesis*
proof −
{
fix i
assume $f0$: $length\ rlst \le i \wedge i < length\ elst$
have $length\ (elst\ !\ i) \ge 2 \wedge getspc\text{-}es\ (elst\ !\ i\ !\ 0) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (elst\ !\ i\ !\ 1) \ne EvtSys\ es$
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 $d0$: $\neg(getspc\text{-}es\ esc = EvtSys\ es \wedge length\ esl1 > 0 \wedge getspc\text{-}es\ (esl1!0) \ne EvtSys\ es)$
then have $d1$: $parse\text{-}es\text{-}cpts\text{-}i2\ (esc\ \#\ esl1)\ es\ rlst =$
$parse\text{-}es\text{-}cpts\text{-}i2\ esl1\ es\ (list\text{-}update\ rlst\ (length\ rlst - 1)\ (last\ rlst\ @\ [esc]))$ **by** *auto*
with $b2$ **have** $d2$: $elst = parse\text{-}es\text{-}cpts\text{-}i2\ esl1\ es\ (list\text{-}update\ rlst\ (length\ rlst - 1)\ (last\ rlst\ @\ [esc]))$
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*]:
$[\![esl{\in}cpts\text{-}es;\ rlst \ne [];\ elst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ rlst]\!] \Longrightarrow$
$\forall\, i.\ i \ge length\ rlst \wedge i < length\ elst \longrightarrow length\ (elst!i) \ge 2$
$\wedge\ getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \ne EvtSys\ es$
using *parse-es-cpts-i2-start-withlen* **by** *fastforce*

lemma *parse-es-cpts-i2-fstempty*: $[\![esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;\ esl{\in}cpts\text{-}es;$
$rlst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]]\!] \Longrightarrow rlst!0 = []$
proof −
assume $p0$: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and $p1$: $esl{\in}cpts\text{-}es$
and $p2$: $rlst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]$

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*
  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*;
  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* ∧ *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* ∧ *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* ∧ *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*
          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* **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;
    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
      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;
      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∈cpts-es
      and p2: elst = parse-es-cpts-i2 esl es [[]]
    from p1 p2 have a0: ∀ i. i ≥ length [[]] ∧ i < length elst ⟶ length (elst!i) ≥ 2  ∧
          getspc-es (elst!i!0) = EvtSys es ∧ getspc-es (elst!i!1) ≠ 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) ≥ 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-es (elst!Suc i!0) = EvtSys es ∧ getspc-es (elst!Suc i!1) ≠ 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) ≥ 2 ∧ getspc-es ((tl elst)!i!0) = EvtSys es
          ∧ getspc-es ((tl elst)!i!1) ≠ 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∈cpts-es;
    elst = tl (parse-es-cpts-i2 esl es [[]]); Suc i < length elst; esl1 = elst!i@[elst!Suc i!0]⟧ ⟹
     ¬(∃j. j > 0 ∧ Suc j < length esl1 ∧
          getspc-es (esl1!j) = EvtSys es ∧ getspc-es (esl1!Suc j) ≠ EvtSys es)
 proof −
  assume p0: esl = (EvtSys es, s, x) # (EvtSeq e (EvtSys es), s1,x1) # xs
   and p1: esl∈cpts-es
   and p2: elst = tl (parse-es-cpts-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 ¬(∃j. j > 0 ∧ Suc j < length ?esl2 ∧
          getspc-es (?esl2!j) = EvtSys es ∧ getspc-es (?esl2!Suc j) ≠ EvtSys es)
   using parse-es-cpts-i2-noent-mid[of esl es s x e s1 x1 xs elst]
    by (meson Suc-lessD parse-es-cpts-i2-noent-mid)
  moreover
  from p0 p1 p2 p3 have getspc-es (elst!Suc i!0) = EvtSys es
   using parse-es-cpts-i2-start-aux[of esl es s x e s1 x1 xs
      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;
     elst = tl (parse-es-cpts-i2 esl es [[]])⟧ ⟹
     ∀i. i < length elst ⟶ concat (drop i elst)∈cpts-es
 proof −
  assume p0: esl = (EvtSys es, s, x) # (EvtSeq e (EvtSys es), s1,x1) # xs
   and p1: esl∈cpts-es
   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
    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
      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 concat.simps(2))
     from c0 c1 have concat (drop j elst) ∈ cpts-es by simp
     with c1 c2 show ?case
      using cpts-es-dropi2[of concat (drop j elst) 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:
```

$\llbracket esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;\ esl{\in}cpts\text{-}es;$
$\quad elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])\rrbracket \Longrightarrow$
$\quad \forall\, i.\ Suc\ i < length\ elst \longrightarrow (elst!i)@[elst!Suc\ i!0] \in cpts\text{-}es$

proof −
  assume *p0*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
    and *p1*: $esl{\in}cpts\text{-}es$
    and *p2*: $elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
  then have *p3*: $concat\ elst = esl$ **using** *parse-es-cpts-i2-concat3* **by** *metis*
  from *p0 p1 p2* **have** *p4*: $\forall\, i.\ i < length\ elst \longrightarrow length\ (elst!i) \geq 2$
    using *parse-es-cpts-i2-start-aux*[*of esl es s x e s1 x1 xs parse-es-cpts-i2 esl es* [[]]]
      by *simp*

  {
    fix *i*
    assume *a0*: $Suc\ i < length\ elst$
    have $(elst!i)@[elst!Suc\ i!0] \in cpts\text{-}es$
      proof(*cases* $i = 0$)
        assume *b0*: $i = 0$
        with *a0 p4* **have** *b1*: $length\ (elst!1) \geq 2$ **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 length* (*elst* ! 0)]
          by (*metis One-nat-def Suc-lessD add.right-neutral add-Suc-right le-less-linear take-all*)
      next
        assume $i{\neq}0$
        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*: $length\ (?elst!1) \geq 2$ **by** *auto*
        from *p0 p1 p2 a0 b1* **have** *b2*: $?esl{\in}cpts\text{-}es$
          using *parse-es-cpts-i2-drop-cptes*[*of esl es s x e s1 x1 xs 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 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*;
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
last elst ∈*cpts-es*
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es*
and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
then have ∀ *i*. *i* < *length elst* ⟶ *concat* (*drop i elst*)∈*cpts-es*
using *parse-es-cpts-i2-drop-cptes*[*of esl es s x e s1 x1 xs 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*; *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*
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* ∧ *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 −
{
fix *m*
assume *a0*: *esl* ∈ *cpts-es*
and *a1*: *getspc-es* (*esl* ! *0*) = *EvtSys es*
and *a2*: *Suc m* ≤ *length esl*
and *a3*: ∃ *i*. *i* ≤ *m* ∧ *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*
then have ∃ *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(*induct m*)
case *0* **show** *?case* **using** *0.prems*(*4*) *p1* **by** *auto*
next
case (*Suc n*)
assume *b0*: *esl* ∈ *cpts-es* ⟹
getspc-es (*esl* ! *0*) = *EvtSys es* ⟹
Suc n ≤ *length esl* ⟹
∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) ≠ *EvtSys es* ⟹
∃ *i*. (*i* < *n* ∧ *getspc-es* (*esl* ! *i*) = *EvtSys es*
∧ *getspc-es* (*esl* ! *Suc i*) ≠ *EvtSys es*)
∧ (∀ *j*<*i*. *getspc-es* (*esl* ! *j*) = *EvtSys es*)
and *b1*: *esl* ∈ *cpts-es*
and *b2*: *getspc-es* (*esl* ! *0*) = *EvtSys es*
and *b3*: *Suc* (*Suc n*) ≤ *length esl*
and *b4*: ∃ *i*≤*Suc n*. *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*
show *?case*
proof(*cases* ∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) ≠ *EvtSys es*)
assume *c0*: ∃ *i*≤*n*. *getspc-es* (*esl* ! *i*) ≠ *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*: $\neg(\exists i{\le}n.\ \textit{getspc-es}\ (\textit{esl} \ !\ i) \neq \textit{EvtSys es})$
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*:
$[\![\textit{esl}{\in}\textit{cpts-es};\ \textit{length esl} > 0;\ \exists e.\ \textit{getspc-es}\ (\textit{esl}!0) = \textit{EvtSeq}\ e\ (\textit{EvtSys es});$
$\neg(\exists j.\ \textit{Suc}\ j < \textit{length esl} \wedge \textit{getspc-es}\ (\textit{esl}!j) = \textit{EvtSys es} \wedge \textit{getspc-es}\ (\textit{esl}!\textit{Suc}\ j) \neq \textit{EvtSys es})\ ]\!]$
$\Longrightarrow \textit{rm-evtsys esl} \in \textit{cpts-ev}$
proof −
assume *p0*: $\textit{esl}{\in}\textit{cpts-es}$
and *p1*: $\textit{length esl} > 0$
and *p2*: $\exists e.\ \textit{getspc-es}\ (\textit{esl}!0) = \textit{EvtSeq}\ e\ (\textit{EvtSys es})$
and *p3*: $\neg(\exists j.\ \textit{Suc}\ j < \textit{length esl} \wedge \textit{getspc-es}\ (\textit{esl}!j) = \textit{EvtSys es} \wedge \textit{getspc-es}\ (\textit{esl}!\textit{Suc}\ j) \neq \textit{EvtSys es})$
have $\forall \textit{esl e es}\ .\textit{esl}{\in}\textit{cpts-es} \wedge \textit{length esl} > 0 \wedge (\exists e.\ \textit{getspc-es}\ (\textit{esl}!0) = \textit{EvtSeq}\ e\ (\textit{EvtSys es})) \wedge$
$\neg(\exists j.\ \textit{Suc}\ j < \textit{length esl} \wedge \textit{getspc-es}\ (\textit{esl}!j) = \textit{EvtSys es} \wedge \textit{getspc-es}\ (\textit{esl}!\textit{Suc}\ j) \neq \textit{EvtSys es})$
$\longrightarrow \textit{rm-evtsys esl} \in \textit{cpts-ev}$
proof −
{
fix *esl e es*
assume *a0*: $\textit{esl}{\in}\textit{cpts-es}$
and *a1*: $\textit{length esl} > 0$
and *a2*: $\exists e.\ \textit{getspc-es}\ (\textit{esl}!0) = \textit{EvtSeq}\ e\ (\textit{EvtSys es})$
and *a3*: $\neg(\exists j.\ \textit{Suc}\ j < \textit{length esl} \wedge \textit{getspc-es}\ (\textit{esl}!j) = \textit{EvtSys es} \wedge \textit{getspc-es}\ (\textit{esl}!\textit{Suc}\ j) \neq \textit{EvtSys es})$
from *a0 a1 a2 a3* **have** $\textit{rm-evtsys esl} \in \textit{cpts-ev}$
proof(*induct esl*)
case (*CptsEsOne es1 s x*)
show *?case*
proof(*induct es1*)
case (*EvtSeq x1 es1*)
have $\textit{rm-evtsys}\ [(\textit{EvtSeq x1 es1},\ s,\ x)] = [(\textit{x1},\ s,\ x)]$
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 $\textit{rm-evtsys}\ [(\textit{EvtSys xa},\ s,\ x)] = [(\textit{AnonyEvent None},\ s,\ x)]$
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*: $(\textit{es1},\ t,\ x)\ \#\ \textit{xs} \in \textit{cpts-es}$

```
    and b1: 0 < length ((es1, t, x) # xs) ⟹
              ∃e. getspc-es (((es1, t, x) # xs) ! 0) = EvtSeq e (EvtSys es) ⟹
              ¬ (∃j. Suc j < length ((es1, t, x) # xs) ∧
              getspc-es (((es1, t, x) # xs) ! j) = EvtSys es ∧
              getspc-es (((es1, t, x) # xs) ! Suc j) ≠ EvtSys es) ⟹
                rm-evtsys ((es1, t, x) # xs) ∈ cpts-ev
    and b2: 0 < length ((es1, s, y) # (es1, t, x) # xs)
    and b3: ∃e. getspc-es (((es1, s, y) # (es1, t, x) # xs) ! 0) = EvtSeq e (EvtSys es)
    and b4: ¬ (∃j. Suc j < length ((es1, s, y) # (es1, t, x) # xs) ∧
                    getspc-es (((es1, s, y) # (es1, t, x) # xs) ! j) = EvtSys es ∧
                    getspc-es (((es1, s, y) # (es1, t, x) # xs) ! Suc j) ≠ EvtSys es)
  from b4 have ¬ (∃j. Suc j < length ((es1, t, x) # xs) ∧
                    getspc-es (((es1, t, x) # xs) ! j) = EvtSys es ∧
                    getspc-es (((es1, t, x) # xs) ! Suc j) ≠ EvtSys es) by force
  moreover have ∃e. getspc-es (((es1, t, x) # xs) ! 0) = EvtSeq e (EvtSys es)
    proof –
      from b3 obtain e where getspc-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-evtsys ((es1, t, x) # xs) ∈ cpts-ev using b1 b3 by blast
  then have b4: rm-evtsys1 (es1, t, x) # rm-evtsys xs ∈ cpts-ev by (simp add:rm-evtsys-def)
  have b5: rm-evtsys ((es1, s, y) # (es1, t, x) # xs) =
          rm-evtsys1 (es1, s, y) # rm-evtsys1 (es1, t, x) # rm-evtsys xs
      by (simp add:rm-evtsys-def)
  from b4 show ?case
    proof(induct es1)
      case(EvtSeq x1 es2)
      assume c0: rm-evtsys1 (EvtSeq x1 es2, t, x) # rm-evtsys xs ∈ cpts-ev
      have rm-evtsys ((EvtSeq x1 es2, s, y) # (EvtSeq x1 es2, t, x) # xs) =
              (x1,s,y) # (x1, 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 (x1, t, x) # rm-evtsys xs ∈ cpts-ev
        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-evtsys1 (EvtSys xa, t, x) # rm-evtsys xs ∈ cpts-ev
      have rm-evtsys ((EvtSys xa, s, y) # (EvtSys xa, t, x) # xs) =
              (AnonyEvent None, s, y) # (AnonyEvent None, 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 None,t, x) # rm-evtsys xs ∈ cpts-ev
        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: (e1, s1, x1) −es−et→ (e2, t1, y1)
    and b1: (e2, t1, y1) # xs1 ∈ cpts-es
    and b2: 0 < length ((e2, t1, y1) # xs1) ⟹
              ∃e. getspc-es (((e2, t1, y1) # xs1) ! 0) = EvtSeq e (EvtSys es) ⟹
              ¬ (∃j. Suc j < length ((e2, t1, y1) # xs1) ∧
                      getspc-es (((e2, t1, y1) # xs1) ! j) = EvtSys es ∧
                      getspc-es (((e2, t1, y1) # xs1) ! Suc j) ≠ EvtSys es) ⟹
                        rm-evtsys ((e2, t1, y1) # xs1) ∈ cpts-ev
    and b3: 0 < length ((e1, s1, x1) # (e2, t1, y1) # xs1)
    and b4: ∃e. getspc-es (((e1, s1, x1) # (e2, t1, y1) # xs1) ! 0) = EvtSeq e (EvtSys es)
```

```
  and b5: ¬ (∃ j. Suc j < length ((e1, s1, x1) # (e2, t1, y1) # xs1) ∧
                  getspc-es (((e1, s1, x1) # (e2, t1, y1) # xs1) ! j) = EvtSys es ∧
                  getspc-es (((e1, s1, x1) # (e2, t1, y1) # xs1) ! Suc j) ≠ 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 ∃ e. e2 = EvtSeq e (EvtSys es))
    assume c0: ∃ e. e2 = EvtSeq e (EvtSys es)
    then obtain e where c1: e2 = EvtSeq e (EvtSys es) by auto
    then have c2: ∃ e. getspc-es (((e2, t1, y1) # xs1) ! 0) = EvtSeq e (EvtSys es)
      by (simp add:getspc-es-def)
    moreover from b5 have ¬ (∃ j. Suc j < length ((e2, t1, y1) # xs1) ∧
                    getspc-es (((e2, t1, y1) # xs1) ! j) = EvtSys es ∧
                    getspc-es (((e2, t1, y1) # xs1) ! Suc j) ≠ EvtSys es) by force
    ultimately have c3: rm-evtsys ((e2, t1, y1) # xs1) ∈ cpts-ev using b2 by blast
    then have c5: rm-evtsys1 (e2, t1, y1) # rm-evtsys xs1 ∈ cpts-ev by (simp add:rm-evtsys-def)

    from b0 c1 b7 have ∃ t. (e′, s1, x1) −et−t→ (e, t1, y1)
      using evtseq-tran-exist-etran by simp
    then obtain t where c8: (e′, s1, x1) −et−t→ (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: ¬(∃ 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 ∈ cpts-ev
      proof −
        from b5 have d0: ¬ (∃ j. Suc j < length ((e2, t1, y1) # xs1) ∧
              getspc-es (((e2, t1, y1) # xs1) ! j) = EvtSys es ∧
              getspc-es (((e2, t1, y1) # xs1) ! Suc j) ≠ EvtSys es) by force
        have d00: ∀ j. j < length xs1 ⟶ 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 ⟹ 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: ∀ j. j < length (rm-evtsys xs1) ⟶ getspc-e ((rm-evtsys xs1)!j) = AnonyEvent None
          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 None*, *t1*, *y1*)
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def getspc-e-def*)
with *d1* **have** ∀ *i*. *i* < *length* (*rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys xs1*) ⟶
getspc-e ((*rm-evtsys1* (*e2*, *t1*, *y1*) # *rm-evtsys xs1*)!*i*) = *AnonyEvent None*
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 None*, *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** ∃ *t*. (*e′*, *s1*, *x1*) −*et*−*t*→ (*AnonyEvent None*, *t1*, *y1*)
using *evtseq-tran-0-exist-etran* **by** *simp*
then obtain *t* **where** (*e′*, *s1*, *x1*) −*et*−*t*→ (*AnonyEvent None*, *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*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
(*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*
proof −
assume *p0*: *esl*∈*cpts-es*
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: (*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* **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* **using** *rm-evtsys-in-cptse0* **by** *blast*

with *a1 a2 a3 a4* **have** *a5*: *rm-evtsys ?esl1* ∈ *cpts-ev* **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** (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*ev*, *s1*, *x1*)
using *evtsysent-evtent*[*of 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; 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⟧ ⟹
    ∀i. i > 0 ∧ i < length el ⟶
        (getspc-es (esl!i) = EvtSys es ∧ getspc-e (el!i) = AnonyEvent None)
          ∨ (getspc-es (esl!i) = EvtSeq (getspc-e (el!i)) (EvtSys es))
proof −
  assume p0: esl∈cpts-es
    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
  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 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 None)
          ∨ (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 None)
          ∨ (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 None ∨
                      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 None ∨
                      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 None)
              assume d0: getspc-es (?esl1 ! j) = EvtSys es ∧
                      getspc-e (?el1 ! j) = AnonyEvent None
              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 None
      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 None)
    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 None, 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 None
          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∈cpts-es; 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 e s x. el∈cpts-of-ev (BasicEvent e) s x ∧ e-sim-es esl el es e
proof –
  assume p0: esl∈cpts-es
    and p1: esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs
    and p3: ¬(∃ j. j > 0 ∧ Suc j < length esl ∧ getspc-es (esl!j) = EvtSys es
              ∧ getspc-es (esl!Suc j) ≠ EvtSys es)
  from p1 have ∃ t. (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: (EvtSys es, s, x) −es−t→ (EvtSeq ev (EvtSys es), s1,x1) by auto
  then have ∃ evt e. evt ∈ es ∧ evt = BasicEvent e ∧ Act t = EvtEnt (BasicEvent e) ∧
        (BasicEvent e, s, x) −et−t→ (ev, s1, x1) using evtsysent-evtent0 by fastforce
  then obtain evt and e where a2: evt ∈ es ∧ evt = BasicEvent e ∧ Act t = EvtEnt (BasicEvent e) ∧
        (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 (BasicEvent e, s, x) −et−(at♯ak)→ (ev, s1, x1)
    using get-actk-def by (metis actk.cases)
  with p0 p1 p3 a1 a2 have a4: ?el ∈ cpts-ev
    using rm-evtsys-in-cptse [of esl 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: ∀ 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 $i \neq 0$
  with *p1 p3 a3 c1* **show** *?thesis* **by** (*simp add*: *same-s-x-def*)
qed
}
then show *?thesis* **by** *auto*
qed
qed
moreover
have $\forall i.\ i > 0 \wedge i < length\ ?el \longrightarrow$
  $(getspc\text{-}es\ (esl!i) = EvtSys\ es \wedge getspc\text{-}e\ (?el!i) = AnonyEvent\ None)$
  $\vee\ (getspc\text{-}es\ (esl!i) = EvtSeq\ (getspc\text{-}e\ (?el!i))\ (EvtSys\ es))$
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*:
$[\![esl \in cpts\text{-}es;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$(EvtSys\ es,\ s,\ x)\ -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1);$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es);$
$el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs);\ el \in cpts\text{-}ev]\!] \Longrightarrow$
$e\text{-}sim\text{-}es\ esl\ el\ es\ e$
proof –
assume *p0*: $esl \in cpts\text{-}es$
and *p1*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p2*: $(EvtSys\ es,\ s,\ x)\ -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)$
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*: $el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs)$
and *p5*: $el \in cpts\text{-}ev$
from *p2* **have** *a2*: $(BasicEvent\ e,\ s,\ x)\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ (ev,\ s1,\ x1)$
using *evtsysent-evtent*[*of es s x e k ev s1 x1*] **by** *auto*
let $?esl1 = (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
let $?el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ?esl1$
let $?el1 = rm\text{-}evtsys\ ?esl1$
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*: $?el \in cpts\text{-}ev$
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*: $getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es$ **by** (*simp add*:*getspc-es-def*)
moreover
have *b3*: $getspc\text{-}e\ (?el\ !\ 0) = BasicEvent\ e$ **by** (*simp add*:*getspc-e-def*)
moreover
from *a3 b1* **have** *b4*: $\forall i.\ i < length\ ?el \longrightarrow$
  $gets\text{-}e\ (?el\ !\ i) = gets\text{-}es\ (esl\ !\ i)\ \wedge$
  $getx\text{-}e\ (?el\ !\ i) = getx\text{-}es\ (esl\ !\ i)$
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 None)
            ∨ (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; esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs;
    (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(pre,rely)⟧
    ⟹ el∈assume-e(pre,rely)
 proof −
  assume p0: esl∈cpts-es
    and  p1: esl = (EvtSys es, s, x) # (EvtSeq ev (EvtSys es), s1,x1) # xs
    and  p2: (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(pre,rely)
  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

  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 ⟶
       esl!i −ese→ esl!(Suc i) ⟶ (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 ⟶ el!i −ee→ el!(Suc i)
      ⟶ (gets-e (el!i), gets-e (el!Suc i)) ∈ rely
  proof −
  {
    fix i
    assume c0: Suc i<length el
      and  c1: el!i −ee→ el!(Suc i)
    with a2 have ¬(el!0 −ee→ el!1)
        by (metis One-nat-def eetran.simps evtsysent-evtent0
            no-tran2basic0 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 None)
                      ∨ (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 None)
               ∨ (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 None)
        assume d0: getspc-es (esl!i) = EvtSys es ∧ getspc-e (el!i) = AnonyEvent None
        with c2 p3-1 c0 a3 have getspc-es (esl!Suc i) = EvtSys es by auto
        with d0 have 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 None)
        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 None)
            assume e0: ?ei = AnonyEvent None
            with c1 have e1: getspc-e (el!Suc i) = AnonyEvent None
              using eetran-eqconf1 by fastforce
            show ?thesis
              proof(cases getspc-es (esl!Suc i) = EvtSys es ∧ getspc-e (el!Suc i) = AnonyEvent None)
                assume f0: getspc-es (esl!Suc i) = EvtSys es ∧ getspc-e (el!Suc i) = AnonyEvent None
                with d0 have getspc-e (el!i) ≠ AnonyEvent None
                  proof −
                    let ?esl' = drop i esl
                    from p0 have ?esl'∈cpts-es
                      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 None*)
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 *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 None*
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 *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*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
(*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*(*guar*,*post*)⟧
⟹ *esl*∈*commit-es*(*guar*,*post*)
proof −
assume *p0*: *esl*∈*cpts-es*
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: (*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*(*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* **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*. *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(guar,post)
  proof −
    have ∀ i. Suc i<length esl ⟶ (∃ t. 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. 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 (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 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: (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. (e, si, xi) −et−t→ (AnonyEvent (None), si',xi')
                      by (simp add: evtseq-tran-0-exist-etran getspc-es-def)
                    then obtain et where f1: (e, si, xi) −et−et→ (AnonyEvent (None), 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 less-Suc-eq-0-disj list.simps(9) nth-Cons-Suc nth-map snd-conv)
                    moreover
```

from *p1 p4 a3 c0 d0 e2 f0* **have** *f3*:*el*!*Suc i* = (*AnonyEvent* (*None*), *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** ∃ *t*. (*e*, *si*, *xi*) −*et*−*t*→ (*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 less-Suc-eq-0-disj list.simps*(*9*) *nth-Cons-Suc nth-map 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'*)∈*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*∈*cpts-es*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
(*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∈*assume-e*(*pre*,*rely*) ⟶ *el*∈*commit-e*(*guar*,*post*) ⟧
⟹ *esl*∈*assume-es*(*pre*,*rely*) ⟶ *esl*∈*commit-es*(*guar*,*post*)
proof −
assume *p0*: *esl*∈*cpts-es*
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: (*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*∈*assume-e*(*pre*,*rely*) ⟶ *el*∈*commit-e*(*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* **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*(*pre*,*rely*)
with *p0 p1 p2 p3 p4 a1* **have** *b2*: *el*∈*assume-e*(*pre*,*rely*) **using** *e-sim-es-same-assume* **by** *metis*
with *p5* **have** *b3*: *el*∈*commit-e*(*guar*,*post*) **by** *simp*
with *p0 p1 p2 p3 p4 a1* **show** *esl*∈*commit-es*(*guar*,*post*) **using** *e-sim-es-same-commit* **by** *metis*
qed
qed

lemma *EventSys-sound-aux1*:
⟦∀ *ef*∈*es*. ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es*;*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*(*Pre m*,*Rely m*) ⟶ *esl*∈*commit-es*(*Guar m*,*Post m*))
∧ (∃ *k*. *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
proof −
assume *p0*: ∀ *ef*∈*es*. ⊨ *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*

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** ∃ *e k*. (*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** *c4*:
(*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
by *auto*
let *?el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)

from *c1 c3 c4 c41* **have** *c5*: *?el* ∈ *cpts-ev* **using** *rm-evtsys-in-cptse* **by** *metis*
from *c4* **have** ∃ *ei*∈*es*. *ei* = *BasicEvent e* **using** *evtsysent-evtent* **by** *metis*
then obtain *ei* **where** *c6*: *ei*∈*es* ∧ *ei* = *BasicEvent e* **by** *auto*
from *c3 c4 c6* **have** *c61*: *esl*!*0*−*es*−(*EvtEnt ei*)♯*k*→*esl*!*1* **by** *simp*
have *c8*: *?el*∈*assume-e*(*Pre ei*, *Rely ei*) ⟶ *?el*∈*commit-e*(*Guar ei*,*Post ei*)
proof
assume *d0*: *?el*∈*assume-e*(*Pre ei*, *Rely ei*)
moreover
from *p0 c6* **have** *d1*: ⊨ *ei* sat_e [*Pre ei*, *Rely ei*, *Guar ei*, *Post ei*] **by** *auto*
moreover
from *c5* **have** *?el*∈*cpts-of-ev* (*BasicEvent e*) *s x* **by** (*simp add*:*cpts-of-ev-def*)
ultimately show *?el*∈*commit-e*(*Guar ei*,*Post ei*) **using** *evt-validity-def c6*
by *fastforce*
qed
with *c1 c3 c4 c41* **have** *c7*: *esl*∈*assume-es*(*Pre ei*, *Rely ei*) ⟶ *esl*∈*commit-es*(*Guar ei*,*Post ei*)
using *rm-evtsys-assum-comm* **by** *metis*
then show *?thesis* **using** *c6 c61* **by** *blast*
qed

lemma *EventSys-sound-aux1-forall*:
⟦∀ *ef*∈*es*. ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es*;*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*. (∃ *k*. *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
⟶ (*esl*∈*assume-es*(*Pre m*,*Rely m*) ⟶ *esl*∈*commit-es*(*Guar m*,*Post m*))
proof −
assume *p0*: ∀ *ef*∈*es*. ⊨ *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*
then show *?thesis*
proof −
{
fix *m*
assume *c01*: *m*∈*es*
and *c02*: ∃ *k*. *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*. (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt m*)♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *simp*
then obtain *k* **where** *c4*: (*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* **using** *rm-evtsys-in-cptse* **by** *metis*

from *c3 c4 c40* **have** *c61*: *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1* **by** *simp*
have *c8*: *?el*∈*assume-e*(*Pre m*, *Rely m*) ⟶ *?el*∈*commit-e*(*Guar m*,*Post m*)
proof
assume *d0*: *?el*∈*assume-e*(*Pre m*, *Rely m*)
moreover
from *p0 c01 c40* **have** *d1*: ⊨ *m* sat_e [*Pre m*, *Rely m*, *Guar m*, *Post m*] **by** *auto*
moreover
from *c5 c40* **have** *?el*∈*cpts-of-ev* (*BasicEvent e*) *s x* **by** (*simp add*:*cpts-of-ev-def*)
ultimately show *?el*∈*commit-e*(*Guar m*,*Post m*) **using** *evt-validity-def c40*
by *fastforce*
qed
with *c1 c3 c4 c40 c41* **have** *c7*: *esl*∈*assume-es*(*Pre m*, *Rely m*) ⟶ *esl*∈*commit-es*(*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*;*length esl* ≥ *2*; *getspc-es* (*esl*!*0*) = *EvtSys es*; *getspc-es* (*esl*!*1*) ≠ *EvtSys es*⟧
⟹ ∃ *m*∈*es*. (∃ *k*. *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
proof −
assume *p0*: *esl*∈*cpts-es*
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** $\exists$ *e k*. (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
using *fst-esys-snd-eseq-exist-evtent2* **by** *fastforce*
then obtain *e* **and** *k* **where** *a3*:
(*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
by *auto*
from *a3* **have** $\exists$ *i*$\in$*es*. *i* = *BasicEvent e* **using** *evtsysent-evtent* **by** *metis*
then obtain *ei* **where** *c6*: *ei*$\in$ *es* $\wedge$ *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*:
$[\![\forall$ *ef*$\in$*es*. $\models$ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl$\in$*cpts-es*;*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*)$]\!]$
$\Longrightarrow$ $\forall$ *ei*$\in$*es*. ($\exists$ *k*. *esl*!*0*$-es-$(*EvtEnt ei*)$\sharp k\rightarrow$*esl*!*1*)
$\longrightarrow$ (*esl*$\in$*assume-es*(*Pre ei*,*Rely ei*) $\longrightarrow$ *esl*$\in$*commit-es*(*Guar ei*,*Post ei*)
$\wedge$ *gets-es* (*last esl*) $\in$ *Post ei*)
proof $-$
assume *p0*: $\forall$ *ef*$\in$*es*. $\models$ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *a0*: *length esl* $\geq$ *2* $\wedge$ *getspc-es* (*esl*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*1*) $\neq$ *EvtSys es*
and *p6*: *getspc-es* (*last esl*) = *EvtSys es*
and *c41*: $\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*)
and *c1*: *esl*$\in$*cpts-es*
then show *?thesis*
proof$-$
{
fix *ei*
assume *c01*: *ei*$\in$*es*
and *c02*: $\exists$ *k*. *esl*!*0*$-es-$(*EvtEnt ei*)$\sharp k\rightarrow$*esl*!*1*

from *a0 c1* **have** *c2*: $\exists$ *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*. (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt ei*)$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *simp*
then obtain *k* **where** *c4*: (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt ei*)$\sharp k\rightarrow$ (*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* **using** *rm-evtsys-in-cptse* **by** *metis*

from *c3 c4 c6* **have** *c61*: *esl*!*0*$-es-$(*EvtEnt ei*)$\sharp k\rightarrow$*esl*!*1* **by** *simp*
have *c8*: *?el*$\in$*assume-e*(*Pre ei*, *Rely ei*) $\longrightarrow$ *?el*$\in$*commit-e*(*Guar ei*,*Post ei*)
proof
assume *d0*: *?el*$\in$*assume-e*(*Pre ei*, *Rely ei*)
moreover
from *p0 c01 c6* **have** *d1*: $\models$ *ei sat*$_e$ [*Pre ei*, *Rely ei*, *Guar ei*, *Post ei*] **by** *auto*
moreover
from *c5 c6* **have** *?el*$\in$*cpts-of-ev* (*BasicEvent e*) *s x* **by** (*simp add*:*cpts-of-ev-def*)
ultimately show *?el*$\in$*commit-e*(*Guar ei*,*Post ei*) **using** *evt-validity-def c6*
by *fastforce*
qed
with *c1 c3 c4 c41 c6* **have** *c7*: *esl*$\in$*assume-es*(*Pre ei*, *Rely ei*) $\longrightarrow$ *esl*$\in$*commit-es*(*Guar ei*,*Post ei*)
using *rm-evtsys-assum-comm* **by** *metis*
moreover
have *esl*$\in$*assume-es*(*Pre ei*, *Rely ei*) $\longrightarrow$ *gets-es* (*last esl*) $\in$ *Post ei*

```
        proof
          assume d0: esl∈assume-es(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(Pre ei, Rely ei)
            using e-sim-es-same-assume by metis
          with c8 have d1: ?el∈commit-e(Guar ei,Post ei) by auto

          have d4: getspc-e (last ?el) = AnonyEvent None
            proof −
              from a0 d2 have e1: length ?el = length esl by (simp add: e-sim-es-def)
              with d2 have ∀i. i > 0 ∧ i < length ?el ⟶
                                    (getspc-es (esl!i) = EvtSys es ∧ getspc-e (?el!i) = AnonyEvent None)
                                     ∨ (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 ∧ getspc-e (last ?el) = AnonyEvent None)
                                     ∨ (getspc-es (last esl) = EvtSeq (getspc-e (last ?el)) (EvtSys es))
                by (metis (no-types, hide-lams) c3 last-length length-Cons length-tl lessI list.sel(3) zero-less-Suc)
              with p6 show ?thesis by simp
            qed
          with d1 have gets-e (last ?el) ∈ 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 ∀i. i < length ?el ⟶ gets-e (?el ! i) = gets-es (esl ! i) ∧
                                                      getx-e (?el ! i) = getx-es (esl ! i)
                by (simp add: e-sim-es-def)
              with a0 e1 show ?thesis by (metis (no-types, hide-lams) c3 last-length
                    length-Cons length-tl lessI list.sel(3))
            qed
          ultimately show gets-es (last esl) ∈ Post ei by simp
        qed

      ultimately have (esl∈assume-es(Pre ei,Rely ei) ⟶ esl∈commit-es(Guar ei,Post ei)
                              ∧ gets-es (last esl) ∈ Post ei) by simp
    }
    then show ?thesis by auto
    qed
  qed

lemma EventSys-sound-seg-aux0:
    ⟦∀ef∈es. ⊨ ef sat_e [Pre ef, Rely ef, Guar ef, Post ef];
     esl∈cpts-es;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)⟧
      ⟹ ∃m∈es. (esl∈assume-es(Pre m,Rely m) ⟶ esl∈commit-es(Guar m,Post m)
                          ∧ gets-es (last esl) ∈ Post m)
                 ∧ (∃k. esl!0−es−(EvtEnt m)♯k→esl!1)
  proof −
    assume p0: ∀ef∈es. ⊨ ef sat_e [Pre ef, Rely ef, Guar ef, Post ef]
      and p1: length esl ≥ 2 ∧ getspc-es (esl!0) = EvtSys es ∧ getspc-es (esl!1) ≠ EvtSys es
      and p2: getspc-es (last esl) = EvtSys es
      and p3: ¬(∃j. j > 0 ∧ Suc j < length esl ∧ getspc-es (esl!j) = EvtSys es ∧ getspc-es (esl!Suc j) ≠ EvtSys es)
      and p4: esl∈cpts-es
    then have ∃m∈es. (∃k. esl!0−es−(EvtEnt m)♯k→esl!1)
      using EventSys-sound-seg-aux0-exist[of esl es] by simp
    then obtain m where a1: m∈ es ∧ (∃k. esl!0−es−(EvtEnt m)♯k→esl!1) by auto
    with p0 p1 p2 p3 p4 have (esl∈assume-es(Pre m,Rely m) ⟶ esl∈commit-es(Guar m,Post m)
```

$\wedge$ *gets-es* (*last esl*) $\in$ *Post m*)
using *EventSys-sound-seg-aux0-forall* [*of es Pre Rely Guar Post esl*] **by** *simp*
with *a1* **show** *?thesis* **by** *auto*
qed

lemma *EventSys-sound-aux-i-forall*:
$[\![\forall ef{\in}es.\ \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall ef{\in}es.\ pre \subseteq Pre\ ef;\ \forall ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall ef{\in}es.\ Post\ ef \subseteq post;$
$\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$esl{\in}cpts\text{-}es;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$esl{\in}assume\text{-}es(pre,rely);$
$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])]\!]$
$\Longrightarrow \forall i.\ Suc\ i < length\ elst \longrightarrow$
$(\forall ei{\in}es.\ (\exists k.\ (elst!i@[(elst!Suc\ i)!0])!0{-}es{-}(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es(Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei)$
proof −
assume *p0*: $\forall ef{\in}es.\ \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
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*: $esl{\in}cpts\text{-}es$
and *p9*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p10*: $esl{\in}assume\text{-}es(pre,rely)$
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 $\forall ei{\in}es.\ (\exists k.\ (elst!i@[(elst!Suc\ i)!0])!0{-}es{-}(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es(Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei$
proof(*induct i*)
case *0*
assume *c0*: $Suc\ 0 < length\ elst$
let $?els = elst\ !\ 0\ @\ [elst\ !\ Suc\ 0\ !\ 0]$
have *c1*: $?els \in cpts\text{-}es$
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** $\exists m\ n.\ m \leq length\ esl \wedge n \leq length\ esl \wedge m \leq n \wedge ?els = take\ (n - m)\ (drop\ m\ esl)$
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 c0*)
have *c3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length ?els* ∧ *getspc-es* (*?els*!*j*) = *EvtSys es*
∧ *getspc-es* (*?els*!*Suc j*) ≠ *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* ≤ *length ?els* ∧ *getspc-es* (*?els* ! *0*) = *EvtSys es* ∧ *getspc-es* (*?els* ! *1*) ≠ *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*: ∀ *ei*∈*es*. (∃ *k*. *?els*!*0*−*es*−(*EvtEnt ei*)♯*k*→*?els*!*1*)
⟶ (*?els*∈*assume-es*(*Pre ei*,*Rely ei*) ⟶ *?els*∈*commit-es*(*Guar ei*,*Post ei*)
∧ *gets-es* (*last ?els*) ∈ *Post ei*)
using *EventSys-sound-seg-aux0-forall*[*of es Pre Rely Guar Post ?els*] **by** *auto*

from *p10 a2* **have** *?els*∈*assume-es*(*pre*,*rely*)
proof −
from *a0* **have** *d1*: ∀ *i*<*length elst*. *elst* ! *i* ≠ []
using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 c0* **have** ∃ *m n*. *m* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n* ∧ *?els* = *take* (*n* − *m*) (*drop m esl*)
using *concat-i-lm* **by** *blast*
moreover
from *p10* **have** ∀ *i*. *Suc i*<*length esl* ⟶ *esl*!*i* −*ese*→ *esl*!(*Suc i*) ⟶
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have ∀ *i*. *Suc i*<*length ?els* ⟶ *?els*!*i* −*ese*→ *?els*!(*Suc i*) ⟶
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) ∈ *rely*
using *rely-takedrop-rely* **by** *blast*
moreover
have *gets-es* (*?els*!*0*) ∈ *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*) ∈ *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** ∀ *ei*∈*es*. *?els*∈*assume-es*(*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*
⟹ ∀ *ei*∈*es*. (∃ *k*. *?elstjj* ! *0* −*es*−*EvtEnt ei*♯*k*→ *?elstjj* ! *1*) ⟶
?elstjj ∈ *commit-es* (*Guar ei*, *Post ei*) ∧ *gets-es* (*elst* ! *Suc j* ! *0*) ∈ *Post ei*
and *c02*: *Suc* (*Suc j*) < *length elst*
then show *?case*
proof−
{

fix *ei*
assume *d0*: *ei*∈*es*
  and *d1*: ∃*k*. *?els* ! *0* −*es*−*EvtEnt ei*♯*k*→ *?els* ! *1*

from *c02 a0*[*of j*] **have** ∃*m*∈*es*. (∃*k*. *?elstjj*!*0*−*es*−(*EvtEnt m*)♯*k*→*?elstjj*!*1*)
  using *EventSys-sound-seg-aux0-exist*[*of ?elstjj 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′*∈*es* ∧ (∃*k*. *?elstjj*!*0*−*es*−(*EvtEnt ei′*)♯*k*→*?elstjj*!*1*)
  by *auto*
with *c01 c02* **have** *c04*: *?elstjj* ∈ *commit-es* (*Guar ei′*, *Post ei′*)
      ∧ *gets-es* (*elst* ! *Suc j* ! *0*) ∈ *Post ei′*
  by *auto*

have *c1*: *?els* ∈ *cpts-es*
  proof −
    from *a0* **have** *c11*: ∀*i*<*length elst*. *elst* ! *i* ≠ []
      using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
    with *a2 c02* **have** ∃*m n*. *m* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n* ∧ *?els* = *take* (*n* − *m*) (*drop m*
esl)
      using *concat-i-lm* **by** *blast*
    then obtain *m* **and** *n* **where** *d1*: *m* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n*
        ∧ *?els* = *take* (*n* − *m*) (*drop m esl*) **by** *auto*
    have *?els* ≠ [] **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*: ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length ?els* ∧ *getspc-es* (*?els*!*j*) = *EvtSys es*
  ∧ *getspc-es* (*?els*!*Suc j*) ≠ *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* ≤ *length ?els* ∧ *getspc-es* (*?els* ! *0*) = *EvtSys es* ∧ *getspc-es* (*?els* ! *1*) ≠ *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*∈*assume-es*(*Pre ei*,*Rely ei*) ⟶ *?els*∈*commit-es*(*Guar ei*,*Post ei*)
      ∧ *gets-es* (*last ?els*) ∈ *Post ei*)
  using *EventSys-sound-seg-aux0-forall*[*of es Pre Rely Guar Post ?els*] **by** *blast*
from *p10 a2* **have** *?els*∈*assume-es*(*Pre ei*,*rely*)
  proof −
    from *a0* **have** *d1*: ∀*i*<*length elst*. *elst* ! *i* ≠ []
      using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
    with *a2 c02* **have** ∃*m n*. *m* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n* ∧ *?els* = *take* (*n* − *m*) (*drop m*
esl)
      using *concat-i-lm* **by** *blast*
    moreover
    from *p10* **have** ∀*i*. *Suc i*<*length esl* ⟶ *esl*!*i* −*ese*→ *esl*!(*Suc i*) ⟶
        (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
    ultimately have ∀*i*. *Suc i*<*length ?els* ⟶ *?els*!*i* −*ese*→ *?els*!(*Suc i*) ⟶
        (*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) ∈ *rely*
        using *rely-takedrop-rely* **by** *blast*
    moreover
    have *gets-es* (*?els*!*0*) ∈ *Pre ei*

proof −
from *p5*[*of ei′ ei*] *d0 c03 c04* **have** *gets-es* (*elst* ! *Suc j* ! *0*) ∈ *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*∈*assume-es*(*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*(*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*. ⊨ *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*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es*(*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*(*Guar m*,*Post m*)
∧ *gets-es* ((*elst*!*Suc i*)!*0*) ∈ *Post m*
∧ (∃ *k*. (*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*0*−*es*−(*EvtEnt m*)♯*k*→(*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*1*))
proof −
assume *p0*: ∀ *ef*∈*es*. ⊨ *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*
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*∈*assume-es*(*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** ∃ m∈es. (∃ k. elst!i!0 −es−(EvtEnt m)♯k→ elst!i!1)
using EventSys-sound-seg-aux0-exist[of elst!i@[(elst!Suc i)!0] es]
parse-es-cpts-i2-in-cptes-i[of esl es s x e s1 x1 xs 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∈es ∧ (∃ k. elst!i!0 −es−(EvtEnt m)♯k→ elst!i!1) **by** auto
with p0 p1 p2 p3 p4 p5 p8 p9 p10 p11 b0
have b2[rule-format]: ∀ i. Suc i < length elst ⟶ (∀ ei∈es.
(∃ k. (elst ! i @ [elst ! Suc i ! 0]) ! 0 −es−EvtEnt ei♯k→ (elst ! i @ [elst ! Suc i ! 0]) ! 1) ⟶
elst ! i @ [elst ! Suc i ! 0] ∈ commit-es (Guar ei, Post ei) ∧ gets-es (elst ! Suc i ! 0) ∈ Post ei)
using EventSys-sound-aux-i-forall[of es 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]∈commit-es(Guar m,Post m)
∧ gets-es ((elst!Suc i)!0) ∈ 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** ∃ m∈es. elst!i@[(elst!Suc i)!0]∈commit-es(Guar m,Post m)
∧ gets-es ((elst!Suc i)!0) ∈ Post m
∧ (∃ k. (elst!i@[(elst!Suc i)!0])!0 −es−(EvtEnt m)♯k→ (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:
⟦∀ ef∈es. ⊨ 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; esl = (EvtSys es, s, x) # (EvtSeq e (EvtSys es), s1,x1) # xs;
esl∈assume-es(pre,rely);
elst = tl (parse-es-cpts-i2 esl es [[]])⟧
⟹ ∀ ei∈es. (∃ k. (last elst)!0 −es−(EvtEnt ei)♯k→ (last elst)!1)
⟶ last elst∈commit-es(Guar ei,Post ei)
proof −
assume p0: ∀ ef∈es. ⊨ 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
and p9: esl = (EvtSys es, s, x) # (EvtSeq e (EvtSys es), s1,x1) # xs
and p10: esl∈assume-es(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
with p9 **have** a3: elst ≠ [] **by** auto
show ?thesis

proof –
{
fix *ei*
assume *a01*: *ei*∈*es*
and *a02*: ∃ *k*. (*last elst*)!*0*−*es*−(*EvtEnt ei*)♯*k*→(*last elst*)!*1*
have *last elst*∈*commit-es*(*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* ∈ *cpts-es* **using** *b1 a3 last-conv-nth* **by** *fastforce*

from *a1 b0* **have** *c3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length ?els* ∧ *getspc-es* (*?els*!*j*) = *EvtSys es*
∧ *getspc-es* (*?els*!*Suc j*) ≠ *EvtSys es*) **by** *simp*
from *a0 b0* **have** *c4*: *2* ≤ *length ?els* ∧ *getspc-es* (*?els* ! *0*) = *EvtSys es* ∧ *getspc-es* (*?els* ! *1*) ≠ *EvtSys es*
by *simp*

with *p0 c1 c3* **have** *c5*: ∀ *m*∈*es*. (∃ *k*. *?els*!*0*−*es*−(*EvtEnt m*)♯*k*→*?els*!*1*)
⟶ (*?els*∈*assume-es*(*Pre m*,*Rely m*) ⟶ *?els*∈*commit-es*(*Guar m*,*Post m*))
using *EventSys-sound-aux1-forall*[*of es Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** *?els*∈*assume-es*(*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* ⟶ *esl*!*i* −*ese*→ *esl*!(*Suc i*) ⟶
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have ∀ *i*. *Suc i*<*length ?els* ⟶ *?els*!*i* −*ese*→ *?els*!(*Suc i*) ⟶
(*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*(*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*(*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*
proof –
from *a2 b0* **have** ∃ *m* . *m* ≤ *length esl* ∧ *?els* = *drop m esl*

by (*simp add*: *concat-last-lm a3*)

then obtain m **where** $d1$: $m \leq$ *length esl* $\wedge$ *?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*: $\forall m \in es.$ ($\exists k.$ *?els*!*0*$-es-$(*EvtEnt m*)$\sharp k\rightarrow$*?els*!*1*)
$\longrightarrow$ (*?els*$\in$*assume-es*(*Pre m*,*Rely m*) $\longrightarrow$ *?els*$\in$*commit-es*(*Guar m*,*Post m*))
using *EventSys-sound-aux1-forall*[*of es Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** *c6*: *?els*$\in$*assume-es*(*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* $\longrightarrow$ *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* $\longrightarrow$ *?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* $\longrightarrow$
($\forall ei \in es.$ ($\exists k.$ (*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*(*Guar ei*,*Post ei*)
$\wedge$ *gets-es* ((*elst*!*Suc i*)!*0*) $\in$ *Post ei*)
using *EventSys-sound-aux-i-forall*[*of es 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*
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* ∧ *getspc-es* (*?els1* ! *1*) ≠ *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 ∃ *m*∈*es*. (∃ *k*. *?els1*!*0*−*es*−(*EvtEnt m*)♯*k*→*?els1*!*1*)
using *EventSys-sound-seg-aux0-exist*[*of ?els1 es*] **by** *simp*
then obtain *m* **where** *d2*: *m*∈*es* ∧ (∃ *k*. *?els1*!*0*−*es*−(*EvtEnt m*)♯*k*→*?els1*!*1*)
by *auto*
then have *gets-es* (*elst* ! (*length elst* − *1*) ! *0*) ∈ *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*) ∈ *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* ⊆ *Rely ei* **by** (*simp add*: *a01*)
ultimately have *?els*∈*assume-es*(*Pre ei*, *Rely ei*)
using *assume-es-imp* **by** *blast*
with *c5* **have** *c6*: *?els*∈*commit-es*(*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*:
⟦∀ *ef*∈*es*. ⊨ *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*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es*(*pre*,*rely*);
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧
⟹ ∃ *m*∈*es*. *last elst*∈*commit-es*(*Guar m*,*Post m*)
∧ (∃ *k*. (*last elst*)!*0*−*es*−(*EvtEnt m*)♯*k*→(*last elst*)!*1*)
proof −
assume *p0*: ∀ *ef*∈*es*. ⊨ *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*
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*∈*assume-es*(*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*
with *p9* **have** *a3*: *elst* $\neq$ [] **by** *auto*
from *p8 p9 p11 a0*[*of length elst* − *1*] **have** $\exists$ *m*$\in$*es*. ($\exists$ *k*. *last elst*!*0*−*es*−(*EvtEnt m*)$\sharp$*k*$\rightarrow$*last elst*!*1*)
using *EventSys-sound-seg-aux0-exist*[*of last elst es*]
parse-es-cpts-i2-in-cptes-last[*of esl es s x e s1 x1 xs 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*. *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-es*(*Guar m*,*Post m*)
using *EventSys-sound-aux-last-forall*[*of es 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*. $\models$ *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-es*; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl$\in$*assume-es*(*pre*,*rely*)$]\!]$
$\Longrightarrow$ $\forall$ *i*. *Suc i*<*length esl* $\longrightarrow$ ($\exists$ *t*. *esl*!*i* −*es*−*t*$\rightarrow$ *esl*!(*Suc i*)) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *guar*
proof −
assume *p0*: $\forall$ *ef*$\in$*es*. $\models$ *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-es*
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*$\in$*assume-es*(*pre*,*rely*)
let *?elst* = *tl* (*parse-es-cpts-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-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 *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-es*(*Guar m*,*Post m*)
$\wedge$ *gets-es* ((*?elst*!*Suc i*)!*0*) $\in$ *Post m*)
using *EventSys-sound-aux-i*
[*of es 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-es*(*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-es*(*Guar m*,*Post m*)
using *EventSys-sound-aux-last*
[*of es 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-es*(*Guar m*,*Post m*) **by** *auto*
show *?thesis*

proof −
{
fix *i*
assume *b0*: *Suc i < length esl*
and *b1*: ∃ *t. 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. (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** ∀ *i. Suc i<length (last ?elst) ⟶ (∃ t. (last ?elst)!i −es−t→ (last ?elst)!(Suc i))*

$\longrightarrow$ (*gets-es* ((*last ?elst*)!*i*), *gets-es* ((*last ?elst*)!*Suc i*)) $\in$ *Guar m*
by (*simp add*: *commit-es-def*)
with *b1 e1 e2* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *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* $\neq$ *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*$\in$*es*. *?els'*$\in$*commit-es*(*Guar m*,*Post m*)
by *blast*
then obtain *m* **where** *e1*: *m*$\in$*es* $\wedge$ *?els'*$\in$*commit-es*(*Guar m*,*Post m*)
by *auto*
then have *e2*: $\forall$ *i*. *Suc i*<*length ?els'* $\longrightarrow$ ($\exists$ *t*. *?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 *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*. *?els'*!*0* $-es-t\rightarrow$ *?els'*!*1* **by** *simp*
have *length ?els'* $>$ *1* **using** *d2* **by** *auto*
with *e2 e5* **have** (*gets-es* (*?els'*!*0*), *gets-es* (*?els'*!*1*)) $\in$ *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*(*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e7*: *m*$\in$*es* $\wedge$
?els'$\in$*commit-es*(*Guar m*,*Post m*)

by *auto*
then have *e8*: $\forall$ *i*. *Suc i*<*length ?els'* $\longrightarrow$ ($\exists$ *t*. *?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 *b1 e5 e6* **have** *e9*: $\exists$ *t*. *?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*(*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e2*: *m*$\in$*es* $\wedge$ *?els'*$\in$*commit-es*(*Guar m*,*Post m*)
by *auto*
then have *e3*: $\forall$ *i*. *Suc i*<*length ?els'* $\longrightarrow$ ($\exists$ *t*. *?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*. *?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*. $\models$ *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* $]\!]$
$\Longrightarrow$ $\models$ *EvtSys es sat$_s$* [*pre*, *rely*, *guar*, *post*]
proof $-$
assume *p0*: $\forall$ *ef*$\in$*es*. $\models$ *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: ∀ s. (s, s)∈guar
  then have ∀ s x. (cpts-of-es (EvtSys es) s x) ∩ assume-es(pre, rely) ⊆ commit-es(guar, post)
    proof−
    {
      fix s x
      have ∀ esl. esl∈(cpts-of-es (EvtSys es) s x) ∩ assume-es (pre, rely) ⟶ esl∈ commit-es (guar, post)
        proof −
        {
          fix esl
          assume a0: esl∈(cpts-of-es (EvtSys es) s x) ∩ assume-es (pre, rely)
          then have a1: esl∈(cpts-of-es (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 ∈ cpts-es by (simp add:cpts-of-es-def)
          from a0 have a2: esl∈assume-es (pre, rely) by simp
          then have ∀ i. Suc i<length esl ⟶ (∃ t. esl!i −es−t→ esl!(Suc i)) ⟶
                      (gets-es (esl!i), gets-es (esl!Suc i)) ∈ guar
            proof −
            {
              fix i
              assume b0: Suc i<length esl
                and  b1: ∃ t. esl!i −es−t→ esl!(Suc i)
              then obtain t where b2: esl!i −es−t→ esl!(Suc i) by auto
              from a1-2 b0 b1 have (gets-es (esl!i), gets-es (esl!Suc i)) ∈ guar
                proof(cases ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) = EvtSys es)
                  assume c0: ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) = EvtSys es

                  with b0 have getspc-es (esl ! i) = EvtSys es by simp
                  moreover from b0 c0 have getspc-es (esl ! (Suc i)) = EvtSys es by simp
                  ultimately have ¬(∃ t. esl!i −es−t→ 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: ¬ (∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) = EvtSys es)
                  then obtain m where c1: Suc m ≤ length esl ∧ getspc-es (esl ! m) ≠ EvtSys es
                    by auto
                  from a1-1 have c2: getspc-es (esl!0) = EvtSys es by (simp add:getspc-es-def)
                  from c1 have ∃ i. i ≤ m ∧ getspc-es (esl ! i) ≠ EvtSys es by auto
                  with a1-2 a1-1 c1 c2 have ∃ 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)
                    using evtsys-fst-ent by blast
                  then obtain n where c3: (n < m ∧ getspc-es (esl ! n) = EvtSys es
                          ∧ getspc-es (esl ! Suc n) ≠ EvtSys es)
                          ∧ (∀ j. j < n ⟶ getspc-es (esl ! j) = EvtSys es) by auto
                  with b1 have c4: i ≥ 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 ¬(∃ t. esl!i −es−t→ 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 ≥ 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 ∧ getspc-es (?esl!1) ≠ EvtSys es
  by force


from a1-2 c1 c3 have c7: ?esl ∈ cpts-es 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 ∃ 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∈assume-es(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 ≠ 0
    let ?eslh = take (n + 1) esl
    from a2 have d1: ∀ i. Suc i<length ?esl ⟶ ?esl!i −ese→ ?esl!(Suc i)
      ⟶ (gets-es (?esl!i), gets-es (?esl!Suc i)) ∈ rely by (simp add:assume-es-def)
    have gets-es (?esl!0) ∈ pre
      proof −
        from a2 d0 have gets-es (?eslh!0) ∈ pre by (simp add:assume-es-def)
        moreover
        from a2 have ∀ i. Suc i<length ?eslh ⟶ ?eslh!i −ese→ ?eslh!(Suc i)
          ⟶ (gets-es (?eslh!i), gets-es (?eslh!Suc i)) ∈ rely by (simp add:assume-es-def)
        ultimately have ?eslh ∈ assume-es(pre, rely) by (simp add:assume-es-def)
        moreover
        from c3 have ∀ 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 ∀ i<length ?eslh. gets-es (?eslh!i) ∈ pre
          using p6 pre-trans by blast
        with d0 have gets-es (?eslh ! n) ∈ 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: ∀ i. Suc i<length ?esl ⟶ (∃ t. ?esl!i −es−t→ ?esl!(Suc i)) ⟶
      (gets-es (?esl!i), gets-es (?esl!Suc i)) ∈ guar
  using EventSys-sound-0
      [of es 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** $\exists\, t.$ *?esl* ! $(i - n)$ *−es−t→ ?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* (*guar*, *post*) **by** (*simp add*:*commit-es-def*)
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *blast*
qed

then show $\models$ *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*;
$\models$ *esys sat*$_s$ [*pre'*, *rely'*, *guar'*, *post'*]$]\!]$
$\Longrightarrow$ $\models$ *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*: $\models$ *esys sat*$_s$ [*pre'*, *rely'*, *guar'*, *post'*]
from *p4* **have** *p5*: $\forall$ *s x*. (*cpts-of-es esys s x*) $\cap$ *assume-es*(*pre'*, *rely'*) $\subseteq$ *commit-es*(*guar'*, *post'*)
by (*simp add*: *es-validity-def*)
have $\forall$ *s x*. (*cpts-of-es esys s x*) $\cap$ *assume-es*(*pre*, *rely*) $\subseteq$ *commit-es*(*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: *c*$\in$(*cpts-of-es esys s x*) $\cap$ *assume-es*(*pre*, *rely*)
then have *c*$\in$(*cpts-of-es esys s x*) $\wedge$ *c*$\in$*assume-es*(*pre*, *rely*) **by** *simp*
with *p0 p1* **have** *c*$\in$(*cpts-of-es esys s x*) $\wedge$ *c*$\in$*assume-es*(*pre'*, *rely'*)
using *assume-es-imp*[*of pre pre' rely rely' c*] **by** *simp*
with *p5* **have** *c*$\in$*commit-es*(*guar'*, *post'*) **by** *auto*
with *p2 p3* **have** *c*$\in$*commit-es*(*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

theorem *rgsound-es*: $\vdash$ *esf sat*$_s$ [*pre*, *rely*, *guar*, *post*] $\Longrightarrow$ $\models$ *evtsys-spec esf sat*$_s$ [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-es.induct*)
proof −

{
fix *ef esf pre post rely guar*
assume *p0*: $\vdash E_e$ (*ef*::('*l*,'*k*,'*s*) *rgformula-e*) sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
and *p1*: $\vdash$ *fst* (*esf*::('*l*,'*k*,'*s*) *rgformula-ess* $\times$ '*s rgformula*) sat_s [Pre_f (*snd esf*), $Rely_f$ (*snd esf*), $Guar_f$ (*snd esf*), $Post_f$ (*snd esf*)]
and *p2*: $\models$ *evtsys-spec* (*fst esf*) sat_s [Pre_f (*snd esf*), $Rely_f$ (*snd esf*), $Guar_f$ (*snd esf*), $Post_f$ (*snd esf*)]
and *p3*: *pre* = Pre_e *ef*
and *p4*: *post* = $Post_f$ (*snd esf*)
and *p5*: *rely* $\subseteq$ $Rely_e$ *ef*
and *p6*: *rely* $\subseteq$ $Rely_f$ (*snd esf*)
and *p7*: $Guar_e$ *ef* $\subseteq$ *guar*
and *p8*: $Guar_f$ (*snd esf*) $\subseteq$ *guar*
and *p9*: $Post_e$ *ef* $\subseteq$ Pre_f (*snd esf*)
from *p0* **have** *a1*: $\models E_e$ (*ef*::('*l*,'*k*,'*s*) *rgformula-e*) sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
using *rgsound-e* **by** *blast*
have *a2*: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *EvtSeq* (*fst ef*) (*evtsys-spec* (*fst esf*))
using *evtsys-spec-evtseq* **by** (*simp add*:E_e-*def*)
from *p2 p3 p4 p5 p6 p7 p8 p9 a1 a2* **show** $\models$ *evtsys-spec* (*rgf-EvtSeq ef esf*) sat_s [*pre*, *rely*, *guar*, *post*]
using *EventSeq-sound* [*of fst ef pre* $Rely_e$ *ef* $Guar_e$ *ef* $Post_e$ *ef*
evtsys-spec (*fst esf*) Pre_f (*snd esf*) $Rely_f$ (*snd esf*) $Guar_f$ (*snd esf*) *post*
rely guar] **by** (*simp add*:E_e-*def*)
}
next
{
fix *esf pre rely guar post*
assume *p0*: $\forall$ *ef*$\in$*esf*. $\vdash E_e$ *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*
let *?es* = *Domain esf*
let *?RG* = λ*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*. $\models E_e$ *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*. $\models$ *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 $\models$ *EvtSys* (*Domain esf*) sat_s [*pre*, *rely*, *guar*, *post*]

using *p6 p7 EventSys-sound* [*of ?es ?Pre ?Rely ?Guar ?Post pre rely guar post*] **by** *simp*
then show $\models$ *evtsys-spec* (*rgf-EvtSys esf*) sat_s [*pre*, *rely*, *guar*, *post*] **by** *simp*
}
next
{
fix *pre pre′ rely rely′ guar′ guar post′ post esys*
assume *pre* $\subseteq$ *pre′*
and *rely* $\subseteq$ *rely′*
and *guar′* $\subseteq$ *guar*
and *post′* $\subseteq$ *post*
and $\vdash$ *esys* sat_s [*pre′*, *rely′*, *guar′*, *post′*]
and $\models$ *evtsys-spec esys* sat_s [*pre′*, *rely′*, *guar′*, *post′*]
then show $\models$ *evtsys-spec esys* sat_s [*pre*, *rely*, *guar*, *post*]
using *esys-seq-sound*[*of pre pre′ rely rely′ guar′ guar post′ post evtsys-spec esys*] **by** *simp*
}
qed

7.6 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*(*Guar k*, *Post k*);
c $\in$ *cpts-of-pes pes s x*; *c*$\in$*assume-pes*(*pre*, *rely*); *c* $\propto$ *cs*⟧ $\Longrightarrow$
$\forall n\, k.$ $n \leq$ *length* (*cs k*) $\wedge$ $n > 0 \longrightarrow$ *take n* (*cs k*) $\in$ *assume-es*(*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 pes s x*
and *p5*: *c* $\in$ *assume-pes*(*pre*, *rely*)
and *p6*: *c* $\propto$ *cs*
and *p0*: $\forall k.$ *cs k* $\in$ *commit-es*(*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* (*pes k*) *s x* **using** *conjoin-imp-cptses-k* **by** *auto*
then have *p9*: $\forall k.$ *cs k* $\in$ *cpts-es* $\wedge$ *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*(*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* (*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* (*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) ∈ 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 ∀ i. Suc i < length (take (Suc m) (cs k))
        ⟶ (take (Suc m) (cs k)) ! i −ese→ (take (Suc m) (cs k)) ! Suc i
        ⟶ (gets-es ((take (Suc m) (cs k)) ! i), gets-es ((take (Suc m) (cs k)) ! Suc i)) ∈ rely
    by auto
  moreover
  have assume-es (Pre k, Rely k) = {c. gets-es (c ! 0) ∈ Pre k ∧
        (∀ i. Suc i < length c ⟶ c ! i −ese→ c ! Suc i
              ⟶ (gets-es (c ! i), gets-es (c ! Suc i)) ∈ 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 ≠ 0
  then have dd0: m > 0 by simp
  with b0 c0 have dd1: take m (cs k) ∈ assume-es (Pre k, Rely k) by simp

  have gets-es (?esl ! 0) ∈ 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) ∈ pre by (simp add:assume-pes-def)
      ultimately show ?thesis using p1 p8 by auto
    qed
  moreover
  have ∀ i. Suc i<length ?esl ⟶
       ?esl!i −ese→ ?esl!(Suc i) ⟶
       (gets-es (?esl!i), gets-es (?esl!Suc i)) ∈ Rely k
    proof -
    {
      fix i
      assume d0: Suc i<length ?esl
        and  d1: ?esl!i −ese→ ?esl!Suc i
      then have d2: ?esl!i = (cs k)!i ∧ ?esl!Suc i = (cs k)! Suc i
        by auto
      from p6 c3 d0 have d4: (∃ t k. (c!i −pes−(t♯k)→ c!Suc i) ∧
                (∀ k t. (c!i −pes−(t♯k)→ c!Suc i) ⟶ (cs k!i −es−(t♯k)→ cs k! Suc i) ∧
                        (∀ k'. k' ≠ k ⟶ (cs k'!i −ese→ cs k'! Suc i))))
                ∨
                (((c!i) −pese→ (c!Suc i)) ∧ (∀ k. (((cs k)!i) −ese→ ((cs k)! Suc i))))
        by (simp add:conjoin-def compat-tran-def)
      from d1 have d5: ((cs k)!i) −ese→ ((cs k)! Suc i)
           by (simp add: d2)
      from d4 have (gets-es (?esl!i), gets-es (?esl!Suc i)) ∈ Rely k
        proof
          assume e0: ∃ t k. (c!i −pes−(t♯k)→ c!Suc i) ∧
                (∀ k t. (c!i −pes−(t♯k)→ c!Suc i) ⟶ (cs k!i −es−(t♯k)→ cs k! Suc i) ∧
                        (∀ k'. k' ≠ k ⟶ (cs k'!i −ese→ cs k'! Suc i)))
          then obtain ct and k' where e1: ((c!i) −pes−(ct♯k')→ (c!Suc i)) ∧
```

$(((cs\ k')!i)\ -es-(ct\sharp k')\rightarrow\ ((cs\ k')!\ Suc\ i))$ **by** *auto*
with *p6 p8 d0 d5* **have** *e2*: $k \neq k'$
using *conjoin-def[of c cs] same-spec-def[of c cs]*
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: $((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$ **by** *auto*
with *d0* **have** $(?esl!i)\ -ese\rightarrow\ (?esl!\ Suc\ i)$ **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** $\forall i.\ Suc\ i<length\ (take\ m\ (cs\ k)) \longrightarrow$
$(take\ m\ (cs\ k))!i\ -ese\rightarrow\ (take\ m\ (cs\ k))!(Suc\ i) \longrightarrow$
$(gets\text{-}es\ ((take\ m\ (cs\ k))!i),\ gets\text{-}es\ ((take\ m\ (cs\ k))!Suc\ i)) \in Rely\ k$
by (*simp add*:*assume-es-def*)
with *dd0 f0* **have** $(gets\text{-}es\ (take\ m\ (cs\ k)\ !\ i),\ gets\text{-}es\ (take\ m\ (cs\ k)\ !\ Suc\ i)) \in Rely\ k$
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** $take\ m\ (cs\ k') \in assume\text{-}es\ (Pre\ k',\ Rely\ k')$
by (*metis Suc-leD p8*)
moreover
from *e1 f0* **have** $\neg(cs\ k'\ !\ (m-1)\ -ese\rightarrow\ cs\ k'\ !m)$
using *Suc-pred' dd0 es-tran-not-etran1* **by** *fastforce*
ultimately have *f1*: $take\ (Suc\ m)\ (cs\ k') \in assume\text{-}es\ (Pre\ k',\ Rely\ k')$
using *assume-es-one-more[of cs k' m Pre k' Rely k'] p8 p9 c0 dd0*
by (*simp add*: *Suc-le-eq*)
from *p7* **have** $cs\ k' \in cpts\text{-}of\text{-}es\ (pes\ k')\ s\ x$ **by** *simp*
with *p8 c0 dd0* **have** *f2*: $?esl2 \in cpts\text{-}of\text{-}es\ (pes\ k')\ s\ x$
using *cpts-es-take[of cs k' m] cpts-of-es-def[of pes k' s x]*
by (*simp add*: *Suc-le-lessD*)
from *p0 p8 c0* **have** $?esl2 \in commit\text{-}es(Guar\ k',\ Post\ k')$
using *commit-es-take-n[of Suc m cs k' Guar k' Post k']* **by** *auto*
then have $\forall i.\ Suc\ i<length\ ?esl2 \longrightarrow$
$(\exists t.\ ?esl2!i\ -es-t\rightarrow\ ?esl2!(Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (?esl2!i),\ gets\text{-}es\ (?esl2!Suc\ i)) \in Guar\ k'$
by (*simp add*:*commit-es-def*)

with *p8 e1 f0 c0 dd0* **have** $(gets\text{-}es\ (?esl2\ !\ (m-1)),\ gets\text{-}es\ (?esl2\ !\ m)) \in Guar\ k'$
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*: $(((c!i)\ -pese\rightarrow\ (c!Suc\ i)) \wedge (\forall k.\ (((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i))))$
from *p5* **have** $\forall i.\ Suc\ i<length\ c \longrightarrow$
$c!i\ -pese\rightarrow\ c!(Suc\ i) \longrightarrow$
$(gets\ (c!i),\ gets\ (c!Suc\ i)) \in rely$

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*(*Guar k*, *Post k*);
c ∈ *cpts-of-pes pes s x*; *c*∈*assume-pes*(*pre*, *rely*); *c* ∝ *cs*⟧ ⟹
∀ *k*. (*cs k*) ∈ *assume-es*(*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* (*Guar k*, *Post k*)
assume *a5*: *c* ∈ *cpts-of-pes pes s x*
assume *a6*: *c* ∈ *assume-pes* (*pre*, *rely*)
assume *a7*: *c* ∝ *cs*
have *f8*: *c* ≠ []
using *a5 cpts-of-pes-def* **by** *force*
from *a7* **have** *p8*: ∀ *k*. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
{
fix *k*
have (*cs k*) ∈ *assume-es*(*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 Post c pes s x length* (*cs k*) *k*] **by** *force*
}
then show *?thesis* **by** *simp*
qed

lemma *cpts-es-sat-rely*[*rule-format*]:
⟦∀ *k*. ⊨ (*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 pes s x*; *c*∈*assume-pes*(*pre*, *rely*);
c ∝ *cs*; ∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*⟧ ⟹
∀ *n k*. *n* ≤ *length* (*cs k*) ∧ *n* > *0* ⟶ *take n* (*cs k*)∈*assume-es*(*Pre k*, *Rely k*)
proof −
assume *p0*: ∀ *k*. ⊨ (*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 pes s x
  and  p5: c ∈ assume-pes(pre, rely)
  and  p6: c ∝ cs
  and  p7: ∀k. cs k ∈ cpts-of-es (pes k) s x
from p6 have p8: ∀k. length (cs k) = length c by (simp add:conjoin-def same-length-def)
from p7 have p9: ∀k. cs k ∈ cpts-es using cpts-of-es-def mem-Collect-eq by fastforce
from p6 have p10: ∀k j. j < length c ⟶ gets (c!j) = gets-es ((cs k)!j) by (simp add:conjoin-def same-state-def)
{
  fix n
  have ∀k. n ≤ length (cs k) ∧ n > 0 ⟶ take n (cs k)∈assume-es(Pre k, Rely k)
    proof(induct n)
      case 0 then show ?case by simp
    next
      case (Suc m)
      assume b0: ∀k. m ≤ length (cs k) ∧ 0 < m ⟶ take m (cs k) ∈ assume-es (Pre k, Rely k)

      {
        fix k
        assume c0: Suc m ≤ length (cs k) ∧ 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 ∈ assume-es (Pre k, Rely k)
        proof(cases m = 0)
          assume d0: m = 0
          have gets-es (take (Suc m) (cs k)!0) ∈ 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) ∈ 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 ∀i. Suc i < length (take (Suc m) (cs k))
                ⟶ (take (Suc m) (cs k)) ! i −ese→ (take (Suc m) (cs k)) ! Suc i
                ⟶ (gets-es ((take (Suc m) (cs k)) ! i), gets-es ((take (Suc m) (cs k)) ! Suc i)) ∈ rely
            by auto
          moreover
          have assume-es (Pre k, Rely k) = {c. gets-es (c ! 0) ∈ Pre k ∧
                (∀i. Suc i < length c ⟶ c ! i −ese→ c ! Suc i
                      ⟶ (gets-es (c ! i), gets-es (c ! Suc i)) ∈ 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 ≠ 0
          then have dd0: m > 0 by simp
          with b0 c0 have dd1: take m (cs k) ∈ assume-es (Pre k, Rely k) by simp

          have gets-es (?esl ! 0) ∈ 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* $\longrightarrow$
?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*: *?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*. (*c!i* $-pes-(t\sharp k)\rightarrow$ *c!Suc i*) $\wedge$
($\forall$ *k t*. (*c!i* $-pes-(t\sharp k)\rightarrow$ *c!Suc i*) $\longrightarrow$ (*cs k!i* $-es-(t\sharp k)\rightarrow$ *cs k! Suc i*) $\wedge$
($\forall k'$. $k' \neq k$ $\longrightarrow$ (*cs k'!i* $-ese\rightarrow$ *cs k'! Suc i*))))
$\vee$
(((*c!i*) $-pese\rightarrow$ (*c!Suc i*)) $\wedge$ ($\forall$ *k*. (((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*))))
by (*simp add:conjoin-def compat-tran-def*)
from *d1* **have** *d5*: ((*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*. (*c!i* $-pes-(t\sharp k)\rightarrow$ *c!Suc i*) $\wedge$
($\forall$ *k t*. (*c!i* $-pes-(t\sharp k)\rightarrow$ *c!Suc i*) $\longrightarrow$ (*cs k!i* $-es-(t\sharp k)\rightarrow$ *cs k! Suc i*) $\wedge$
($\forall k'$. $k' \neq k$ $\longrightarrow$ (*cs k'!i* $-ese\rightarrow$ *cs k'! Suc i*)))
then obtain *ct* **and** k' **where** *e1*: ((*c!i*) $-pes-(ct\sharp k')\rightarrow$ (*c!Suc i*)) $\wedge$
(((*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 c cs*] *same-spec-def*[*of c cs*]
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: ((*cs k*)!*i*) $-ese\rightarrow$ ((*cs k*)! *Suc i*) **by** *auto*
with *d0* **have** (*?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*)) $\longrightarrow$
(*take m* (*cs k*))!*i* $-ese\rightarrow$ (*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* (*Pre k'*, *Rely k'*)

```
                  by (metis Suc-leD p8)
                moreover
                from e1 f0 have ¬(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 (Pre k′, Rely k′)
                  using assume-es-one-more[of cs k′ m Pre k′ Rely k′] p8 p9 c0 dd0
                  by (simp add: Suc-le-eq)
                from p7 have cs k′ ∈ cpts-of-es (pes k′) s x by simp
                with p8 c0 dd0 have f2: ?esl2∈cpts-of-es (pes k′) s x
                  using cpts-es-take[of cs k′ m] cpts-of-es-def[of pes k′ s x]
                    by (simp add: Suc-le-lessD)
                from p0 have f3: ⊨ pes k′ sat_s [Pre k′, Rely k′, Guar k′, Post k′]  by simp
                with f1 f2 have ?esl2∈commit-es(Guar k′, Post k′)
                  using es-validity-def[of pes k′ Pre k′ Rely k′ Guar k′ Post k′]
                    by auto
                then have ∀ i. Suc i<length ?esl2 ⟶
                             (∃ t. ?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: (((c!i) −pese→ (c!Suc i)) ∧ (∀ k. (((cs k)!i) −ese→ ((cs k)! Suc i))))
              from p5 have ∀ i. Suc i<length c ⟶
                             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. ⊨ (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;
```

$c \in$ *cpts-of-pes pes s x*; $c \in$*assume-pes*(*pre*, *rely*);
$c \propto cs$; $\forall k$. *cs k* $\in$ *cpts-of-es* (*pes k*) *s x* $]\!]$
$\Longrightarrow \forall k\ i\ m.\ m \leq$ *length c* $\longrightarrow$ *Suc i* $<$ *length* (*take m* (*cs k*)) $\longrightarrow$ ($\exists t$.((*take m* (*cs k*))!*i*$-es-t\rightarrow$((*take m* (*cs k*))!*Suc i*)))
$\longrightarrow$ (*gets-es* ((*take m* (*cs k*))!*i*),*gets-es* ((*take m* (*cs k*))!*Suc i*)) $\in$ *Guar k*

proof $-$
assume *p0*: $\forall k. \models$ (*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 pes s x*
and *p5*: $c \in$ *assume-pes*(*pre*, *rely*)
and *p6*: $c \propto cs$
and *p7*: $\forall k$. *cs k* $\in$ *cpts-of-es* (*pes k*) *s x*
from *p6* **have** *p8*: $\forall k$. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
{
fix *k i m*
assume *a0*: $m \leq$ *length c*
and *a1*: *Suc i* $<$ *length* (*take m* (*cs k*))
and *a2*: $\exists t$.((*take m* (*cs k*))!*i*$-es-t\rightarrow$((*take m* (*cs k*))!*Suc i*))
have (*gets-es* ((*take m* (*cs k*))!*i*),*gets-es* ((*take m* (*cs k*))!*Suc i*)) $\in$ *Guar k*
proof(*cases m = 0*)
assume *m = 0* **with** *a1* **show** *?thesis* **by** *auto*
next
assume $m \neq 0$
then have *b0*: $m > 0$ **by** *simp*
let *?esl* = *take m* (*cs k*)
from *p7* **have** *cs k* $\in$ *cpts-of-es* (*pes k*) *s x* **by** *simp*
then have *cs k*!*0*=(*pes k*,*s*,*x*) $\wedge$ *cs k* $\in$ *cpts-es* **by** (*simp add*:*cpts-of-es-def*)
with *b0* **have** *?esl*!*0*=(*pes k*,*s*,*x*) $\wedge$ *?esl* $\in$ *cpts-es*
by (*metis Suc-pred a0 cpts-es-take leD not-less-eq nth-take p8*)
then have *r1*: *?esl* $\in$ *cpts-of-es* (*pes k*) *s x* **by** (*simp add*:*cpts-of-es-def*)
from *p0 p1 p2 p3 p4 p5 p6 p7*
have $\forall n.\ n \leq$ *length* (*cs k*) $\wedge\ n > 0 \longrightarrow$ *take n* (*cs k*)$\in$*assume-es*(*Pre k*, *Rely k*)
using *cpts-es-sat-rely*[*of pes Pre Rely Guar Post pre rely c s x cs*] **by** *auto*
with *p8 a0 b0* **have** *r2*: *?esl*$\in$*assume-es*(*Pre k*, *Rely k*) **by** *auto*

from *p0* **have** (*cpts-of-es* (*pes k*) *s x*) $\cap$ *assume-es*(*Pre k*, *Rely k*) $\subseteq$ *commit-es*(*Guar k*, *Post k*)
by (*simp add*:*es-validity-def*)
with *r1 r2* **have** *?esl* $\in$ *commit-es*(*Guar k*, *Post k*)
using *IntI subsetCE* **by** *auto*
then have $\forall i$. *Suc i*$<$*length ?esl* $\longrightarrow$
($\exists t$. *?esl*!*i* $-es-t\rightarrow$ *?esl*!(*Suc i*)) $\longrightarrow$ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *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*:
$[\![\forall k. \models$ (*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 pes s x*; $c \in$*assume-pes*(*pre*, *rely*);
$c \propto cs$; $\forall k$. *cs k* $\in$ *cpts-of-es* (*pes k*) *s x* $]\!]$

$\Longrightarrow \forall k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists\ t.((cs\ k)!i{-}es{-}t{\rightarrow}(cs\ k)!Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$

proof −
assume *p0*: $\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
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*: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes(pre,\ rely)$
and *p6*: $c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x$
then have $\forall k\ i\ m.\ m \le length\ c \longrightarrow Suc\ i < length\ (take\ m\ (cs\ k)) \longrightarrow (\exists\ t.((take\ m\ (cs\ k))!i{-}es{-}t{\rightarrow}((take\ m\ (cs\ k))!Suc\ i)))$
$\longrightarrow (gets\text{-}es\ ((take\ m\ (cs\ k))!i), gets\text{-}es\ ((take\ m\ (cs\ k))!Suc\ i)) \in Guar\ k$
using *es-tran-sat-guar-aux* [*of 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*:
$[\![\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x;\ c{\in}assume\text{-}pes(pre,\ rely);$
$c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x\ ]\!]$
$\Longrightarrow \forall k.\ cs\ k \in assume\text{-}es(Pre\ k,\ Rely\ k)$
proof −
assume *p0*: $\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
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*: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes(pre,\ rely)$
and *p6*: $c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x$
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*: $\forall k.\ cs\ k \in cpts\text{-}es$ **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 $cs\ k \in assume\text{-}es(Pre\ k,\ Rely\ k)$
using *p0 p1 p2 p3 p4 p5 p6 p7 p13 p11*
cpts-es-sat-rely[*of 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*:
$[\![\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$\forall k.\ Guar\ k \subseteq guar;$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x;\ c{\in}assume\text{-}pes(pre,\ rely)]\!]$

$\Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ c!i\ {-pes-t\rightarrow}\ c!(Suc\ i))$
$\quad\longrightarrow (gets\ (c!i), gets\ (c!Suc\ i)) \in guar$

proof –
assume $p0$: $\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
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 $p5$: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x$
and $p6$: $c{\in}assume\text{-}pes(pre,\ rely)$
{
fix i
assume $a0$: $Suc\ i < length\ c$
and $a1$: $\exists t.\ c!i\ {-pes-t\rightarrow}\ c!(Suc\ i)$
from $p5$ **have** $\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \wedge c \propto cs$
by (*meson cpt-imp-exist-conjoin-cs*)
then obtain cs **where** $a2$: $(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \wedge c \propto cs$ **by** *auto*
then have *compat-tran c cs* **by** (*simp add*:*conjoin-def*)
with $a0$ **have** $a3$: $(\exists t\ k.\ (c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i))))$
$\vee$
$(((c!i)\ {-pese\rightarrow}\ (c!Suc\ i)) \wedge (\forall k.\ (((cs\ k)!i)\ {-ese\rightarrow}\ ((cs\ k)!\ Suc\ i))))$
by (*simp add*:*compat-tran-def*)
from $a1$ **have** $\neg((c!i)\ {-pese\rightarrow}\ (c!Suc\ i))$
using *pes-tran-not-etran1* **by** *blast*
with $a3$ **have** $\exists t\ k.\ (c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i)))$
by *simp*
then obtain t **and** k **where** $a4$: $(c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i)))$
by *auto*
from $p0\ p1\ p2\ p3\ p4\ p5\ p6\ a2$ **have**
$\forall k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists t.((cs\ k)!i{-es-t\rightarrow}(cs\ k)!Suc\ i))$
$\quad\longrightarrow (gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$
using *es-tran-sat-guar* [*of pes Pre Rely Guar Post pre rely c s x cs*] **by** *simp*
then have $a5$: $Suc\ i < length\ (cs\ k) \longrightarrow (\exists t.((cs\ k)!i{-es-t\rightarrow}(cs\ k)!Suc\ i))$
$\quad\longrightarrow (gets\text{-}es\ ((cs\ k)!i), gets\text{-}es\ ((cs\ k)!Suc\ i)) \in Guar\ k$ **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*:
$[\![\forall k.\ \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$\forall k.\ Guar\ k \subseteq guar;$
$\forall k.\ Post\ k \subseteq post]\!]$
$\Longrightarrow\ \models pes\ SAT\ [pre,\ rely,\ guar,\ post]$

proof −
assume *p0*: ∀ *k*. ⊨ (*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*: ∀ *k*. *Guar k* ⊆ *guar*
and *p5*: ∀ *k*. *Post k* ⊆ *post*
have ∀ *s x*. (*cpts-of-pes pes s x*) ∩ *assume-pes*(*pre*, *rely*) ⊆ *commit-pes*(*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: *c*∈(*cpts-of-pes pes s x*) ∩ *assume-pes*(*pre*, *rely*)
then have *a1*: *c*∈(*cpts-of-pes pes s x*) ∧ *c*∈*assume-pes*(*pre*, *rely*) **by** *simp*
with *p0 p1 p2 p3 p4* **have** ∀ *i*. *Suc i* < *length c* ⟶ (∃ *t*. *c*!*i* −*pes*−*t*→ *c*!(*Suc i*))
⟶ (*gets* (*c*!*i*),*gets* (*c*!*Suc i*)) ∈ *guar*
using *pes-tran-sat-guar* [*of pes Pre Rely Guar Post pre rely guar c s x*] **by** *simp*
then have *c*∈*commit-pes*(*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* ⊆ *pre′*; *rely* ⊆ *rely′*; *guar′* ⊆ *guar*; *post′* ⊆ *post*;
⊨ *pes SAT* [*pre′*, *rely′*, *guar′*, *post′*]⟧
⟹ ⊨ *pes SAT* [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: *pre* ⊆ *pre′*
and *p1*: *rely* ⊆ *rely′*
and *p2*: *guar′* ⊆ *guar*
and *p3*: *post′* ⊆ *post*
and *p4*: ⊨ *pes SAT* [*pre′*, *rely′*, *guar′*, *post′*]
from *p4* **have** *p5*: ∀ *s x*. (*cpts-of-pes pes s x*) ∩ *assume-pes*(*pre′*, *rely′*) ⊆ *commit-pes*(*guar′*, *post′*)
by (*simp add*: *pes-validity-def*)
have ∀ *s x*. (*cpts-of-pes pes s x*) ∩ *assume-pes*(*pre*, *rely*) ⊆ *commit-pes*(*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: *c*∈(*cpts-of-pes pes s x*) ∩ *assume-pes*(*pre*, *rely*)
then have *c*∈(*cpts-of-pes pes s x*) ∧ *c*∈*assume-pes*(*pre*, *rely*) **by** *simp*
with *p0 p1* **have** *c*∈(*cpts-of-pes pes s x*) ∧ *c*∈*assume-pes*(*pre′*, *rely′*)
using *assume-pes-imp*[*of pre pre′ rely rely′ c*] **by** *simp*
with *p5* **have** *c*∈*commit-pes*(*guar′*, *post′*) **by** *auto*
with *p2 p3* **have** *c*∈*commit-pes*(*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

theorem *rgsound-pes*: ⊢ *rgf-par SAT* [*pre*, *rely*, *guar*, *post*] ⟹ ⊨ *paresys-spec rgf-par SAT* [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-pes.induct*)
proof −
{

fix *pes pre rely guar post*
assume *p0*: $\forall k.\ \vdash fst\ ((pes::'k \Rightarrow ('l,'k,'s)\ rgformula\text{-}es)\ k)\ sat_s\ [Pre_{es}\ (pes\ k),\ Rely_{es}\ (pes\ k),\ Guar_{es}\ (pes\ k),\ Post_{es}\ (pes\ k)]$
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*: $\forall k.\ Post_{es}\ (pes\ k) \subseteq post$
from *p0* **have** $\forall k.\ \models evtsys\text{-}spec\ (fst\ (pes\ k))\ sat_s\ [Pre_{es}\ (pes\ k),\ Rely_{es}\ (pes\ k),\ Guar_{es}\ (pes\ k),\ Post_{es}\ (pes\ k)]$
proof −
{
fix *k*
from *p0* **have** $\vdash fst\ (pes\ k)\ sat_s\ [Pre_{es}\ (pes\ k),\ Rely_{es}\ (pes\ k),\ Guar_{es}\ (pes\ k),\ Post_{es}\ (pes\ k)]$
by *simp*
then have $\models evtsys\text{-}spec\ (fst\ (pes\ k))\ sat_s\ [Pre_{es}\ (pes\ k),\ Rely_{es}\ (pes\ k),\ Guar_{es}\ (pes\ k),\ Post_{es}\ (pes\ k)]$
using *rgsound-es* [*of 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** $\models$ *paresys-spec pes SAT* [*pre*, *rely*, *guar*, *post*]
using *parallel-sound* [*of 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*)
}
next
{
fix *pre pre′ rely rely′ guar′ guar post′ post pesf*
assume $pre \subseteq pre'$
and $rely \subseteq rely'$
and $guar' \subseteq guar$
and $post' \subseteq post$
and $\vdash$ *pesf SAT* [*pre′*, *rely′*, *guar′*, *post′*]
and $\models$ *paresys-spec pesf SAT* [*pre′*, *rely′*, *guar′*, *post′*]
then show $\models$ *paresys-spec pesf SAT* [*pre*, *rely*, *guar*, *post*]
using *parallel-seq-sound*[*of pre pre′ rely rely′ guar′ guar post′ post paresys-spec pesf*] **by** *simp*
}
qed

end

8 Rely-guarantee Reasoning

theory *PiCore-RG-Prop*
imports *PiCore-Hoare*
begin

fun *all-evts-es* :: (*′l*,*′k*,*′s*) *rgformula-ess* ⇒ (*′l*,*′k*,*′s*) *rgformula-e set*
where *all-evts-es-seq*: *all-evts-es* (*rgf-EvtSeq e es*) = *insert e* (*all-evts-es* (*fst es*)) |
all-evts-es-esys: *all-evts-es* (*rgf-EvtSys es*) = *es*

fun *all-evts-esspec* :: (*′l*,*′k*,*′s*) *esys* ⇒ (*′l*,*′k*,*′s*) *event set*
where *all-evts-esspec* (*EvtSeq e es*) = *insert e* (*all-evts-esspec es*) |
all-evts-esspec (*EvtSys es*) = *es*

fun *all-basicevts-es* :: (*′l*,*′k*,*′s*) *esys* ⇒ (*′l*,*′k*,*′s*) *event set*
where *all-basicevts-es* (*EvtSeq e es*) = (*if is-basicevt e then*

insert e (*all-basicevts-es es*)
else all-basicevts-es es) |
all-basicevts-es (*EvtSys es*) = $\{x.\ x \in es \wedge$ *is-basicevt x*$\}$

definition *all-evts* :: ($'l,'k,'s$) *rgformula-par* $\Rightarrow$ ($'l,'k,'s$) *rgformula-e set*
where *all-evts parsys* $\equiv \bigcup k.$ *all-evts-es* (*fst* (*parsys k*))

definition *all-basicevts* :: ($'l,'k,'s$) *paresys* $\Rightarrow$ ($'l,'k,'s$) *event set*
where *all-basicevts parsys* $\equiv \bigcup k.$ *all-basicevts-es* (*parsys k*)

lemma *all-evts-same*: *Domain* (*all-evts-es rgfes*) = *all-evts-esspec* (*evtsys-spec rgfes*)
apply(*induct rgfes*)
using *all-evts-esspec.simps all-evts-es.simps evtsys-spec.simps*
E_e*-def eq-fst-iff fsts.intros* **apply** *fastforce*
using *all-evts-esspec.simps all-evts-es.simps evtsys-spec.simps*
E_e*-def fsts.intros* **apply** *force*
done

lemma *allbasicevts-es-blto-allevts*: *all-basicevts-es esys* $\subseteq$ *all-evts-esspec esys*
apply(*induct esys*)
apply *auto*[*1*]
by *auto*

lemma *allevts-es-blto-allevts*: $\forall k.$ *all-evts-esspec* (*evtsys-spec* (*fst* (*pesrgf k*))) $\subseteq$ *Domain* (*all-evts pesrgf*)
proof −
{
fix *k*
have *all-evts-esspec* (*evtsys-spec* (*fst* (*pesrgf k*))) = *Domain* (*all-evts-es* (*fst* (*pesrgf k*)))
using *all-evts-same* **by** *auto*
moreover
have *all-evts-es* (*fst* (*pesrgf k*)) $\subseteq$ *all-evts pesrgf*
using *all-evts-def UNIV-I UN-upper* **by** *blast*
ultimately have *all-evts-esspec* (*evtsys-spec* (*fst* (*pesrgf k*))) $\subseteq$ *Domain* (*all-evts pesrgf*)
by *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *etran-nchg-curevt*:
$c \propto cs \Longrightarrow \forall k\ i.$ *Suc i* < *length* (*cs k*) $\wedge$ ($\exists$ *actk*. *c*!*i*−*pes*−*actk*→*c*!*Suc i*)
$\wedge$ (*cs k* ! *i* −*ese*→ *cs k* ! *Suc i*)
$\longrightarrow$ *getx-es* (*cs k* ! *i*) *k* = *getx-es* (*cs k* ! *Suc i*) *k*
proof −
assume *p0*: $c \propto cs$
{
fix *k i*
assume *a0*: *Suc i* < *length* (*cs k*)
and *a1*: $\exists$ *actk*. *c*!*i*−*pes*−*actk*→*c*!*Suc i*
and *a2*: *cs k* ! *i* −*ese*→ *cs k* ! *Suc i*
from *p0* **have** *a3*: $\forall k.$ *length c* = *length* (*cs k*)
using *conjoin-def*[*of c cs*] *same-length-def*[*of c cs*] **by** *simp*
from *a1* **have** ¬(*c*!*i*−*pese*→*c*!*Suc i*) **using** *pes-tran-not-etran1* **by** *blast*
with *p0 a0 a1 a3* **have** $\exists$ *t k*. (*c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) $\wedge$
($\forall$ *k t*. (*c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) $\longrightarrow$ (*cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) $\wedge$
($\forall$ *k'*. *k'* $\neq$ *k* $\longrightarrow$ (*cs k'*!*i* −*ese*→ *cs k'*! *Suc i*)))
using *conjoin-def*[*of c cs*] *compat-tran-def*[*of c cs*] **by** *auto*
then obtain *t1* **and** *k1* **where** *a4*: (*c*!*i* −*pes*−(*t1*♯*k1*)→ *c*!*Suc i*) $\wedge$
($\forall$ *k t*. (*c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) $\longrightarrow$ (*cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) $\wedge$

$(\forall k'. k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$ **by** *auto*
from *p0 a0 a3* **have** *a5*: $getx\text{-}es\ (cs\ k\ !\ i) = getx\text{-}es\ (cs\ k1\ !\ i)$
$\wedge\ getx\text{-}es\ (cs\ k\ !\ Suc\ i) = getx\text{-}es\ (cs\ k1\ !\ Suc\ i)$
using *conjoin-def[of c cs] same-state-def[of c cs] same-spec-def[of c cs]* **by** *auto*
from *a2 a4* **have** *a6*: $k \neq k1$ **using** *es-tran-not-etran1* **by** *blast*
from *a4* **have** $getx\text{-}es\ (cs\ k\ !\ i)\ k = getx\text{-}es\ (cs\ k\ !\ Suc\ i)\ k$
proof(*induct t1*)
case (*Cmd x*)
then show *?case*
using *cmd-ines-nchg-x2[of cs k1 ! i x k1 cs k1 ! Suc i] a5* **by** *auto*
next
case (*EvtEnt x*)
then show *?case*
using *a5 a6 entevt-ines-notchg-otherx2[of cs k1 ! i x k1 cs k1 ! Suc i]* **by** *auto*
qed

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

lemma *compt-notevtent-iscmd*:
$c \propto cs \Longrightarrow \forall k\ i.\ Suc\ i < length\ (cs\ k) \wedge (\exists\ actk.\ c!i{-}pes{-}actk{\rightarrow}c!Suc\ i)$
$\wedge\ (\neg\ (\exists\ e.\ cs\ k\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ cs\ k\ !\ Suc\ i))$
$\longrightarrow (\exists\ cmd.\ cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i) \vee cs\ k\ !\ i\ -ese\rightarrow\ cs\ k\ !\ Suc\ i$
proof −
assume *p0*: $c \propto cs$
{
fix *k i*
assume *a0*: $Suc\ i < length\ (cs\ k)$
and *a1*: $\exists\ actk.\ c!i{-}pes{-}actk{\rightarrow}c!Suc\ i$
and *a2*: $\neg\ (\exists\ e.\ cs\ k\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ cs\ k\ !\ Suc\ i)$
from *p0* **have** *a3*: $\forall k.\ length\ c = length\ (cs\ k)$
using *conjoin-def[of c cs] same-length-def[of c cs]* **by** *simp*
from *a1* **have** $\neg(c!i{-}pese{\rightarrow}c!Suc\ i)$ **using** *pes-tran-not-etran1* **by** *blast*
with *p0 a0 a1 a3* **have** $\exists\ t\ k.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\ k\ t.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (cs\ k!i\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'. k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$
using *conjoin-def[of c cs] compat-tran-def[of c cs]* **by** *auto*
then obtain *t1* **and** *k1* **where** *a4*: $(c!i\ -pes-(t1\sharp k1)\rightarrow\ c!Suc\ i) \wedge$
$(\forall\ k\ t.\ (c!i\ -pes-(t\sharp k)\rightarrow\ c!Suc\ i) \longrightarrow (cs\ k!i\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'. k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i)))$ **by** *auto*
have $(\exists\ cmd.\ cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i) \vee cs\ k\ !\ i\ -ese\rightarrow\ cs\ k\ !\ Suc\ i$
proof(*cases* $k = k1$)
assume *b0*: $k = k1$
with *a2 a4* **have** $\exists\ cmd.\ cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i$
proof(*induct t1*)
case (*Cmd x*) **then show** *?case* **by** *auto*
next
case (*EvtEnt x*) **then show** *?case* **by** *auto*
qed
then show *?thesis* **by** *auto*
next
assume *b0*: $k \neq k1$
with *a4* **have** $cs\ k\ !\ i\ -ese\rightarrow\ cs\ k\ !\ Suc\ i$ **by** *auto*
then show *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *auto*

qed

lemma *evtent-impl-curevt-in-cpts-es*[*rule-format*]:
$[\![c \propto cs;\ \forall j.\ Suc\ j < length\ c \longrightarrow (\exists\, actk.\ c!j{-}pes{-}actk{\rightarrow}c!Suc\ j)]\!]$
$\Longrightarrow \forall\, k\ i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k)!i\ {-}es{-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$
$\longrightarrow (\forall\, j.\ j > Suc\ i \wedge Suc\ j < length\ (cs\ k)$
$\wedge\ (\forall\, m.\ m > i \wedge m < j \longrightarrow \neg(\exists\, e.\ (cs\ k)!m\ {-}es{-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ m)))$
$\longrightarrow (\forall\, m.\ m > i \wedge m \le j \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = e))$
proof −
assume *p1*: $c \propto cs$
and *p3*: $\forall j.\ Suc\ j < length\ c \longrightarrow (\exists\, actk.\ c!j{-}pes{-}actk{\rightarrow}c!Suc\ j)$
from *p1 p3* **have** $\forall\, i\ k.\ Suc\ i < length\ (cs\ k) \wedge (\exists\, actk.\ c\ !\ i\ {-}pes{-}actk{\rightarrow}\ c\ !\ Suc\ i)$
$\wedge\ \neg\ (\exists\, e.\ cs\ k\ !\ i\ {-}es{-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ i)$
$\longrightarrow (\exists\, cmd.\ cs\ k\ !\ i\ {-}es{-}Cmd\ cmd\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ i) \vee cs\ k\ !\ i\ {-}ese{\rightarrow}\ cs\ k\ !\ Suc\ i$
using *compt-notevtent-iscmd* [*of c cs*] **by** *auto*
then have *p5*: $\bigwedge i\ k.\ Suc\ i < length\ (cs\ k) \wedge (\exists\, actk.\ c\ !\ i\ {-}pes{-}actk{\rightarrow}\ c\ !\ Suc\ i)$
$\wedge\ \neg\ (\exists\, e.\ cs\ k\ !\ i\ {-}es{-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ i)$
$\Longrightarrow (\exists\, cmd.\ cs\ k\ !\ i\ {-}es{-}Cmd\ cmd\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ i)$
$\vee\ cs\ k\ !\ i\ {-}ese{\rightarrow}\ cs\ k\ !\ Suc\ i$ **by** *auto*
from *p1* **have** $\forall\, k\ i.\ Suc\ i < length\ (cs\ k) \wedge (\exists\, actk.\ c\ !\ i\ {-}pes{-}actk{\rightarrow}\ c\ !\ Suc\ i)$
$\wedge\ cs\ k\ !\ i\ {-}ese{\rightarrow}\ cs\ k\ !\ Suc\ i \longrightarrow$
$getx\text{-}es\ (cs\ k\ !\ i)\ k = getx\text{-}es\ (cs\ k\ !\ Suc\ i)\ k$
using *etran-nchg-curevt* [*of c cs*] **by** *simp*
then have *p6*: $\bigwedge i\ k.\ Suc\ i < length\ (cs\ k) \wedge (\exists\, actk.\ c\ !\ i\ {-}pes{-}actk{\rightarrow}\ c\ !\ Suc\ i)$
$\wedge\ cs\ k\ !\ i\ {-}ese{\rightarrow}\ cs\ k\ !\ Suc\ i \Longrightarrow$
$getx\text{-}es\ (cs\ k\ !\ i)\ k = getx\text{-}es\ (cs\ k\ !\ Suc\ i)\ k$ **by** *auto*
then show *?thesis*
proof −
{
fix *k i*
assume *a0*: $Suc\ i < length\ (cs\ k) \wedge ((cs\ k)!i\ {-}es{-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$
then obtain *es1* **and** *s1* **and** *x1* **where** *a01*: $(cs\ k)!i = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
from *a0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a02*: $(cs\ k)!Suc\ i = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
from *p1* **have** *a2*: $\forall\, k.\ length\ c = length\ (cs\ k)$ **using** *conjoin-def*[*of c cs*] *same-length-def*[*of c cs*] **by** *simp*
from *a0* **have** $\forall\, j.\ j > Suc\ i \wedge Suc\ j < length\ (cs\ k)$
$\wedge\ (\forall\, m.\ m > i \wedge m < j \longrightarrow \neg(\exists\, e.\ (cs\ k)!m\ {-}es{-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ m)))$
$\longrightarrow (\forall\, m.\ m > i \wedge m \le j \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = e)$
proof−
{
fix *j*
assume *b0*: $j > Suc\ i \wedge Suc\ j < length\ (cs\ k)$
and *b1*: $\forall\, m.\ m > i \wedge m < j \longrightarrow \neg(\exists\, e.\ (cs\ k)!m\ {-}es{-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ m))$
then have $\forall\, m.\ m > i \wedge m \le j \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = e$
proof(*induct j*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc sj*)
assume *c0*: $Suc\ i < sj \wedge Suc\ sj < length\ (cs\ k) \Longrightarrow$
$(\forall\, m.\ i < m \wedge m < sj \longrightarrow \neg\ (\exists\, e.\ cs\ k\ !\ m\ {-}es{-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ m)) \Longrightarrow$
$(\forall\, m.\ i < m \wedge m \le sj \longrightarrow getx\text{-}es\ (cs\ k\ !\ m)\ k = e)$
and *c1*: $Suc\ i < Suc\ sj \wedge Suc\ (Suc\ sj) < length\ (cs\ k)$
and *c2*: $\forall\, m.\ i < m \wedge m < Suc\ sj \longrightarrow \neg\ (\exists\, e.\ cs\ k\ !\ m\ {-}es{-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ m)$
show *?case*
proof(*cases Suc i = sj*)
assume *d0*: $Suc\ i = sj$
then show *?thesis*

```
    proof−
    {
      fix m
      assume e0: i < m ∧ m ≤ Suc sj
      from a0 have e1: getx-es (cs k ! Suc i) k = e
        using entevt-ines-chg-selfx2[of cs k ! i e k cs k ! Suc i] by simp
      have getx-es (cs k ! m) k = e
        proof(cases m = Suc i)
          assume f0: m = Suc i
          with e1 show ?thesis by simp
        next
          assume m ≠ Suc i
          with d0 e0 have f0: m = Suc (Suc i) by auto
          with c2 d0 have f1: ¬ (∃ e. cs k ! Suc i −es−EvtEnt e♯k→ cs k ! Suc (Suc i))
            by auto
          from p3 a2 b0 have ∃ actk. c ! Suc i −pes−actk→ c ! Suc (Suc i) by auto
          with p3 b0 f1 have (∃ cmd. cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i)) ∨
                    cs k ! Suc i −ese→ cs k ! Suc (Suc i) using p5 [of Suc i k] by auto
          then show ?thesis
            proof
              assume ∃ cmd. cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i)
              then obtain cmd where g0: cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i) by auto
              with e1 f0 have getx-es (cs k ! Suc (Suc i)) k = e
                using cmd-ines-nchg-x2 [of cs k ! Suc i cmd k cs k ! Suc (Suc i)] by simp
              with f0 show ?thesis by simp
            next
              assume g0: cs k ! Suc i −ese→ cs k ! Suc (Suc i)
              from p3 a2 b0 have g1: ∃ actk. c ! Suc i −pes−actk→ c ! Suc (Suc i) by auto
              from b0 e1 f0 g0 g1 show ?thesis using p6 [of Suc i k] by auto
            qed
        qed
    }
    then show ?thesis by auto qed
next
  assume d0: Suc i ≠ sj
  with c1 have d1: Suc i < sj by auto
  with c0 c1 c2 have d2: ∀ m. i < m ∧ m ≤ sj ⟶ getx-es (cs k ! m) k = e by auto
  then show ?thesis
    proof −
    {
      fix m
      assume e0: i < m ∧ m ≤ Suc sj
      have  getx-es (cs k ! m) k = e
        proof(cases i < m ∧ m < Suc sj)
          assume f0: i < m ∧ m < Suc sj
          with d2 show ?thesis by auto
        next
          assume f0: ¬(i < m ∧ m < Suc sj)
          with e0 have f1: m = Suc sj by simp
          from d1 d2 have f2: getx-es (cs k ! sj) k = e by auto
          from f1 c1 c2 have f3: ¬ (∃ e. cs k ! sj −es−EvtEnt e♯k→ cs k ! Suc sj)
            by auto
          from c2 d1 have ¬ (∃ e. cs k ! sj −es−EvtEnt e♯k→ cs k ! Suc sj) by auto
          from p3 a2 c1 have ∃ actk. c ! sj −pes−actk→ c ! Suc sj by auto
          with p3 b0 c1 f1 f3 have (∃ cmd. cs k ! sj −es−Cmd cmd♯k→ cs k ! Suc sj) ∨
                    cs k ! sj −ese→ cs k ! Suc sj using p5 [of sj k] by auto
          then show ?thesis
            proof
```

```
                    assume (∃ cmd. cs k ! sj −es−Cmd cmd♯k→ cs k ! Suc sj)
                    then obtain cmd where g0: cs k !sj −es−Cmd cmd♯k→ cs k ! Suc sj by auto
                    with f2 have getx-es (cs k ! Suc sj) k = e
                      using cmd-ines-nchg-x2 [of cs k ! sj cmd k cs k ! Suc sj] by simp
                    with f1 show ?thesis by simp
                  next
                    assume g0: cs k ! sj −ese→ cs k ! Suc sj
                    from p3 a2 c1 have g1: ∃ actk. c ! sj −pes−actk→ c ! Suc sj by auto
                    from b0 c1 f1 f2 g0 g1 show ?thesis using p6 [of sj k] by auto
                  qed
                qed
              }
              then show ?thesis by auto qed
            qed
          qed
        }
        then show ?thesis by auto qed
      }
      then show ?thesis by auto qed
    qed

lemma evtent-impl-curevt-in-cpts-es1[rule-format]:
  ⟦c ∝ cs; ∀ j. Suc j < length c ⟶ (∃ actk. c!j−pes−actk→c!Suc j)⟧
    ⟹ ∀ k i. Suc i < length (cs k) ∧ ((cs k)!i −es−((EvtEnt e)♯k)→ (cs k)!(Suc i))
          ⟶ (∀ j. j ≥ Suc i ∧ Suc j ≤ length (cs k)
              ∧ (∀ m. m > i ∧ m < j ⟶ ¬(∃ e. (cs k)!m −es−((EvtEnt e)♯k)→ (cs k)!(Suc m)))
              ⟶ (∀ m. m > i ∧ m ≤ j ⟶ getx-es ((cs k)!m) k = e))
  proof −
    assume p1: c ∝ cs
      and  p3: ∀ j. Suc j < length c ⟶ (∃ actk. c!j−pes−actk→c!Suc j)
    from p1 p3 have ∀ i k. Suc i < length (cs k) ∧ (∃ actk. c ! i −pes−actk→ c ! Suc i)
                    ∧ ¬ (∃ e. cs k ! i −es−EvtEnt e♯k→ cs k ! Suc i)
                        ⟶ (∃ cmd. cs k ! i −es−Cmd cmd♯k→ cs k ! Suc i) ∨ cs k ! i −ese→ cs k ! Suc i
                  using compt-notevtent-iscmd [of c cs] by auto
    then have p5: ⋀i k. Suc i < length (cs k) ∧ (∃ actk. c ! i −pes−actk→ c ! Suc i)
                  ∧ ¬ (∃ e. cs k ! i −es−EvtEnt e♯k→ cs k ! Suc i)
                      ⟹ (∃ cmd. cs k ! i −es−Cmd cmd♯k→ cs k ! Suc i)
                        ∨ cs k ! i −ese→ cs k ! Suc i by auto
    from p1 have ∀ k i. Suc i < length (cs k) ∧ (∃ actk. c ! i −pes−actk→ c ! Suc i)
                  ∧ cs k ! i −ese→ cs k ! Suc i ⟶
                  getx-es (cs k ! i) k = getx-es (cs k ! Suc i) k
      using etran-nchg-curevt [of c cs] by simp
    then have p6: ⋀i k. Suc i < length (cs k) ∧ (∃ actk. c ! i −pes−actk→ c ! Suc i)
                  ∧ cs k ! i −ese→ cs k ! Suc i ⟹
                  getx-es (cs k ! i) k = getx-es (cs k ! Suc i) k by auto
    then show ?thesis
      proof −
      {
        fix k i
        assume a0: Suc i < length (cs k) ∧ ((cs k)!i −es−((EvtEnt e)♯k)→ (cs k)!(Suc i))
        then obtain es1 and s1 and x1 where a01: (cs k)!i = (es1,s1,x1)
          using prod-cases3 by blast
        from a0 obtain es2 and s2 and x2 where a02: (cs k)!Suc i = (es2,s2,x2)
          using prod-cases3 by blast
        from p1 have a2: ∀ k. length c = length (cs k) using conjoin-def[of c cs] same-length-def[of c cs] by simp
        from a0 have ∀ j. j ≥ Suc i ∧ Suc j ≤ length (cs k)
                  ∧ (∀ m. m > i ∧ m < j ⟶ ¬(∃ e. (cs k)!m −es−((EvtEnt e)♯k)→ (cs k)!(Suc m)))
                  ⟶ (∀ m. m > i ∧ m ≤ j ⟶ getx-es ((cs k)!m) k = e)
```

```
proof−
{
  fix j
  assume b0: j ≥ Suc i ∧ Suc j ≤ length (cs k)
    and  b1: ∀m. m > i ∧ m < j ⟶ ¬(∃e. (cs k)!m −es−((EvtEnt e)♯k)→ (cs k)!(Suc m))
  then have ∀m. m > i ∧ m ≤ j ⟶ getx-es ((cs k)!m) k = e
    proof(induct j)
      case 0 show ?case by simp
    next
      case (Suc sj)
      assume c0: Suc i ≤ sj ∧ Suc sj ≤ length (cs k) ⟹
                 (∀m. i < m ∧ m < sj ⟶ ¬ (∃e. cs k ! m −es−EvtEnt e♯k→ cs k ! Suc m)) ⟹
                 (∀m. i < m ∧ m ≤ sj ⟶ getx-es (cs k ! m) k = e)
        and  c1: Suc i ≤ Suc sj ∧ Suc (Suc sj) ≤ length (cs k)
        and  c2: ∀m. i < m ∧ m < Suc sj ⟶ ¬ (∃e. cs k ! m −es−EvtEnt e♯k→ cs k ! Suc m)
      show ?case
        proof(cases Suc i = Suc sj)
          assume d0: Suc i = Suc sj
          then show ?thesis
            proof−
            {
              fix m
              assume e0: i < m ∧ m ≤ Suc sj
              from a0 have e1: getx-es (cs k ! Suc i) k = e
                using entevt-ines-chg-selfx2[of cs k ! i e k cs k ! Suc i] by simp
              have getx-es (cs k ! m) k = e
                proof(cases m = Suc i)
                  assume f0: m = Suc i
                  with e1 show ?thesis by simp
                next
                  assume m ≠ Suc i
                  with d0 e0 have f0: m = Suc (Suc i) by auto
                  with c2 d0 have f1: ¬ (∃e. cs k ! Suc i −es−EvtEnt e♯k→ cs k ! Suc (Suc i))
                    using Suc-n-not-le-n e0 by blast
                  from p3 a2 b0 have ∃actk. c ! Suc i −pes−actk→ c ! Suc (Suc i)
                    using Suc-le-lessD c1 d0 Suc-n-not-le-n e0 f0 by blast
                  with p3 b0 f1 have (∃cmd. cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i)) ∨
                            cs k ! Suc i −ese→ cs k ! Suc (Suc i) using p5 [of Suc i k]
                              using Suc-le-eq c1 d0 Suc-n-not-le-n e0 f0 by blast
                  then show ?thesis
                    proof
                      assume ∃cmd. cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i)
                      then obtain cmd where g0: cs k ! Suc i −es−Cmd cmd♯k→ cs k ! Suc (Suc i) by auto
                      with e1 f0 have getx-es (cs k ! Suc (Suc i)) k = e
                        using cmd-ines-nchg-x2 [of cs k ! Suc i cmd k cs k ! Suc (Suc i)] by simp
                      with f0 show ?thesis by simp
                    next
                      assume g0: cs k ! Suc i −ese→ cs k ! Suc (Suc i)
                      from p3 a2 b0 have g1: ∃actk. c ! Suc i −pes−actk→ c ! Suc (Suc i)
                        using ‹∃actk. c ! Suc i −pes−actk→ c ! Suc (Suc i)› by blast
                      from b0 e1 f0 g0 g1 show ?thesis using p6 [of Suc i k]
                        Suc-n-not-le-n d0 e0 by blast
                    qed
                qed
            }
            then show ?thesis by auto qed
        next
          assume d0: Suc i ≠ Suc sj
```

with *c1* **have** *d1*: $Suc\ i < Suc\ sj$ **by** *auto*
with *c0 c1 c2* **have** *d2*: $\forall m.\ i < m \wedge m \leq sj \longrightarrow getx\text{-}es\ (cs\ k\ !\ m)\ k = e$ **by** *auto*
then show *?thesis*
proof –
{
fix *m*
assume *e0*: $i < m \wedge m \leq Suc\ sj$
have $getx\text{-}es\ (cs\ k\ !\ m)\ k = e$
proof(*cases* $i < m \wedge m < Suc\ sj$)
assume *f0*: $i < m \wedge m < Suc\ sj$
with *d2* **show** *?thesis* **by** *auto*
next
assume *f0*: $\neg(i < m \wedge m < Suc\ sj)$
with *e0* **have** *f1*: $m = Suc\ sj$ **by** *simp*
from *d1 d2* **have** *f2*: $getx\text{-}es\ (cs\ k\ !\ sj)\ k = e$ **by** *auto*
from *f1 c1 c2* **have** *f3*: $\neg\ (\exists e.\ cs\ k\ !\ sj\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ sj)$
using *Suc-less-SucD d1 lessI* **by** *blast*
from *c2 d1* **have** $\neg\ (\exists e.\ cs\ k\ !\ sj\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ sj)$ **by** *auto*
from *p3 a2 c1* **have** $\exists actk.\ c\ !\ sj\ {-pes-}actk{\rightarrow}\ c\ !\ Suc\ sj$ **by** *auto*
with *p3 b0 c1 f1 f3* **have** $(\exists cmd.\ cs\ k\ !\ sj\ {-es-}Cmd\ cmd\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ sj) \vee$
$cs\ k\ !\ sj\ {-ese\rightarrow}\ cs\ k\ !\ Suc\ sj$ **using** *p5* [*of sj k*] **by** *auto*
then show *?thesis*
proof
assume $(\exists cmd.\ cs\ k\ !\ sj\ {-es-}Cmd\ cmd\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ sj)$
then obtain *cmd* **where** *g0*: $cs\ k\ !sj\ {-es-}Cmd\ cmd\sharp k{\rightarrow}\ cs\ k\ !\ Suc\ sj$ **by** *auto*
with *f2* **have** $getx\text{-}es\ (cs\ k\ !\ Suc\ sj)\ k = e$
using *cmd-ines-nchg-x2* [*of cs k ! sj cmd k cs k ! Suc sj*] **by** *simp*
with *f1* **show** *?thesis* **by** *simp*
next
assume *g0*: $cs\ k\ !\ sj\ {-ese\rightarrow}\ cs\ k\ !\ Suc\ sj$
from *p3 a2 c1* **have** *g1*: $\exists actk.\ c\ !\ sj\ {-pes-}actk{\rightarrow}\ c\ !\ Suc\ sj$ **by** *auto*
from *b0 c1 f1 f2 g0 g1* **show** *?thesis* **using** *p6* [*of sj k*] **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto* **qed**
qed
qed
}
then show *?thesis* **by** *auto* **qed**
}
then show *?thesis* **by** *auto* **qed**
qed

lemma *evtent-impl-curevt-in-cpts-es2*[*rule-format*]:
$[\![c \propto cs;\ \forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c!j{-pes-}actk{\rightarrow}c!Suc\ j)]\!]$
$\Longrightarrow \forall k\ i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k)!i\ {-es-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$
$\longrightarrow (\forall j.\ j > i \wedge Suc\ j < length\ (cs\ k)$
$\wedge\ (\forall m.\ m > i \wedge m < j \longrightarrow \neg(\exists e.\ (cs\ k)!m\ {-es-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ m)))$
$\longrightarrow (\forall m.\ m > i \wedge m \leq j \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = e))$
proof –
assume *p1*: $c \propto cs$
and *p3*: $\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c!j{-pes-}actk{\rightarrow}c!Suc\ j)$
then show *?thesis*
proof –
{
fix *k i*
assume *a0*: $Suc\ i < length\ (cs\ k) \wedge ((cs\ k)!i\ {-es-}((EvtEnt\ e)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$

then have $\forall j.\ j > i \wedge \mathit{Suc}\ j < \mathit{length}\ (\mathit{cs}\ k)$
$\wedge\ (\forall m.\ m > i \wedge m < j \longrightarrow \neg(\exists e.\ (\mathit{cs}\ k)!m\ -\mathit{es}-((\mathit{EvtEnt}\ e)\sharp k)\rightarrow (\mathit{cs}\ k)!(\mathit{Suc}\ m)))$
$\longrightarrow (\forall m.\ m > i \wedge m \leq j \longrightarrow \mathit{getx\text{-}es}\ ((\mathit{cs}\ k)!m)\ k = e)$
proof $-$
{
fix j
assume $\mathit{b0}$: $j > i \wedge \mathit{Suc}\ j < \mathit{length}\ (\mathit{cs}\ k)$
and $\mathit{b1}$: $\forall m.\ m > i \wedge m < j \longrightarrow \neg(\exists e.\ (\mathit{cs}\ k)!m\ -\mathit{es}-((\mathit{EvtEnt}\ e)\sharp k)\rightarrow (\mathit{cs}\ k)!(\mathit{Suc}\ m))$
then have $\forall m.\ m > i \wedge m \leq j \longrightarrow \mathit{getx\text{-}es}\ ((\mathit{cs}\ k)!m)\ k = e$
proof(*cases* $j = \mathit{Suc}\ i$)
assume $\mathit{c0}$: $j = \mathit{Suc}\ i$
then show *?thesis* **by** (*metis a0 entevt-ines-chg-selfx2 le-SucE not-less*)
next
assume $\mathit{c0}$: $j \neq \mathit{Suc}\ i$
with *b0* **have** $j > \mathit{Suc}\ i$ **by** *simp*
with *p1 p3 a0 b0 b1* **show** *?thesis* **using** *evtent-impl-curevt-in-cpts-es*[*of c cs i k e j*] **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *anonyevtseq-and-noet-impl-allanonyevtseq-bef*:
$\mathit{esl} \in \mathit{cpts\text{-}es} \Longrightarrow$
$\forall m < \mathit{length}\ \mathit{esl}.\ (\exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}!m) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e)$
$\longrightarrow (\forall i < m.\ \neg\ (\exists e\ k.\ \mathit{esl}\ !\ i\ -\mathit{es}-\mathit{EvtEnt}\ e\sharp k\rightarrow \mathit{esl}\ !\ \mathit{Suc}\ i))$
$\longrightarrow (\forall i < m.\ \exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}!i) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e)$
proof $-$
assume $\mathit{p0}$: $\mathit{esl} \in \mathit{cpts\text{-}es}$
{
fix m
assume $\mathit{a0}$: $m < \mathit{length}\ \mathit{esl}$
and $\mathit{a1}$: $\exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}!m) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e$
and $\mathit{a2}$: $\forall i < m.\ \neg\ (\exists e\ k.\ \mathit{esl}\ !\ i\ -\mathit{es}-\mathit{EvtEnt}\ e\sharp k\rightarrow \mathit{esl}\ !\ \mathit{Suc}\ i)$
then have $\forall i < m.\ \exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}!i) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e$
proof(*induct m*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n*)
assume $\mathit{b0}$: $n < \mathit{length}\ \mathit{esl} \Longrightarrow$
$\exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ n) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e \Longrightarrow$
$\forall i < n.\ \neg\ (\exists e\ k.\ \mathit{esl}\ !\ i\ -\mathit{es}-\mathit{EvtEnt}\ e\sharp k\rightarrow \mathit{esl}\ !\ \mathit{Suc}\ i) \Longrightarrow$
$\forall i < n.\ \exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ i) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e$
and $\mathit{b1}$: $\mathit{Suc}\ n < \mathit{length}\ \mathit{esl}$
and $\mathit{b2}$: $\exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ \mathit{Suc}\ n) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e$
and $\mathit{b3}$: $\forall i < \mathit{Suc}\ n.\ \neg\ (\exists e\ k.\ \mathit{esl}\ !\ i\ -\mathit{es}-\mathit{EvtEnt}\ e\sharp k\rightarrow \mathit{esl}\ !\ \mathit{Suc}\ i)$
then show *?case*
proof(*cases* $n = 0$)
assume $\mathit{c0}$: $n = 0$
with *b3* **have** $\neg\ (\exists e\ k.\ \mathit{esl}\ !\ 0\ -\mathit{es}-\mathit{EvtEnt}\ e\sharp k\rightarrow \mathit{esl}\ !\ 1)$ **by** *auto*
with *p0 b1 c0* **have** $\mathit{esl}\ !\ 0\ -\mathit{ese}\rightarrow \mathit{esl}\ !\ 1 \vee (\exists c\ k.\ \mathit{esl}\ !\ 0\ -\mathit{es}-\mathit{Cmd}\ c\sharp k\rightarrow \mathit{esl}\ !\ 1)$
using *notevtent-cptses-isenvorcmd*[*of esl*] **by** *auto*
then have $\exists e\ \mathit{es}.\ \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ 0) = \mathit{EvtSeq}\ e\ \mathit{es} \wedge \mathit{is\text{-}anonyevt}\ e$
proof
assume $\mathit{d0}$: $\mathit{esl}\ !\ 0\ -\mathit{ese}\rightarrow \mathit{esl}\ !\ 1$

with *b2 c0* **show** *?thesis* **using** *esetran-eqconf1*[*of esl* ! *0 esl* ! *1*] **by** *simp*
next
assume *d0*: $\exists\, c\ k.\ esl\ !\ 0\ -es-Cmd\ c\sharp k\rightarrow\ esl\ !\ 1$
then obtain *c* **and** *k* **where** $esl\ !\ 0\ -es-Cmd\ c\sharp k\rightarrow\ esl\ !\ 1$ **by** *auto*
then show *?thesis* **using** *cmd-enable-impl-anonyevt2*[*of esl* ! *0 c k esl* ! *1*] **by** *auto*
qed
with *c0* **show** *?thesis* **by** *auto*
next
assume $n \neq 0$
then have *c0*: $n > 0$ **by** *auto*
from *b1 b3* **have** *b4*: $\neg\ (\exists\, e\ k.\ esl\ !\ n\ -es-EvtEnt\ e\sharp k\rightarrow\ esl\ !\ Suc\ n)$ **by** *auto*
moreover
from *p0 b1* **have** $drop\ n\ esl \in cpts\text{-}es$ **using** *cpts-es-dropi2*[*of esl n*] **by** *simp*
moreover
from *b1* **have** $2 \leq length\ (drop\ n\ esl)$ **by** *simp*
moreover
from *b1* **have** $drop\ n\ esl\ !\ 0 = esl\ !\ n$ **by** *auto*
moreover
from *b1 c0* **have** $drop\ n\ esl\ !\ 1 = esl\ !\ Suc\ n$ **by** *auto*
ultimately have $esl\ !\ n\ -ese\rightarrow\ esl\ !\ Suc\ n \lor (\exists\, c\ k.\ esl\ !\ n\ -es-Cmd\ c\sharp k\rightarrow\ esl\ !\ Suc\ n)$
using *notevtent-cptses-isenvorcmd*[*of drop n esl*] **by** *auto*
then show *?case*
proof
assume *d0*: $esl\ !\ n\ -ese\rightarrow\ esl\ !\ Suc\ n$
with *b2 c0* **have** *d1*: $\exists\, e\ es.\ getspc\text{-}es\ (esl\ !\ n) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e$
using *esetran-eqconf1*[*of esl* ! *n esl* ! *Suc n*] **by** *auto*
with *b0 b1 b2 b3* **have** $\forall\, i < n.\ \exists\, e\ es.\ getspc\text{-}es\ (esl\ !\ i) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e$
by *auto*
with *d1* **show** *?thesis* **by** (*simp add*: *less-Suc-eq*)
next
assume *d0*: $\exists\, c\ k.\ esl\ !\ n\ -es-Cmd\ c\sharp k\rightarrow\ esl\ !\ Suc\ n$
then obtain *c1* **and** *k1* **where** $esl\ !\ n\ -es-Cmd\ c1\sharp k1\rightarrow\ esl\ !\ Suc\ n$ **by** *auto*
then have *d1*: $\exists\, e\ e'\ es1.\ getspc\text{-}es\ (esl\ !\ n) = EvtSeq\ e\ es1 \land e = AnonyEvent\ e'$
using *cmd-enable-impl-anonyevt2*[*of* (*esl* ! *n*) *c1 k1 esl* ! *Suc n*] **by** *simp*
with *b0 b1 b2 b3* **have** $\forall\, i < n.\ \exists\, e\ es.\ getspc\text{-}es\ (esl\ !\ i) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e$
by *auto*
with *d1* **show** *?thesis* **using** *is-anonyevt.simps*(*1*) *less-Suc-eq* **by** *auto*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed

lemma *anonyevtseq-and-noet-impl-allanonyevtseq-bef3*:
$[\![c \propto cs;\ cs\ k \in cpts\text{-}es;\ m < length\ (cs\ k)]\!] \Longrightarrow$
$(\exists\, e\ es.\ getspc\text{-}es\ ((cs\ k)!m) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e)$
$\longrightarrow (\forall\, i < m.\ \neg\ (\exists\, e.\ (cs\ k)\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ (cs\ k)\ !\ Suc\ i))$
$\longrightarrow (\forall\, i < m.\ \exists\, e\ es.\ getspc\text{-}es\ ((cs\ k)!i) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e)$
proof −
assume *p0*: $(cs\ k) \in cpts\text{-}es$
and *p1*: $c \propto cs$
and *p2*: $m < length\ (cs\ k)$
{
assume *a1*: $\exists\, e\ es.\ getspc\text{-}es\ ((cs\ k)!m) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e$
and *a2*: $\forall\, i < m.\ \neg\ (\exists\, e.\ (cs\ k)\ !\ i\ -es-EvtEnt\ e\sharp k\rightarrow\ (cs\ k)\ !\ Suc\ i)$
with *p2* **have** $\forall\, i < m.\ \exists\, e\ es.\ getspc\text{-}es\ ((cs\ k)!i) = EvtSeq\ e\ es \land is\text{-}anonyevt\ e$
proof(*induct m*)

```
        case 0 then show ?case by simp
      next
        case (Suc n)
        assume b0: n < length (cs k) ⟹
                  ∃ e es. getspc-es ((cs k) ! n) = EvtSeq e es ∧ is-anonyevt e ⟹
                  ∀ i < n. ¬ (∃ e. (cs k) ! i −es−EvtEnt e♯k→ (cs k) ! Suc i) ⟹
                  ∀ i < n. ∃ e es. getspc-es ((cs k) ! i) = EvtSeq e es ∧ is-anonyevt e
          and  b1: Suc n < length (cs k)
          and  b2: ∃ e es. getspc-es ((cs k) ! Suc n) = EvtSeq e es ∧ is-anonyevt e
          and  b3: ∀ i < Suc n. ¬ (∃ e. (cs k) ! i −es−EvtEnt e♯k→ (cs k) ! Suc i)
        then show ?case
          proof(cases n = 0)
            assume c0: n = 0
            with b3 have ¬ (∃ e. (cs k) ! 0 −es−EvtEnt e♯k→ (cs k) ! 1) by auto
            with p0 p1 b1 c0 have (cs k) ! 0 −ese→ (cs k) ! 1 ∨ (∃ c. (cs k) ! 0 −es−Cmd c♯k→ (cs k) ! 1)
              using acts-in-conjoin-cpts by (metis One-nat-def)
            then have ∃ e es. getspc-es ((cs k) ! 0) = EvtSeq e es ∧ is-anonyevt e
              proof
                assume d0: (cs k) ! 0 −ese→ (cs k) ! 1
                with b2 c0 show ?thesis using esetran-eqconf1[of (cs k) ! 0 (cs k) ! 1] by simp
              next
                assume d0: ∃ c. (cs k) ! 0 −es−Cmd c♯k→ (cs k) ! 1
                then obtain c and k where (cs k) ! 0 −es−Cmd c♯k→ (cs k) ! 1 by auto
                then show ?thesis using cmd-enable-impl-anonyevt2[of (cs k) ! 0 c k (cs k) ! 1]
                  by (metis cmd-enable-impl-anonyevt2 d0 is-anonyevt.simps(1))
              qed
            with c0 show ?thesis by auto
          next
            assume n ≠ 0
            then have c0: n > 0 by auto
            from b1 b3 have b4: ¬ (∃ e. (cs k) ! n −es−EvtEnt e♯k→ (cs k) ! Suc n) by auto
            with p1 b1 have (cs k) ! n −ese→ (cs k) ! Suc n ∨ (∃ c. (cs k) ! n −es−Cmd c♯k→ (cs k) ! Suc n)
              using acts-in-conjoin-cpts by fastforce
            then show ?case
              proof
                assume d0: (cs k) ! n −ese→ (cs k) ! Suc n
                with b2 c0 have d1: ∃ e es. getspc-es ((cs k) ! n) = EvtSeq e es ∧ is-anonyevt e
                  using esetran-eqconf1[of (cs k) ! n (cs k) ! Suc n] by auto
                with b0 b1 b2 b3 have ∀ i < n. ∃ e es. getspc-es ((cs k) ! i) = EvtSeq e es ∧ is-anonyevt e
                  by auto
                with d1 show ?thesis by (simp add: less-Suc-eq)
              next
                assume d0: ∃ c. (cs k) ! n −es−Cmd c♯k→ (cs k) ! Suc n
                then obtain c1 where (cs k) ! n −es−Cmd c1♯k→ (cs k) ! Suc n by auto
                then have d1: ∃ e e' es1. getspc-es ((cs k) ! n) = EvtSeq e es1 ∧ e = AnonyEvent e'
                  using cmd-enable-impl-anonyevt2[of ((cs k) ! n) c1 k (cs k) ! Suc n] by simp
                with b0 b1 b2 b3 have ∀ i < n. ∃ e es. getspc-es ((cs k) ! i) = EvtSeq e es ∧ is-anonyevt e
                  by auto
                with d1 show ?thesis using is-anonyevt.simps(1) less-Suc-eq by auto
              qed
          qed
      qed
    }
    then show ?thesis by auto
  qed

lemma evtseq-noesys-allevtseq: ⟦esl∈cpts-es; esl = (EvtSeq ev esys, s, x) # esl1;
      (∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys)⟧
```

```
      ⟹ (∀ i < length esl. ∃ e'. getspc-es (esl ! i) = EvtSeq e' esys)
  proof −
    assume p0: esl∈cpts-es
      and p1: esl = (EvtSeq ev esys, s, x) # esl1
      and p2: ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys
    {
      fix i
      assume a0: i < length esl
      then have ∃ e'. getspc-es (esl ! i) = EvtSeq e' esys
        proof(induct i)
          case 0
          from p1 show ?case using getspc-es-def fst-conv nth-Cons-0 by fastforce
        next
          case (Suc ii)
          assume b0: ii < length esl ⟹ ∃ e'. getspc-es (esl ! ii) = EvtSeq e' esys
            and b1: Suc ii < length esl
          then obtain e' where getspc-es (esl ! ii) = EvtSeq e' esys by auto
          with p0 have getspc-es (esl!Suc ii) = esys ∨ (∃ e. getspc-es (esl!Suc ii) = EvtSeq e esys)
            using evtseq-next-in-cpts[of esl e' esys] b1 by auto
          with p2 b1 show ?case by auto
        qed
    }
    then show ?thesis by auto
  qed

lemma evtseq-noesys-allevtseq2: ⟦esl∈cpts-es; esl = (EvtSeq ev esys, s, x) # esl1; ¬ is-basicevt ev;
      (∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys)⟧
      ⟹ (∀ i < length esl. ∃ e'. ¬ is-basicevt e' ∧ getspc-es (esl ! i) = EvtSeq e' esys)
  proof −
    assume p0: esl∈cpts-es
      and p1: esl = (EvtSeq ev esys, s, x) # esl1
      and p2: ¬ is-basicevt ev
      and p3: ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys
    {
      fix i
      assume a0: i < length esl
      then have ∃ e'. ¬ is-basicevt e' ∧ getspc-es (esl ! i) = EvtSeq e' esys
        proof(induct i)
          case 0
          with p1 p2 show ?case using getspc-es-def fst-conv nth-Cons-0 by fastforce
        next
          case (Suc ii)
          assume b0: ii < length esl ⟹ ∃ e'. ¬ is-basicevt e' ∧ getspc-es (esl ! ii) = EvtSeq e' esys
            and b1: Suc ii < length esl
          then have b2: ∃ e'. ¬ is-basicevt e' ∧ getspc-es (esl ! ii) = EvtSeq e' esys by auto
          then obtain e' where b3: ¬ is-basicevt e' ∧ getspc-es (esl ! ii) = EvtSeq e' esys by auto
          from b1 b2 have getspc-es (esl!Suc ii) = esys ∨ (∃ e. getspc-es (esl!Suc ii) = EvtSeq e esys)
            using evtseq-next-in-cpts [of esl] p0 by blast
          with p3 b1 have ∃ e. getspc-es (esl!Suc ii) = EvtSeq e esys by auto
          then obtain e where b4: getspc-es (esl!Suc ii) = EvtSeq e esys by auto
          with p0 b2 have ¬ is-basicevt e
            proof −
            {
              assume c0: is-basicevt e
              then obtain be where e = BasicEvent be by (metis event.exhaust is-basicevt.simps(1))
              with p0 b1 b3 b4 have getspc-es (esl ! ii) = EvtSeq (BasicEvent be) esys
                using only-envtran-to-basicevt[of esl esys be] by fastforce
              with b3 c0 have False using is-basicevt-def by auto
```

```
          }
          then show ?thesis by auto
          qed
        with b4 show ?case by simp
      qed
    }
    then show ?thesis by auto
  qed

lemma evtseq-evtent-befaft: ⟦esl∈cpts-es; esl = (EvtSeq ev esys, s, x) # esl1;
    (∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys);
    (∃ e k. m <length esl − 1 ∧ esl ! m −es−EvtEnt e♯k→ esl ! Suc m)⟧ ⟹
     is-basicevt ev ∧ (∀ i. i ≤ m ⟶ getspc-es (esl ! i) = EvtSeq ev esys)
     ∧ (∀ i. i > m ∧ i < length esl ⟶ (∃ e'. ¬ is-basicevt e' ∧ getspc-es (esl ! i) = EvtSeq e' esys))
  proof −
    assume p0: esl∈cpts-es
      and p1: esl = (EvtSeq ev esys, s, x) # esl1
      and p2: ∀ i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys
      and p3: ∃ e k. m <length esl − 1 ∧ esl ! m −es−EvtEnt e♯k→ esl ! Suc m
    then have a0: ∀ i < length esl. ∃ e'. getspc-es (esl ! i) = EvtSeq e' esys
      using evtseq-noesys-allevtseq[of esl ev esys s x esl1] by simp
    from p3 obtain e and k where a1: m <length esl − 1 ∧ esl ! m −es−EvtEnt e♯k→ esl ! Suc m by auto
    with a0 obtain e' where a2: getspc-es (esl ! m) = EvtSeq e' esys
      using length-Cons length-tl less-SucI list.sel(3) p1 by fastforce
    with a0 a1 have a3: e = e' ∧ (∃ e''. e' = BasicEvent e'')
      using evtent-is-basicevt-inevtseq2[of esl ! m e k esl ! Suc m e' esys] by auto
    then obtain be where a4: e' = BasicEvent be by auto
    then have a5: ∀ i. i ≤ m ⟶ getspc-es ((drop (m − i) esl) ! 0) = EvtSeq e esys
      proof−
      {
        fix i
        assume b0: i ≤ m
        then have getspc-es ((drop (m − i) esl) ! 0) = EvtSeq e esys
          proof(induct i)
            case 0
            with a1 a2 a3 show ?case by auto
          next
            case (Suc ii)
            assume c0: ii ≤ m ⟹ getspc-es (drop (m − ii) esl ! 0) = EvtSeq e esys
              and c1: Suc ii ≤ m
            from p0 have ∀ i. Suc i < length esl ∧
                  (∃ e. getspc-es (esl ! i) = EvtSeq e esys) ∧ getspc-es (esl ! Suc i) = EvtSeq (BasicEvent be) esys ⟶
                  getspc-es (esl ! i) = EvtSeq (BasicEvent be) esys
                using only-envtran-to-basicevt[of esl esys be] by simp
            then have c01: ⋀i. Suc i < length esl ∧
                  (∃ e. getspc-es (esl ! i) = EvtSeq e esys) ∧ getspc-es (esl ! Suc i) = EvtSeq (BasicEvent be) esys ⟶
                  getspc-es (esl ! i) = EvtSeq (BasicEvent be) esys by simp
            from c0 c1 have c2: getspc-es (drop (m − ii) esl ! 0) = EvtSeq e esys by simp
            moreover
            from a1 c1 have drop (m − Suc ii) esl ! 0 = esl ! (m − Suc ii) by force
            moreover
            from a1 c1 have drop (m − ii) esl ! 0 = esl ! (m − ii) by force
            moreover
            from a0 a1 c1 have (∃ e. getspc-es (esl ! (m − Suc ii)) = EvtSeq e esys) by auto
            ultimately show ?case using p0 a0 a1 a3 a4 c0 c1 c01[of (m − Suc ii)]
              Suc-diff-Suc Suc-le-lessD length-Cons length-tl less-SucI less-imp-diff-less
              list.sel(3) p1 by auto
          qed
```

}
then show *?thesis* by *auto*
qed
then have *getspc-es* (*esl* ! *0*) = *EvtSeq e esys* by *auto*
with *p1* have *a51*: *ev* = *e* using *getspc-es-def* by (*metis esys.inject*(*1*) *fst-conv nth-Cons-0*)
with *a5* have *r1*: $\forall$ *i*. *i* $\leq$ *m* $\longrightarrow$ *getspc-es* (*esl* ! *i*) = *EvtSeq ev esys*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc a1 diff-diff-cancel diff-le-self*
le-less-trans length-Cons length-tl less-SucI list.sel(*3*) *nth-Cons-0 p1*)

let *?esl* = *drop* (*Suc m*) *esl*
from *p0 p1 a1* have *a6*: *?esl*$\in$*cpts-es*
using *Suc-mono cpts-es-dropi length-Cons length-tl list.sel*(*3*) by *fastforce*
from *a1* obtain *esc1* and *s1* and *x1* and *esc2* and *s2* and *x2*
where *a7*: *esl* ! *m* = (*esc1*,*s1*,*x1*) $\wedge$ *esl* ! *Suc m* = (*esc2*,*s2*,*x2*) $\wedge$ (*esc1*,*s1*,*x1*) $-es-EvtEnt\ e\sharp k\rightarrow$ (*esc2*,*s2*,*x2*)
using *prod-cases3* by *metis*
from *a7* have $\exists$ *e*. $\neg$ *is-basicevt e* $\wedge$ *getspc-es* (*?esl*!*0*) = *EvtSeq e esys*
apply(*simp add*:*is-basicevt-def*)
apply(*rule estran.cases*)
apply *auto*
apply (*metis a2 esys.simps*(*4*) *fst-conv getspc-es-def*)
using *get-actk-def* apply (*smt Cons-nth-drop-Suc Suc-mono a1 a2 a3 ent-spec2′*
esys.inject(*1*) *event.simps*(*7*) *fst-conv getspc-es-def length-Cons length-tl list.sel*(*3*) *nth-Cons-0 p1*)
by (*metis* (*no-types*, *lifting*) *Suc-leI Suc-le-mono a1 a2 esys.inject*(*1*) *fst-conv*
getspc-es-def length-Cons length-tl list.sel(*3*) *p1 p2*)
then obtain *e1* and *s3* and *x3* where *a7*: $\neg$ *is-basicevt e1* $\wedge$ *?esl*!*0* = (*EvtSeq e1 esys*,*s3*,*x3*)
by (*metis fst-conv getspc-es-def surj-pair*)
from *p2* have $\forall$ *i*. *Suc i* $\leq$ *length ?esl* $\longrightarrow$ *getspc-es* (*?esl* ! *i*) $\neq$ *esys* by *auto*
with *p2 a6 a7* have *a8*: $\forall$ *i* < *length ?esl*. $\exists$ *e′*. $\neg$ *is-basicevt e′* $\wedge$ *getspc-es* (*?esl* ! *i*) = *EvtSeq e′ esys*
using *evtseq-noesys-allevtseq2*[*of ?esl e1 esys s3 x3*] by (*metis* (*no-types*, *lifting*)
Cons-nth-drop-Suc Suc-mono a1 length-Cons length-tl list.sel(*3*) *nth-Cons-0 p1*)
then have $\forall$ *i*. *i* > *m* $\wedge$ *i* < *length esl* $\longrightarrow$ ($\exists$ *e′*. $\neg$ *is-basicevt e′* $\wedge$ *getspc-es* (*esl* ! *i*) = *EvtSeq e′ esys*)
proof −
{
fix *i*
assume *b0*: *i* > *m* $\wedge$ *i* < *length esl*
with *a1* have *esl* ! *i* = *?esl* ! (*i* − *Suc m*) by *auto*
from *b0* have *i* − *Suc m* $\geq$ *0* by *auto*
moreover
from *b0* have *i* − *Suc m* < *length ?esl* by *auto*
ultimately have $\exists$ *e′*. $\neg$ *is-basicevt e′* $\wedge$ *getspc-es* (*?esl* ! (*i* − *Suc m*)) = *EvtSeq e′ esys* using *a8* by *auto*
}
then show *?thesis* by *auto*
qed

with *a1 a3 a4 a51 r1* show *?thesis* by *auto*
qed

lemma *evtsys-allevtseqorevtsys*:
⟦*esl*$\in$*cpts-es*; *esl* = (*EvtSys es*, *s*, *x*) # *esl1*⟧
$\Longrightarrow$ ($\forall$ *i* < *length esl*. *getspc-es* (*esl* ! *i*) = *EvtSys es*
$\vee$ ($\exists$ *e′*. *is-anonyevt e′* $\wedge$ *getspc-es* (*esl* ! *i*) = *EvtSeq e′* (*EvtSys es*)))
proof −
assume *p0*: *esl*$\in$*cpts-es*
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # *esl1*
{
fix *i*
assume *a0*: *i* < *length esl*
then have *getspc-es* (*esl* ! *i*) = *EvtSys es* $\vee$

(∃ e′. is-anonyevt e′ ∧ getspc-es (esl ! i) = EvtSeq e′ (EvtSys es))
proof(induct i)
case 0 **then show** ?case **using** p1 getspc-es-def fst-conv nth-Cons-0 **by** force
next
case (Suc ii)
assume b0: ii < length esl ⟹ getspc-es (esl ! ii) = EvtSys es ∨
(∃ e′. is-anonyevt e′ ∧ getspc-es (esl ! ii) = EvtSeq e′ (EvtSys es))
and b1: Suc ii < length esl
from a0 **obtain** esc1 **and** s1 **and** x1 **where** b2: esl ! ii = (esc1,s1,x1)
using prod-cases3 **by** blast
from a0 **obtain** esc2 **and** s2 **and** x2 **where** b3: esl ! Suc ii = (esc2,s2,x2)
using prod-cases3 **by** blast
from p0 b1 b2 b3 **have** b4: (esc1,s1,x1) −ese→ (esc2,s2,x2) ∨ (∃ et. (esc1,s1,x1) −es−et→ (esc2,s2,x2))
using incpts-es-impl-evnorcomptran[of esl] **by** auto
from b0 b1 **have** getspc-es (esl ! ii) = EvtSys es ∨
(∃ e′. is-anonyevt e′ ∧ getspc-es (esl ! ii) = EvtSeq e′ (EvtSys es))
by auto
then show ?case
proof
assume c0: getspc-es (esl ! ii) = EvtSys es
with b2 **have** c1: esc1 = EvtSys es **using** getspc-es-def **by** (metis fst-conv)
from b4 **have** esc2 = EvtSys es ∨ (∃ e′. is-anonyevt e′ ∧ esc2 = EvtSeq e′ (EvtSys es))
proof
assume (esc1,s1,x1) −ese→ (esc2,s2,x2)
then have esc1 = esc2 **by** (simp add: esetran-eqconf)
with c1 **show** ?thesis **by** simp
next
assume ∃ et. (esc1,s1,x1) −es−et→ (esc2,s2,x2)
then obtain et **where** (esc1,s1,x1) −es−et→ (esc2,s2,x2) **by** auto
with c1 **have** ∃ e′. is-anonyevt e′ ∧ esc2 = EvtSeq e′ (EvtSys es)
apply(clarsimp simp:is-anonyevt-def)
apply(rule estran.cases)
apply(simp add:get-actk-def)+
apply(rule etran.cases)
apply simp+
done
then show ?thesis **by** auto
qed
with b2 b3 **show** ?thesis **using** getspc-es-def fst-conv **by** fastforce
next
assume c0: ∃ e′. is-anonyevt e′ ∧ getspc-es (esl ! ii) = EvtSeq e′ (EvtSys es)
then obtain e′ **where** c2: is-anonyevt e′ ∧ getspc-es (esl ! ii) = EvtSeq e′ (EvtSys es) **by** auto
with b2 **have** c1: esc1 = EvtSeq e′ (EvtSys es) **using** getspc-es-def **by** (metis fst-conv)
from b4 **have** esc2 =EvtSys es ∨ (∃ e′. is-anonyevt e′ ∧ esc2 = EvtSeq e′ (EvtSys es))
proof
assume d0:(esc1,s1,x1) −ese→ (esc2,s2,x2)
then have esc1 = esc2 **by** (simp add: esetran-eqconf)
with c1 c2 d0 **show** ?thesis **by** auto
next
assume ∃ et. (esc1,s1,x1) −es−et→ (esc2,s2,x2)
then obtain et **where** (esc1,s1,x1) −es−et→ (esc2,s2,x2) **by** auto
with c1 c2 **show** ?thesis
apply(clarsimp simp:is-anonyevt-def)
apply(rule estran.cases)
apply(simp add:get-actk-def)+
apply(rule etran.cases)
apply simp+
done

qed
with *b2 b3* **show** *?thesis* **using** *getspc-es-def fst-conv* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed

lemma *evtsys-befevtent-isevtsys*:
⟦*esl*∈*cpts-es*; *esl* = (*EvtSys es*, *s*, *x*) # *esl1*⟧
⟹ ∀ *i*. *Suc i* < *length esl* ∧ (∃ *e k*. *esl* ! *i* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc i*) ⟶ *getspc-es* (*esl*!*i*) = *EvtSys es*
proof −
assume *p0*: *esl*∈*cpts-es*
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # *esl1*
{
fix *i*
assume *a0*: *Suc i* < *length esl*
and *a1*: (∃ *e k*. *esl* ! *i* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc i*)
with *p0 p1* **have** *a00*: *getspc-es* (*esl* ! *i*) = *EvtSys es* ∨ (∃ *e*′. *is-anonyevt e*′ ∧ *getspc-es* (*esl* ! *i*) = *EvtSeq e*′ (*EvtSys es*))
using *evtsys-allevtseqorevtsys*[*of esl es s x esl1*] **by** *auto*
from *a0* **obtain** *esc1* **and** *s1* **and** *x1* **where** *a2*: *esl* ! *i* = (*esc1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
from *a0* **obtain** *esc2* **and** *s2* **and** *x2* **where** *a3*: *esl* ! *Suc i* = (*esc2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
from *a1 a2 a3* **obtain** *e* **and** *k* **where** *a4*: (*esc1*,*s1*,*x1*)−*es*−*EvtEnt e*♯*k*→(*esc2*,*s2*,*x2*) **by** *auto*
from *a00 a2* **have** *a5*: *esc1* = *EvtSys es* ∨ (∃ *e*′. *is-anonyevt e*′ ∧ *esc1* = *EvtSeq e*′ (*EvtSys es*))
using *getspc-es-def* **by** (*metis fst-conv*)
with *a4* **have** ¬(∃ *e*′. *is-anonyevt e*′ ∧ *esc1* = *EvtSeq e*′ (*EvtSys es*))
apply(*simp add*:*get-actk-def is-anonyevt-def*)
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
with *a5* **have** *esc1* = *EvtSys es* **by** *simp*
with *a2* **have** *getspc-es* (*esl*!*i*) = *EvtSys es* **using** *getspc-es-def* **by** (*metis fst-conv*)
}
then show *?thesis* **by** *auto*
qed

lemma *allentev-isin-basicevts*:
∀ *esl esc s x esl1 e k*. *esl*∈*cpts-es* ∧ *esl* = (*esc*,*s*,*x*)#*esl1* ⟶
(∀ *m*<*length esl* − *1*. (*esl* ! *m* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc m*) ⟶ *e*∈*all-basicevts-es esc*)
proof −
{
fix *esc*
have ∀ *esl s x esl1 e k*. *esl*∈*cpts-es* ∧ *esl* = (*esc*,*s*,*x*)#*esl1* ⟶
(∀ *m*<*length esl* − *1*. (*esl* ! *m* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc m*) ⟶ *e*∈*all-basicevts-es esc*)
proof(*induct esc*)
case (*EvtSeq ev esys*)
assume *a0*: ∀ *esl s x esl1 e k*.
esl ∈ *cpts-es* ∧ *esl* = (*esys*, *s*, *x*) # *esl1* ⟶
(∀ *i*<*length esl* − *1*. (*esl* ! *i* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc i*) ⟶ *e* ∈ *all-basicevts-es esys*)
then have *a1*: ⋀*esl s x esl1 e k*.
esl ∈ *cpts-es* ∧ *esl* = (*esys*, *s*, *x*) # *esl1* ⟹

```
        (∀i<length esl − 1. (esl ! i −es−EvtEnt e♯k→ esl ! Suc i) ⟶ e ∈ all-basicevts-es esys) by auto
{
  fix esl s x esl1 e k
  assume b0: esl ∈ cpts-es ∧ esl = (EvtSeq ev esys, s, x) # esl1
  {
    fix m
    assume c0: m<length esl − 1
      and c1: esl ! m −es−EvtEnt e♯k→ esl ! Suc m
    have e ∈ all-basicevts-es (EvtSeq ev esys)
      proof(cases ∀i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys)
        assume d0: ∀i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys
        with b0 c0 c1 have d1: is-basicevt ev ∧ (∀i. i ≤ m ⟶ getspc-es (esl ! i) = EvtSeq ev esys)
          using evtseq-evtent-befaft[of esl ev esys s x esl1 m] by auto
        then have getspc-es (esl ! m) = EvtSeq ev esys by simp
        with c1 have e = ev using evtent-is-basicevt-inevtseq2 by fastforce
        with d1 show ?thesis using all-basicevts-es.simps(1)
          by (simp add: insertI1)
      next
        assume d0: ¬(∀i. Suc i ≤ length esl ⟶ getspc-es (esl ! i) ≠ esys)
        then have ∃m. Suc m ≤ length esl ∧ getspc-es (esl ! m) = esys by auto
        then obtain m1 where d1: Suc m1 ≤ length esl ∧ getspc-es (esl ! m1) = esys by auto
        then have ∃i. i ≤ m1 ∧ getspc-es (esl ! i) = esys by auto
        with b0 d1 have d2: ∃i. (i ≤ m1 ∧ getspc-es (esl ! i) = esys)
                      ∧ (∀j. j < i ⟶ getspc-es (esl ! j) ≠ esys)
          using evtseq-fst-finish[of esl ev esys m1] getspc-es-def fst-conv nth-Cons' by force
        then obtain n where d3: (n ≤ m1 ∧ getspc-es (esl ! n) = esys)
                            ∧ (∀j. j < n ⟶ getspc-es (esl ! j) ≠ esys)
          by auto
        from b0 d3 have n ≠ 0 by (metis (no-types, lifting) Groups.add-ac(2)
            Suc-n-not-le-n add.right-neutral add-Suc-right esys.size(3) fst-conv
            getspc-es-def le-add1 nth-Cons')
        then have d4:n > 0 by simp

        show ?thesis
          proof(cases m < n)
            assume e0: m < n
            let ?esl0 = take n esl
            from d1 d3 d4 have e1: ?esl0 ∈ cpts-es
              by (metis (no-types, lifting) Suc-le-lessD Suc-pred' b0 cpts-es-take less-trans)

            from b0 d1 d3 d4 obtain esl2 where e2: ?esl0 = (EvtSeq ev esys, s, x) # esl2
              by (simp add: take-Cons')

            from d1 d3 d4 have e3: ∀i. Suc i ≤ length ?esl0 ⟶ getspc-es (?esl0 ! i) ≠ esys
              by (simp add: drop-take leD le-less-linear not-less-eq)

            have e4: Suc m ≠ n
              proof −
              {
                assume f0: Suc m = n
                from d1 d3 d4 e0 have m < length ?esl0 by auto
                with d1 d3 e0 e1 e2 e3 have ∃e'. getspc-es (?esl0 ! m) = EvtSeq e' esys
                  using evtseq-noesys-allevtseq[of ?esl0 ev esys s x esl2] by simp
                then obtain e' where getspc-es (?esl0 ! m) = EvtSeq e' esys by auto
                then obtain s' and x' where f1: ?esl0 ! m = (EvtSeq e' esys, s',x')
                  using getspc-es-def by (metis fst-conv surj-pair)
                moreover
                from d3 obtain s'' and x'' where f2:esl ! n = (esys,s'',x'')
```

using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
moreover
from *d1 d3 e0* **have** *?esl0* ! *m* = *esl* ! *m* **by** *auto*
moreover
with *c1* **have** *f4*:*?esl0* ! *m* −*es*−*EvtEnt e*♯*k*→ *esl* ! *Suc m* **by** *simp*
ultimately have *f3*:(*EvtSeq e′ esys*, *s′*,*x′*)−*es*−*EvtEnt e*♯*k*→(*esys*,*s″*,*x″*) **using** *f0* **by** *simp*
then have *False*
apply(*rule estran.cases*)
apply(*simp add*:*get-actk-def*)
apply(*rule etran.cases*)
apply(*simp add*:*get-actk-def*)+
apply (*metis f3 ent-spec2′ event.inject*(*1*) *evtseq-tran-0-exist-etran noevtent-notran option.distinct*(*1*))
by (*metis f2 f4 f1 ent-spec2′ event.inject*(*1*) *evtent-is-basicevt-inevtseq f0 option.simps*(*3*))
} **then show** *?thesis* **by** *auto*
qed

from *c1 e0 d1 d3 d4 e4* **have** *e5*: *?esl0* ! *m* −*es*−*EvtEnt e*♯*k*→ *?esl0* ! *Suc m*
by (*simp add*: *Suc-lessI*)
from *d1 d3 d4 e0 e4* **have** *m* < *length ?esl0* − *1* **by** *auto*
with *b0 c0 c1 e1 e2 e3 e4 e5* **have** *d1*: *is-basicevt ev* ∧ (∀ *i*. *i* ≤ *m* ⟶ *getspc-es* (*esl* ! *i*) = *EvtSeq ev esys*)
using *evtseq-evtent-befaft*[*of ?esl0 ev esys s x esl2 m*]
by (*smt diff-diff-cancel e0 less-imp-diff-less nth-take*)
then have *getspc-es* (*esl* ! *m*) = *EvtSeq ev esys* **by** *simp*
with *c1* **have** *e* = *ev* **using** *evtent-is-basicevt-inevtseq2* **by** *fastforce*
with *d1* **show** *?thesis* **using** *all-basicevts-es.simps*(*1*)
by (*simp add*: *insertI1*)
next
assume ¬*m* < *n*
then have *e0*: *m* ≥ *n* **by** *auto*
let *?esl0* = *drop n esl*
from *c0 e0* **have** *?esl0* ∈ *cpts-es* **using** *b0 cpts-es-dropi2 length-Cons length-tl less-SucI list.sel*(*3*) **by** *fastforce*
moreover
from *d1 d3* **obtain** *s′* **and** *x′* **and** *esl1* **where** *?esl0* = (*esys*,*s′*,*x′*)#*esl1*
by (*metis* (*no-types*, *hide-lams*) *Cons-nth-drop-Suc getspc-es-def less-le-trans not-less-eq old.prod.exhaust prod.sel*(*1*))
moreover
from *d1 d3 d0 c0 e0* **have** *m* − *n* <*length ?esl0* − *1* **by** *auto*
moreover
from *d1 d3 d0 c0 e0* **have** *esl* ! *m* = *?esl0* ! (*m* − *n*) **by** *auto*
moreover
from *d1 d3 d0 c0 e0* **have** *esl* ! *Suc m* = *?esl0* ! *Suc* (*m* − *n*) **by** *auto*
ultimately have *e* ∈ *all-basicevts-es esys*
using *c1 d1 d3 e0 a1*[*of ?esl0 s′ x′ esl1 e k*] **by** *auto*
then show *?thesis* **using** *all-basicevts-es.simps* **by** *simp*
qed
qed
}
}
then show *?case* **by** *auto*
next
case (*EvtSys es*)
{
fix *esl s x esl1 e k*
assume *b0*: *esl* ∈ *cpts-es* ∧ *esl* = (*EvtSys es*, *s*, *x*) # *esl1*
{

fix *m*
assume *c0*: *m<length esl − 1*
and *c1*: *esl ! m −es−EvtEnt e♯k→ esl ! Suc m*
with *b0* **have** *c00*: *getspc-es (esl!m) = EvtSys es*
using *evtsys-befevtent-isevtsys[of esl es s x esl1]*
Suc-mono length-Cons length-tl list.sel(3) **by** *auto*
from *c0* **obtain** *esc1* **and** *s1* **and** *x1* **where** *c2*: *esl ! m = (esc1,s1,x1)*
using *prod-cases3* **by** *blast*
from *c0* **obtain** *esc2* **and** *s2* **and** *x2* **where** *c3*: *esl ! Suc m = (esc2,s2,x2)*
using *prod-cases3* **by** *blast*
from *c1 c2 c3* **have** *c4*: *(esc1,s1,x1)−es−EvtEnt e♯k→(esc2,s2,x2)* **by** *auto*
with *c00 c2 c3* **have** *c5*: *∃ i∈es. i = e*
using *evtsysent-evtent2[of es s1 x1 e k esc2 s2 x2] getspc-es-def*
by *(metis fst-conv)*
from *c4* **have** *is-basicevt e*
using *evtent-is-basicevt[of esc1 s1 x1 e k esc2 s2 x2] is-basicevt.simps* **by** *auto*
with *c5* **have** *e ∈ all-basicevts-es (EvtSys es)* **using** *all-basicevts-es.simps* **by** *auto*
}
}
then show *?case* **by** *auto*
qed
}
then show *?thesis* **by** *fastforce*
qed

lemma *cmd-impl-evtent-before*:
⟦*c ∝ cs*; *cs k∈cpts-of-es esc s x*; *∀ ef∈all-evts-esspec esc. is-basicevt ef*⟧
⟹ *∀ i. Suc i < length (cs k) ⟶ (∃ cmd. (cs k)!i −es−((Cmd cmd)♯k)→ (cs k)!(Suc i))*
⟶ (∃ m. m < i ∧ (∃ e. (cs k)!m −es−(EvtEnt e♯k)→ (cs k)!(Suc m)))
proof −
assume *p0*: *c ∝ cs*
and *p1*: *cs k∈cpts-of-es esc s x*
and *p2*: *∀ ef∈all-evts-esspec esc. is-basicevt ef*
let *?esl = cs k*
from *p1* **have** *p01*: *?esl ∈ cpts-es ∧ ?esl ! 0 = (esc,s,x)* **by** *(simp add:cpts-of-es-def)*
{
fix *i*
assume *a0*: *Suc i < length ?esl*
and *a1*: *∃ cmd. ?esl!i −es−((Cmd cmd)♯k)→ ?esl!(Suc i)*

then obtain *cmd* **where** *a2*: *?esl!i −es−((Cmd cmd)♯k)→ ?esl!(Suc i)* **by** *auto*
then obtain *esc1* **and** *s1* **and** *x1* **and** *esc2* **and** *s2* **and** *x2* **where** *a3*:
?esl!i = (esc1,s1,x1) ∧ ?esl!Suc i = (esc2,s2,x2)
by *(meson prod-cases3)*
with *a2* **have** *a4*: *∃ e′ es. esc1 = EvtSeq e′ es ∧ is-anonyevt e′*
using *cmd-enable-impl-anonyevt[of esc1 s1 x1 cmd k esc2 s2 x2] is-anonyevt.simps* **by** *auto*
from *p01 p2 a3 a4* **have** *a5*: *i ≠ 0* **by** *(metis all-evts-esspec.simps(1) anonyevt-isnot-basic fst-conv insertI1)*
have *∃ m. m < i ∧ (∃ e. ?esl!m −es−(EvtEnt e♯k)→ ?esl!(Suc m))*
proof−
{
assume *b0*: *¬(∃ m. m < i ∧ (∃ e. ?esl!m −es−(EvtEnt e♯k)→ ?esl!(Suc m)))*
then have *b1*: *∀ j. j < i ⟶ ¬(∃ e. ?esl!j −es−(EvtEnt e♯k)→ ?esl!(Suc j))* **by** *auto*
with *p0 p01 a0 a1 a3 a4* **have** *∀ j < i. ∃ e es. getspc-es (?esl!j) = EvtSeq e es ∧ is-anonyevt e*
using *anonyevtseq-and-noet-impl-allanonyevtseq-bef3[of c cs k i] getspc-es-def*
by *(metis Suc-lessD fst-conv)*
with *a5* **have** *∃ e es. getspc-es (?esl!0) = EvtSeq e es ∧ is-anonyevt e* **by** *simp*
with *p01 p1 p2* **have** *False* **by** *(metis all-evts-esspec.simps(1) anonyevt-isnot-basic getspc-es-def insertI1 prod.sel(1))*

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

lemma *cmd-impl-evtent-before-and-cmds*:
$[\![c \propto cs;\ cs\ k \in cpts\text{-}of\text{-}es\ esc\ s\ x;\ \forall ef \in all\text{-}evts\text{-}esspec\ esc.\ is\text{-}basicevt\ ef]\!]$
$\Longrightarrow \forall i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists cmd.\ (cs\ k)!i\ {-es-}((Cmd\ cmd)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$
$\longrightarrow (\exists m.\ m < i \wedge (\exists e.\ (cs\ k)!m\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ (cs\ k)!(Suc\ m))$
$\wedge (\forall j.\ j > m \wedge j < i \longrightarrow \neg(\exists e.\ (cs\ k)!j\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ (cs\ k)!(Suc\ j))))$
proof −
assume *p0*: $c \propto cs$
and *p1*: $cs\ k \in cpts\text{-}of\text{-}es\ esc\ s\ x$
and *p2*: $\forall ef \in all\text{-}evts\text{-}esspec\ esc.\ is\text{-}basicevt\ ef$
let $?esl = cs\ k$
from *p1* **have** *p01*: $?esl \in cpts\text{-}es \wedge ?esl\ !\ 0 = (esc,s,x)$ **by** (*simp add*:*cpts-of-es-def*)
{
fix *i*
assume *a0*: $Suc\ i < length\ ?esl$
and *a1*: $\exists cmd.\ ?esl!i\ {-es-}((Cmd\ cmd)\sharp k){\rightarrow}\ ?esl!(Suc\ i)$
from *p0 p1 p2 a0 a1* **have** $\exists m.\ m < i \wedge (\exists e.\ ?esl!m\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ ?esl!(Suc\ m))$
using *cmd-impl-evtent-before*[*of c cs k esc s x*] **by** *auto*
then obtain *m* **where** *a2*: $m < i \wedge (\exists e.\ ?esl!m\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ ?esl!(Suc\ m))$ **by** *auto*
with *a0* **have** $\exists m.\ m < i \wedge (\exists e.\ ?esl!m\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ ?esl!(Suc\ m))$
$\wedge (\forall j.\ j > m \wedge j < i \longrightarrow \neg(\exists e.\ ?esl!j\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ ?esl!(Suc\ j)))$
proof(*induct i*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc ii*)
assume *b0*: $Suc\ ii < length\ ?esl \Longrightarrow$
$m < ii \wedge (\exists e.\ ?esl\ !\ m\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ m) \Longrightarrow$
$\exists m{<}ii.\ (\exists e.\ ?esl\ !\ m\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ m) \wedge$
$(\forall j.\ m < j \wedge j < ii \longrightarrow \neg\ (\exists e.\ ?esl\ !\ j\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ j))$
and *b1*: $Suc\ (Suc\ ii) < length\ ?esl$
and *b2*: $m < Suc\ ii \wedge (\exists e.\ ?esl\ !\ m\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ m)$
then show *?case*
proof(*cases* $m = ii$)
assume *c0*: $m = ii$
with *b2* **show** *?case* **using** *not-less-eq* **by** *auto*
next
assume $m \neq ii$
with *b2* **have** *c0*: $m < ii$ **by** *simp*
with *b0 b1 b2* **have** *c1*: $\exists m{<}ii.\ (\exists e.\ ?esl\ !\ m\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ m) \wedge$
$(\forall j.\ m < j \wedge j < ii \longrightarrow \neg\ (\exists e.\ ?esl\ !\ j\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ j))$ **by** *auto*
then obtain *m1* **where** *c2*: $m1{<}ii \wedge (\exists e.\ ?esl\ !\ m1\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ m1) \wedge$
$(\forall j.\ m1 < j \wedge j < ii \longrightarrow \neg\ (\exists e.\ ?esl\ !\ j\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ j))$ **by** *auto*
show *?case*
proof(*cases* $\exists e.\ ?esl\ !\ ii\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ ii$)
assume *d0*: $\exists e.\ ?esl\ !\ ii\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ ii$
then show *?thesis* **using** *lessI not-less-eq* **by** *auto*
next
assume *d0*: $\neg\ (\exists e.\ ?esl\ !\ ii\ {-es-}EvtEnt\ e\sharp k{\rightarrow}\ ?esl\ !\ Suc\ ii)$
with *c2* **show** *?thesis* **by** (*metis less-Suc-eq*)
qed
qed
qed

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

lemma *cur-evt-in-cpts-es*:
$\llbracket c \in cpts\text{-}of\text{-}pes\ (paresys\text{-}spec\ pesrgf)\ s\ x;\ c \propto cs;$
$(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (evtsys\text{-}spec\ (fst\ (pesrgf\ k)))\ s\ x);$
$\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c!j{-}pes{-}actk{\rightarrow}c!Suc\ j);$
$\forall ef \in all\text{-}evts\ pesrgf.\ is\text{-}basicevt\ (E_e\ ef)\rrbracket$
$\Longrightarrow \forall k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists cmd.\ (cs\ k)!i\ {-}es{-}((Cmd\ cmd)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i))$
$\longrightarrow (\exists ef \in all\text{-}evts\text{-}es\ (fst\ (pesrgf\ k)).\ getx\text{-}es\ ((cs\ k)!i)\ k = E_e\ ef)$
proof −
assume *p0*: $c \in cpts\text{-}of\text{-}pes\ (paresys\text{-}spec\ pesrgf)\ s\ x$
and *p1*: $c \propto cs$
and *p2*: $(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (evtsys\text{-}spec\ (fst\ (pesrgf\ k)))\ s\ x)$
and *p3*: $\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c!j{-}pes{-}actk{\rightarrow}c!Suc\ j)$
and *p4*: $\forall ef \in all\text{-}evts\ pesrgf.\ is\text{-}basicevt\ (E_e\ ef)$
{
fix *k i*
assume *a0*: $Suc\ i < length\ (cs\ k)$
and *a1*: $\exists cmd.\ (cs\ k)!i\ {-}es{-}((Cmd\ cmd)\sharp k){\rightarrow}\ (cs\ k)!(Suc\ i)$
from *p4* **have** *a2*: $\forall ef \in all\text{-}evts\text{-}esspec\ (evtsys\text{-}spec\ (fst\ (pesrgf\ k))).\ is\text{-}basicevt\ ef$
using *allevts-es-blto-allevts*[*of pesrgf*]
by (*metis* (*no-types*, *hide-lams*) *DomainE* E_e*-def prod.sel*(*1*) *subsetCE*)
from *p2* **have** *a3*: $cs\ k \in cpts\text{-}of\text{-}es\ (evtsys\text{-}spec\ (fst\ (pesrgf\ k)))\ s\ x$ **by** *simp*
with *p1 a0 a1 a2 a3* **have** $(\exists m.\ m < i \wedge (\exists e.\ cs\ k!m\ {-}es{-}(EvtEnt\ e\sharp k){\rightarrow}\ cs\ k!(Suc\ m))$
$\wedge (\forall j.\ j > m \wedge j < i \longrightarrow \neg(\exists e.\ cs\ k!j\ {-}es{-}(EvtEnt\ e\sharp k){\rightarrow}\ cs\ k!(Suc\ j))))$
using *cmd-impl-evtent-before-and-cmds*[*of c cs k evtsys-spec* (*fst* (*pesrgf k*)) *s x*] **by** *auto*
then obtain *m* **and** *e* **where** *a4*: $m < i \wedge (cs\ k!m\ {-}es{-}(EvtEnt\ e\sharp k){\rightarrow}\ cs\ k!(Suc\ m))$
$\wedge (\forall j.\ j > m \wedge j < i \longrightarrow \neg(\exists e.\ cs\ k!j\ {-}es{-}(EvtEnt\ e\sharp k){\rightarrow}\ cs\ k!(Suc\ j)))$ **by** *auto*
with *p1 p3 a0* **have** *a5*: $\forall j.\ j > m \wedge j \le i \longrightarrow getx\text{-}es\ ((cs\ k)!j)\ k = e$
using *evtent-impl-curevt-in-cpts-es*[*of c cs m k e i*]
by (*smt Suc-lessD Suc-lessI entevt-ines-chg-selfx2 less-trans-Suc not-less*)
with *a4* **have** *a6*: $getx\text{-}es\ ((cs\ k)!i)\ k = e$ **by** *auto*
from *a3* **have** $cs\ k \in cpts\text{-}es \wedge (\exists esl1.\ cs\ k = (evtsys\text{-}spec\ (fst\ (pesrgf\ k)),\ s,\ x)\#esl1)$
using *cpts-of-es-def* **by** (*smt a0 hd-Cons-tl list.size*(*3*) *mem-Collect-eq not-less0 nth-Cons-0*)
with *a0 a4* **have** $e \in all\text{-}basicevts\text{-}es\ (evtsys\text{-}spec\ (fst\ (pesrgf\ k)))$
using *allentev-isin-basicevts* **by** (*smt Suc-lessE diff-Suc-1 le-less-trans less-imp-le-nat*)
with *a6* **have** $\exists ef \in all\text{-}evts\text{-}es\ (fst\ (pesrgf\ k)).\ getx\text{-}es\ ((cs\ k)!i)\ k = E_e\ ef$
using *allbasicevts-es-blto-allevts*[*of evtsys-spec* (*fst* (*pesrgf k*))]
by (*metis* (*no-types*, *hide-lams*) *DomainE* E_e*-def all-evts-same fst-conv set-mp*)
}
then show *?thesis* **by** *auto*
qed

lemma *cur-evt-in-specevts*:
$\llbracket pesl \in cpts\text{-}of\text{-}pes\ (paresys\text{-}spec\ pesf)\ s\ x;$
$\forall j.\ Suc\ j < length\ pesl \longrightarrow (\exists actk.\ pesl!j{-}pes{-}actk{\rightarrow}pesl!Suc\ j);$
$\forall ef \in all\text{-}evts\ pesf.\ is\text{-}basicevt\ (E_e\ ef)\rrbracket \Longrightarrow$
$(\forall k\ i.\ Suc\ i < length\ pesl \longrightarrow (\exists c.\ (pesl!i\ {-}pes{-}((Cmd\ c)\sharp k){\rightarrow}\ pesl!(Suc\ i)))$
$\longrightarrow (\exists ef \in all\text{-}evts\ pesf.\ getx\ (pesl!i)\ k = E_e\ ef)\ )$
proof −
assume *p0*: $pesl \in cpts\text{-}of\text{-}pes\ (paresys\text{-}spec\ pesf)\ s\ x$
and *p1*: $\forall j.\ Suc\ j < length\ pesl \longrightarrow (\exists actk.\ pesl!j{-}pes{-}actk{\rightarrow}pesl!Suc\ j)$
and *p2*: $\forall ef \in all\text{-}evts\ pesf.\ is\text{-}basicevt\ (E_e\ ef)$
then have $\exists cs.\ (\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ ((paresys\text{-}spec\ pesf)\ k)\ s\ x) \wedge pesl \propto cs$
using *par-evtsys-semantics-comp*[*of paresys-spec pesf s x*] **by** *auto*
then obtain *cs* **where** *a1*: $(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ ((paresys\text{-}spec\ pesf)\ k)\ s\ x) \wedge pesl \propto cs$ **by** *auto*

then have *a2*: $\forall k.\ length\ pesl = length\ (cs\ k)$ **by** (*simp add:conjoin-def same-length-def*)
from *a1* **have** *a3*: $\forall k\ j.\ j < length\ pesl \longrightarrow getx\ (pesl!j) = getx\text{-}es\ ((cs\ k)!j)$
by (*simp add:conjoin-def same-state-def*)
{
fix *k i*
assume *b0*: $Suc\ i < length\ pesl$
and *b1*: $\exists c.\ (pesl!i\ {-pes-}((Cmd\ c)\sharp k)\rightarrow\ pesl!(Suc\ i))$
then obtain *c* **where** *b2*: $pesl!i\ {-pes-}((Cmd\ c)\sharp k)\rightarrow\ pesl!(Suc\ i)$ **by** *auto*
from *a1* **have** *b3*: *compat-tran pesl cs* **by** (*simp add:conjoin-def*)
with *b0* **have** *b4*: $\exists t\ k.\ (pesl!i\ {-pes-}(t\sharp k)\rightarrow\ pesl!Suc\ i) \wedge$
$(\forall k\ t.\ (pesl!i\ {-pes-}(t\sharp k)\rightarrow\ pesl!Suc\ i) \longrightarrow (cs\ k!i\ {-es-}(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese}\rightarrow\ cs\ k'!\ Suc\ i)))$
$\vee$
$(((pesl!i)\ {-pese}\rightarrow\ (pesl!Suc\ i)) \wedge (\forall k.\ (((cs\ k)!i)\ {-ese}\rightarrow\ ((cs\ k)!\ Suc\ i))))$
using *compat-tran-def*[*of pesl cs*] **by** *auto*

from *b2* **have** $\exists t\ k1.\ k1 = k \wedge t = Cmd\ c \wedge pesl\ !\ i\ {-pes-}t\sharp k\rightarrow\ pesl\ !\ Suc\ i$ **by** *simp*

then have $\neg(pesl\ !\ i\ {-pese}\rightarrow\ pesl\ !\ Suc\ i)$ **by** (*simp add: pes-tran-not-etran1*)
with *b4* **have** $\exists t\ k.\ (pesl!i\ {-pes-}(t\sharp k)\rightarrow\ pesl!Suc\ i) \wedge$
$(\forall k\ t.\ (pesl!i\ {-pes-}(t\sharp k)\rightarrow\ pesl!Suc\ i) \longrightarrow (cs\ k!i\ {-es-}(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese}\rightarrow\ cs\ k'!\ Suc\ i)))$ **by** *simp*
then obtain *t* **and** *k1* **where** *b5*: $(pesl!i\ {-pes-}(t\sharp k1)\rightarrow\ pesl!Suc\ i) \wedge$
$(\forall k\ t.\ (pesl!i\ {-pes-}(t\sharp k)\rightarrow\ pesl!Suc\ i) \longrightarrow (cs\ k!i\ {-es-}(t\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ {-ese}\rightarrow\ cs\ k'!\ Suc\ i)))$ **by** *auto*
have $cs\ k\ !\ i\ {-es-}((Cmd\ c)\sharp k)\rightarrow\ cs\ k!(Suc\ i)$ **using** *b2 b5* **by** *auto*
with *p0 p1 p2 a1 a2 b0 b1* **have** $\exists ef{\in}all\text{-}evts\text{-}es\ (fst\ (pesf\ k)).\ getx\text{-}es\ ((cs\ k)!i)\ k = E_e\ ef$
using *cur-evt-in-cpts-es*[*of pesl pesf s x cs*] **by** (*metis paresys-spec-def*)
then obtain *ef* **where** $ef{\in}all\text{-}evts\text{-}es\ (fst\ (pesf\ k)) \wedge getx\text{-}es\ ((cs\ k)!i)\ k = E_e\ ef$ **by** *auto*
moreover
have $all\text{-}evts\text{-}es\ (fst\ (pesf\ k)) \subseteq all\text{-}evts\ pesf$ **using** *all-evts-def* **by** *auto*
moreover
from *a2 a3 b0* **have** $getx\text{-}es\ ((cs\ k)!i)\ k = getx\ (pesl!i)\ k$ **by** *auto*
ultimately have $\exists ef{\in}all\text{-}evts\ pesf.\ getx\ (pesl!i)\ k = E_e\ ef$ **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *drop-take-ln*: $[\![l1 = drop\ i\ (take\ j\ l);\ length\ l1 > n]\!] \Longrightarrow j > i + n$
by (*metis add.commute add-lessD1 leI length-drop length-take less-diff-conv less-imp-add-positive min.absorb2 nat-le-linear take-all*)

lemma *drop-take-eq*: $[\![l1 = drop\ i\ (take\ j\ l);\ j \leq length\ l;\ length\ l1 = n;\ n > 0]\!] \Longrightarrow j = i + n$
by *simp*

lemma *drop-take-sametrace*[*rule-format*]: $[\![l1 = drop\ i\ (take\ j\ l)]\!] \Longrightarrow \forall m < length\ l1.\ l1\ !\ m = l\ !\ (i + m)$
by (*simp add: less-imp-le-nat*)

lemma *act-cpts-evtsys-sat-guar-curevt-gen0-new2*[*rule-format*]:
$[\![\vdash (esspc::('l,'k,'s)\ rgformula\text{-}ess)\ sat_s\ [pre,\ rely,\ guar,\ post]]\!]$
$\Longrightarrow \forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq pre \wedge Rely\ k \subseteq rely \wedge guar \subseteq Guar\ k \wedge post \subseteq Post\ k \longrightarrow$
$c{\in}cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c{\in}assume\text{-}pes(pre1,\ rely1) \longrightarrow$
$(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ (cs\ k){\in}commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$

(∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*) ⟶
evtsys-spec esspc = *EvtSys es* ∧ *EvtSys es* = *getspc-es* (*cs k*!*0*) ⟶
(∀ *e*∈*all-evts-es esspc*. *is-basicevt* (E_e *e*)) ⟶
(∀ *e*∈*all-evts-es esspc*. *the* (*evtrgfs* (E_e *e*)) = *snd e*) ⟶
(∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c*!*j*−*pes*−*actk*→*c*!*Suc j*)) ⟶
(∀ *i*. *Suc i* < *length* (*cs k*) ∧ ((*cs k*)!*i* −*es*−((*Cmd cmd*)♯*k*)→ (*cs k*)!(*Suc i*))
⟶ (*gets-es* ((*cs k*)!*i*), *gets-es* ((*cs k*)!(*Suc i*)))∈$Guar_f$ (*the* (*evtrgfs* (*getx-es* ((*cs k*)!*i*) *k*))))

apply(*rule rghoare-es.induct*[*of esspc pre rely guar post*])
apply *simp*
apply *simp*
proof −
{
fix *esf prea relya guara posta*
assume *p0*: ⊢ (*esspc*::(′*l*,′*k*,′*s*) *rgformula-ess*) sat_s [*pre*, *rely*, *guar*, *post*]
and *b5*: ∀ *ef*∈(*esf*::(′*l*,′*k*,′*s*) *rgformula-e set*). ⊢ E_e *ef* sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
and *b6*: ∀ *ef*∈*esf*. *prea* ⊆ Pre_e *ef*
and *b7*: ∀ *ef*∈*esf*. *relya* ⊆ $Rely_e$ *ef*
and *b8*: ∀ *ef*∈*esf*. $Guar_e$ *ef* ⊆ *guara*
and *b9*: ∀ *ef*∈*esf*. $Post_e$ *ef* ⊆ *posta*
and *b10*: ∀ *ef1 ef2*. *ef1* ∈ *esf* ∧ *ef2* ∈ *esf* ⟶ $Post_e$ *ef1* ⊆ Pre_e *ef2*
and *b11*: *stable prea relya*
and *b12*: ∀ *s*. (*s*, *s*) ∈ *guara*
{
fix *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*
assume *b1*: *Pre k* ⊆ *prea*
and *b2*: *Rely k* ⊆ *relya*
and *b3*: *guara* ⊆ *Guar k*
and *b4*: *posta* ⊆ *Post k*
and *p0*: *c* ∈ *cpts-of-pes pes s x*
and *p1*: *c* ∝ *cs*
and *p8*: *c* ∈ *assume-pes* (*pre1*, *rely1*)
and *p2*: (∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*)
and *p3*: ∀ *k*. (*cs k*)∈*commit-es*(*Guar k*, *Post k*)
and *a5*: (∀ *k*. *pre1* ⊆ *Pre k*)
and *a6*: (∀ *k*. *rely1* ⊆ *Rely k*)
and *p4*: (∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*)
and *a0*: *evtsys-spec* (*rgf-EvtSys esf*) = *EvtSys es* ∧ *EvtSys es* = *getspc-es* (*cs k* ! *0*)
∧ (∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *is-basicevt* (E_e *e*))
∧ (∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*)
and *p6*: (∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*))
then have *p30*: (∀ *k*. *cs k* ∈ *assume-es* (*Pre k*, *Rely k*))
using *conjoin-comm-imp-rely*[*of pre1 Pre rely1 Rely Guar cs Post c pes s x*] **by** *auto*
with *p3* **have** *p31*: (∀ *k*. *cs k* ∈ *commit-es* (*Guar k*, *Post k*))
by (*meson IntI contra-subsetD cpts-of-es-def es-validity-def p2*)

from *p1* **have** *p11*: ∀ *k*. *length* (*cs k*) = *length c* **by** (*simp add*:*conjoin-def same-length-def*)
from *p2* **have** *p12*: ∀ *k*. *cs k* ∈ *cpts-es* **using** *cpts-of-es-def mem-Collect-eq* **by** *fastforce*
with *p11* **have** *c* ≠ *Nil* **using** *cpts-es-not-empty length-0-conv* **by** *auto*
then have *p13*: *length c* > *0* **by** *auto*

let *?esl* = *cs k*
let *?esys* = *EvtSys es*

from *p1 p2 a0* **have** *a8*: *?esl* ∈ *cpts-es* ∧ *?esl*!*0* = (*EvtSys es*,*s*,*x*)
by (*simp add*: *cpts-of-es-def eq-fst-iff getspc-es-def*)

then obtain *esll* **where** *a81*: *?esl* = (*EvtSys es*,*s*,*x*)#*esll*
by (*metis hd-Cons-tl length-greater-0-conv nth-Cons-0 p11 p13*)

{
  fix *i*
  assume *a3*: *Suc i* $<$ *length* (*cs k*)
    and *a4*: *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i*
  have (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ *Guar$_f$* (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*)))
    proof(*cases* ∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) = *EvtSys es*)
        assume *c0*: ∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) = *EvtSys es*
        with *a3* **have** *getspc-es* (*?esl* ! *i*) = *EvtSys es* ∧ *getspc-es* (*?esl* ! *Suc i*) = *EvtSys es*
          by *auto*
        with *a4* **show** *?thesis* **using** *evtsys-not-eq-in-tran-aux1* **by** *fastforce*
      next
        assume *c0*: ¬(∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) = *EvtSys es*)
        then obtain *m* **where** *c1*: *Suc m* ≤ *length ?esl* ∧ *getspc-es* (*?esl* ! *m*) ≠ *EvtSys es*
          by *auto*
        from *a8* **have** *c2*: *getspc-es* (*?esl*!*0*) = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
        from *c1* **have** ∃ *i*. *i* ≤ *m* ∧ *getspc-es* (*?esl* ! *i*) ≠ *EvtSys es* **by** *auto*
        with *a8 c1 c2* **have** ∃ *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*)
          using *evtsys-fst-ent* **by** *blast*
        then obtain *n* **where** *c3*: (*n* $<$ *m* ∧ *getspc-es* (*?esl* ! *n*) = *EvtSys es*
                  ∧ *getspc-es* (*?esl* ! *Suc n*) ≠ *EvtSys es*)
                  ∧ (∀ *j*. *j* $<$ *n* ⟶ *getspc-es* (*?esl* ! *j*) = *EvtSys es*) **by** *auto*
        have *c4*: *i* ≥ *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 ¬(∃ *t*. *?esl*!*i* −*es*−*t*→ *?esl*!*Suc i*)
              using *evtsys-not-eq-in-tran getspc-es-def* **by** (*metis surjective-pairing*)
            with *a4* **have** *False* **by** *simp*
          }
          then show *?thesis* **using** *leI* **by** *auto*
          qed

        let *?esl1* = *drop n ?esl*
        let *?eslh* = *take* (*Suc n*) *?esl*
        from *c1 c3* **have** *c5*: *length ?esl1* ≥ *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* (*?esl1*!*0*) = *EvtSys es* ∧ *getspc-es* (*?esl1*!*1*) ≠ *EvtSys es*
          by *force*

        from *a3 a8 c1 c3 c4* **have** *c7*: *?esl1* ∈ *cpts-es* **using** *cpts-es-dropi*
            by (*metis* (*no-types*, *lifting*) *drop-0 dual-order.strict-trans*
                le-0-eq le-SucE le-imp-less-Suc zero-induct)
        from *c5 c6 c7* **have** ∃ *s x ev s1 x1 xs*.
          ?esl1 = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
            using *fst-esys-snd-eseq-exist* **by** *blast*
        then obtain *s0* **and** *x0* **and** *e* **and** *s1* **and** *x1* **and** *xs* **where** *c8*:
            ?esl1 = (*EvtSys es*, *s0*, *x0*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*
        with *c3* **have** *c3-1*: (∀ *j*≤*n*. *getspc-es* (*cs k* ! *j*) = *EvtSys es*) **using** *getspc-es-def*
          using *antisym-conv2* **by** *blast*
        let *?elst* = *tl* (*parse-es-cpts-i2 ?esl1 es* [[]])
        from *c8 c7* **have** *c9*: *concat ?elst* = *?esl1* **using** *parse-es-cpts-i2-concat3* **by** *metis*

from *a0* **have** *c13*: *es* = *Domain esf* **using** *evtsys-spec-evtsys* **by** *auto*
from *b5* **have** *c14*: $\forall$ *i*$\in$*esf*. $\models$ E_e *i* sat_e [Pre_e *i*, $Rely_e$ *i*, $Guar_e$ *i*, $Post_e$ *i*]
  by (*simp add*: *rgsound-e*)

let *?RG* = λ*e*. *SOME rg*. (*e*,*rg*)$\in$*esf*
from *c13* **have** *c131*: $\forall$ *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 *c13 c14* **have** *c16*: $\forall$ *ef*$\in$*es*. $\models$ *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 comp-apply fst-conv snd-conv someI-ex*)
moreover
from *b1 b6* **have** *c17*: $\forall$ *j*$\in$*es*. *prea* $\subseteq$ *?Pre j* **using** Pre_e*-def c131 comp-def* **by** *metis*
moreover
from *b2 b7* **have** *c18*: $\forall$ *j*$\in$*es*. *Rely k* $\subseteq$ *?Rely j* **using** $Rely_e$*-def c131 comp-def* **by** (*metis subsetCE subsetI*)
moreover
from *b3 b8* **have** *c19*: $\forall$ *j*$\in$*es*. *?Guar j* $\subseteq$ *Guar k* **using** $Guar_e$*-def c131 comp-def* **by** (*metis subsetCE subsetI*)
moreover
from *b4 b9* **have** *c20*: $\forall$ *j*$\in$*es*. *?Post j* $\subseteq$ *Post k* **using** *c131 comp-def*
  by (*metis* $Post_e$*-def contra-subsetD subsetI*)
moreover
from *b5 b10* **have** *c21*: $\forall$ *ef1 ef2*. *ef1* $\in$ *es* $\wedge$ *ef2* $\in$ *es* $\longrightarrow$ *?Post ef1* $\subseteq$ *?Pre ef2*
  by (*metis* $Post_e$*-def* Pre_e*-def c131 comp-apply*)
moreover
from *c1 c3-1 p30* **have** *c24*: *?esl1*$\in$*assume-es* (*prea*, *Rely k*)
  proof(*cases n* = *0*)
    assume *d0*: *n* = *0*
    from *b1 p30* **have** *?esl*$\in$*assume-es*(*prea*,*Rely k*)
      using *assume-es-imp*[*of Pre k prea Rely k Rely k ?esl*] **by** *blast*
    with *d0* **show** *?thesis* **by** *auto*
  next
    assume *d0*: *n* $\neq$ *0*
    from *b1 b2 p30* **have** *?esl*$\in$*assume-es*(*prea*,*relya*)
      using *assume-es-imp*[*of Pre k prea Rely k relya ?esl*] **by** *blast*
    then have *?eslh* $\in$ *assume-es*(*prea*,*relya*)
      using *assume-es-take-n*[*of Suc n ?esl prea relya*] *d0 c1 c3* **by** *auto*
    moreover
    from *c3* **have** $\forall$ *i*<*length ?eslh*. *getspc-es* (*?eslh*!*i*) = *EvtSys es*
      proof –
        from *c3* **have** $\forall$ *i*. *Suc i*<*length ?eslh* $\longrightarrow$ *getspc-es* (*?eslh*!*i*) = *EvtSys es*
          using *Suc-le-lessD length-take less-antisym less-imp-le-nat*
          min.bounded-iff nth-take **by** *auto*
        moreover
        from *c3* **have** *getspc-es* (*last ?eslh*) = *EvtSys es*
          by (*metis* (*no-types*, *lifting*) *a3 c4 dual-order.strict-trans*
            getspc-es-def last-snoc le-imp-less-Suc take-Suc-conv-app-nth)
        ultimately show *?thesis*
          by (*metis Suc-lessI diff-Suc-1 last-conv-nth*
            length-greater-0-conv nat.distinct(*1*) *p11 p13 take-eq-Nil*)
      qed
    ultimately have $\forall$ *i*<*length ?eslh*. *gets-es* (*?eslh*!*i*) $\in$ *prea*
      using *b11 pre-trans*[*of ?eslh prea relya EvtSys es*] **by** *blast*

moreover
from *c1 c3* **have** *d1*: *Suc n* ≤ *length ?esl* **by** *auto*
moreover
then have *n* < *length ?eslh* **by** *auto*
ultimately have *gets-es* (*?eslh* ! *n*) ∈ *prea* **by** *simp*
moreover
from *d1* **have** *?eslh* ! *n* = *?esl1* ! *0* **by** (*simp add*: *c8 nth-via-drop*)
ultimately have *gets-es* (*?esl* ! *n*) ∈ *prea* **by** *simp*
with *p30 d1* **show** *?thesis* **using** *assume-es-drop-n*[*of n ?esl Pre k Rely k prea*] **by** *auto*
qed
ultimately
have *ri*[*rule-format*]: ∀ *i*. *Suc i* < *length ?elst* ⟶
(∃ *m*∈*es*. *?elst*!*i*@[(*?elst*!*Suc i*)!*0*]∈*commit-es*(*?Guar m*,*?Post m*)
∧ *gets-es* ((*?elst*!*Suc i*)!*0*) ∈ *?Post m*
∧ (∃ *k*. (*?elst*!*i*@[(*?elst*!*Suc i*)!*0*])!*0*−*es*−(*EvtEnt m*)♯*k*→(*?elst*!*i*@[(*?elst*!*Suc i*)!*0*])!*1*))
using *EventSys-sound-aux-i*[*of es ?Pre ?Rely ?Guar ?Post*
prea Rely k Guar k Post k ?esl1 s0 x0 e s1 x1 xs ?elst]
c7 c8 **by** *force*

from *c16 c17 c18 c19 c20 c21 c24*
have *ri-forall*[*rule-format*]:
∀ *i*. *Suc i* < *length ?elst* ⟶
(∀ *ei*∈*es*. (∃ *k*. (*?elst*!*i*@[(*?elst*!*Suc i*)!*0*])!*0*−*es*−(*EvtEnt ei*)♯*k*→(*?elst*!*i*@[(*?elst*!*Suc i*)!*0*])!*1*)
⟶ *?elst*!*i*@[(*?elst*!*Suc i*)!*0*]∈*commit-es*(*?Guar ei*,*?Post ei*)
∧ *gets-es* ((*?elst*!*Suc i*)!*0*) ∈ *?Post ei*)
using *EventSys-sound-aux-i-forall*[*of es ?Pre ?Rely ?Guar ?Post*
prea Rely k Guar k Post k ?esl1 s0 x0 e s1 x1 xs ?elst]
c7 c8 **by** *simp*

from *c16 c17 c18 c19 c20 c21 b10 c7 c8 c24*
have *rl-forall*: ∀ *ei*∈*es*. (∃ *k*. (*last ?elst*)!*0*−*es*−(*EvtEnt ei*)♯*k*→(*last ?elst*)!*1*)
⟶ *last ?elst*∈*commit-es*(*?Guar ei*,*?Post ei*)
using *EventSys-sound-aux-last-forall*[*of es ?Pre ?Rely ?Guar ?Post*
prea Rely k Guar k Post k ?esl1 s0 x0 e s1 x1 xs ?elst] **by** *simp*

from *c16 c17 c18 c19 c20 c21 b10 c7 c8 c24*
have *rl*: ∃ *m*∈*es*. *last ?elst*∈*commit-es*(*?Guar m*,*?Post m*)
∧ (∃ *k*. (*last ?elst*)!*0*−*es*−(*EvtEnt m*)♯*k*→(*last ?elst*)!*1*)
using *EventSys-sound-aux-last*[*of es ?Pre ?Rely ?Guar ?Post*
prea Rely k Guar k Post k ?esl1 s0 x0 e s1 x1 xs ?elst] **by** *simp*

from *c8 c7* **have** *no-mident*[*rule-format*]: ∀ *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*[*of ?esl1 es s0 x0 e s1 x1 xs parse-es-cpts-i2 ?esl1 es* [[]]]
by *simp*

from *c8 c7* **have** *no-mident-i*[*rule-format*]: ∀ *i*. *Suc i* < *length ?elst* ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*?elst*!*i*@[*?elst*!*Suc i*!*0*]) ∧
getspc-es ((*?elst*!*i*@[*?elst*!*Suc i*!*0*])!*j*) = *EvtSys es* ∧ *getspc-es* ((*?elst*!*i*@[*?elst*!*Suc i*!*0*])!*Suc j*) ≠
EvtSys es)
by (*metis parse-es-cpts-i2-noent-mid-i*)

have *in-cpts-i*[*rule-format*]: ∀ *i*. *Suc i* < *length ?elst* ⟶ (*?elst*!*i*)@[*?elst*!*Suc i*!*0*] ∈*cpts-es*

using *parse-es-cpts-i2-in-cptes-i*[*of ?esl1 es s0 x0 e s1 x1 xs ?elst*] *c7 c8*
by *simp*

have *in-cpts-last*: *last ?elst* ∈*cpts-es*
using *parse-es-cpts-i2-in-cptes-last*[*of ?esl1 es s0 x0 e s1 x1 xs ?elst*] *c7 c8*
by *simp*

then have *in-cpts-last1*: *?elst* ! (*length ?elst* − *1*) ∈ *cpts-es*
by (*metis c7 c9 concat.simps*(*1*) *cpts-es-not-empty last-conv-nth*)

from *c5 c8 c7* **have** *len-start-elst*[*rule-format*]:
∀ *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*[*of ?esl1 es s0 x0 e s1 x1 xs parse-es-cpts-i2 ?esl1 es* [[]]]
by *fastforce*

then have *c30*: ∀ *i*. *Suc i* < *length ?esl1*
⟶ (∃ *k j*. (*Suc k* < *length ?elst* ∧ *Suc j* < *length* (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*]) ∧
?esl1!*i* = (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*])!*j* ∧ *?esl1*!*Suc i* = (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*])!*Suc j*)
∨ (*Suc k* = *length ?elst* ∧ *Suc j* < *length* (*?elst*!*k*) ∧
?esl1!*i* = *?elst*!*k*!*j* ∧ *?esl1*!*Suc i* = *?elst*!*k*!*Suc j*))
using *c9 concat-list-lemma*[*of ?esl1 ?elst*] **by** *fastforce*

from *p12 a3* **have** *c33*[*rule-format*]: ∀ *i*. *i* < *length ?esl*
⟶ *getspc-es* (*?esl*!*i*) = *EvtSys es* ∨ (∃ *e*. *getspc-es* (*?esl*!*i*) = *EvtSeq e* (*EvtSys es*) ∧ *is-anonyevt e*)
using *evtsys-all-es-in-cpts-anony*[*of ?esl es*]
c2 gr0I gr-implies-not0 **by** *blast*

from *a3 c4* **have** *c34*: *?esl*!*i* = *?esl1*!(*i* − *n*)
using *Suc-lessD add-diff-inverse-nat leD less-imp-le-nat nth-drop* **by** *auto*
from *a3 c4* **have** *c340*: *?esl*!*Suc i* = *?esl1*!(*Suc* (*i* − *n*))
using *Suc-lessD add-diff-inverse-nat leD less-imp-le-nat nth-drop* **by** *auto*
from *a3 c4* **have** *Suc* (*i* − *n*) < *length ?esl1*
by (*simp add*: *Suc-diff-le diff-less-mono le-SucI*)
with *c30* **have** ∃ *k j*. (*Suc k* < *length ?elst* ∧ *Suc j* < *length* (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*]) ∧
?esl1!(*i* − *n*) = (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*])!*j* ∧ *?esl1*!*Suc* (*i* − *n*) = (*?elst*!*k*@[(*?elst*!*Suc k*)!*0*])!*Suc j*)
∨ (*Suc k* = *length ?elst* ∧ *Suc j* < *length* (*?elst*!*k*) ∧
?esl1!(*i* − *n*) = *?elst*!*k*!*j* ∧ *?esl1*!*Suc* (*i* − *n*) = *?elst*!*k*!*Suc j*)
by *auto*
then obtain *kk* **and** *j* **where** *c35*: (*Suc kk* < *length ?elst* ∧ *Suc j* < *length* (*?elst*!*kk*@[(*?elst*!*Suc kk*)!*0*]) ∧
?esl1!(*i* − *n*) = (*?elst*!*kk*@[(*?elst*!*Suc kk*)!*0*])!*j* ∧ *?esl1*!*Suc* (*i* − *n*) = (*?elst*!*kk*@[(*?elst*!*Suc kk*)!*0*])!*Suc j*)
∨ (*Suc kk* = *length ?elst* ∧ *Suc j* < *length* (*?elst*!*kk*) ∧
?esl1!(*i* − *n*) = *?elst*!*kk*!*j* ∧ *?esl1*!*Suc* (*i* − *n*) = *?elst*!*kk*!*Suc j*)
by *auto*
let *?elstk* = *?elst*!*kk*@[(*?elst*!*Suc kk*)!*0*]
have *c36*: *length ?elstk* > *2* **using** *len-start-elst*[*of kk*] *c35*
by (*metis Suc-lessD le-imp-less-Suc length-append-singleton lessI*)

let *?elstl* = *?elst*!*kk*
have *c37*: *length ?elstl* ≥ *2* **using** *len-start-elst*[*of kk*] *c35*
by (*metis Suc-lessD lessI*)

from *c35* **have** *c38*: *Suc kk* ≤ *length ?elst* **using** *less-or-eq-imp-le* **by** *blast*

from *c38* **have** ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*?elst*!*kk*) ∧
getspc-es (*?elst*!*kk*!*j*) = *EvtSys es* ∧ *getspc-es* (*?elst*!*kk*!*Suc j*) ≠ *EvtSys es*)

```
    using no-mident by auto
  then have d1: ∀j. j > 0 ∧ Suc j < length (?elst!kk) ⟶ getspc-es ((?elst!kk) ! j) = EvtSys es
        ⟶ getspc-es ((?elst!kk) ! Suc j) = EvtSys es using noevtent-inmid-eq by auto

  have d43: length ?esl = n + length ?esl1
      using ‹Suc (i − n) < length (drop n (cs k))› by auto

  from c35 show ?thesis
    proof
      assume d0: (Suc kk < length ?elst ∧ Suc j < length ?elstk ∧
              ?esl1!(i − n) = ?elstk!j ∧ ?esl1!Suc (i − n) = ?elstk!Suc j)

      have d01: j ≠ 0
        proof
          assume e0: j = 0
          with len-start-elst[of kk] have e1: getspc-es (?elstk!j) = EvtSys es
               ∧ getspc-es (?elstk!Suc j) ≠ EvtSys es
            by (metis (no-types, hide-lams) One-nat-def Suc-1 Suc-le-lessD c34 d0 less-imp-le-nat nth-append)
          moreover
          from a4 have ¬(∃ess. getspc-es (?esl ! i) = EvtSys ess)
            using cmd-enable-impl-notesys2[of ?esl ! i cmd k ?esl ! Suc i] by simp
          moreover
          from d0 have ?esl!i = ?elstk!j
            by (simp add: c34)
          ultimately show False by simp
        qed


      have d1-1: ∀ii. ii > 0 ∧ Suc ii < length ?elstk
          ⟶ ¬(∃e. (?elstk!ii) −es−((EvtEnt e)♯k)→ (?elstk!(Suc ii)))
        proof −
        {
          fix ii
          assume e0: ii > 0 ∧ Suc ii < length ?elstk
          have ¬(∃e. (?elstk!ii) −es−((EvtEnt e)♯k)→ (?elstk!(Suc ii)))
            proof(cases getspc-es (?elstk!ii) = EvtSys es)
              assume f0: getspc-es (?elstk!ii) = EvtSys es
              with d1 d0 have getspc-es (?elstk!(Suc ii)) = EvtSys es
                by (smt Suc-lessI Suc-less-eq c7 c8 e0 length-append-singleton
                  nth-append nth-append-length parse-es-cpts-i2-start-aux)
              with f0 show ?thesis
                using evtsys-not-eq-in-tran-aux1 by fastforce
            next
              assume f0: getspc-es (?elstk!ii) ≠ EvtSys es
              from d0 e0 in-cpts-i[of kk] have f1: ?elstk ∈ cpts-es by simp
              moreover
              from d0 f1 len-start-elst[of kk] have
                length ?elstk > 0 ∧ getspc-es (?elstk!0) = EvtSys es
                by (metis (no-types, lifting) Suc-lessD cpts-es-not-empty length-greater-0-conv
                    list.size(3) not-numeral-le-zero nth-append)
              ultimately have ∃e. getspc-es (?elstk!ii) = EvtSeq e (EvtSys es)
                                ∧ is-anonyevt e
                using evtsys-all-es-in-cpts-anony[of ?elstk es] e0 f0 Suc-lessD by blast
              then show ?thesis using incpts-es-eseq-not-evtent[of ?elstk ii]
                in-cpts-i[of kk] d0 e0 by blast
            qed
        }
        then show ?thesis by auto
```

```
  qed

have d2: getspc-es (?elstk!0) = EvtSys es ∧ getspc-es (?elstk!1) ≠ EvtSys es
  using len-start-elst[of 0] by (metis (no-types, hide-lams) One-nat-def
    Suc-1 Suc-le-lessD Suc-lessD d0 len-start-elst nth-append)

from c9 d0 len-start-elst
  have ∃ si ti. si = length (concat (take kk ?elst)) ∧ ti = Suc (length (concat (take (Suc kk) ?elst))) ∧
    si ≤ length ?esl1 ∧ ti < length ?esl1 ∧ si < ti ∧ drop si (take ti ?esl1) = ?elstk
  using concat-list-lemma-withnextfst3[of ?esl1 ?elst kk]
    Suc-1 Suc-le-lessD by presburger
then obtain si and ti where d4: si = length (concat (take kk ?elst))
    ∧ ti = Suc (length (concat (take (Suc kk) ?elst)))
    ∧ si ≤ length ?esl1 ∧ ti < length ?esl1
    ∧ si < ti ∧ drop si (take ti ?esl1) = ?elstk by auto
then have d42: si + (length ?elstk) = ti
  using drop-take-eq[of ?elstk si ti ?esl1 length ?elstk] c36
    by (metis cpts-es-not-empty d0 in-cpts-i length-greater-0-conv less-imp-le-nat)

from d4 have ti < length ?esl1 by simp
with d43 have d41: n + ti < length ?esl by simp

from d4 have d5: ?elstk = drop (si+n) (take (ti+n) ?esl)
  by (metis (no-types, lifting) drop-drop take-drop)
then have d6: ?elstk!0 = ?esl!(si+n)
  by (metis (no-types, lifting) Nat.add-0-right
      append-is-Nil-conv append-take-drop-id drop-eq-Nil
      leI nat-le-linear not-Cons-self2 nth-append nth-drop)

from d5 have ?elstk!1 = drop (si+n) (take (ti+n) ?esl) ! 1 by simp
moreover
from d0 d5 have drop (si+n) (take (ti+n) ?esl) ! 1 = ?esl!(Suc (si+n))
  by (metis (no-types, lifting) One-nat-def Suc-eq-plus1 Suc-leI Suc-lessI
    add-diff-cancel-left' append-is-Nil-conv append-take-drop-id
    drop-eq-Nil length-drop not-less nth-append nth-drop zero-less-Suc)
ultimately have d7: ?elstk!1 = ?esl!(Suc (si+n)) by simp


from c36 d4 have d71: ti > si + 2 using drop-take-ln[of ?elstk si ti ?esl1 2] by fastforce
with c1 c3 d4 have d72: Suc (si+n) < length ?esl
  proof −
    have si + 2 < length (cs k) − n
      using ‹ti < length (drop n (cs k))› d71 by auto
    then have Suc (Suc (si + n)) < length (cs k)
      by linarith
    then show ?thesis
      by (metis Suc-le-lessD order.strict-implies-order)
  qed

with p1 d2 d6 d7 have ∃ e. ?esl!(si+n) −es−((EvtEnt e)♯k)→ ?esl!(Suc (si+n))
  using entevt-in-conjoin-cpts[of c cs si+n k es] by simp
then obtain ente where d8: ?esl!(si+n) −es−((EvtEnt ente)♯k)→ ?esl!(Suc (si+n)) by auto
with d2 d6 have ∃ ei∈es. ente = ei
  using evtsysent-evtent3[of ?esl!(si+n) ente k ?esl!(Suc (si+n)) es] by auto
then obtain ei where d9: ei∈es ∧ ente = ei by auto

from ri-forall[of kk ei] d0 d6 d7 d8 d9
  have d10: ?elstk ∈ commit-es(?Guar ei,?Post ei) by auto
```

from *d0* **have** *d11*: *cs k* ! *i* = *?elstk* ! *j* **by** (*simp add*: *c34*)
moreover
from *d0* **have** *d12*: *cs k* ! *Suc i* = *?elstk* ! *Suc j* **by** (*simp add*: *c340*)
ultimately have *d13*: *?elstk* ! *j* −*es*−*Cmd cmd*♯*k*→ *?elstk* ! *Suc j* **using** *a4* **by** *auto*

have *d14*: (*gets-es* (*?elstk* ! *j*), *gets-es* (*?elstk* ! *Suc j*)) ∈ *?Guar ei*
  proof −
    from *d10* **have** ∀ *i*. *Suc i*<*length ?elstk* ⟶
        (∃ *t*. *?elstk*!*i* −*es*−*t*→ *?elstk*!(*Suc i*)) ⟶
          (*gets-es* (*?elstk*!*i*), *gets-es* (*?elstk*!*Suc i*)) ∈ *?Guar ei*
      by (*simp add*:*commit-es-def*)
    with *d0 d13* **show** *?thesis* **by** *auto*
  qed

with *d11 d12* **have** *d15*: (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ *?Guar ei*
  by *simp*

from *d0 no-mident-i*[*of kk*] **have** ¬(∃ *m*. *m* > *0* ∧ *Suc m* < *length ?elstk* ∧
      getspc-es (*?elstk*!*m*) = *EvtSys es* ∧ *getspc-es* (*?elstk*!*Suc m*) ≠ *EvtSys es*)
  by *simp*
then have *d16*[*rule-format*]: ∀ *m*. *m* > *0* ∧ *Suc m* < *length ?elstk*
    ⟶ ¬(*getspc-es* (*?elstk*!*m*) = *EvtSys es* ∧ *getspc-es* (*?elstk*!*Suc m*) ≠ *EvtSys es*)
  by *auto*
have *d17*: ∀ *m*. *m* > (*si* + *n*) ∧ *m* < *ti* + *n* − *1* ⟶
      ¬(*getspc-es* (*?esl*!*m*) = *EvtSys es* ∧ *getspc-es* (*?esl*!*Suc m*) ≠ *EvtSys es*)
  proof −
  {
    fix *m*
    assume *e0*: *m* > (*si* + *n*) ∧ *m* < *ti* + *n* − *1*
    then have *e1*: *m* − (*n*+*si*) > *0* **by** *auto*
    moreover
    have *e2*: *Suc* (*m* − (*n*+*si*)) < *length ?elstk*
      proof −
        from *e0* **have** *m* − (*n* + *si*) < *ti* − *si* − *1* **by** *auto*
        then have *Suc* (*m* − (*n* + *si*)) < *ti* − *si* **by** *auto*
        with *d42* **show** *?thesis* **by** *auto*
      qed
    ultimately have ¬(*getspc-es* (*?elstk*!(*m*−(*n*+*si*))) = *EvtSys es*
        ∧ *getspc-es* (*?elstk*!*Suc* (*m*−(*n*+*si*))) ≠ *EvtSys es*)
      using *d16*[*of m* − (*n*+*si*)] **by** *simp*
    moreover
    from *e1 e2 d5* **have** *?esl*!*m* = *?elstk*!(*m* − (*n*+*si*))
      using *drop-take-sametrace*[*of ?elstk si*+*n ti*+*n ?esl m* − (*n*+*si*)] **by** *auto*
    moreover
    from *e1 e2 d5* **have** *?esl*!*Suc m* = *?elstk*!*Suc* (*m* − (*n*+*si*))
      using *drop-take-sametrace*[*of ?elstk si*+*n ti*+*n ?esl Suc* (*m* − (*n*+*si*))] **by** *auto*
    ultimately have ¬(*getspc-es* (*?esl*!*m*) = *EvtSys es* ∧ *getspc-es* (*?esl*!*Suc m*) ≠ *EvtSys es*)
      by *simp*
  }
  then show *?thesis* **by** *auto*
  qed
have *d18*: ∀ *m*. *m* > (*si* + *n*) ∧ *m* < *ti* + *n* − *1* ⟶
      ¬(∃ *e*. *?esl*!*m* −*es*−((*EvtEnt e*)♯*k*)→ *?esl*!*Suc m*)
  proof −
  {
    fix *m*

assume *e0*: $m > (si + n) \wedge m < ti + n - 1$
with *d17* **have** $\neg(getspc\text{-}es\ (?esl!m) = EvtSys\ es \wedge getspc\text{-}es\ (?esl!Suc\ m) \neq EvtSys\ es)$
by *auto*
with *p1 a8 a81 d41 e0* **have** $\neg(\exists\, e.\ ?esl!m\ -es-((EvtEnt\ e)\sharp k)\rightarrow\ ?esl!Suc\ m)$
using *notentevt-in-conjoin-cpts[of c cs m k es] evtsys-allevtseqorevtsys[of ?esl es s x esll]*
by *auto*
}
then show *?thesis* **by** *auto*
qed

from *d71* **have** $Suc\ (si + n) < ti + n - 1$
using *Suc-eq-plus1 add.assoc add-2-eq-Suc add-diff-cancel-right′ less-diff-conv* **by** *linarith*
moreover
from *d41* **have** $Suc\ (ti + n - 1) < length\ (cs\ k)$ **using** *calculation d41* **by** *linarith*
ultimately
have *d19[rule-format]*:$\forall\, m.\ m > (si + n) \wedge m \leq (ti + n - 1) \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = ente$
using *evtent-impl-curevt-in-cpts-es[of c cs si + n k ente ti + n − 1]*
d18 p1 p6 d8 d41 d71 d72 **by** *auto*
from *d0 d42* **have** $si + n + j \leq ti + n - 1$ **by** *auto*
with *d19[of si + n + j] d01* **have** $getx\text{-}es\ ((cs\ k)!(si + n + j))\ k = ente$ **by** *auto*
with *d11 d5* **have** $getx\text{-}es\ ((cs\ k)!i)\ k = ente$
by (*metis Suc-lessD d0 drop-take-sametrace*)
moreover
from *a0* **have** $the\ (evtrgfs\ (ei)) = (?RG\ ei)$
using *all-evts-es-esys d9 c13 c131* **by** (*metis Domain.cases* E_e*-def prod.sel(1) snd-conv someI-ex*)
moreover
from *d9 c13 c131* **have** $?Guar\ ei = Guar_f\ (?RG\ ei)$ **by** (*simp add*: $Guar_f$*-def*)
ultimately show *?thesis* **using** *d15 d9* **by** *simp*
next
assume *d0*: $Suc\ kk = length\ ?elst \wedge Suc\ j < length\ ?elstl \wedge$
$?esl1!(i - n) = ?elstl!j \wedge ?esl1!Suc\ (i - n) = ?elstl!Suc\ j$
have *d01*: $j \neq 0$
proof
assume *e0*: $j = 0$
with *len-start-elst[of kk]* **have** *e1*: $getspc\text{-}es\ (?elstl!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (?elstl!Suc\ j) \neq EvtSys\ es$
using *One-nat-def d0 lessI* **by** *fastforce*
moreover
from *a4* **have** $\neg(\exists\, ess.\ getspc\text{-}es\ (?esl\ !\ i) = EvtSys\ ess)$
using *cmd-enable-impl-notesys2[of ?esl ! i cmd k ?esl ! Suc i]* **by** *simp*
moreover
from *d0* **have** $?esl!i = ?elstl!j$
by (*simp add*: *c34*)
ultimately show *False* **by** *simp*
qed

have *d1-1*: $\forall\, ii.\ ii > 0 \wedge Suc\ ii < length\ ?elstl$
$\longrightarrow \neg(\exists\, e.\ (?elstl!ii)\ -es-((EvtEnt\ e)\sharp k)\rightarrow\ (?elstl!(Suc\ ii)))$
proof −
{
fix *ii*
assume *e0*: $ii > 0 \wedge Suc\ ii < length\ ?elstl$
have $\neg(\exists\, e.\ (?elstl!ii)\ -es-((EvtEnt\ e)\sharp k)\rightarrow\ (?elstl!(Suc\ ii)))$
proof(*cases* $getspc\text{-}es\ (?elstl!ii) = EvtSys\ es$)
assume *f0*: $getspc\text{-}es\ (?elstl!ii) = EvtSys\ es$
with *d1 d0* **have** $getspc\text{-}es\ (?elstl!(Suc\ ii)) = EvtSys\ es$
by (*smt Suc-lessI Suc-less-eq c7 c8 e0 length-append-singleton*

nth-append nth-append-length parse-es-cpts-i2-start-aux)
with *f0* **show** *?thesis*
using *evtsys-not-eq-in-tran-aux1* **by** *fastforce*
next
assume *f0*: *getspc-es* (*?elstl*!*ii*) ≠ *EvtSys es*
from *d0* **have** *f1*: *Suc kk* = *length ?elst* **by** *simp*
with *in-cpts-last1* **have** *f2*: *?elstl* ∈ *cpts-es*
by (*metis diff-Suc-1*)
moreover
from *f1 len-start-elst*[*of kk*] **have**
length ?elstl > *0* ∧ *getspc-es* (*?elstl*!*0*) = *EvtSys es*
using *Suc-le-lessD c38 d0 gr-implies-not0* **by** *blast*
ultimately have ∃ *e*. *getspc-es* (*?elstl*!*ii*) = *EvtSeq e* (*EvtSys es*)
∧ *is-anonyevt e*
using *evtsys-all-es-in-cpts-anony*[*of ?elstl es*] *e0 f0 Suc-lessD* **by** *blast*
then show *?thesis* **using** *incpts-es-eseq-not-evtent*[*of ?elstl ii*]
in-cpts-last1 f2 d0 e0 **by** *blast*
qed
}
then show *?thesis* **by** *auto*
qed

from *d0* **have** *d2*: *getspc-es* (*?elstl*!*0*) = *EvtSys es* ∧ *getspc-es* (*?elstl*!*1*) ≠ *EvtSys es*
using *len-start-elst*[*of kk*] **by** *auto*

from *c9 d0 len-start-elst*[*of kk*]
have ∃ *si ti*. *si* = *length* (*concat* (*take kk ?elst*)) ∧ *ti* = *length* (*concat* (*take* (*Suc kk*) *?elst*)) ∧
si ≤ *length ?esl1* ∧ *ti* ≤ *length ?esl1* ∧ *si* < *ti* ∧ *drop si* (*take ti ?esl1*) = *?elstl*
using *concat-list-lemma3*[*of ?esl1 ?elst kk*]
using *Suc-1 Suc-le-lessD c38* **by** *presburger*

then obtain *si* **and** *ti* **where** *d4*: *si* = *length* (*concat* (*take kk ?elst*))
∧ *ti* = *length* (*concat* (*take* (*Suc kk*) *?elst*))
∧ *si* ≤ *length ?esl1* ∧ *ti* ≤ *length ?esl1* ∧ *si* < *ti*
∧ *drop si* (*take ti ?esl1*) = *?elstl* **by** *auto*
then have *d42*: *si* + (*length ?elstl*) = *ti*
using *drop-take-eq*[*of ?elstl si ti ?esl1 length ?elstl*] *c37*
by (*metis d0 gr-implies-not0 not-gr0*)

from *d0 d4* **have** *ti* = *length ?esl1* **by** (*simp add*: *c38 c9*)
with *d43* **have** *d41*: *n* + *ti* = *length ?esl* **by** *simp*

from *d4* **have** *d5*: *?elstl* = *drop* (*si*+*n*) (*take* (*ti*+*n*) *?esl*)
by (*metis* (*no-types*, *lifting*) *drop-drop take-drop*)
then have *d6*: *?elstl*!*0* = *?esl*!(*si*+*n*)
by (*metis Cons-nth-drop-Suc* ‹*ti* = *length* (*drop n* (*cs k*))› *d4*
drop-drop drop-eq-Nil linorder-not-less nth-Cons-0 take-all)

from *d5* **have** *?elstl*!*1* = *drop* (*si*+*n*) (*take* (*ti*+*n*) *?esl*) ! *1* **by** *simp*
moreover
from *d0 d5* **have** *drop* (*si*+*n*) (*take* (*ti*+*n*) *?esl*) ! *1* = *?esl*!(*Suc* (*si*+*n*))
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-eq-plus1 Suc-leI Suc-lessI*
add-diff-cancel-left′ append-is-Nil-conv append-take-drop-id
drop-eq-Nil length-drop not-less nth-append nth-drop zero-less-Suc)
ultimately have *d7*: *?elstl*!*1* = *?esl*!(*Suc* (*si*+*n*)) **by** *simp*

from *c37 d4* **have** *d71*: *ti* > *si* + *2* **using** *drop-take-ln*[*of ?elstl si ti ?esl1 2*]
by (*metis Suc-inject d0 d01 le-eq-less-or-eq less-2-cases nat.distinct*(*1*))

```
with c1 c3 d4 have d72: Suc (si+n) < length ?esl
  using Suc-leI Suc-n-not-le-n add.commute add-2-eq-Suc' add-Suc-right
    d41 leI le-antisym less-trans-Suc nat-add-left-cancel-less
    nat-le-linear not-less by linarith

with p1 d2 d6 d7 have ∃ e. ?esl!(si+n) −es−((EvtEnt e)♯k)→ ?esl!(Suc (si+n))
  using entevt-in-conjoin-cpts[of c cs si+n k es] by simp
then obtain ente where d8: ?esl!(si+n) −es−((EvtEnt ente)♯k)→ ?esl!(Suc (si+n)) by auto
with d2 d6 have ∃ ei∈es. ente = ei
  using evtsysent-evtent3[of ?esl!(si+n) ente k ?esl!(Suc (si+n)) es] by auto
then obtain ei where d9: ei∈es ∧ ente = ei by auto

from d0 d6 d7 d8 d9
  have d10: ?elstl ∈ commit-es(?Guar ei,?Post ei)
    by (metis c7 c9 concat.simps(1) cpts-es-not-empty diff-Suc-1 last-conv-nth rl-forall)

from d0 have d11: cs k ! i = ?elstl ! j by (simp add: c34)
moreover
from d0 have d12: cs k ! Suc i = ?elstl ! Suc j by (simp add: c340)
ultimately have d13: ?elstl ! j −es−Cmd cmd♯k→ ?elstl ! Suc j using a4 by auto

have d14: (gets-es (?elstl ! j), gets-es (?elstl ! Suc j)) ∈ ?Guar ei
  proof −
    from d10 have ∀ i. Suc i<length ?elstl ⟶
            (∃ t. ?elstl!i −es−t→ ?elstl!(Suc i)) ⟶
               (gets-es (?elstl!i), gets-es (?elstl!Suc i)) ∈ ?Guar ei
      by (simp add:commit-es-def)
    with d0 d13 show ?thesis by auto
  qed

with d11 d12 have d15: (gets-es (cs k ! i), gets-es (cs k ! Suc i)) ∈ ?Guar ei
  by simp

from d0 no-mident[of kk] have ¬(∃ m. m > 0 ∧ Suc m < length ?elstl ∧
           getspc-es (?elstl!m) = EvtSys es ∧ getspc-es (?elstl!Suc m) ≠ EvtSys es)
  by simp
then have d16[rule-format]: ∀ m. m > 0 ∧ Suc m < length ?elstl
    ⟶ ¬(getspc-es (?elstl!m) = EvtSys es ∧ getspc-es (?elstl!Suc m) ≠ EvtSys es)
  by auto
have d17: ∀ m. m > (si + n) ∧ m < ti + n − 1 ⟶
           ¬(getspc-es (?esl!m) = EvtSys es ∧ getspc-es (?esl!Suc m) ≠ EvtSys es)
  proof −
  {
    fix m
    assume e0: m > (si + n) ∧ m < ti + n − 1
    then have e1: m − (n+si) > 0 by auto
    moreover
    have e2: Suc (m − (n+si)) < length ?elstl
      proof −
        from e0 have m − (n + si) < ti − si − 1 by auto
        then have Suc (m − (n + si)) < ti − si by auto
        with d42 show ?thesis by auto
      qed
    ultimately have ¬(getspc-es (?elstl!(m−(n+si))) = EvtSys es
        ∧ getspc-es (?elstl!Suc (m−(n+si))) ≠ EvtSys es)
      using d16[of m − (n+si)] by simp
    moreover
    from e1 e2 d5 have ?esl!m = ?elstl!(m − (n+si))
```

using *drop-take-sametrace*[*of ?elstl si*+*n ti*+*n ?esl m* − (*n*+*si*)] **by** *auto*
moreover
from *e1 e2 d5* **have** *?esl*!*Suc m* = *?elstl*!*Suc* (*m* − (*n*+*si*))
using *drop-take-sametrace*[*of ?elstl si*+*n ti*+*n ?esl Suc* (*m* − (*n*+*si*))] **by** *auto*
ultimately have ¬(*getspc-es* (*?esl*!*m*) = *EvtSys es* ∧ *getspc-es* (*?esl*!*Suc m*) ≠ *EvtSys es*)
by *simp*
}
then show *?thesis* **by** *auto*
qed

have *d18*: ∀ *m*. *m* > (*si* + *n*) ∧ *m* < *ti* + *n* − *1* ⟶
¬(∃ *e*. *?esl*!*m* −*es*−((*EvtEnt e*)♯*k*)→ *?esl*!*Suc m*)
proof −
{
fix *m*
assume *e0*: *m* > (*si* + *n*) ∧ *m* < *ti* + *n* − *1*
with *d17* **have** ¬(*getspc-es* (*?esl*!*m*) = *EvtSys es* ∧ *getspc-es* (*?esl*!*Suc m*) ≠ *EvtSys es*)
by *auto*
with *p1 a8 a81 d41 e0* **have** ¬(∃ *e*. *?esl*!*m* −*es*−((*EvtEnt e*)♯*k*)→ *?esl*!*Suc m*)
using *notentevt-in-conjoin-cpts*[*of c cs m k es*] *evtsys-allevtseqorevtsys*[*of ?esl es s x esll*]
by *auto*
}
then show *?thesis* **by** *auto*
qed

from *d71* **have** *Suc* (*si* + *n*) < *ti* + *n* − *1*
using *Suc-eq-plus1 add.assoc add-2-eq-Suc add-diff-cancel-right′ less-diff-conv* **by** *linarith*
moreover
from *d41* **have** *Suc* (*ti* + *n* − *1*) = *length* (*cs k*) **using** *calculation d41* **by** *linarith*
ultimately
have *d19*[*rule-format*]:∀ *m*. *m* > (*si* + *n*) ∧ *m* ≤ (*ti* + *n* − *1*) ⟶ *getx-es* ((*cs k*)!*m*) *k* = *ente*
using *evtent-impl-curevt-in-cpts-es1*[*of c cs si* + *n k ente ti* + *n* − *1*]
d18 p1 p6 d8 d41 d71 d72 **by** *auto*
from *d0 d42* **have** *si* + *n* + *j* ≤ *ti* + *n* − *1* **by** *auto*
with *d19*[*of si* + *n* + *j*] *d01* **have** *getx-es* ((*cs k*)!(*si* + *n* + *j*)) *k* = *ente* **by** *auto*
with *d11 d5* **have** *getx-es* ((*cs k*)!*i*) *k* = *ente*
by (*metis Suc-lessD d0 drop-take-sametrace*)
moreover
from *a0* **have** *the* (*evtrgfs* (*ei*)) = (*?RG ei*)
using *all-evts-es-esys d9 c13 c131* **by** (*metis Domain.cases* E_e*-def prod.sel*(*1*) *snd-conv someI-ex*)
moreover
from *d9 c13 c131* **have** *?Guar ei* = $Guar_f$ (*?RG ei*) **by** (*simp add*: $Guar_f$*-def*)
ultimately show *?thesis* **using** *d15 d9* **by** *simp*
qed
qed
}
then have ∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))) **by** *auto*
}
then show ∀ *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*.
Pre k ⊆ *prea* ∧ *Rely k* ⊆ *relya* ∧ *guara* ⊆ *Guar k* ∧ *posta* ⊆ *Post k* ⟶
c ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*) ⟶
(∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*) ⟶
(∀ *k*. (*cs k*)∈*commit-es*(*Guar k*, *Post k*)) ⟶
(∀ *k*. *pre1* ⊆ *Pre k*) ⟶
(∀ *k*. *rely1* ⊆ *Rely k*) ⟶
(∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*) ⟶
evtsys-spec (*rgf-EvtSys esf*) = *EvtSys es* ∧ *EvtSys es* = *getspc-es* (*cs k* ! *0*) ⟶

$(\forall e{\in}all\text{-}evts\text{-}es\ (rgf\text{-}EvtSys\ esf).\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e{\in}all\text{-}evts\text{-}es\ (rgf\text{-}EvtSys\ esf).\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ {-pes-actk\rightarrow}\ c\ !\ Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ {-es-Cmd\ cmd\sharp k\rightarrow}\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$ **by** *fastforce*
}
next
{
fix *prea pre′ relya rely′ guar′ guara post′ posta esys*
assume $p0{:} \vdash (esspc{::}('l,'k,'s)\ rgformula\text{-}ess)\ sat_s\ [pre,\ rely,\ guar,\ post]$
and $p1{:}\ prea \subseteq pre'$
and $p2{:}\ relya \subseteq rely'$
and $p3{:}\ guar' \subseteq guara$
and $p4{:}\ post' \subseteq posta$
and $p5{:} \vdash esys\ sat_s\ [pre',\ rely',\ guar',\ post']$
and $p6[rule\text{-}format]{:}\ \forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq pre' \wedge Rely\ k \subseteq rely' \wedge guar' \subseteq Guar\ k \wedge post' \subseteq Post\ k \longrightarrow$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1) \longrightarrow$
$(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ (cs\ k){\in}commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ esys = EvtSys\ es \wedge EvtSys\ es = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e{\in}all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e{\in}all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ {-pes-actk\rightarrow}\ c\ !\ Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ {-es-Cmd\ cmd\sharp k\rightarrow}\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$
{
fix *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*
assume $a0{:}\ Pre\ k \subseteq prea \wedge Rely\ k \subseteq relya \wedge guara \subseteq Guar\ k \wedge posta \subseteq Post\ k$
and $a1{:}\ c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1)$
and $a2{:}\ (\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x)$
and $a3{:}\ \forall k.\ (cs\ k){\in}commit\text{-}es(Guar\ k,\ Post\ k)$
and $a5{:}\ (\forall k.\ pre1 \subseteq Pre\ k)$
and $a6{:}\ (\forall k.\ rely1 \subseteq Rely\ k)$
and $a7{:}\ (\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k)$
and $a8{:}\ evtsys\text{-}spec\ esys = EvtSys\ es \wedge EvtSys\ es = getspc\text{-}es\ (cs\ k\ !\ 0)$
and $a9{:}\ (\forall e{\in}all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e))$
and $a10{:}\ (\forall e{\in}all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e)$
and $a11{:}\ (\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ {-pes-actk\rightarrow}\ c\ !\ Suc\ j))$
from *a0 p1 p2 p3 p4* **have** $Pre\ k \subseteq pre' \wedge Rely\ k \subseteq rely' \wedge guar' \subseteq Guar\ k \wedge post' \subseteq Post\ k$ **by** *auto*
with *a1 a2 a3 a5 a6 a7 a8 a9 a10 a11 p1 p2 p3 p4 p6[of Pre k Rely Guar Post c pes s x cs pre1 rely1]*
have $\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ {-es-Cmd\ cmd\sharp k\rightarrow}\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k)))$ **by** *force*
}
then show $\forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq prea \wedge Rely\ k \subseteq relya \wedge guara \subseteq Guar\ k \wedge posta \subseteq Post\ k \longrightarrow$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1) \longrightarrow$
$(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ (cs\ k){\in}commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ esys = EvtSys\ es \wedge EvtSys\ es = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e{\in}all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e{\in}all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$

$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ -pes-actk\rightarrow\ c\ !\ Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$ **by** *fastforce*

}
qed

lemma *act-cpts-evtseq-sat-guar-curevt-fstseg-new2*[*rule-format*]:
assumes *b51*: $\vdash (E_e\ ef)\ sat_e\ [Pre_e\ ef,\ Rely_e\ ef,\ Guar_e\ ef,\ Post_e\ ef]$
and *b52*: $\vdash (fst\ esf)\ sat_s\ [Pre_f\ (snd\ esf),\ Rely_f\ (snd\ esf),\ Guar_f\ (snd\ esf),\ Post_f\ (snd\ esf)]$
and *b6*: $pre = Pre_e\ ef$
and *b7*: $post = Post_f\ (snd\ esf)$
and *b8*: $rely \subseteq Rely_e\ ef$
and *b9*: $rely \subseteq Rely_f\ (snd\ esf)$
and *b10*: $Guar_e\ ef \subseteq guar$
and *b11*: $Guar_f\ (snd\ esf) \subseteq guar$
and *b12*: $Post_e\ ef \subseteq Pre_f\ (snd\ esf)$
and *b1*: $Pre\ k \subseteq pre$
and *b2*: $Rely\ k \subseteq rely$
and *b3*: $guar \subseteq Guar\ k$
and *b4*: $post \subseteq Post\ k$
and *p0*: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x$
and *p1*: $c \propto cs$
and *p8*: $c \in assume\text{-}pes(pre1,\ rely1)$
and *p2*: $\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x$
and *p16*: $\forall k.\ (cs\ k) \in commit\text{-}es(Guar\ k,\ Post\ k)$
and *p9*: $\forall k.\ pre1 \subseteq Pre\ k$
and *p10*: $\forall k.\ rely1 \subseteq Rely\ k$
and *p4*: $\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k$
and *a5*: $evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf) = getspc\text{-}es\ (cs\ k\ !\ 0)\ \wedge$
$(\forall i.\ Suc\ i \leq length\ (cs\ k) \longrightarrow getspc\text{-}es\ ((cs\ k)\ !\ i) \neq evtsys\text{-}spec\ (fst\ esf))$
and *a2*: $\forall e \in all\text{-}evts\text{-}es\ (rgf\text{-}EvtSeq\ ef\ esf).\ is\text{-}basicevt\ (E_e\ e)$
and *a01*: $\forall e \in all\text{-}evts\text{-}es\ (rgf\text{-}EvtSeq\ ef\ esf).\ the\ (evtrgfs\ (E_e\ e)) = snd\ e$
and *p6*: $\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ ((c\ !\ j)\ -pes-actk\rightarrow\ (c\ !\ Suc\ j)))$
shows $\forall i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k\ !\ i)\ -es-(Cmd\ cmd)\sharp k\rightarrow\ (cs\ k\ !\ Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k)))$
proof −
from *p1* **have** *p11*[*rule-format*]: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add:conjoin-def same-length-def*)
from *p2* **have** *p12*: $\forall k.\ cs\ k \in cpts\text{-}es$ **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*

from *p16 p0 p1 p2 p4 p8 p9 p10* **have** *p14*: $\forall k.\ (cs\ k) \in assume\text{-}es(Pre\ k,\ Rely\ k)$
using *conjoin-comm-imp-rely* **by** (*metis* (*mono-tags*, *lifting*))
{
fix *i*
let *?esys* = *evtsys-spec* (*rgf-EvtSeq ef esf*)
let *?esl* = *cs k*

assume *a3*: $Suc\ i < length\ ?esl$
and *a4*: $(?esl!i\ -es-((Cmd\ cmd)\sharp k)\rightarrow\ ?esl!(Suc\ i))$

from *a5* **have** $\exists e\ es\ ess.\ ?esys = EvtSeq\ e\ es \wedge getspc\text{-}es\ (cs\ k\ !\ 0) = EvtSeq\ e\ es$
using *evtsys-spec-evtseq*[*of ef esf*] **by** *fastforce*
then obtain *e* **and** *es* **where** *a6*: $?esys = EvtSeq\ e\ es \wedge getspc\text{-}es\ (cs\ k\ !\ 0) = EvtSeq\ e\ es$ **by** *auto*

from *p2 a6* **have** *a8*: $?esl \in cpts\text{-}es \wedge ?esl!0 = (EvtSeq\ e\ es,s,x)$

using *cpts-of-es-def*[*of pes k s x*]
by (*metis* (*mono-tags*, *lifting*) *fst-conv getspc-es-def mem-Collect-eq*)
then obtain *esl1* **where** *a9*: *?esl* = (*EvtSeq e es*,*s*,*x*)#*esl1*
by (*metis Suc-pred length-Suc-conv nth-Cons-0 p11 p13*)

from *a6* **have** *b17*: E_e *ef* = *e* **using** *evtsys-spec-evtseq* **by** *simp*
from *a6* **have** *b18*: *evtsys-spec* (*fst esf*) = *es* **using** *evtsys-spec-evtsys* **by** *simp*

have *b19*: *ef*∈*all-evts-es* (*rgf-EvtSeq ef esf*)
using *all-evts-es-seq*[*of ef esf*] **by** *simp*

from *a5 b18* **have** *c0*: ∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) ≠ *es* **by** *simp*
with *a8* **have** ∃ *el*. (*el* ∈ *cpts-of-ev e s x* ∧ *length ?esl* = *length el* ∧ *e-eqv-einevtseq ?esl el es*)
by (*simp add*: *evtseq-nfin-samelower cpts-of-es-def*)
then obtain *el* **where** *c1*: *el* ∈ *cpts-of-ev e s x* ∧ *length ?esl* = *length el* ∧ *e-eqv-einevtseq ?esl el es*
by *auto*
from *p14* **have** *?esl* ∈ *assume-es*(*Pre k*, *Rely k*) **by** *simp*
with *b1 b2 b6 b8* **have** *?esl* ∈ *assume-es*(Pre_e *ef*, $Rely_e$ *ef*)
by (*metis assume-es-imp equalityE*)
with *c1* **have** *c2*: *el* ∈ *assume-e*(Pre_e *ef*, $Rely_e$ *ef*)
using *e-eqv-einevtseq-def*[*of ?esl el es*] *assume-es-def assume-e-def*
by (*smt Suc-leI a3 eetran-eqconf1 eqconf-esetran less-or-eq-imp-le less-trans-Suc mem-Collect-eq old.prod.case zero-less-Suc*)
with *b51 b17 c1* **have** *c3*: *el* ∈ *commit-e*($Guar_e$ *ef*, $Post_e$ *ef*)
by (*meson Int-iff contra-subsetD evt-validity-def rgsound-e*)

from *a3 c1* **have** *c4*: *getspc-es* (*?esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)

from *a3 c1* **have** *c5*: *getspc-es* (*?esl* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)

from *a4* **have** *getspc-es* (*?esl* ! *i*) ≠ *getspc-es* (*?esl* ! *Suc i*)
using *evtsys-not-eq-in-tran-aux getspc-es-def* **by** (*metis surjective-pairing*)

with *c4 c5* **have** *getspc-e* (*el* ! *i*) ≠ *getspc-e* (*el* ! *Suc i*) **by** *simp*
with *a3 c1* **have** ∃ *t*. (*el* ! *i*) −*et*−*t*→ (*el* ! *Suc i*)
using *cpts-of-ev-def notran-confeqi* **by** *fastforce*

with *a3 c1 c3* **have** *c6*: (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*))∈$Guar_e$ *ef* **by** (*simp add*:*commit-e-def*)

from *p2 a5* **have** *b0*: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *pes k*
using *cpts-of-es-def*[*of pes k s x*] *getspc-es-def*[*of cs k* ! *0*] **by** *force*

from *a2* **have** ∀ *ef*∈*all-evts-esspec* (*evtsys-spec* (*rgf-EvtSeq ef esf*)). *is-basicevt ef*
using *evtsys-spec-evtseq*[*of ef esf*] *all-evts-same*[*of rgf-EvtSeq ef esf*]
by (*metis DomainE* E_e*-def prod.sel*(*1*))
with *p1 p2 a6 a2 a3 a4 b0* **have** ∃ *ie*. *ie* < *i* ∧ (∃ *e*. (*cs k*)!*ie* −*es*−(*EvtEnt e*♯*k*)→ (*cs k*)!(*Suc ie*)) ∧ (∀ *j*. *j* > *ie* ∧ *j* < *i* ⟶ ¬(∃ *e*. (*cs k*)!*j* −*es*−(*EvtEnt e*♯*k*)→ (*cs k*)!(*Suc j*)))
using *cmd-impl-evtent-before-and-cmds*[*of c cs k evtsys-spec* (*rgf-EvtSeq ef esf*) *s x*] **by** *auto*
then obtain *ie* **and** *ev* **where** *c4*: *ie* < *i* ∧ ((*cs k*)!*ie* −*es*−(*EvtEnt ev*♯*k*)→ (*cs k*)!(*Suc ie*)) ∧ (∀ *j*. *j* > *ie* ∧ *j* < *i* ⟶ ¬(∃ *e*. (*cs k*)!*j* −*es*−(*EvtEnt e*♯*k*)→ (*cs k*)!(*Suc j*))) **by** *auto*
with *p1 p6 a3* **have** ∀ *m*. *m* > *ie* ∧ *m* ≤ *i* ⟶ *getx-es* ((*cs k*)!*m*) *k* = *ev*
using *evtent-impl-curevt-in-cpts-es2*[*of c cs ie k ev i*] **by** *auto*
with *c4* **have** *c7*: *getx-es* ((*cs k*)!*i*) *k* = *ev* **by** *simp*

have *is-basicevt e* **using** *a2 b0 b17* **by** *auto*

from *a3 a8 a9 c0 c4* **have** $\forall i.\ i \leq ie \longrightarrow$ *getspc-es* (*?esl* ! *i*) = *EvtSeq e es*
using *evtseq-evtent-befaft*[*of ?esl e es s x esl1 ie*]
by (*smt Suc-diff-1 Suc-lessD Suc-less-eq less-trans-Suc p11 p13*)

with *c4* **have** *c8*: *ev* = *e* **by** (*metis evtent-is-basicevt-inevtseq2 leI*)

from *a3 c1 c6* **have** (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) $\in$ $Guar_e$ *ef*
using *e-eqv-einevtseq-def*[*of ?esl el es*] *Suc-leI less-imp-le-nat* **by** *fastforce*
moreover
from *a01 b17 b19 c7 c8* **have** $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))) = $Guar_e$ *ef*
using $Guar_f$*-def* $Guar_e$*-def* **by** *metis*
ultimately have (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) $\in$ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))) **by** *simp*

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

lemma *act-cpts-evtseq-sat-guar-curevt-fstseg-new2-withlst* [*rule-format*]:
assumes *b51*: $\vdash$ (E_e *ef*) sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
and *b52*: $\vdash$ (*fst esf*) sat_s [Pre_f (*snd esf*), $Rely_f$ (*snd esf*), $Guar_f$ (*snd esf*), $Post_f$ (*snd esf*)]
and *b6*: *pre* = Pre_e *ef*
and *b7*: *post* = $Post_f$ (*snd esf*)
and *b8*: *rely* $\subseteq$ $Rely_e$ *ef*
and *b9*: *rely* $\subseteq$ $Rely_f$ (*snd esf*)
and *b10*: $Guar_e$ *ef* $\subseteq$ *guar*
and *b11*: $Guar_f$ (*snd esf*) $\subseteq$ *guar*
and *b12*: $Post_e$ *ef* $\subseteq$ Pre_f (*snd esf*)
and *b1*: *Pre k* $\subseteq$ *pre*
and *b2*: *Rely k* $\subseteq$ *rely*
and *b3*: *guar* $\subseteq$ *Guar k*
and *b4*: *post* $\subseteq$ *Post k*
and *p0*: *c*$\in$*cpts-of-pes pes s x*
and *p1*: *c* $\propto$ *cs*
and *p8*: *c*$\in$*assume-pes*(*pre1*, *rely1*)
and *p2*: $\forall k.$ (*cs k*) $\in$ *cpts-of-es* (*pes k*) *s x*
and *p16*: $\forall k.$ (*cs k*)$\in$*commit-es*(*Guar k*, *Post k*)
and *p9*: $\forall k.$ *pre1* $\subseteq$ *Pre k*
and *p10*: $\forall k.$ *rely1* $\subseteq$ *Rely k*
and *p4*: $\forall k\ j.\ j \neq k \longrightarrow$ *Guar j* $\subseteq$ *Rely k*
and *a5*: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *getspc-es* (*cs k* ! *0*) $\wedge$
($\forall i.$ *Suc i* < *length* (*cs k*) $\longrightarrow$ *getspc-es* ((*cs k*) ! *i*) $\neq$ *evtsys-spec* (*fst esf*)) $\wedge$
getspc-es(*last* (*cs k*)) = *evtsys-spec* (*fst esf*)
and *a2*: $\forall$ *e*$\in$*all-evts-es* (*rgf-EvtSeq ef esf*). *is-basicevt* (E_e *e*)
and *a01*: $\forall$ *e*$\in$*all-evts-es* (*rgf-EvtSeq ef esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*
and *p6*: $\forall j.$ *Suc j* < *length c* $\longrightarrow$ ($\exists$ *actk*. ((*c* ! *j*) $-pes-$*actk*$\rightarrow$ (*c* ! *Suc j*)))
shows ($\forall i.$ *Suc i* < *length* (*cs k*) $\wedge$ ((*cs k* ! *i*) $-es-$(*Cmd cmd*)$\sharp$*k*$\rightarrow$ (*cs k* ! *Suc i*)) $\longrightarrow$
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) $\in$ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))))
proof −
from *p1* **have** *p11*[*rule-format*]: $\forall k.$ *length* (*cs k*) = *length c* **by** (*simp add:conjoin-def same-length-def*)
from *p2* **have** *p12*: $\forall k.$ *cs k* $\in$ *cpts-es* **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*

from *p16 p0 p1 p2 p4 p8 p9 p10* **have** *p14*: $\forall k.$ (*cs k*) $\in$ *assume-es*(*Pre k*, *Rely k*)
using *conjoin-comm-imp-rely* **by** (*metis* (*mono-tags*, *lifting*))
{

fix *i*
let *?esys* = *evtsys-spec* (*rgf-EvtSeq ef esf*)
let *?esl* = *cs k*
let *?n* = *length ?esl*
let *?eslh* = *take* (*?n* − *1*) *?esl*
assume *a3*: *Suc i* < *length ?esl*
  and *a4*: (*?esl*!*i* −*es*−((*Cmd cmd*)♯*k*)→ *?esl*!(*Suc i*))

from *a5* **have** ∃ *e es ess*. *?esys* = *EvtSeq e es* ∧ *getspc-es* (*cs k* ! *0*) = *EvtSeq e es*
  using *evtsys-spec-evtseq*[*of ef esf*] **by** *fastforce*
then obtain *e* **and** *es* **where** *a6*: *?esys* = *EvtSeq e es* ∧ *getspc-es* (*cs k* ! *0*) = *EvtSeq e es* **by** *auto*

from *p2 a6* **have** *a8*: *?esl* ∈ *cpts-es* ∧ *?esl*!*0* = (*EvtSeq e es*,*s*,*x*)
  using *cpts-of-es-def*[*of pes k s x*]
    by (*metis* (*mono-tags*, *lifting*) *fst-conv getspc-es-def mem-Collect-eq*)
then obtain *esl1* **where** *a9*: *?esl* = (*EvtSeq e es*,*s*,*x*)#*esl1*
  by (*metis Suc-pred length-Suc-conv nth-Cons-0 p11 p13*)

from *a5* **have** *a10*: *?n* > *1* **using** *a3* **by** *linarith*

from *a8 a10* **have** *a81*: *?eslh* ∈ *cpts-es*
  by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc butlast-conv-take cpts-es-take diff-less p11 p13 zero-less-Suc*)
from *a10 a8* **have** *a82*: *?eslh*!*0* = (*EvtSeq e es*,*s*,*x*)
  by (*simp add*: *nth-butlast p11*)
obtain *esl2* **where** *a83*: *?eslh* = (*EvtSeq e es*,*s*,*x*)#*esl2*
  by (*metis Suc-diff-Suc a10 a9 take-Suc-Cons*)

from *a6* **have** *b17*: E_e *ef* = *e* **using** *evtsys-spec-evtseq* **by** *simp*
from *a6* **have** *b18*: *evtsys-spec* (*fst esf*) = *es* **using** *evtsys-spec-evtsys* **by** *simp*

have *b19*: *ef*∈*all-evts-es* (*rgf-EvtSeq ef esf*)
  using *all-evts-es-seq*[*of ef esf*] **by** *simp*

from *a5 b18* **have** *c0*: ∀ *i*. *Suc i* ≤ *length ?eslh* ⟶ *getspc-es* (*?eslh* ! *i*) ≠ *es*
  using *Suc-diff-1 Suc-le-lessD Suc-less-eq length-take min.bounded-iff*
    nth-take p11 p13 **by** *auto*

with *a81 a82* **have** ∃ *el*. (*el* ∈ *cpts-of-ev e s x* ∧ *length ?eslh* = *length el* ∧ *e-eqv-einevtseq ?eslh el es*)
  using *evtseq-nfin-samelower*[*of ?eslh e es s x*] *cpts-of-es-def*[*of EvtSeq e es s x*] **by** *auto*
then obtain *el* **where** *c1*: *el* ∈ *cpts-of-ev e s x* ∧ *length ?eslh* = *length el* ∧ *e-eqv-einevtseq ?eslh el es*
  by *auto*
then have *c2*: *el* ∈ *cpts-ev* **by** (*simp add*:*cpts-of-ev-def*)

from *a5 b18* **have** ∃ *sn xn*. *last* (*cs k*) = (*es*, *sn*, *xn*)
  using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
then obtain *sn* **and** *xn* **where** *d2*: *last* (*cs k*) = (*es*, *sn*, *xn*)
  by *auto*

let *?el1* = *el* @ [(*AnonyEvent* (*None*),*sn*, *xn*)]

from *c1* **have** *c23*: *length ?el1* = *?n*
  using *a9 butlast-conv-take diff-Suc-1 length-Cons length-append-singleton length-butlast* **by** *auto*

from *c1* **have** *d3*: *getspc-es* (*last ?eslh*) = *EvtSeq* (*getspc-e* (*last el*)) *es*
  using *e-eqv-einevtseq-def*[*rule-format*, *of ?eslh el es*] *a10*
    by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc butlast-conv-take diff-Suc-1 diff-is-0-eq*

last-conv-nth length-butlast length-greater-0-conv not-le order-refl p11 p13 take-eq-Nil)

then have ∃ *sn1 xn1*. *last ?eslh* = (*EvtSeq* (*getspc-e* (*last el*)) *es*, *sn1*, *xn1*)
using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
then obtain *sn1* **and** *xn1* **where** *d4*: *last ?eslh* = (*EvtSeq* (*getspc-e* (*last el*)) *es*, *sn1*, *xn1*)
by *auto*

with *c1* **have** *d41*: *gets-e* (*last el*) = *sn1* ∧ *getx-e* (*last el*) = *xn1*
using *e-eqv-einevtseq-def*[*of ?eslh el es*]
by (*smt Suc-diff-Suc a10 a9 diff-Suc-1 diff-is-0-eq fst-conv gets-es-def getx-es-def last-conv-nth le-refl length-0-conv list.distinct*(*1*) *not-le snd-conv take-eq-Nil*)
then have *d42*: *last el* = (*getspc-e* (*last el*), *sn1*, *xn1*)
by (*metis gets-e-def getspc-e-def getx-e-def prod.collapse*)

have *d51*: *last ?eslh* = *?esl* ! (*?n* − *2*)
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-diff-Suc a10 butlast-conv-take diff-Suc-eq-diff-pred last-conv-nth length-butlast length-greater-0-conv lessI nth-butlast p11 p13 take-eq-Nil*)

have *d52*: *last ?esl* = *?esl* ! (*?n* − *1*)
by (*simp add*: *a9 last-conv-nth*)
from *a8 a10* **have** *drop* (*?n*−*2*) *?esl* ∈ *cpts-es* **using** *cpts-es-dropi2*[*of ?esl ?n* − *2*]
using *Suc-1 diff-Suc-less p11 p13* **by** *linarith*
with *d2 d4 b18 d51 d52* **have** *d6*: ∃ *est*. *?esl* ! (*?n*−*2*) −*es*−*est*→ *?esl* ! (*?n*−*1*)
using *exist-estran*[*of EvtSeq* (*getspc-e* (*last el*)) *es sn1 xn1 es sn xn* []]
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-1 Suc-diff-Suc a10 a5 d3 diff-Suc-less exist-estran p11 p13*)

then obtain *est* **where** *?esl* ! (*?n*−*2*) −*es*−*est*→ *?esl* ! (*?n*−*1*) **by** *auto*
with *d2 d4 d51 d52 b18* **have** *d7*: ∃ *t*. (*getspc-e* (*last el*), *sn1*, *xn1*) −*et*−*t*→(*AnonyEvent* (*None*),*sn*, *xn*)
using *evtseq-tran-0-exist-etran*[*of getspc-e* (*last el*) *es sn1 xn1 est sn xn*] **by** *auto*
with *a10 c1 c2 d41 d42* **have** *d8*:*?el1* ∈ *cpts-ev*
using *cpts-ev-onemore* **by** (*metis diff-is-0-eq last-conv-nth length-greater-0-conv not-le p11 p13 take-eq-Nil*)

from *d8* **have** *d9*: *?el1* ∈ *cpts-of-ev e s x*
by (*metis* (*no-types*, *lifting*) *a10 butlast-conv-take c1 cpts-of-ev-def length-butlast mem-Collect-eq nth-append zero-less-diff*)

from *p14* **have** *?esl* ∈ *assume-es*(*Pre k*, *Rely k*) **by** *simp*
with *b1 b2 b6 b8* **have** *?esl* ∈ *assume-es*(Pre_e *ef*, $Rely_e$ *ef*)
by (*metis assume-es-imp equalityE*)
then have *?eslh* ∈ *assume-es*(Pre_e *ef*, $Rely_e$ *ef*)
using *assume-es-take-n*[*of ?n*−*1 ?esl* Pre_e *ef* $Rely_e$ *ef*]
by (*metis a10 butlast-conv-take diff-le-self zero-less-diff*)
with *c1* **have** *c21*: *el* ∈ *assume-e*(Pre_e *ef*, $Rely_e$ *ef*)
using *e-eqv-einevtseq-def*[*of ?eslh el es*] *assume-es-def assume-e-def*
by (*smt Suc-leI a10 diff-is-0-eq eetran-eqconf1 eqconf-esetran length-greater-0-conv less-imp-le-nat mem-Collect-eq not-le p11 p13 prod.simps*(*2*) *take-eq-Nil*)
have *?el1* ∈ *assume-e*(Pre_e *ef*, $Rely_e$ *ef*)
proof −
have *gets-e* (*?el1*!*0*) ∈ Pre_e *ef*
proof −
from *c21* **have** *gets-e* (*el*!*0*) ∈ Pre_e *ef* **by** (*simp add*:*assume-e-def*)
then show *?thesis* **by** (*metis a10 butlast-conv-take c1 length-butlast nth-append zero-less-diff*)
qed
moreover
have ∀ *i*. *Suc i*<*length ?el1* ⟶ *?el1*!*i* −*ee*→ *?el1*!(*Suc i*) ⟶

$(gets\text{-}e\ (?el1!i),\ gets\text{-}e\ (?el1!Suc\ i)) \in Rely_e\ ef$
proof −
{
fix *i*
assume *e0*: *Suc i<length ?el1*
and *e1*: $?el1!i\ {-ee\rightarrow}\ ?el1!(Suc\ i)$
from *c21* **have** *e2*: $\forall i.\ Suc\ i<length\ el \longrightarrow\ el!i\ {-ee\rightarrow}\ el!(Suc\ i) \longrightarrow$
$(gets\text{-}e\ (el!i),\ gets\text{-}e\ (el!Suc\ i)) \in Rely_e\ ef$ **by** (*simp add:assume-e-def*)
have $(gets\text{-}e\ (?el1!i),\ gets\text{-}e\ (?el1!Suc\ i)) \in Rely_e\ ef$
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*: $Suc\ i \geq length\ ?el1 - 1$ **by** *simp*
with *e0* **have** *f1*: *Suc i = length ?el1 − 1* **by** *simp*
then have *f2*: *?el1!(Suc i) = (AnonyEvent None, sn, xn)* **by** *simp*
from *f1* **have** *f3*: *?el1!i = (getspc-e (last el), sn1, xn1)*
by (*metis* (*no-types*, *lifting*) *a10 c1 d42 diff-Suc-1 diff-is-0-eq*
last-conv-nth length-append-singleton length-greater-0-conv
lessI not-le nth-append p11 p13 take-eq-Nil)

with *d7 f2* **have** $getspc\text{-}e\ (?el1!i) \neq getspc\text{-}e\ (?el1!(Suc\ i))$
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

with *d9 b51* **have** *d10*: $?el1 \in commit\text{-}e(Guar_e\ ef,\ Post_e\ ef)$
using $evt\text{-}validity\text{-}def[of\ E_e\ ef\ Pre_e\ ef\ Rely_e\ ef\ Guar_e\ ef\ Post_e\ ef]$
Int-iff b17 contra-subsetD rgsound-e **by** *fastforce*

have *getspc-e (last ?el1) = AnonyEvent None* **using** *getspc-e-def*[*of last ?el1*] **by** *simp*
moreover
have *gets-e (last ?el1) = sn* **using** *gets-e-def*[*of last ?el1*] **by** *simp*
ultimately have $sn \in Post_e\ ef$ **using** *d10* **by** (*simp add:commit-e-def*)
with *d2* **have** *d101*: $gets\text{-}es\ (last\ (cs\ k)) \in Post_e\ ef$ **by** (*simp add:gets-es-def*)

from *a2* **have** $\forall ef \in all\text{-}evts\text{-}esspec\ (evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf)).\ is\text{-}basicevt\ ef$
using *evtsys-spec-evtseq*[*of ef esf*] *all-evts-same*[*of rgf-EvtSeq ef esf*]
by ($metis\ DomainE\ E_e\text{-}def\ prod.sel(1)$)
with *p1 p2 a6 a2 a3 a4 a8* **have** $\exists ie.\ ie < i \wedge (\exists e.\ (cs\ k)!ie\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ (cs\ k)!(Suc\ ie))$
$\wedge (\forall j.\ j > ie \wedge j < i \longrightarrow \neg(\exists e.\ (cs\ k)!j\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ (cs\ k)!(Suc\ j)))$
using *cmd-impl-evtent-before-and-cmds*[*of c cs k evtsys-spec (rgf-EvtSeq ef esf) s x*]
cpts-of-es-def[*of EvtSeq e es s x*] **by** *auto*
then obtain *ie* **and** *ev* **where** *c4*: $ie < i \wedge ((cs\ k)!ie\ {-es-}(EvtEnt\ ev\sharp k){\rightarrow}\ (cs\ k)!(Suc\ ie))$
$\wedge (\forall j.\ j > ie \wedge j < i \longrightarrow \neg(\exists e.\ (cs\ k)!j\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ (cs\ k)!(Suc\ j)))$ **by** *auto*
with *p1 p6 a3* **have** $\forall m.\ m > ie \wedge m \leq i \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = ev$
using *evtent-impl-curevt-in-cpts-es2*[*of c cs ie k ev i*] **by** *auto*

with *c4* **have** *c7*: *getx-es* ((*cs k*)!*i*) *k* = *ev* **by** *simp*

from *a3 c4* **have** *c8*: *ie* < *i* ∧ (*?eslh*!*ie* −*es*−(*EvtEnt ev*♯*k*)→ *?eslh*!(*Suc ie*))
∧ (∀ *j*. *j* > *ie* ∧ *j* < *i* ⟶ ¬(∃ *e*. *?eslh*!*j* −*es*−(*EvtEnt e*♯*k*)→ *?eslh*!(*Suc j*))) **by** *force*
from *a3 a81 a82 a83 c8 c0* **have** ∀ *i*. *i* ≤ *ie* ⟶ *getspc-es* (*?eslh* ! *i*) = *EvtSeq e es*
using *evtseq-evtent-befaft*[*of ?eslh e es s x esl2 ie*]
by (*smt Suc-diff-1 Suc-diff-Suc Suc-less-eq a10 butlast-conv-take*
diff-Suc-eq-diff-pred length-butlast less-trans-Suc p11 p13)

with *c8* **have** *c10*: *ev* = *e* **by** (*metis evtent-is-basicevt-inevtseq2 order-refl*)

have *c11*: $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))) = $Guar_e$ *ef*
using $Guar_f$*-def* $Guar_e$*-def* **by** (*metis a01 b17 b19 c10 c7*)

have (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*)))
proof(*cases Suc i* < *?n* − *1*)
assume *e0*: *Suc i* < *?n* − *1*
have *e1*: *getspc-es* (*?eslh* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*metis a3 c1 e0 e-eqv-einevtseq-def length-take less-imp-le-nat min.bounded-iff*)

have *e2*: *getspc-es* (*?eslh* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*metis Suc-leI a3 c1 e0 e-eqv-einevtseq-def length-take min.bounded-iff*)

from *a3 a4* **have** *getspc-es* (*?eslh* ! *i*) ≠ *getspc-es* (*?eslh* ! *Suc i*)
by (*metis Suc-lessD e0 evtsys-not-eq-in-tran-aux1 nth-take*)

with *e1 e2* **have** *getspc-e* (*el* ! *i*) ≠ *getspc-e* (*el* ! *Suc i*) **by** *simp*
with *c1 c2 e0* **have** *e4*: ∃ *t*. (*el* ! *i*) −*et*−*t*→ (*el* ! *Suc i*)
using *cpts-of-ev-def*[*of e s x*] *notran-confeqi*[*of el i*]
using *a3 length-take less-eq-Suc-le min.bounded-iff* **by** *fastforce*

from *e0 a3 c1* **have** *e5*: *Suc i* < *length ?el1* **by** *auto*
moreover
from *e0 a3 c23 e4 e5* **have** ∃ *t*. *?el1* ! *i* −*et*−*t*→ *?el1* ! *Suc i*
by (*metis* (*no-types*, *lifting*) *Suc-lessD butlast-snoc length-butlast nth-append*)
ultimately have *c6*: (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*))∈$Guar_e$ *ef*
using *d10* **by** (*simp add*:*commit-e-def*)

then have (*gets-es* (*?eslh* ! *i*), *gets-es* (*?eslh* ! *Suc i*)) ∈ $Guar_e$ *ef*
using *e-eqv-einevtseq-def*[*of ?eslh el es*]
by (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessE a3 c1 c23 diff-Suc-1*
e0 length-append-singleton nth-append)
with *c11* **show** *?thesis* **by** (*metis Suc-lessD e0 nth-take*)
next
assume ¬ (*Suc i* < *?n* − *1*)
then have *e0*: *Suc i* = *?n* − *1*
using *Suc-pred′ a3 less-antisym p11 p13* **by** *linarith*
then have *e1*: *Suc i* < *length ?el1* **using** *a3 c23* **by** *linarith*
have ∃ *t*. (*?el1* ! *i*) −*et*−*t*→(*?el1* ! *Suc i*)
proof −
have *f1*: *Suc i* = *length* (*butlast* (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]))
by (*metis c23 e0 length-butlast*)
have *f2*: *length el* = *length* (*cs k*) − *1*
using *c23* **by** *auto*
have (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]) ! *i* = *el* ! *i*
using *f1* **by** (*simp add*: *nth-append*)
then have (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]) ! *i* = *last el*
using *f2* **by** (*metis a83 c1 diff-Suc-1 e0 last-conv-nth length-greater-0-conv list.simps*(*3*))

then show *?thesis*
using *f1 d42 d7* by *auto*
qed

with *d10 e1* have (*gets-e* (*?el1* ! *i*), *gets-e* (*?el1* ! *Suc i*)) $\in$ *Guar$_e$ ef*
by (*simp add:commit-e-def*)
moreover
from *e0 c23* have *?el1* !*i* = *last el*
by (*metis* (*no-types*, *lifting*) *a10 butlast-snoc diff-Suc-1 diff-is-0-eq*
last-conv-nth length-0-conv length-butlast lessI not-le nth-append)
moreover
from *e0 c23* have *?el1* ! *Suc i* = (*AnonyEvent None*, *sn*, *xn*)
by (*metis* (*no-types*, *lifting*) *butlast-snoc length-butlast nth-append-length*)
ultimately have (*sn1*,*sn*)$\in$*Guar$_e$ ef* using *d42 gets-e-def*[*of* (*getspc-e* (*last el*), *sn1*, *xn1*)]
gets-e-def[*of* (*AnonyEvent None*, *sn*, *xn*)] by (*metis fst-conv snd-conv*)
moreover
from *d2 d52 e0* have *gets-es* (*cs k* ! *Suc i*) = *sn* using *gets-es-def*
using *fst-conv snd-conv* by *force*
moreover
from *e0 e1 c1 d42* have *gets-es* (*cs k* ! *i*) = *sn1* using *e-eqv-einevtseq-def*[*of ?eslh el es*]
by (*metis Suc-1 d4 d51 diff-Suc-1 diff-Suc-eq-diff-pred fst-conv gets-es-def snd-conv*)
ultimately show *?thesis* using *c11* by *simp*
qed
}
then show *?thesis* by *auto*
qed

lemma *act-cpts-evtseq-sat-guar-curevt-fstseg-new2-withlst-pst* [*rule-format*]:
assumes *b51*: $\vdash$ (*E$_e$ ef*) *sat$_e$* [*Pre$_e$ ef*, *Rely$_e$ ef*, *Guar$_e$ ef*, *Post$_e$ ef*]
and *b52*: $\vdash$ (*fst esf*) *sat$_s$* [*Pre$_f$* (*snd esf*), *Rely$_f$* (*snd esf*), *Guar$_f$* (*snd esf*), *Post$_f$* (*snd esf*)]
and *b6*: *pre* = *Pre$_e$ ef*
and *b7*: *post* = *Post$_f$* (*snd esf*)
and *b8*: *rely* $\subseteq$ *Rely$_e$ ef*
and *b9*: *rely* $\subseteq$ *Rely$_f$* (*snd esf*)
and *b10*: *Guar$_e$ ef* $\subseteq$ *guar*
and *b11*: *Guar$_f$* (*snd esf*) $\subseteq$ *guar*
and *b12*: *Post$_e$ ef* $\subseteq$ *Pre$_f$* (*snd esf*)
and *b1*: *Pre k* $\subseteq$ *pre*
and *b2*: *Rely k* $\subseteq$ *rely*
and *b3*: *guar* $\subseteq$ *Guar k*
and *b4*: *post* $\subseteq$ *Post k*
and *p0*: *c*$\in$*cpts-of-pes pes s x*
and *p1*: *c* $\propto$ *cs*
and *p8*: *c*$\in$*assume-pes*(*pre1*, *rely1*)
and *p2*: $\forall$ *k*. (*cs k*) $\in$ *cpts-of-es* (*pes k*) *s x*
and *p16*: $\forall$ *k*. (*cs k*)$\in$*commit-es*(*Guar k*, *Post k*)
and *p9*: $\forall$ *k*. *pre1* $\subseteq$ *Pre k*
and *p10*: $\forall$ *k*. *rely1* $\subseteq$ *Rely k*
and *p4*: $\forall$ *k j*. *j* $\neq$ *k* $\longrightarrow$ *Guar j* $\subseteq$ *Rely k*
and *a5*: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *getspc-es* (*cs k* ! *0*) $\wedge$
($\forall$ *i*. *Suc i* < *length* (*cs k*) $\longrightarrow$ *getspc-es* ((*cs k*) ! *i*) $\neq$ *evtsys-spec* (*fst esf*)) $\wedge$
getspc-es(*last* (*cs k*)) = *evtsys-spec* (*fst esf*)
and *a2*: $\forall$ *e*$\in$*all-evts-es* (*rgf-EvtSeq ef esf*). *is-basicevt* (*E$_e$ e*)
and *a01*: $\forall$ *e*$\in$*all-evts-es* (*rgf-EvtSeq ef esf*). *the* (*evtrgfs* (*E$_e$ e*)) = *snd e*
and *p6*: $\forall$ *j*. *Suc j* < *length c* $\longrightarrow$ ($\exists$ *actk*. ((*c* ! *j*) $-$*pes*$-$*actk*$\rightarrow$ (*c* ! *Suc j*)))
shows ($\forall$ *i*. *Suc i* < *length* (*cs k*) $\wedge$ ((*cs k* ! *i*) $-$*es*$-$(*Cmd cmd*)$\sharp$*k*$\rightarrow$ (*cs k* ! *Suc i*)) $\longrightarrow$
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) $\in$ *Guar$_f$* (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))))

∧ gets-es (last (cs k))∈$Post_e$ ef
proof −
from p1 **have** p11[rule-format]: ∀ k. length (cs k) = length c **by** (simp add:conjoin-def same-length-def)
from p2 **have** p12: ∀ k. cs k ∈ cpts-es **using** cpts-of-es-def mem-Collect-eq **by** fastforce
with p11 **have** c ≠ Nil **using** cpts-es-not-empty length-0-conv **by** auto
then have p13: length c > 0 **by** auto

let ?esys = evtsys-spec (rgf-EvtSeq ef esf)
let ?esl = cs k
let ?n = length ?esl
let ?eslh = take (?n − 1) ?esl

from a5 **have** ∃ e es ess. ?esys = EvtSeq e es ∧ getspc-es (cs k ! 0) = EvtSeq e es
using evtsys-spec-evtseq[of ef esf] **by** fastforce
then obtain e **and** es **where** a6: ?esys = EvtSeq e es ∧ getspc-es (cs k ! 0) = EvtSeq e es **by** auto

from a6 **have** b17: E_e ef = e **using** evtsys-spec-evtseq **by** simp
from a6 **have** b18: evtsys-spec (fst esf) = es **using** evtsys-spec-evtsys **by** simp

from p2 a6 **have** a8: ?esl ∈ cpts-es ∧ ?esl!0 = (EvtSeq e es,s,x)
using cpts-of-es-def[of pes k s x]
by (metis (mono-tags, lifting) fst-conv getspc-es-def mem-Collect-eq)
then obtain esl1 **where** a9: ?esl = (EvtSeq e es,s,x)#esl1
by (metis Suc-pred length-Suc-conv nth-Cons-0 p11 p13)

from a5 a6 b18 **have** a10: ?n > 1 **using** evtseq-ne-es
using a9 diff-is-0-eq last-conv-nth leI list.simps(3) **by** force

from a8 a10 **have** a81: ?eslh ∈ cpts-es
by (metis (no-types, lifting) Suc-diff-Suc butlast-conv-take cpts-es-take diff-less p11 p13 zero-less-Suc)
from a10 a8 **have** a82: ?eslh!0 = (EvtSeq e es,s,x)
by (simp add: nth-butlast p11)
obtain esl2 **where** a83: ?eslh = (EvtSeq e es,s,x)#esl2
by (metis Suc-diff-Suc a10 a9 take-Suc-Cons)

from p16 p0 p1 p2 p4 p8 p9 p10 **have** p14: ∀ k. (cs k) ∈ assume-es(Pre k, Rely k)
using conjoin-comm-imp-rely **by** (metis (mono-tags, lifting))

have b19: ef∈all-evts-es (rgf-EvtSeq ef esf)
using all-evts-es-seq[of ef esf] **by** simp

from a5 b18 **have** c0: ∀ i. Suc i ≤ length ?eslh ⟶ getspc-es (?eslh ! i) ≠ es
using Suc-diff-1 Suc-le-lessD Suc-less-eq length-take min.bounded-iff
nth-take p11 p13 **by** auto

with a81 a82 **have** ∃ el. (el ∈ cpts-of-ev e s x ∧ length ?eslh = length el ∧ e-eqv-einevtseq ?eslh el es)
using evtseq-nfin-samelower[of ?eslh e es s x] cpts-of-es-def[of EvtSeq e es s x] **by** auto
then obtain el **where** c1: el ∈ cpts-of-ev e s x ∧ length ?eslh = length el ∧ e-eqv-einevtseq ?eslh el es
by auto
then have c2: el ∈ cpts-ev **by** (simp add:cpts-of-ev-def)

from a5 b18 **have** ∃ sn xn. last (cs k) = (es, sn, xn)
using getspc-es-def **by** (metis fst-conv surj-pair)
then obtain sn **and** xn **where** d2: last (cs k) = (es, sn, xn)
by auto

let *?el1 = el @ [(AnonyEvent (None),sn, xn)]*

from *c1* **have** *c23: length ?el1 = ?n*
using *a9 butlast-conv-take diff-Suc-1 length-Cons length-append-singleton length-butlast* **by** *auto*

from *c1* **have** *d3: getspc-es (last ?eslh) = EvtSeq (getspc-e (last el)) es*
using *e-eqv-einevtseq-def[rule-format, of ?eslh el es] a10*
by *(metis (no-types, lifting) Suc-diff-Suc butlast-conv-take diff-Suc-1 diff-is-0-eq last-conv-nth length-butlast length-greater-0-conv not-le order-refl p11 p13 take-eq-Nil)*

then have *∃ sn1 xn1. last ?eslh = (EvtSeq (getspc-e (last el)) es, sn1, xn1)*
using *getspc-es-def* **by** *(metis fst-conv surj-pair)*
then obtain *sn1* **and** *xn1* **where** *d4: last ?eslh = (EvtSeq (getspc-e (last el)) es, sn1, xn1)*
by *auto*

with *c1* **have** *d41: gets-e (last el) = sn1 ∧ getx-e (last el) = xn1*
using *e-eqv-einevtseq-def[of ?eslh el es]*
by *(smt Suc-diff-Suc a10 a9 diff-Suc-1 diff-is-0-eq fst-conv gets-es-def getx-es-def last-conv-nth le-refl length-0-conv list.distinct(1) not-le snd-conv take-eq-Nil)*
then have *d42: last el = (getspc-e (last el), sn1, xn1)*
by *(metis gets-e-def getspc-e-def getx-e-def prod.collapse)*

have *d51: last ?eslh = ?esl ! (?n − 2)*
by *(metis (no-types, lifting) Suc-1 Suc-diff-Suc a10 butlast-conv-take diff-Suc-eq-diff-pred last-conv-nth length-butlast length-greater-0-conv lessI nth-butlast p11 p13 take-eq-Nil)*

have *d52: last ?esl = ?esl ! (?n − 1)*
by *(simp add: a9 last-conv-nth)*
from *a8 a10* **have** *drop (?n−2) ?esl ∈ cpts-es* **using** *cpts-es-dropi2[of ?esl ?n − 2]*
using *Suc-1 diff-Suc-less p11 p13* **by** *linarith*
with *d2 d4 b18 d51 d52* **have** *d6: ∃ est. ?esl ! (?n−2) −es−est→ ?esl ! (?n−1)*
using *exist-estran[of EvtSeq (getspc-e (last el)) es sn1 xn1 es sn xn []]*
by *(metis (no-types, lifting) Cons-nth-drop-Suc One-nat-def Suc-1 Suc-diff-Suc a10 a5 d3 diff-Suc-less exist-estran p11 p13)*

then obtain *est* **where** *?esl ! (?n−2) −es−est→ ?esl ! (?n−1)* **by** *auto*
with *d2 d4 d51 d52 b18* **have** *d7: ∃ t. (getspc-e (last el), sn1, xn1) −et−t→(AnonyEvent (None),sn, xn)*
using *evtseq-tran-0-exist-etran[of getspc-e (last el) es sn1 xn1 est sn xn]* **by** *auto*
with *a10 c1 c2 d41 d42* **have** *d8:?el1 ∈ cpts-ev*
using *cpts-ev-onemore* **by** *(metis diff-is-0-eq last-conv-nth length-greater-0-conv not-le p11 p13 take-eq-Nil)*

from *d8* **have** *d9: ?el1 ∈ cpts-of-ev e s x*
by *(metis (no-types, lifting) a10 butlast-conv-take c1 cpts-of-ev-def length-butlast mem-Collect-eq nth-append zero-less-diff)*

from *p14* **have** *?esl ∈ assume-es(Pre k, Rely k)* **by** *simp*
with *b1 b2 b6 b8* **have** *?esl ∈ assume-es(Pre_e ef, $Rely_e$ ef)*
by *(metis assume-es-imp equalityE)*
then have *?eslh ∈ assume-es(Pre_e ef, $Rely_e$ ef)*
using *assume-es-take-n[of ?n−1 ?esl Pre_e ef $Rely_e$ ef]*
by *(metis a10 butlast-conv-take diff-le-self zero-less-diff)*
with *c1* **have** *c21: el ∈ assume-e(Pre_e ef, $Rely_e$ ef)*
using *e-eqv-einevtseq-def[of ?eslh el es] assume-es-def assume-e-def*
by *(smt Suc-leI a10 diff-is-0-eq eetran-eqconf1 eqconf-esetran length-greater-0-conv less-imp-le-nat mem-Collect-eq not-le p11 p13 prod.simps(2) take-eq-Nil)*

have *?el1* ∈ *assume-e*(*Pre*$_e$ *ef*, *Rely*$_e$ *ef*)
  proof −
    have *gets-e* (*?el1*!*0*) ∈ *Pre*$_e$ *ef*
      proof −
        from *c21* **have** *gets-e* (*el*!*0*) ∈ *Pre*$_e$ *ef* **by** (*simp add*:*assume-e-def*)
        then show *?thesis* **by** (*metis a10 butlast-conv-take c1 length-butlast nth-append zero-less-diff*)
      qed
    moreover
    have ∀ *i*. *Suc i*<*length ?el1* ⟶ *?el1*!*i* −*ee*→ *?el1*!(*Suc i*) ⟶
        (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *Rely*$_e$ *ef*
      proof −
      {
        fix *i*
        assume *e0*: *Suc i*<*length ?el1*
          and *e1*: *?el1*!*i* −*ee*→ *?el1*!(*Suc i*)
        from *c21* **have** *e2*: ∀ *i*. *Suc i*<*length el* ⟶ *el*!*i* −*ee*→ *el*!(*Suc i*) ⟶
          (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *Rely*$_e$ *ef* **by** (*simp add*:*assume-e-def*)
        have (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *Rely*$_e$ *ef*
          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 None*, *sn*, *xn*) **by** *simp*
            from *f1* **have** *f3*: *?el1*!*i* = (*getspc-e* (*last el*), *sn1*, *xn1*)
              by (*metis* (*no-types*, *lifting*) *a10 c1 d42 diff-Suc-1 diff-is-0-eq*
                last-conv-nth length-append-singleton length-greater-0-conv
                lessI not-le nth-append p11 p13 take-eq-Nil)

            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

with *d9 b51* **have** *d10*: *?el1* ∈ *commit-e*(*Guar*$_e$ *ef*, *Post*$_e$ *ef*)
   using *evt-validity-def*[*of E*$_e$ *ef Pre*$_e$ *ef Rely*$_e$ *ef Guar*$_e$ *ef Post*$_e$ *ef*]
    Int-iff b17 contra-subsetD rgsound-e **by** *fastforce*

have *getspc-e* (*last ?el1*) = *AnonyEvent None* **using** *getspc-e-def*[*of last ?el1*] **by** *simp*
moreover
have *gets-e* (*last ?el1*) = *sn* **using** *gets-e-def*[*of last ?el1*] **by** *simp*
ultimately have *sn*∈*Post*$_e$ *ef* **using** *d10* **by** (*simp add*:*commit-e-def*)
with *d2* **have** *d101*: *gets-es* (*last* (*cs k*)) ∈ *Post*$_e$ *ef* **by** (*simp add*:*gets-es-def*)

{

fix i

assume $a3$: $Suc\ i < length\ ?esl$
and $a4$: $(?esl!i\ -es-((Cmd\ cmd)\sharp k)\rightarrow\ ?esl!(Suc\ i))$

from $a2$ **have** $\forall\, ef{\in}all\text{-}evts\text{-}esspec\ (evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf)).\ is\text{-}basicevt\ ef$
using $evtsys\text{-}spec\text{-}evtseq[of\ ef\ esf]\ all\text{-}evts\text{-}same[of\ rgf\text{-}EvtSeq\ ef\ esf]$
by $(metis\ DomainE\ E_e\text{-}def\ prod.sel(1))$
with $p1\ p2\ a6\ a2\ a3\ a4\ a8$ **have** $\exists\, ie.\ ie < i \wedge (\exists\, e.\ (cs\ k)!ie\ -es-(EvtEnt\ e\sharp k)\rightarrow\ (cs\ k)!(Suc\ ie))$
$\wedge (\forall\, j.\ j > ie \wedge j < i \longrightarrow \neg(\exists\, e.\ (cs\ k)!j\ -es-(EvtEnt\ e\sharp k)\rightarrow\ (cs\ k)!(Suc\ j)))$
using $cmd\text{-}impl\text{-}evtent\text{-}before\text{-}and\text{-}cmds[of\ c\ cs\ k\ evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf)\ s\ x]$
$cpts\text{-}of\text{-}es\text{-}def[of\ EvtSeq\ e\ es\ s\ x]$ **by** $auto$
then obtain ie **and** ev **where** $c4$: $ie < i \wedge ((cs\ k)!ie\ -es-(EvtEnt\ ev\sharp k)\rightarrow\ (cs\ k)!(Suc\ ie))$
$\wedge (\forall\, j.\ j > ie \wedge j < i \longrightarrow \neg(\exists\, e.\ (cs\ k)!j\ -es-(EvtEnt\ e\sharp k)\rightarrow\ (cs\ k)!(Suc\ j)))$ **by** $auto$
with $p1\ p6\ a3$ **have** $\forall\, m.\ m > ie \wedge m \le i \longrightarrow getx\text{-}es\ ((cs\ k)!m)\ k = ev$
using $evtent\text{-}impl\text{-}curevt\text{-}in\text{-}cpts\text{-}es2[of\ c\ cs\ ie\ k\ ev\ i]$ **by** $auto$
with $c4$ **have** $c7$: $getx\text{-}es\ ((cs\ k)!i)\ k = ev$ **by** $simp$

from $a3\ c4$ **have** $c8$: $ie < i \wedge (?eslh!ie\ -es-(EvtEnt\ ev\sharp k)\rightarrow\ ?eslh!(Suc\ ie))$
$\wedge (\forall\, j.\ j > ie \wedge j < i \longrightarrow \neg(\exists\, e.\ ?eslh!j\ -es-(EvtEnt\ e\sharp k)\rightarrow\ ?eslh!(Suc\ j)))$ **by** $force$
from $a3\ a81\ a82\ a83\ c8\ c0$ **have** $\forall\, i.\ i \le ie \longrightarrow getspc\text{-}es\ (?eslh\ !\ i) = EvtSeq\ e\ es$
using $evtseq\text{-}evtent\text{-}befaft[of\ ?eslh\ e\ es\ s\ x\ esl2\ ie]$
by $(smt\ Suc\text{-}diff\text{-}1\ Suc\text{-}diff\text{-}Suc\ Suc\text{-}less\text{-}eq\ a10\ butlast\text{-}conv\text{-}take$
$diff\text{-}Suc\text{-}eq\text{-}diff\text{-}pred\ length\text{-}butlast\ less\text{-}trans\text{-}Suc\ p11\ p13)$

with $c8$ **have** $c10$: $ev = e$ **by** $(metis\ evtent\text{-}is\text{-}basicevt\text{-}inevtseq2\ order\text{-}refl)$

have $c11$: $Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))) = Guar_e\ ef$
using $Guar_f\text{-}def\ Guar_e\text{-}def$ **by** $(metis\ a01\ b17\ b19\ c10\ c7)$

have $(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k)))$
proof$(cases\ Suc\ i < ?n - 1)$
assume $e0$: $Suc\ i < ?n - 1$
have $e1$: $getspc\text{-}es\ (?eslh\ !\ i) = EvtSeq\ (getspc\text{-}e\ (el\ !\ i))\ es$
by $(metis\ a3\ c1\ e0\ e\text{-}eqv\text{-}einevtseq\text{-}def\ length\text{-}take\ less\text{-}imp\text{-}le\text{-}nat\ min.bounded\text{-}iff)$

have $e2$: $getspc\text{-}es\ (?eslh\ !\ Suc\ i) = EvtSeq\ (getspc\text{-}e\ (el\ !\ Suc\ i))\ es$
by $(metis\ Suc\text{-}leI\ a3\ c1\ e0\ e\text{-}eqv\text{-}einevtseq\text{-}def\ length\text{-}take\ min.bounded\text{-}iff)$

from $a3\ a4$ **have** $getspc\text{-}es\ (?eslh\ !\ i) \neq getspc\text{-}es\ (?eslh\ !\ Suc\ i)$
by $(metis\ Suc\text{-}lessD\ e0\ evtsys\text{-}not\text{-}eq\text{-}in\text{-}tran\text{-}aux1\ nth\text{-}take)$

with $e1\ e2$ **have** $getspc\text{-}e\ (el\ !\ i) \neq getspc\text{-}e\ (el\ !\ Suc\ i)$ **by** $simp$
with $c1\ c2\ e0$ **have** $e4$: $\exists\, t.\ (el\ !\ i)\ -et-t\rightarrow\ (el\ !\ Suc\ i)$
using $cpts\text{-}of\text{-}ev\text{-}def[of\ e\ s\ x]\ notran\text{-}confeqi[of\ el\ i]$
using $a3\ length\text{-}take\ less\text{-}eq\text{-}Suc\text{-}le\ min.bounded\text{-}iff$ **by** $fastforce$

from $e0\ a3\ c1$ **have** $e5$: $Suc\ i < length\ ?el1$ **by** $auto$
moreover
from $e0\ a3\ c23\ e4\ e5$ **have** $\exists\, t.\ ?el1\ !\ i\ -et-t\rightarrow\ ?el1\ !\ Suc\ i$
by $(metis\ (no\text{-}types,\ lifting)\ Suc\text{-}lessD\ butlast\text{-}snoc\ length\text{-}butlast\ nth\text{-}append)$
ultimately have $c6$: $(gets\text{-}e\ (?el1!i),\ gets\text{-}e\ (?el1!Suc\ i)){\in}Guar_e\ ef$
using $d10$ **by** $(simp\ add{:}commit\text{-}e\text{-}def)$

then have $(gets\text{-}es\ (?eslh\ !\ i),\ gets\text{-}es\ (?eslh\ !\ Suc\ i)) \in Guar_e\ ef$
using $e\text{-}eqv\text{-}einevtseq\text{-}def[of\ ?eslh\ el\ es]$
by $(metis\ (no\text{-}types,\ lifting)\ Suc\text{-}leI\ Suc\text{-}lessE\ a3\ c1\ c23\ diff\text{-}Suc\text{-}1$
$e0\ length\text{-}append\text{-}singleton\ nth\text{-}append)$

with *c11* **show** *?thesis* **by** (*metis Suc-lessD e0 nth-take*)
next
assume ¬ (*Suc i* < *?n* − *1*)
then have *e0*: *Suc i* = *?n* − *1*
using *Suc-pred′ a3 less-antisym p11 p13* **by** *linarith*
then have *e1*: *Suc i* < *length ?el1* **using** *a3 c23* **by** *linarith*
have ∃ *t*. (*?el1* ! *i*) −*et*−*t*→(*?el1* ! *Suc i*)
proof −
have *f1*: *Suc i* = *length* (*butlast* (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]))
by (*metis c23 e0 length-butlast*)
have *f2*: *length el* = *length* (*cs k*) − *1*
using *c23* **by** *auto*
have (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]) ! *i* = *el* ! *i*
using *f1* **by** (*simp add*: *nth-append*)
then have (*el* @ [(*AnonyEvent None*, *sn*, *xn*)]) ! *i* = *last el*
using *f2* **by** (*metis a83 c1 diff-Suc-1 e0 last-conv-nth length-greater-0-conv list.simps*(*3*))
then show *?thesis*
using *f1 d42 d7* **by** *auto*
qed

with *d10 e1* **have** (*gets-e* (*?el1* ! *i*), *gets-e* (*?el1* ! *Suc i*)) ∈ $Guar_e$ *ef*
by (*simp add*:*commit-e-def*)
moreover
from *e0 c23* **have** *?el1* !*i* = *last el*
by (*metis* (*no-types*, *lifting*) *a10 butlast-snoc diff-Suc-1 diff-is-0-eq*
last-conv-nth length-0-conv length-butlast lessI not-le nth-append)
moreover
from *e0 c23* **have** *?el1* ! *Suc i* = (*AnonyEvent None*, *sn*, *xn*)
by (*metis* (*no-types*, *lifting*) *butlast-snoc length-butlast nth-append-length*)
ultimately have (*sn1*,*sn*)∈$Guar_e$ *ef* **using** *d42 gets-e-def*[*of* (*getspc-e* (*last el*), *sn1*, *xn1*)]
gets-e-def[*of* (*AnonyEvent None*, *sn*, *xn*)] **by** (*metis fst-conv snd-conv*)
moreover
from *d2 d52 e0* **have** *gets-es* (*cs k* ! *Suc i*) = *sn* **using** *gets-es-def*
using *fst-conv snd-conv* **by** *force*
moreover
from *e0 e1 c1 d42* **have** *gets-es* (*cs k* ! *i*) = *sn1* **using** *e-eqv-einevtseq-def*[*of ?eslh el es*]
by (*metis Suc-1 d4 d51 diff-Suc-1 diff-Suc-eq-diff-pred fst-conv gets-es-def snd-conv*)
ultimately show *?thesis* **using** *c11* **by** *simp*
qed
}
then show *?thesis* **using** *d101* **by** *auto*
qed

lemma *act-cpts-evtseq-sat-guar-curevt-new2*:
assumes *b51*: ⊢ (E_e *ef*) sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
and *b52*: ⊢ (*fst esf*) sat_s [Pre_f (*snd esf*), $Rely_f$ (*snd esf*), $Guar_f$ (*snd esf*), $Post_f$ (*snd esf*)]
and *b6*: *prea* = Pre_e *ef*
and *b7*: *posta* = $Post_f$ (*snd esf*)
and *b8*: *relya* ⊆ $Rely_e$ *ef*
and *b9*: *relya* ⊆ $Rely_f$ (*snd esf*)
and *b10*: $Guar_e$ *ef* ⊆ *guara*
and *b11*: $Guar_f$ (*snd esf*) ⊆ *guara*
and *b12*: $Post_e$ *ef* ⊆ Pre_f (*snd esf*)
and *b1*: *Pre k* ⊆ *prea*
and *b2*: *Rely k* ⊆ *relya*
and *b3*: *guara* ⊆ *Guar k*
and *b4*: *posta* ⊆ *Post k*

and $p0$: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x$
and $p1$: $c \propto cs$
and $p8$: $c \in assume\text{-}pes(pre1,\ rely1)$
and $p2$: $\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x$
and $p16$: $\forall k.\ cs\ k \in\ commit\text{-}es(Guar\ k,\ Post\ k)$
and $p9$: $\forall k.\ pre1 \subseteq Pre\ k$
and $p10$: $\forall k.\ rely1 \subseteq Rely\ k$
and $p4$: $\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k$
and $a0$: $evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf) = getspc\text{-}es\ (cs\ k\ !\ 0)$
and $a2$: $\forall e \in all\text{-}evts\text{-}es\ (rgf\text{-}EvtSeq\ ef\ esf).\ is\text{-}basicevt\ (E_e\ e)$
and $a02$: $\forall e \in all\text{-}evts\text{-}es\ (rgf\text{-}EvtSeq\ ef\ esf).\ the\ (evtrgfs\ (E_e\ e)) = snd\ e$
and $p6$: $\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ ((c\ !\ j)\ {-pes-}actk{\rightarrow}\ (c\ !\ Suc\ j)))$
and $pp[rule\text{-}format]$: $\forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq Pre_f\ (snd\ esf) \wedge Rely\ k \subseteq Rely_f\ (snd\ esf)$
$\wedge\ Guar_f\ (snd\ esf) \subseteq Guar\ k \wedge Post_f\ (snd\ esf) \subseteq Post\ k \longrightarrow$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1) \longrightarrow$
$(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ cs\ k \in\ commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ (fst\ esf) = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ (fst\ esf).\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ (fst\ esf).\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ ((c\ !\ j)\ {-pes-}actk{\rightarrow}\ (c\ !\ Suc\ j)))) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k\ !\ i)\ {-es-}(Cmd\ cmd)\sharp k{\rightarrow}\ (cs\ k\ !\ Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$
shows $\forall i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k\ !\ i)\ {-es-}(Cmd\ cmd)\sharp k{\rightarrow}\ (cs\ k\ !\ Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k)))$
proof −
from $p1$ **have** $p11[rule\text{-}format]$: $\forall k.\ length\ (cs\ k) = length\ c$ **by** $(simp\ add{:}conjoin\text{-}def\ same\text{-}length\text{-}def)$
from $p2$ **have** $p12$: $\forall k.\ cs\ k \in cpts\text{-}es$ **using** $cpts\text{-}of\text{-}es\text{-}def\ mem\text{-}Collect\text{-}eq$ **by** $fastforce$
with $p11$ **have** $c \neq Nil$ **using** $cpts\text{-}es\text{-}not\text{-}empty\ length\text{-}0\text{-}conv$ **by** $auto$
then have $p13$: $length\ c > 0$ **by** $auto$

from $p0\ p1\ p2\ p4\ p8\ p9\ p10\ p16$ **have** $p14$: $\forall k.\ (cs\ k) \in assume\text{-}es(Pre\ k,\ Rely\ k)$
using $conjoin\text{-}comm\text{-}imp\text{-}rely$ **by** $(metis\ (mono\text{-}tags,\ lifting))$

from $p0$ **have** $p15$: $c \in cpts\text{-}pes \wedge c!0=(pes,s,x)$ **by** $(simp\ add{:}cpts\text{-}of\text{-}pes\text{-}def)$

let $?esys = evtsys\text{-}spec\ (rgf\text{-}EvtSeq\ ef\ esf)$
let $?esl = cs\ k$

from $a0$ **have** $\exists e\ es\ ess.\ ?esys = EvtSeq\ e\ es \wedge getspc\text{-}es\ (cs\ k\ !\ 0) = EvtSeq\ e\ es$
using $evtsys\text{-}spec\text{-}evtseq[of\ ef\ esf]$ **by** $fastforce$
then obtain e **and** es **where** $a6$: $?esys = EvtSeq\ e\ es \wedge getspc\text{-}es\ (cs\ k\ !\ 0) = EvtSeq\ e\ es$ **by** $auto$

from $p2\ a6$ **have** $a8$: $?esl \in cpts\text{-}es \wedge ?esl!0 = (EvtSeq\ e\ es,s,x)$
using $cpts\text{-}of\text{-}es\text{-}def[of\ pes\ k\ s\ x]$
by $(metis\ (mono\text{-}tags,\ lifting)\ fst\text{-}conv\ getspc\text{-}es\text{-}def\ mem\text{-}Collect\text{-}eq)$
then obtain $esl1$ **where** $a9$: $?esl = (EvtSeq\ e\ es,s,x)\#esl1$
by $(metis\ Suc\text{-}pred\ length\text{-}Suc\text{-}conv\ nth\text{-}Cons\text{-}0\ p11\ p13)$

from $a6$ **have** $b17$: $E_e\ ef = e$ **using** $evtsys\text{-}spec\text{-}evtseq$ **by** $simp$
from $a6$ **have** $b18$: $evtsys\text{-}spec\ (fst\ esf) = es$ **using** $evtsys\text{-}spec\text{-}evtsys$ **by** $simp$

{

fix *i*
assume *a3*: *Suc i* < *length ?esl*
and *a4*: (*?esl*!*i* −*es*−((*Cmd cmd*)♯*k*)→ *?esl*!(*Suc i*))
then have (*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ *Guar*
proof(*cases* ∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) ≠ *es*)
assume *c0*: ∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) ≠ *es*
with *p0 p1 p8 p2 p9 p10 p4 p6 p16* **show** *?thesis*
using *act-cpts-evtseq-sat-guar-curevt-fstseg-new2*[*of ef esf prea*
posta relya guara Pre k Rely Guar Post c pes s x cs pre1 rely1 evtrgfs i cmd]
a02 a2 b18 a3 a4 b1 b2 b3 b4 b6 b7 b8 b9 b10 b11 b12 b51 b52 c0 b18 a6 **by** *auto*
next
assume *c0*: ¬(∀ *i*. *Suc i* ≤ *length ?esl* ⟶ *getspc-es* (*?esl* ! *i*) ≠ *es*)
then have ∃ *m*. *Suc m* ≤ *length ?esl* ∧ *getspc-es* (*?esl* ! *m*) = *es* **by** *auto*
then obtain *m* **where** *c1*: *Suc m* ≤ *length ?esl* ∧ *getspc-es* (*?esl* ! *m*) = *es* **by** *auto*
then have ∃ *i*. *i* ≤ *m* ∧ *getspc-es* (*?esl* ! *i*) = *es* **by** *auto*
with *a8 c1* **have** *c2*: ∃ *i*. (*i* ≤ *m* ∧ *getspc-es* (*?esl* ! *i*) = *es*)
∧ (∀ *j*. *j* < *i* ⟶ *getspc-es* (*?esl* ! *j*) ≠ *es*)
using *evtseq-fst-finish*[*of ?esl e es m*] *getspc-es-def fst-conv* **by** *force*
then obtain *n* **where** *c3*: (*n* ≤ *m* ∧ *getspc-es* (*?esl* ! *n*) = *es*)
∧ (∀ *j*. *j* < *n* ⟶ *getspc-es* (*?esl* ! *j*) ≠ *es*)
by *auto*
with *a8* **have** *c4*: *n* ≠ *0* **using** *getspc-es-def*[*of cs k* ! *0*]
by (*metis* (*no-types*, *hide-lams*) *add.commute add.right-neutral fst-conv*
add-Suc dual-order.irrefl esys.size(*3*) *le-add1 le-imp-less-Suc*)
from *c1 c3* **have** *c5*: *n* < *length ?esl* **by** *simp*
let *?c1* = *take n c*
let *?cs1* = λ*k*. *take n* (*cs k*)
let *?c2* = *drop n c*
let *?cs2* = λ*k*. *drop n* (*cs k*)
let *?cs1k* = *?cs1 k*
let *?cs2k* = *?cs2 k*

from *c1 c3 p11* **have** *c5-1*: *length ?c1* = *n* **using** *less-le-trans* **by** *auto*
have *c6*: *?c1* @ *?c2* = *c* **by** *simp*
have *c7*: *?esl* = *?cs1k* @ *?cs2k* **by** *simp*

have *c8*: *?cs1k* ! *0* = (*EvtSeq e es*, *s*, *x*) **using** *a9 c4* **by** *auto*
have *c9*: *getspc-es* (*?cs2k* ! *0*) = *es*
by (*simp add*: *c3 c5 less-or-eq-imp-le*)

let *?c12* = *take* (*Suc n*) *c*
let *?cs12* = λ*k*. *take* (*Suc n*) (*cs k*)
from *p15 p11 c1 c3 c4 c5-1 c5* **have** *d1*: *?c12*∈*cpts-pes* **using** *cpts-pes-take*[*of c n*]
by (*metis* (*no-types*, *lifting*))
moreover
with *p15 c4* **have** *d2*: *?c12*∈*cpts-of-pes pes s x*
using *cpts-of-pes-def*[*of pes s x*]
append-take-drop-id length-greater-0-conv mem-Collect-eq
nth-append take-eq-Nil **by** *auto*
moreover
from *p1 p11 c1 c3* **have** *?c12* ∝ *?cs12* **using** *take-n-conjoin*[*of c cs Suc n ?c12 ?cs12*] **by** *auto*
moreover
from *p8 c1 c3 p11* **have** *?c12* ∈ *assume-pes*(*pre1*, *rely1*)
using *assume-pes-take-n*[*of Suc n c pre1 rely1*] **by** *auto*
moreover
from *p2 c1 c3 p11* **have** ∀ *k*. (*?cs12 k*) ∈ *cpts-of-es* (*pes k*) *s x*

proof −
{
fix k'
from *p2 c1 c3 p11* **have** $(?cs12\ k')!0 = (pes\ k',\ s,\ x)$
using *cpts-of-es-def*[*of pes* k' *s x*]
Suc-leI less-le-trans mem-Collect-eq nth-take zero-less-Suc **by** *auto*
moreover
from *p2* **have** $cs\ k' \in cpts\text{-}es$
using *cpts-of-es-def*[*of pes* k' *s x*] **by** *auto*
moreover
with *c1 c3 p11* **have** $(?cs12\ k') \in cpts\text{-}es$ **using** *cpts-es-take*[*of cs* k' *n*]
Suc-diff-1 Suc-le-lessD c4 c5-1 dual-order.trans le-SucI
length-0-conv length-greater-0-conv **by** *auto*
ultimately have $(?cs12\ k') \in cpts\text{-}of\text{-}es\ (pes\ k')\ s\ x$
by (*simp add*:*cpts-of-es-def*)
}
then show *?thesis* **by** *auto*
qed
moreover
from *p6* **have** $\forall j.\ Suc\ j < length\ ?c12 \longrightarrow (\exists\, actk.\ ?c12!j{-}pes{-}actk{\rightarrow}?c12!Suc\ j)$
using *Suc-lessD length-take min-less-iff-conj nth-take* **by** *auto*
moreover
from *c3 b18* **have** $(\forall i.\ Suc\ i < length\ (?cs12\ k) \longrightarrow$
$getspc\text{-}es\ ((?cs12\ k)\ !\ i) \neq evtsys\text{-}spec\ (fst\ esf))$
by (*metis* (*no-types*, *lifting*) *Suc-le-lessD Suc-lessD Suc-lessI*
append-take-drop-id ex-least-nat-le gr-implies-not0 length-take
lessI less-antisym min.bounded-iff nth-append)
moreover
from *c3 c4 c5 b18* **have** $getspc\text{-}es(last\ (?cs12\ k)) = evtsys\text{-}spec\ (fst\ esf)$
proof −
from *c4 c5* **have** $last\ (?cs12\ k) = cs\ k\ !\ n$
by (*simp add*: *take-Suc-conv-app-nth*)
with *c3 b18* **show** *?thesis* **by** *simp*
qed
moreover
from *p16 c5* **have** $\forall k.\ ?cs12\ k \in commit\text{-}es\ (Guar\ k,\ Post\ k)$
using *commit-es-take-n*[*of Suc n*]
by (*metis Suc-leI p11 zero-less-Suc*)
ultimately
have *r1*[*rule-format*]: $(\forall i.\ Suc\ i < length\ (?cs12\ k) \wedge ((?cs12\ k\ !\ i)\ {-}es{-}(Cmd\ cmd)\sharp k{\rightarrow}\ (?cs12\ k\ !\ Suc\ i))$
$\longrightarrow$
$(gets\text{-}es\ (?cs12\ k\ !\ i),\ gets\text{-}es\ (?cs12\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (?cs12\ k\ !\ i)\ k))))$
$\wedge\ gets\text{-}es\ (last\ (?cs12\ k)) \in Post_e\ ef$
using *act-cpts-evtseq-sat-guar-curevt-fstseg-new2-withlst-pst*[*of ef esf prea*
posta relya guara Pre k Rely Guar Post ?c12 pes s x ?cs12 pre1 rely1 evtrgfs]
p9 p10 p4 p6 p16 a02 a2 b18 a3 a4 b1 b2 b3 b4
b6 b7 b8 b9 b10 b11 b12 b51 b52 c0 b18 a6 c4 **by** *auto*

then have *r2*: $\forall i.\ Suc\ i < length\ (?cs12\ k) \wedge ((?cs12\ k\ !\ i)\ {-}es{-}(Cmd\ cmd)\sharp k{\rightarrow}\ (?cs12\ k\ !\ Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (?cs12\ k\ !\ i),\ gets\text{-}es\ (?cs12\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (?cs12\ k\ !\ i)\ k)))$
by *auto*

show *?thesis*
proof(*cases* $Suc\ i \leq n$)
assume *d0*: $Suc\ i \leq n$
with *r2*[*rule-format*,*of i*] *a3 a4*
have $(gets\text{-}es\ ((?cs12\ k)!i),\ gets\text{-}es\ ((?cs12\ k)!(Suc\ i))) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ ((?cs12\ k)!i)\ k)))$
by *auto*

then show *?thesis* **using** *d0* **by** *auto*
next
assume *d0*: $\neg(Suc\ i \leq n)$

let $?c2 = drop\ n\ c$
let $?cs2 = \lambda k.\ drop\ n\ (cs\ k)$

from *d0* **have** *e0*: $Suc\ i > n$ **by** *simp*

let $?pes = \lambda k.\ getspc\text{-}es\ (?cs2\ k!0)$
let $?s = gets\ (?c2!0)$
let $?x = getx\ (?c2!0)$
let $?pre1 = \{?s\}$
let $?Pre = \lambda k.\ \{?s\}$

from *p1 p11 c5* **have** *e1*: $?c2 \propto ?cs2$ **using** *drop-n-conjoin*[*of c cs n ?c2 ?cs2*] **by** *auto*

from *p15 p11 c1 c3 c4 c5-1* **have** $?c2 \in cpts\text{-}pes$ **using** *cpts-pes-dropi*[*of c n−1*]
a3 e0 less-Suc-eq-0-disj less-trans **by** *auto*
moreover
have $?c2!0 = (?pes,\ ?s,\ ?x)$
proof −
from *c5 e1* **have** $\forall k.\ getspc\ (drop\ n\ c\ !\ 0)\ k = getspc\text{-}es\ (drop\ n\ (cs\ k)\ !\ 0)$
using *conjoin-def*[*of ?c2 ?cs2*] *same-spec-def*[*of ?c2 ?cs2*]
by (*metis length-drop p11 zero-less-diff*)
then have $getspc\ (?c2!0) = ?pes$ **by** *auto*
then show *?thesis* **using** *pesconf-trip*[*of ?c2!0 ?s ?pes ?x*] **by** *simp*
qed
ultimately have *e2*: $?c2 \in cpts\text{-}of\text{-}pes\ ?pes\ ?s\ ?x$
using *cpts-of-pes-def*[*of ?pes ?s ?x*] **by** *simp*

from *p8 p11 c5* **have** *e3*: $?c2 \in assume\text{-}pes(?pre1,\ rely1)$
using *assume-pes-drop-n*[*of n c pre1 rely1 ?pre1*]
by (*simp add*: *hd-conv-nth hd-drop-conv-nth not-le singleton-iff*)
have *e4*: $\forall k1.\ (?cs2\ k1) \in cpts\text{-}of\text{-}es\ (?pes\ k1)\ ?s\ ?x$
proof −
{
fix *k1*
from *p11 p12 c5* **have** *d1*: $?cs2\ k1 \in cpts\text{-}es$ **by** (*simp add*: *cpts-es-dropi2*)

have $getspc\text{-}es\ ((?cs2\ k1)!0) = ?pes\ k1$ **by** *simp*
moreover
have $gets\text{-}es\ ((?cs2\ k1)!0) = ?s$
using *conjoin-def*[*of ?c2 ?cs2*] *same-state-def*[*of ?c2 ?cs2*]
by (*metis c5 e1 length-drop p11 zero-less-diff*)
moreover
have $getx\text{-}es\ ((?cs2\ k1)!0) = ?x$
using *conjoin-def*[*of ?c2 ?cs2*] *same-state-def*[*of ?c2 ?cs2*]
by (*metis c5 e1 length-drop p11 zero-less-diff*)
ultimately have $(?cs2\ k1)!0 = (?pes\ k1,\ ?s,\ ?x)$
using *esconf-trip*[*of (?cs2 k1)!0 ?s ?pes k1 ?x*] **by** *simp*
with *d1* **have** $?cs2\ k1 \in cpts\text{-}of\text{-}es\ (?pes\ k1)\ ?s\ ?x$ **using** *cpts-of-es-def*[*of ?pes k1 ?s ?x*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed

```
have ∀n k. n ≤ length (cs k) ∧ n > 0
                        ⟶ take n (cs k)∈assume-es(Pre k, Rely k)
  using conjoin-comm-imp-rely-n[of pre1 Pre rely1 Rely Guar cs Post c pes s x]
    p16 p9 p10 p4 p0 p8 p1 p2 by auto
with p11 p12 p13 have e6: ∀k. cs k∈assume-es(Pre k, Rely k)
  using order-refl take-all by auto
then have e7: ∀k. cs k∈commit-es(Guar k, Post k)
  by (meson IntI contra-subsetD es-validity-def p16 p2)
from e6 p11 c5 have e8: ∀k. (?cs2 k)∈assume-es(?Pre k, Rely k)
  using assume-es-drop-n[of n] by (smt Un-insert-right conjoin-def drop-0
      hd-drop-conv-nth insertI1 length-drop p1 same-state-def zero-less-diff)
from e7 p11 c5 have e9: ∀k. ?cs2 k∈commit-es(Guar k, Post k)
  using commit-es-drop-n[of n] by smt

have e10: ∀k. ?pre1 ⊆ ?Pre k by simp

from p6 c5 p11 have e11: ∀j. Suc j < length ?c2 ⟶ (∃actk. ?c2!j−pes−actk→?c2!Suc j)
  proof −
  {
    fix j
    assume f0: Suc j < length ?c2
    with p11 c5 have f1: Suc (n + j) < length c
      by (metis Suc-diff-Suc Suc-eq-plus1 Suc-neq-Zero add-diff-inverse-nat
        diff-add-0 diff-diff-add length-drop)
    with p6 have ∃actk. c!(n+j)−pes−actk→c!Suc (n+j) by auto
    moreover
    from p11 c5 f0 f1 have c ! (n + j) = drop n c ! j
      by (metis Suc-leD less-imp-le-nat nth-drop)
    moreover
    from p11 c5 f0 f1 have c ! Suc (n + j) = drop n c ! Suc j
      by (simp add: less-or-eq-imp-le)
    ultimately have ∃actk. ?c2!j−pes−actk→?c2!Suc j by simp
  }
  then show ?thesis by auto qed

from p1 have gets (c!n) = gets-es (cs k ! n)
  using conjoin-def[of c cs] same-state-def[of c cs] c5 p11 by auto
moreover
from c5 have gets-es (last (take (Suc n) (cs k))) = gets-es (cs k ! n)
  by (simp add: take-Suc-conv-app-nth)
moreover
from c5 have gets (drop n c ! 0) = gets (c!n) using c5-1 by auto
ultimately have e12: ?s∈Pre_f (snd esf) using r1 b12 by auto

from b18 c3 have e13: evtsys-spec (fst esf) = getspc-es (?cs2 k ! 0)
  using c5 drop-eq-Nil hd-conv-nth hd-drop-conv-nth not-less by auto
from a2 have e14: ∀e ∈ all-evts-es (fst esf). is-basicevt (E_e e)
  using all-evts-es-seq[of ef esf] by simp
from a02 have e15: ∀e ∈ all-evts-es (fst esf). the (evtrgfs (E_e e)) = snd e
  using all-evts-es-seq[of ef esf] by simp

{
  fix ii
  from e2 e1 e3 e4 e8 e9 e10 p10 p4 e11 e12 b1 b2 b3 b4 b6 b7 b8 b9 b10 b11 b12 p9 p10 p4
    e13 e14 e15
  have Suc ii < length (?cs2 k) ∧ ((?cs2 k)!ii −es−((Cmd cmd)♯k)→ (?cs2 k)!(Suc ii))
        ⟶ (gets-es ((?cs2 k)!ii), gets-es ((?cs2 k)!(Suc ii)))∈Guar_f (the (evtrgfs (getx-es ((?cs2 k)!ii) k)))
    using pp[of ?Pre k Rely Guar Post ?c2 ?pes ?s ?x ?cs2 ?pre1 rely1 ii cmd] by force
```

}
then have $\forall i.\ Suc\ i < length\ (?cs2\ k) \wedge ((?cs2\ k)!i\ -es-((Cmd\ cmd)\sharp k)\rightarrow (?cs2\ k)!(Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((?cs2\ k)!i),\ gets\text{-}es\ ((?cs2\ k)!(Suc\ i)))\in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ ((?cs2\ k)!i)\ k)))$
by *auto*
moreover
from *a3 e0* **have** $cs\ k\ !\ i = (?cs2\ k)!(i - n)$
using *Suc-lessD add-diff-inverse-nat less-imp-le-nat not-less-eq nth-drop* **by** *auto*
moreover
from *a3 e0* **have** $cs\ k\ !\ Suc\ i = (?cs2\ k)!Suc\ (i - n)$
by (*simp add*: *Suc-diff-le add-diff-inverse-nat d0 less-Suc-eq-le less-or-eq-imp-le*)
ultimately show *?thesis* **using** *a3 e0 a4 c5*
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc*
diff-Suc-Suc length-drop less-diff-iff less-imp-le-nat)

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

lemma *act-cpts-es-sat-guar-curevt-new2*[*rule-format*]:
$[\![\vdash esspc\ sat_s\ [pre,\ rely,\ guar,\ post]]\!]$
$\Longrightarrow \forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq pre \wedge Rely\ k \subseteq rely \wedge guar \subseteq Guar\ k \wedge post \subseteq Post\ k \longrightarrow$
$c\in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c\in assume\text{-}pes(pre1,\ rely1) \longrightarrow$
$(\forall k.\ (cs\ k) \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ cs\ k \in commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ esspc = getspc\text{-}es\ (cs\ k!0) \longrightarrow$
$(\forall e\in all\text{-}evts\text{-}es\ esspc.\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e\in all\text{-}evts\text{-}es\ esspc.\ the\ ((evtrgfs::('l,'k,'s)\ event \Rightarrow 's\ rgformula\ option)\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c!j-pes-actk\rightarrow c!Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge ((cs\ k)!i\ -es-((Cmd\ cmd)\sharp k)\rightarrow (cs\ k)!(Suc\ i))$
$\longrightarrow (gets\text{-}es\ ((cs\ k)!i),\ gets\text{-}es\ ((cs\ k)!(Suc\ i)))\in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ ((cs\ k)!i)\ k))))$
apply(*rule rghoare-es.induct*[*of esspc pre rely guar post*])
apply *simp*
proof −
{
fix *ef esf prea posta relya guara*
assume *p0*: $\vdash esspc\ sat_s\ [pre,\ rely,\ guar,\ post]$
and *p1*: $\vdash E_e\ (ef::('l,'k,'s)\ rgformula\text{-}e)\ sat_e\ [Pre_e\ ef,\ Rely_e\ ef,\ Guar_e\ ef,\ Post_e\ ef]$
and *p2*: $\vdash fst\ (esf::('l,'k,'s)\ rgformula\text{-}es)\ sat_s$
$[Pre_f\ (snd\ esf),\ Rely_f\ (snd\ esf),\ Guar_f\ (snd\ esf),\ Post_f\ (snd\ esf)]$
and *p3*: $\forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq Pre_f\ (snd\ esf) \wedge Rely\ k \subseteq Rely_f\ (snd\ esf)$
$\wedge\ Guar_f\ (snd\ esf) \subseteq Guar\ k \wedge Post_f\ (snd\ esf) \subseteq Post\ k \longrightarrow$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1) \longrightarrow$
$(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ cs\ k \in commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ (fst\ esf) = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e\in all\text{-}evts\text{-}es\ (fst\ esf).\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e\in all\text{-}evts\text{-}es\ (fst\ esf).\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$

(∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*)) ⟶
(∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))))
and *p4*: *prea* = Pre_e *ef*
and *p5*: *posta* = $Post_f$ (*snd esf*)
and *p6*: *relya* ⊆ $Rely_e$ *ef*
and *p7*: *relya* ⊆ $Rely_f$ (*snd esf*)
and *p8*: $Guar_e$ *ef* ⊆ *guara*
and *p9*: $Guar_f$ (*snd esf*) ⊆ *guara*
and *p10*: $Post_e$ *ef* ⊆ Pre_f (*snd esf*)
then have *p11*: ⊢ (*rgf-EvtSeq ef esf*) sat_s [*prea*, *relya*, *guara*, *posta*]
using *EvtSeq-h*[*of ef esf prea posta relya guara*] **by** *simp*

{
fix *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*
assume *a0*: *Pre k* ⊆ *prea* ∧ *Rely k* ⊆ *relya* ∧ *guara* ⊆ *Guar k* ∧ *posta* ⊆ *Post k*
and *a1*: *c* ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*)
and *a2*: (∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*)
and *a3*: (∀ *k*. *cs k* ∈ *commit-es*(*Guar k*, *Post k*))
and *a4*: (∀ *k*. *pre1* ⊆ *Pre k*)
and *a5*: (∀ *k*. *rely1* ⊆ *Rely k*)
and *a6*: (∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*)
and *a7*: *evtsys-spec* (*rgf-EvtSeq ef esf*) = *getspc-es* (*cs k* ! *0*)
and *a8*: (∀ *e*∈*all-evts-es* (*rgf-EvtSeq ef esf*). *is-basicevt* (E_e *e*))
and *a9*: (∀ *e*∈*all-evts-es* (*rgf-EvtSeq ef esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*)
and *a10*: (∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*))
then have ∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*)))
using *p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 act-cpts-evtseq-sat-guar-curevt-new2*
[*of ef esf prea posta relya guara Pre k Rely Guar*
Post c pes s x cs pre1 rely1 evtrgfs cmd] **by** *blast*
}

then show ∀ *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*.
Pre k ⊆ *prea* ∧ *Rely k* ⊆ *relya* ∧ *guara* ⊆ *Guar k* ∧ *posta* ⊆ *Post k* ⟶
c ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*) ⟶
(∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*) ⟶
(∀ *k*. *cs k* ∈ *commit-es*(*Guar k*, *Post k*)) ⟶
(∀ *k*. *pre1* ⊆ *Pre k*) ⟶
(∀ *k*. *rely1* ⊆ *Rely k*) ⟶
(∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*) ⟶
evtsys-spec (*rgf-EvtSeq ef esf*) = *getspc-es* (*cs k* ! *0*) ⟶
(∀ *e*∈*all-evts-es* (*rgf-EvtSeq ef esf*). *is-basicevt* (E_e *e*)) ⟶
(∀ *e*∈*all-evts-es* (*rgf-EvtSeq ef esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*) ⟶
(∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*)) ⟶
(∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))))
by *fastforce*
}
next
{
fix *esf prea relya guara posta*
assume *a0*: ⊢ *esspc* sat_s [*pre*, *rely*, *guar*, *post*]
and *a1*: ∀ *ef*∈(*esf*::('*l*,'*k*,'*s*) *rgformula-e set*).
⊢ E_e *ef* sat_e [Pre_e *ef*, $Rely_e$ *ef*, $Guar_e$ *ef*, $Post_e$ *ef*]
and *a2*: ∀ *ef*∈*esf*. *prea* ⊆ Pre_e *ef*
and *a3*: ∀ *ef*∈*esf*. *relya* ⊆ $Rely_e$ *ef*
and *a4*: ∀ *ef*∈*esf*. $Guar_e$ *ef* ⊆ *guara*

and *a5*: ∀ *ef*∈*esf*. $Post_e$ *ef* ⊆ *posta*
and *a6*: ∀ *ef1 ef2*. *ef1* ∈ *esf* ∧ *ef2* ∈ *esf* ⟶ $Post_e$ *ef1* ⊆ Pre_e *ef2*
and *a7*: *stable prea relya*
and *a8*: ∀ *s*. (*s*, *s*) ∈ *guara*
then have *a9*: ⊢ *rgf-EvtSys esf* sat_s [*prea*, *relya*, *guara*, *posta*]
using *EvtSys-h*[*of esf prea relya guara posta*] **by** *simp*

{
fix *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*
assume *b0*: *Pre k* ⊆ *prea* ∧ *Rely k* ⊆ *relya* ∧ *guara* ⊆ *Guar k* ∧ *posta* ⊆ *Post k*
and *b1*: *c* ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*)
and *b2*: (∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*)
and *b3*: (∀ *k*. (*cs k*)∈*commit-es*(*Guar k*, *Post k*))
and *b4*: (∀ *k*. *pre1* ⊆ *Pre k*)
and *b5*: (∀ *k*. *rely1* ⊆ *Rely k*)
and *b6*: (∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*)
and *b7*: *evtsys-spec* (*rgf-EvtSys esf*) = *getspc-es* (*cs k* ! *0*)
and *b8*: (∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *is-basicevt* (E_e *e*))
and *b9*: (∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*)
and *b10*: (∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*))
from *b7* **have** ∃ *es*. *evtsys-spec* (*rgf-EvtSys esf*) = *EvtSys es*
using *evtsys-spec-evtsys* **by** *blast*
then obtain *es* **where** *b11*: *evtsys-spec* (*rgf-EvtSys esf*) = *EvtSys es* **by** *auto*

with *a9 b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10*
have ∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*)))
using *act-cpts-evtsys-sat-guar-curevt-gen0-new2*[*of rgf-EvtSys esf prea*
relya guara posta Pre k Rely Guar Post c pes s x cs pre1 rely1 es evtrgfs] **by** *fastforce*
}
then show ∀ *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*.
Pre k ⊆ *prea* ∧ *Rely k* ⊆ *relya* ∧ *guara* ⊆ *Guar k* ∧ *posta* ⊆ *Post k* ⟶
c ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*) ⟶
(∀ *k*. *cs k* ∈ *cpts-of-es* (*pes k*) *s x*) ⟶
(∀ *k*. (*cs k*)∈*commit-es*(*Guar k*, *Post k*)) ⟶
(∀ *k*. *pre1* ⊆ *Pre k*) ⟶
(∀ *k*. *rely1* ⊆ *Rely k*) ⟶
(∀ *k j*. *j* ≠ *k* ⟶ *Guar j* ⊆ *Rely k*) ⟶
evtsys-spec (*rgf-EvtSys esf*) = *getspc-es* (*cs k* ! *0*) ⟶
(∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *is-basicevt* (E_e *e*)) ⟶
(∀ *e*∈*all-evts-es* (*rgf-EvtSys esf*). *the* (*evtrgfs* (E_e *e*)) = *snd e*) ⟶
(∀ *j*. *Suc j* < *length c* ⟶ (∃ *actk*. *c* ! *j* −*pes*−*actk*→ *c* ! *Suc j*)) ⟶
(∀ *i*. *Suc i* < *length* (*cs k*) ∧ *cs k* ! *i* −*es*−*Cmd cmd*♯*k*→ *cs k* ! *Suc i* ⟶
(*gets-es* (*cs k* ! *i*), *gets-es* (*cs k* ! *Suc i*)) ∈ $Guar_f$ (*the* (*evtrgfs* (*getx-es* (*cs k* ! *i*) *k*))))
by *fastforce*
}
next
{
fix *prea pre′ relya rely′ guar′ guara post′ posta esys*
assume *a0*: ⊢ *esspc* sat_s [*pre*, *rely*, *guar*, *post*]
and *a1*: *prea* ⊆ *pre′*
and *a2*: *relya* ⊆ *rely′*
and *a3*: *guar′* ⊆ *guara*
and *a4*: *post′* ⊆ *posta*
and *a5*: ⊢ *esys* sat_s [*pre′*, *rely′*, *guar′*, *post′*]
and *a6*[*rule-format*]: ∀ *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*.
Pre k ⊆ *pre′* ∧ *Rely k* ⊆ *rely′* ∧ *guar′* ⊆ *Guar k* ∧ *post′* ⊆ *Post k* ⟶
c ∈ *cpts-of-pes pes s x* ∧ *c* ∝ *cs* ∧ *c* ∈ *assume-pes* (*pre1*, *rely1*) ⟶

$(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ (cs\ k) \in commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ esys = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ -pes-actk\rightarrow\ c\ !\ Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$

{

fix *c pes s x cs pre1 rely1 Pre Rely Guar Post k cmd*
assume *b0*: $Pre\ k \subseteq prea \wedge Rely\ k \subseteq relya \wedge guara \subseteq Guar\ k \wedge posta \subseteq Post\ k$
and *b1*: $c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1)$
and *b2*: $(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x)$
and *b3*: $(\forall k.\ (cs\ k) \in commit\text{-}es(Guar\ k,\ Post\ k))$
and *b4*: $(\forall k.\ pre1 \subseteq Pre\ k)$
and *b5*: $(\forall k.\ rely1 \subseteq Rely\ k)$
and *b6*: $(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k)$
and *b7*: $evtsys\text{-}spec\ esys = getspc\text{-}es\ (cs\ k\ !\ 0)$
and *b8*: $(\forall e \in all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e))$
and *b9*: $(\forall e \in all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e)$
and *b10*: $(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ -pes-actk\rightarrow\ c\ !\ Suc\ j))$
from *a1 a2 a3 a4 b0* **have** $Pre\ k \subseteq pre' \wedge Rely\ k \subseteq rely' \wedge guar' \subseteq Guar\ k \wedge post' \subseteq Post\ k$ **by** *auto*
with *a1 a2 a3 a5 a6[of Pre k Rely Guar Post c pes s x cs pre1 rely1] b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10*
have $\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k)))$ **by** *force*

}

then show $\forall c\ pes\ s\ x\ cs\ pre1\ rely1\ Pre\ Rely\ Guar\ Post\ k\ cmd.$
$Pre\ k \subseteq prea \wedge Rely\ k \subseteq relya \wedge guara \subseteq Guar\ k \wedge posta \subseteq Post\ k \longrightarrow$
$c \in cpts\text{-}of\text{-}pes\ pes\ s\ x \wedge c \propto cs \wedge c \in assume\text{-}pes\ (pre1,\ rely1) \longrightarrow$
$(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ (pes\ k)\ s\ x) \longrightarrow$
$(\forall k.\ (cs\ k) \in commit\text{-}es(Guar\ k,\ Post\ k)) \longrightarrow$
$(\forall k.\ pre1 \subseteq Pre\ k) \longrightarrow$
$(\forall k.\ rely1 \subseteq Rely\ k) \longrightarrow$
$(\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k) \longrightarrow$
$evtsys\text{-}spec\ esys = getspc\text{-}es\ (cs\ k\ !\ 0) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ esys.\ is\text{-}basicevt\ (E_e\ e)) \longrightarrow$
$(\forall e \in all\text{-}evts\text{-}es\ esys.\ the\ (evtrgfs\ (E_e\ e)) = snd\ e) \longrightarrow$
$(\forall j.\ Suc\ j < length\ c \longrightarrow (\exists actk.\ c\ !\ j\ -pes-actk\rightarrow\ c\ !\ Suc\ j)) \longrightarrow$
$(\forall i.\ Suc\ i < length\ (cs\ k) \wedge cs\ k\ !\ i\ -es-Cmd\ cmd\sharp k\rightarrow\ cs\ k\ !\ Suc\ i \longrightarrow$
$(gets\text{-}es\ (cs\ k\ !\ i),\ gets\text{-}es\ (cs\ k\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ (cs\ k\ !\ i)\ k))))$
by *fastforce*

}

qed

lemma *act-cptpes-sat-guar-curevt-new2*:
$[\![\vdash (pesf::('l,'k,'s)\ rgformula\text{-}par)\ SAT\ [pre,\ \{\},\ UNIV,\ post]]\!] \Longrightarrow$
$s0 \in pre \longrightarrow$
$(\forall ef \in all\text{-}evts\ pesf.\ is\text{-}basicevt\ (E_e\ ef)) \longrightarrow$
$(\forall erg \in all\text{-}evts\ pesf.\ the\ (evtrgfs\ (E_e\ erg)) = snd\ erg) \longrightarrow$
$pesl \in cpts\text{-}of\text{-}pes\ (paresys\text{-}spec\ pesf)\ s0\ x0 \longrightarrow$
$(\forall j.\ Suc\ j < length\ pesl \longrightarrow (\exists actk.\ pesl!j-pes-actk\rightarrow pesl!Suc\ j)) \longrightarrow$
$(\forall k\ i.\ Suc\ i < length\ pesl \longrightarrow (\exists c.\ (pesl!i\ -pes-((Cmd\ c)\sharp k)\rightarrow\ pesl!(Suc\ i)))$
$\longrightarrow (gets\ (pesl!i),gets\ (pesl!Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\ (pesl!i)\ k))))$
apply(*rule rghoare-pes.induct[of pesf pre {} UNIV post]*)

```
apply simp
prefer 2
apply blast
proof −
{
  fix pesfa prea rely guar posta
  assume a0: ⊢ pesf SAT [pre, {}, UNIV, post]
     and  a4: ∀k. ⊢ fst ((pesfa::('l,'k,'s) rgformula-par) k)
                  sat_s [Pre_es (pesfa k), Rely_es (pesfa k), Guar_es (pesfa k), Post_es (pesfa k)]
     and  a5: ∀k. prea ⊆ Pre_es (pesfa k)
     and  a6: ∀k. rely ⊆ Rely_es (pesfa k)
     and  a7: ∀k j. j ≠ k ⟶ Guar_es (pesfa j) ⊆ Rely_es (pesfa k)
     and  a8: ∀k. Guar_es (pesfa k) ⊆ guar
     and  a9: ∀k. Post_es (pesfa k) ⊆ posta

  show s0 ∈ prea ⟶
        (∀ef∈all-evts pesfa. is-basicevt (E_e ef)) ⟶
        (∀erg∈all-evts pesfa. the (evtrgfs (E_e erg)) = snd erg) ⟶
        pesl ∈ cpts-of-pes (paresys-spec pesfa) s0 x0 ⟶
     (∀j. Suc j < length pesl ⟶ (∃actk. pesl ! j −pes−actk→ pesl ! Suc j)) ⟶
     (∀k i. Suc i < length pesl ⟶
           (∃c. pesl ! i −pes−Cmd c♯k→ pesl ! Suc i) ⟶
           (gets (pesl ! i), gets (pesl ! Suc i)) ∈ Guar_f (the (evtrgfs (getx (pesl ! i) k))))
    proof −
    {
      assume b0: pesl ∈ cpts-of-pes (paresys-spec pesfa) s0 x0
        and  b1: ∀j. Suc j < length pesl ⟶ (∃actk. pesl ! j −pes−actk→ pesl ! Suc j)
        and  b2: ∀ef∈all-evts pesfa. is-basicevt (E_e ef)
        and  b3: ∀erg∈all-evts pesfa. the (evtrgfs (E_e erg)) = snd erg
        and  b4: s0 ∈ prea

      from b0 have b5: pesl∈cpts-pes ∧ pesl!0 = (paresys-spec pesfa, s0, x0)
        by (simp add:cpts-of-pes-def)
      let ?pes = paresys-spec pesfa
      from b0 have ∃cs. (∀k. (cs k) ∈ cpts-of-es (?pes k) s0 x0) ∧ pesl ∝ cs
        using par-evtsys-semantics-comp[of ?pes s0 x0] by auto
      then obtain cs where b6: (∀k. (cs k) ∈ cpts-of-es (?pes k) s0 x0) ∧ pesl ∝ cs by auto
      then have b7: ∀k. length (cs k) = length pesl
        using conjoin-def[of pesl cs] same-length-def[of pesl cs] by auto

      have b8: pesl∈assume-pes(prea,rely)
        proof −
          from b4 have gets (paresys-spec pesfa, s0, x0) ∈ prea using gets-def
            by (metis fst-conv snd-conv)
          moreover
          from b1 have ∀i. Suc i < length pesl ⟶ ¬(pesl ! i −pese→ pesl ! Suc i)
            using pes-tran-not-etran1 by blast
          ultimately show ?thesis using b5 by (simp add:assume-pes-def)
        qed

      {
        fix k i
        assume c0: Suc i < length pesl
          and  c1: ∃c. pesl ! i −pes−Cmd c♯k→ pesl ! Suc i

        from c1 obtain c where c2: pesl ! i −pes−Cmd c♯k→ pesl ! Suc i by auto
        from c1 have c3: ¬((pesl!i) −pese→ (pesl!Suc i)) using pes-tran-not-etran1 by blast
        with b6 c0 c1 have (∀k t. (pesl ! i −pes−t♯k→ pesl ! Suc i) ⟶
```

$(cs\ k\ !\ i\ -es-t\sharp k\rightarrow\ cs\ k\ !\ Suc\ i) \wedge (\forall k'.\ k' \neq k \longrightarrow cs\ k'\ !\ i\ -ese\rightarrow\ cs\ k'\ !\ Suc\ i))$
using *conjoin-def* [*of pesl cs*] *compat-tran-def* [*of pesl cs*] **by** *auto*
with *c2* **have** *c4*: $(cs\ k!i\ -es-(Cmd\ c\sharp k)\rightarrow\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i))$ **by** *auto*
from *c0 b6* **have** *c5*: $gets\ (pesl!i) = gets\text{-}es\ ((cs\ k)!i) \wedge getx\ (pesl!i) = getx\text{-}es\ ((cs\ k)!i)$
using *conjoin-def* [*of pesl cs*] *same-state-def* [*of pesl cs*] **by** *auto*
from *c0 b6* **have** *c6*: $gets\ (pesl!Suc\ i) = gets\text{-}es\ ((cs\ k)!Suc\ i)$
$\wedge\ getx\ (pesl!Suc\ i) = getx\text{-}es\ ((cs\ k)!Suc\ i)$
using *conjoin-def* [*of pesl cs*] *same-state-def* [*of pesl cs*] **by** *auto*

from *a4* **have** $\vdash fst\ (pesfa\ k)\ sat_s\ [Pre_{es}\ (pesfa\ k),\ Rely_{es}\ (pesfa\ k),\ Guar_{es}\ (pesfa\ k),\ Post_{es}\ (pesfa\ k)]$ **by** *auto*
moreover
from *a4* **have** *c7*: $\forall k. \models paresys\text{-}spec\ pesfa\ k\ sat_s\ [(Pre_{es} \circ pesfa)\ k,\ (Rely_{es} \circ pesfa)\ k,$
$(Guar_{es} \circ pesfa)\ k,\ (Post_{es} \circ pesfa)\ k]$
by (*simp add*: *paresys-spec-def rgsound-es*)
moreover
from *b5 b6* **have** *c8*: $evtsys\text{-}spec\ (fst\ (pesfa\ k)) = getspc\text{-}es\ (cs\ k\ !\ 0)$
using *conjoin-def* [*of pesl cs*] *same-spec-def* [*of pesl cs*] *paresys-spec-def* [*of pesfa*]
by (*metis* (*no-types*, *lifting*) *c0 dual-order.strict-trans fst-conv getspc-def zero-less-Suc*)
moreover
from *b2* **have** $\forall e.\ e \in all\text{-}evts\text{-}es\ (fst\ (pesfa\ k)) \longrightarrow is\text{-}basicevt\ (E_e\ e)$
using *all-evts-def* [*of pesfa*] **by** *auto*
moreover
from *b3* **have** $\forall e.\ e \in all\text{-}evts\text{-}es\ (fst\ (pesfa\ k)) \longrightarrow the\ (evtrgfs\ (E_e\ e)) = snd\ e$
using *all-evts-def* [*of pesfa*] **by** *auto*
moreover
have $\forall k.\ cs\ k \in commit\text{-}es\ ((Guar_{es} \circ pesfa)\ k,\ (Post_{es} \circ pesfa)\ k)$
proof −
have $\forall k.\ cs\ k \in assume\text{-}es((Pre_{es} \circ pesfa)\ k,\ (Rely_{es} \circ pesfa)\ k)$
using *conjoin-es-sat-assume* [*of paresys-spec pesfa* $Pre_{es} \circ pesfa\ Rely_{es} \circ pesfa$
$Guar_{es} \circ pesfa\ Post_{es} \circ pesfa$ *prea rely pesl s0 x0 cs*] *c7 a5 a6 a7 b0 b6 b8* **by** *auto*
with *c7 c8* **show** *?thesis* **using** *paresys-spec-def* [*of pesfa*]
by (*meson IntI b6 contra-subsetD cpts-of-es-def es-validity-def*)
qed
ultimately
have $(gets\text{-}es\ ((cs\ k)!i),\ gets\text{-}es\ ((cs\ k)!(Suc\ i))) \in Guar_f\ (the\ (evtrgfs\ (getx\text{-}es\ ((cs\ k)!i)\ k)))$
using *act-cpts-es-sat-guar-curevt-new2* [*of fst* $(pesfa\ k)\ Pre_{es}\ (pesfa\ k)$
$Rely_{es}\ (pesfa\ k)\ Guar_{es}\ (pesfa\ k)\ Post_{es}\ (pesfa\ k)\ Pre_{es} \circ pesfa\ k\ Rely_{es} \circ pesfa$
$Guar_{es} \circ pesfa\ Post_{es} \circ pesfa$ *pesl paresys-spec pesfa s0 x0 cs prea rely evtrgfs i c*]
a5 a6 a7 a8 a9 b0 b1 b4 b6 b8 c4 c0 b7 **by** *auto*

with *c5 c6* **have** $(gets\ (pesl\ !\ i),\ gets\ (pesl\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\ (pesl\ !\ i)\ k)))$
by *simp*
}
then have $\forall k\ i.\ Suc\ i < length\ pesl \longrightarrow$
$(\exists c.\ pesl\ !\ i\ -pes-Cmd\ c\sharp k\rightarrow\ pesl\ !\ Suc\ i) \longrightarrow$
$(gets\ (pesl\ !\ i),\ gets\ (pesl\ !\ Suc\ i)) \in Guar_f\ (the\ (evtrgfs\ (getx\ (pesl\ !\ i)\ k)))$ **by** *auto*
}
then show *?thesis* **by** *auto*
qed
}
qed

end

9 Rely-guarantee-based Safety Reasoning

```
theory PiCore-RG-Invariant
imports PiCore-RG-Prop
begin

type-synonym 's invariant = 's set

definition no-environment :: ('l,'k,'s) pesconfs ⇒ bool
  where no-environment pesl ≡ (∀ j. Suc j < length pesl ⟶ (∃ actk. pesl!j−pes−actk→pesl!Suc j))

definition invariant-of-pares::('l,'k,'s) paresys ⇒ 's set ⇒ 's invariant ⇒ bool
  where invariant-of-pares pares init invar ≡
        ∀ s0 x0 pesl. s0∈init ∧ pesl∈cpts-of-pes pares s0 x0 ∧ no-environment pesl
                ⟶ (∀ i<length pesl. gets (pesl!i) ∈ invar)

theorem invariant-theorem:
  assumes parsys-sat-rg: ⊢ pesf SAT [init, {}, UNIV, UNIV]
    and all-evts-are-basic: ∀ ef∈all-evts pesf. is-basicevt (E_e ef)
    and evt-in-parsys-in-evtrgfs: ∀ erg∈all-evts pesf. the (evtrgfs (E_e erg)) = snd erg
    and stb-invar: ∀ ef∈all-evts pesf. stable invar (Guar_e ef)
    and init-in-invar: init⊆invar
  shows invariant-of-pares (paresys-spec pesf) init invar
  proof −
  {
    fix s0 x0 pesl
    assume a0: s0∈init
      and a1: pesl∈cpts-of-pes (paresys-spec pesf) s0 x0
      and no-environment pesl
   then have a2: ∀ j. Suc j < length pesl ⟶ (∃ actk. pesl!j−pes−actk→pesl!Suc j) by (simp add:no-environment-def)
    from a1 have a3: pesl!0 = (paresys-spec pesf, s0, x0) ∧ pesl∈cpts-pes by (simp add:cpts-of-pes-def)

    {
      fix i
      assume b0: i<length pesl
      then have gets (pesl!i) ∈ 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) ∈ invar
            and c1: Suc ni < length pesl
          then have c2: gets (pesl ! ni) ∈ invar by auto
          from a3 c1 have pesl ! ni −pese→ pesl ! Suc ni ∨ (∃ et. pesl ! ni −pes−et→ pesl ! Suc ni)
            using incpts-pes-impl-evnorcomptran by blast
          then show ?case
            proof
              assume d0: pesl ! ni −pese→ pesl ! Suc ni
              then show ?thesis using a2 c1 pes-tran-not-etran1 by blast
            next
              assume ∃ et. pesl ! ni −pes−et→ pesl ! Suc ni
              then obtain et where d0: pesl ! ni −pes−et→ pesl ! Suc ni by auto
              then obtain act and k where d1: et = act♯k using get-actk-def by (metis actk.cases)
              then show ?thesis
                proof(induct act)
                  case (Cmd x)
```

assume *e0*: *et* = *Cmd x♯k*
have *e1*: (*gets* (*pesl*!*ni*),*gets* (*pesl*!*Suc ni*))∈ $Guar_f$ (*the* (*evtrgfs* (*getx* (*pesl*!*ni*) *k*)))
using *act-cptpes-sat-guar-curevt-new2*[*of pesf init UNIV s0 evtrgfs pesl x0*]
parsys-sat-rg a0 all-evts-are-basic evt-in-parsys-in-evtrgfs a1 a2 c1 d0 e0 **by** *auto*

have ∃ *ef*∈*all-evts pesf*. *getx* (*pesl*!*ni*) *k* = E_e *ef*
using *cur-evt-in-specevts*[*of pesl pesf s0 x0*] *a1 a2 all-evts-are-basic c1 d0 e0* **by** *auto*
then obtain *ef* **where** *e2*: *ef*∈*all-evts pesf* ∧ *getx* (*pesl*!*ni*) *k* = E_e *ef* **by** *auto*
with *e1* **have** (*gets* (*pesl*!*ni*),*gets* (*pesl*!*Suc ni*))∈$Guar_e$ *ef* **using** *evt-in-parsys-in-evtrgfs*
by (*simp add*: $Guar_e$*-def* $Guar_f$*-def*)
with *stb-invar e2 c2* **show** *?case* **by** (*meson stable-def*)
next
case (*EvtEnt x*)
assume *e0*: *et* = *EvtEnt x♯k*
with *c2 d0* **show** *?case* **using** *evtent-in-pes-notchgstate2*[*of pesl* ! *ni x k pesl* ! *Suc ni*] **by** *simp*
qed
qed
qed
}
}
then show *?thesis* **using** *invariant-of-pares-def* **by** *blast*
qed

end

10 Concrete Syntax of PiCore Language

theory *PiCore-Syntax*
imports *PiCore-Language*

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*

notation *Seq* ((-;;/ -) [*60*,*61*] *60*)

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) event ((EVENT - WHEN - THEN - END) [0,0,0] 61)
-Event2    :: ['a, 'a, 'a] ⇒ ('l,'k,'s) event ((EVENT - THEN - END) [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|},bd))
EVENT l THEN bd END ⇌ EVENT l WHEN CONST True THEN bd END
```

Translations for variables before and after a transition:

syntax

```
-before :: id ⇒ 'a (ᵒ-)
-after  :: id ⇒ 'a (ᵃ-)
```

translations

```
ᵒx ⇌ x ´CONST fst
ᵃx ⇌ 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*

lemma *assert-int* [*intro*!]: $x \in \{|A|\} \Longrightarrow x \in \{|B|\} \Longrightarrow x \in \{|A \wedge B|\}$

by *blast*

end

11 Formal Specification and Reasoning of ARINC653 Multicore Microkernel

theory *ARINC653-MultiCore-QueIPC*
imports *PiCore-Syntax PiCore-RG-Invariant*
begin

11.1 functional specification

typedecl *Part*
typedecl *Sched*
typedecl *Message*
typedecl *Port*
typedecl *Core*

typedecl *QChannel*

record *Config* = *c2s* :: *Core* ⇒ *Sched*
  p2s :: *Part* ⇒ *Sched*
  p2p :: *Port* ⇒ *Part*
  chsrc :: *QChannel* ⇒ *Port*
  chdest :: *QChannel* ⇒ *Port*
  chmax :: *QChannel* ⇒ *nat*

axiomatization *conf*::*Config*
  where *bij-c2s*: *bij* (*c2s conf*)
    and *portsrc-disj*: ∀ *c1 c2*. *c1* ≠ *c2* ⟶ (*chsrc conf*) *c1* ≠ (*chsrc conf*) *c2*
    and *portdest-disj*: ∀ *c1 c2*. *c1* ≠ *c2* ⟶ (*chdest conf*) *c1* ≠ (*chdest conf*) *c2*
    and *portsrcdest-disj*: ∀ *c1 c2*. (*chsrc conf*) *c1* ≠ (*chdest conf*) *c2*

lemma *inj-surj-c2s*: *inj* (*c2s conf*) ∧ *surj* (*c2s conf*)
  using *bij-c2s* **by** (*simp add*: *bij-def*)

definition *is-src-qport* :: *Config* ⇒ *Port* ⇒ *bool*
  where *is-src-qport sc p* ≡ (*p*∈*range* (*chsrc sc*))

definition *is-dest-qport* :: *Config* ⇒ *Port* ⇒ *bool*
  where *is-dest-qport sc p* ≡ (*p*∈*range* (*chdest sc*))

definition *port-of-part* :: *Config* ⇒ *Port* ⇒ *Part* ⇒ *bool*
  where *port-of-part sc po pa* ≡ ((*p2p sc*) *po* = *pa*)

definition *ch-srcqport* :: *Config* ⇒ *Port* ⇒ *QChannel*
  where *ch-srcqport sc p* ≡ *SOME c*. (*chsrc sc*) *c* = *p*

datatype *PartMode* = *IDLE* | *READY* | *RUN*

record *State* = *cur* :: *Sched* ⇒ *Part option*
  qbuf :: *QChannel* ⇒ *Message list*
  qbufsize :: *QChannel* ⇒ *nat*
  partst :: *Part* ⇒ *PartMode*

```
datatype EL = Core-InitE | ScheduleE | Send-Que-MessageE |  Recv-Que-MessageE

datatype parameter = Port Port | Message Message | Partition Part

type-synonym EventLabel = EL × (parameter list × Core)

definition get-evt-label :: EL ⇒ parameter list ⇒ Core ⇒ EventLabel (- - @ - [0,0,0] 20)
  where get-evt-label el ps k ≡ (el,(ps,k))

definition Core-Init :: Core ⇒ (EventLabel, Core, State) event
  where Core-Init k≡
    EVENT Core-InitE [] @ k
    THEN
      ´partst := (λp. if p2s conf p = c2s conf k ∧ ´partst p = IDLE
                  then READY else ´partst p)
    END

definition System-Init :: Config ⇒ (State × (EventLabel, Core, State) x)
  where System-Init cfg ≡ ((|cur=(λc. None ),
                            qbuf = (λc. []),
                            qbufsize = (λc. 0),
                            partst = (λp. IDLE)|),
                            (λk. Core-Init k))


definition Schedule :: Core ⇒ Part ⇒ (EventLabel, Core, State) event
  where Schedule k p ≡
    EVENT ScheduleE [Partition p] @ k
    WHEN
      p2s conf p = c2s conf k
      ∧ (´partst p ≠ IDLE)
      ∧ (´cur((c2s conf) k) = None
         ∨ p2s conf (the (´cur((c2s conf) k))) = c2s conf k)
    THEN
      IF (´cur((c2s conf) k) ≠ None) THEN
        ATOMIC
          ´partst := ´partst(the (´cur ((c2s conf) k)) := READY);;
          ´cur := ´cur((c2s conf) k := None)
        END
      ELSE SKIP FI;;

      ATOMIC
        ´cur := ´cur((c2s conf) k := Some p);;
        ´partst := ´partst(p := RUN)
      END

    END

definition Send-Que-Message :: Core ⇒ Port ⇒ Message ⇒ (EventLabel, Core, State) event
  where Send-Que-Message k p m ≡
    EVENT Send-Que-MessageE [Port p, Message m] @ k
    WHEN
      is-src-qport conf p
      ∧ ´cur ((c2s conf) k) ≠ None
      ∧ port-of-part conf p (the (´cur ((c2s conf) k)))
    THEN
      AWAIT ´qbufsize (ch-srcqport conf p) < chmax conf (ch-srcqport conf p) THEN
```

´*qbuf* := ´*qbuf* (*ch-srcqport conf p* := ´*qbuf* (*ch-srcqport conf p*) @ [*m*]);;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* := ´*qbufsize* (*ch-srcqport conf p*) + *1*)
END
END

definition *Recv-Que-Message* :: *Core* ⇒ *Port* ⇒ (*EventLabel*, *Core*, *State*) *event*
where *Recv-Que-Message k p* ≡
EVENT Recv-Que-MessageE [*Port p*] @ *k*
WHEN
is-dest-qport conf p
∧ ´*cur* ((*c2s conf*) *k*) ≠ *None*
∧ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))
THEN
AWAIT ´*qbufsize* (*ch-srcqport conf p*) > *0 THEN*
´*qbuf* := ´*qbuf* (*ch-srcqport conf p* := *tl* (´*qbuf* (*ch-srcqport conf p*)));;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* := ´*qbufsize* (*ch-srcqport conf p*) − *1*)
END
END

11.2 Rely-guarantee condition of events

definition *Core-Init-RGCond* :: *Core* ⇒ (*State*) *rgformula*
where *Core-Init-RGCond k* ≡
RG[⦃∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ´*partst p* = *IDLE*⦄,
⦃(∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ^{a}*partst p* = ^{o}*partst p*)⦄,
(⦃^{a}*cur*= ^{o}*cur* ∧ ^{a}*qbuf*= ^{o}*qbuf* ∧ ^{a}*qbufsize*= ^{o}*qbufsize*
∧ (∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ^{o}*partst p* = *IDLE* ∧ ^{a}*partst p* = *READY*)
∧ (∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ ^{a}*partst p* = ^{o}*partst p*)⦄ ∪ *Id*),
⦃*True*⦄]

definition *Schedule-RGCond* :: *Core* ⇒ *Part* ⇒ (*State*) *rgformula*
where *Schedule-RGCond k p* ≡
(*RG*[⦃*True*⦄,
⦃^{a}*cur* (*c2s conf k*) = ^{o}*cur* (*c2s conf k*) ∧
(∀ *p*. *p2s conf p* = *c2s conf k* ⟶ ^{a}*partst p* = ^{o}*partst p*)⦄,
(⦃(^{a}*cur* = ^{o}*cur*(*c2s conf k* := *Some p*)
∧ ^{a}*partst* = ^{o}*partst*(*the* (^{a}*cur*(*c2s conf k*)) := *RUN*)
∧ *p2s conf p* = *c2s conf k*
∨ (^{a}*cur* = ^{o}*cur*(*c2s conf k* := *None*)
∧ ^{a}*partst* = ^{o}*partst*(*the* (^{o}*cur* (*c2s conf k*)) := *READY*)))
∧ (∀ *c*. *c* ≠ *k* ⟶ ^{a}*cur* (*c2s conf c*) = ^{o}*cur* (*c2s conf c*))
∧ (∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ ^{a}*partst p* = ^{o}*partst p*)
∧ ^{a}*qbuf*= ^{o}*qbuf*
∧ ^{a}*qbufsize*= ^{o}*qbufsize*⦄ ∪ *Id*),
⦃*True*⦄])

lemma *id-belong*[*simp*]: *Id* ⊆⦃^{a}*x*= ^{o}*x*⦄
by (*simp add*: *Collect-mono Id-fstsnd-eq*)

definition *Send-Que-Message-RGCond* :: *Core* ⇒ *Port* ⇒ *Message* ⇒ (*State*) *rgformula*
where *Send-Que-Message-RGCond k p m* ≡ (
RG[⦃*True*⦄,
⦃^{a}*cur* (*c2s conf k*) = ^{o}*cur* (*c2s conf k*)⦄,
(⦃^{a}*cur* = ^{o}*cur* ∧ ^{a}*partst* = ^{o}*partst* ∧
(^{o}*qbufsize* (*ch-srcqport conf p*) = *length* (^{o}*qbuf* (*ch-srcqport conf p*))
⟶ ^{a}*qbufsize* (*ch-srcqport conf p*) = *length* (^{a}*qbuf* (*ch-srcqport conf p*))) ∧

$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbuf } c = {}^{o}\textit{qbuf } c) \wedge$
$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbufsize } c = {}^{o}\textit{qbufsize } c)\}),$
$\{\textit{True}\}])$

definition *Recv-Que-Message-RGCond* :: *Core* ⇒ *Port* ⇒ (*State*) *rgformula*
where *Recv-Que-Message-RGCond k p* ≡
$RG[\{\textit{True}\},$
$\{{}^{a}\textit{cur } (\textit{c2s conf } k) = {}^{o}\textit{cur } (\textit{c2s conf } k)\},$
$(\{{}^{a}\textit{cur} = {}^{o}\textit{cur} \wedge {}^{a}\textit{partst} = {}^{o}\textit{partst} \wedge$
$({}^{o}\textit{qbufsize } (\textit{ch-srcqport conf } p) = \textit{length } ({}^{o}\textit{qbuf } (\textit{ch-srcqport conf } p))$
$\longrightarrow {}^{a}\textit{qbufsize } (\textit{ch-srcqport conf } p) = \textit{length } ({}^{a}\textit{qbuf } (\textit{ch-srcqport conf } p))) \wedge$
$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbuf } c = {}^{o}\textit{qbuf } c) \wedge$
$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbufsize } c = {}^{o}\textit{qbufsize } c)\}),$
$\{\textit{True}\}]$

definition *Core-Init-RGF* :: *Core* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-e*
where *Core-Init-RGF k* ≡ (*Core-Init k*, *Core-Init-RGCond k*)

definition *Schedule-RGF* :: *Core* ⇒ *Part* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-e*
where *Schedule-RGF k p* ≡ (*Schedule k p*, *Schedule-RGCond k p*)

definition *Send-Que-Message-RGF* :: *Core* ⇒ *Port* ⇒ *Message* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-e*
where *Send-Que-Message-RGF k p m* ≡ (*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)

definition *Recv-Que-Message-RGF* :: *Core* ⇒ *Port* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-e*
where *Recv-Que-Message-RGF k p* ≡ (*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)

definition *EvtSys1-on-Core-RGF* :: *Core* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-es*
where *EvtSys1-on-Core-RGF k* ≡
$(\textit{rgf-EvtSys } (\bigcup p.\{\textit{Schedule-RGF } k\ p\} \cup$
$(\bigcup (p, m).\ \{\textit{Send-Que-Message-RGF } k\ p\ m\}) \cup$
$(\bigcup p.\{\textit{Recv-Que-Message-RGF } k\ p\})),$
$RG[\{\textit{True}\},$
$\{{}^{a}\textit{cur } (\textit{c2s conf } k) = {}^{o}\textit{cur } (\textit{c2s conf } k) \wedge$
$(\forall p.\ \textit{p2s conf } p = \textit{c2s conf } k \longrightarrow {}^{a}\textit{partst } p = {}^{o}\textit{partst } p)\},$
$((\bigcup p.\{({}^{a}\textit{cur} = {}^{o}\textit{cur}(\textit{c2s conf } k := \textit{Some } p)$
$\wedge {}^{a}\textit{partst} = {}^{o}\textit{partst}(\textit{the } ({}^{a}\textit{cur}(\textit{c2s conf } k)) := \textit{RUN})$
$\wedge \textit{p2s conf } p = \textit{c2s conf } k$
$\vee ({}^{a}\textit{cur} = {}^{o}\textit{cur}(\textit{c2s conf } k := \textit{None})$
$\wedge {}^{a}\textit{partst} = {}^{o}\textit{partst}(\textit{the } ({}^{o}\textit{cur } (\textit{c2s conf } k)) := \textit{READY})))$
$\wedge (\forall c.\ c \neq k \longrightarrow {}^{a}\textit{cur } (\textit{c2s conf } c) = {}^{o}\textit{cur } (\textit{c2s conf } c))$
$\wedge (\forall c\ p.\ c \neq k \wedge \textit{p2s conf } p = \textit{c2s conf } c \longrightarrow {}^{a}\textit{partst } p = {}^{o}\textit{partst } p)$
$\wedge {}^{a}\textit{qbuf} = {}^{o}\textit{qbuf}$
$\wedge {}^{a}\textit{qbufsize} = {}^{o}\textit{qbufsize}\}) \cup$
$(\bigcup p.\{{}^{a}\textit{cur} = {}^{o}\textit{cur} \wedge {}^{a}\textit{partst} = {}^{o}\textit{partst} \wedge$
$({}^{o}\textit{qbufsize } (\textit{ch-srcqport conf } p) = \textit{length } ({}^{o}\textit{qbuf } (\textit{ch-srcqport conf } p))$
$\longrightarrow {}^{a}\textit{qbufsize } (\textit{ch-srcqport conf } p) = \textit{length } ({}^{a}\textit{qbuf } (\textit{ch-srcqport conf } p))) \wedge$
$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbuf } c = {}^{o}\textit{qbuf } c) \wedge$
$(\forall c.\ c \neq \textit{ch-srcqport conf } p \longrightarrow {}^{a}\textit{qbufsize } c = {}^{o}\textit{qbufsize } c)\}) \cup$
$\textit{Id}),$
$\{\textit{True}\}])$

definition *EvtSys-on-Core-RGF* :: *Core* ⇒ (*EventLabel*, *Core*, *State*) *rgformula-es*
where *EvtSys-on-Core-RGF k* ≡
$(\textit{rgf-EvtSeq } (\textit{Core-Init-RGF } k)\ (\textit{EvtSys1-on-Core-RGF } k),$
$RG[\{\forall p.\ \textit{p2s conf } p = \textit{c2s conf } k \longrightarrow \acute{}\textit{partst } p = \textit{IDLE}\},$
$\{{}^{a}\textit{cur } (\textit{c2s conf } k) = {}^{o}\textit{cur } (\textit{c2s conf } k) \wedge$

$$
\begin{array}{l}
\quad (\forall p.\ p2s\ conf\ p = c2s\ conf\ k \longrightarrow {}^{a}partst\ \ p = {}^{o}partst\ p)\}\!\!\},\\
((\bigcup p.\{\!\!\{({}^{a}cur = {}^{o}cur(c2s\ conf\ k := Some\ p)\\
\qquad \wedge\ {}^{a}partst = {}^{o}partst(the\ ({}^{a}cur(c2s\ conf\ k)) := RUN)\\
\qquad \wedge\ p2s\ conf\ p = c2s\ conf\ k\\
\qquad\ \vee\ ({}^{a}cur = {}^{o}cur(c2s\ conf\ k := None)\\
\qquad\quad \wedge\ {}^{a}partst = {}^{o}partst(the\ ({}^{o}cur\ (c2s\ conf\ k)) := READY)))\\
\quad \wedge\ (\forall c.\ c \neq k \longrightarrow {}^{a}cur\ (c2s\ conf\ c) = {}^{o}cur\ \ (c2s\ conf\ c))\\
\quad \wedge\ (\forall c\ p.\ c \neq k \wedge p2s\ conf\ p = c2s\ conf\ c \longrightarrow {}^{a}partst\ \ p = {}^{o}partst\ p\ )\\
\quad \wedge\ {}^{a}qbuf = {}^{o}qbuf\\
\quad \wedge\ {}^{a}qbufsize = {}^{o}qbufsize\}\!\!\}) \cup\\
\quad (\bigcup p.\{\!\!\{{}^{a}cur = {}^{o}cur \wedge {}^{a}partst = {}^{o}partst \wedge\\
\qquad ({}^{o}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{o}qbuf\ \ (ch\text{-}srcqport\ conf\ p))\\
\qquad\ \longrightarrow {}^{a}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{a}qbuf\ \ (ch\text{-}srcqport\ conf\ p))) \wedge\\
\qquad (\forall c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbuf\ c = {}^{o}qbuf\ c) \wedge\\
\qquad (\forall c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbufsize\ c = {}^{o}qbufsize\ \ c)\}\!\!\}) \cup\\
\quad Id \cup\\
\quad \{\!\!\{{}^{a}cur = {}^{o}cur \wedge {}^{a}qbuf = {}^{o}qbuf \wedge {}^{a}qbufsize = {}^{o}qbufsize\\
\qquad \wedge\ (\forall p.\ p2s\ conf\ p = c2s\ conf\ k \longrightarrow {}^{o}partst\ p = IDLE \wedge {}^{a}partst\ p = READY)\\
\qquad \wedge\ (\forall c\ p.\ c \neq k \wedge p2s\ conf\ p = c2s\ conf\ c \longrightarrow {}^{a}partst\ \ p = {}^{o}partst\ p)\}\!\!\}),\\
\{\!\!\{True\}\!\!\}])
\end{array}
$$

definition *ARINCXKernel-Spec* :: (*EventLabel*, *Core*, *State*) *rgformula-par*
where *ARINCXKernel-Spec* ≡ (λ*k*. *EvtSys-on-Core-RGF k*)

11.3 Functional correctness by rely guarantee proof

consts *s0*::*State*
definition *s0-witness*::*State*
where *s0-witness* ≡ *fst* (*System-Init conf*)

specification (*s0*)
s0-init: *s0* ≡ *fst* (*System-Init conf*)
by *simp*

lemma *neq-coreinit*: *k1*≠*k2* ⟹ *Core-Init k1*≠*Core-Init k2*
by (*simp add*:*Core-Init-def get-evt-label-def*)

lemma *neq-schedule*: (*k1*≠*k2* ∨ *p1*≠*p2*) ⟹ *Schedule k1 p1* ≠ *Schedule k2 p2*
by (*simp add*:*Schedule-def get-evt-label-def*)

lemma *neq-wrt-samp*: (*k1*≠*k2* ∨ *p1*≠*p2* ∨ *m1*≠*m2*)
⟹ *Send-Que-Message k1 p1 m1* ≠ *Send-Que-Message k2 p2 m2*
apply (*clarsimp*, *simp add*:*Send-Que-Message-def*)
by (*simp add*:*get-evt-label-def*)

lemma *neq-rd-samp*: (*k1*≠*k2* ∨ *p1*≠*p2*) ⟹ *Recv-Que-Message k1 p1* ≠ *Recv-Que-Message k2 p2*
apply (*clarsimp*, *simp add*:*Recv-Que-Message-def*)
by (*simp add*:*get-evt-label-def*)

lemma *neq-coreinit-sched*: *Core-Init k1* ≠ *Schedule k2 p*
by (*simp add*:*Schedule-def Core-Init-def get-evt-label-def*)

lemma *neq-coreinit-wrtsamp*: *Core-Init k1* ≠ *Send-Que-Message k2 p m*
by (*simp add*:*Send-Que-Message-def Core-Init-def get-evt-label-def*)

lemma *neq-coreinit-rdsamp*: *Core-Init k1* ≠ *Recv-Que-Message k2 p*
by (*simp add*:*Recv-Que-Message-def Core-Init-def get-evt-label-def*)

lemma *neq-sched-wrtsamp*: *Schedule k1 p1* $\neq$ *Send-Que-Message k2 p m*
by (*simp add*:*Send-Que-Message-def Schedule-def get-evt-label-def*)

lemma *neq-sched-rdsamp*: *Schedule k1 p1* $\neq$ *Recv-Que-Message k2 p*
by (*simp add*:*Recv-Que-Message-def Schedule-def get-evt-label-def*)

lemma *neq-wrtsamp-rdsamp*: *Send-Que-Message k1 p1 m* $\neq$ *Recv-Que-Message k2 p2*
by (*simp add*:*Recv-Que-Message-def Send-Que-Message-def get-evt-label-def*)

definition *evtrgfset* :: ((*EventLabel*, *Core*, *State*) *event* $\times$ (*State rgformula*)) *set*
where *evtrgfset* $\equiv$ ($\bigcup$ *k*.{(*Core-Init k*, *Core-Init-RGCond k*)})
$\cup$ ($\bigcup$ (*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)})
$\cup$ ($\bigcup$ (*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)})
$\cup$ ($\bigcup$ (*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)})

lemma *evtrgfset-eq-allevts-ARINCSpec*: *all-evts ARINCXKernel-Spec* = *evtrgfset*
proof −
have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. *all-evts-es* (*fst* (*ARINCXKernel-Spec k*)))
by (*simp add*:*all-evts-def*)
then have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. *all-evts-es* (*fst* (*EvtSys-on-Core-RGF k*)))
by (*simp add*:*ARINCXKernel-Spec-def*)
then have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. *all-evts-es* (*rgf-EvtSeq* (*Core-Init-RGF k*) (*EvtSys1-on-Core-RGF k*)))
by (*simp add*:*EvtSys-on-Core-RGF-def*)
then have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. {*Core-Init-RGF k*} $\cup$ (*all-evts-es* (*fst* (*EvtSys1-on-Core-RGF k*))))
by *simp*
then have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. {*Core-Init-RGF k*} $\cup$
($\bigcup$ *p*.{*Schedule-RGF k p*} $\cup$
($\bigcup$ (*p*, *m*). {*Send-Que-Message-RGF k p m*}) $\cup$
($\bigcup$ *p*.{*Recv-Que-Message-RGF k p*}))
)
by (*simp add*:*Core-Init-RGF-def EvtSys1-on-Core-RGF-def*)
then have *all-evts ARINCXKernel-Spec* = ($\bigcup$ *k*. {(*Core-Init k*, *Core-Init-RGCond k*)} $\cup$
($\bigcup$ *p*.{(*Schedule k p*, *Schedule-RGCond k p*)}) $\cup$
($\bigcup$ (*p*, *m*). {(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}) $\cup$
($\bigcup$ *p*.{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)})
)
unfolding *Core-Init-RGF-def Schedule-RGF-def Send-Que-Message-RGF-def Recv-Que-Message-RGF-def* **by** *auto*
moreover
have ($\bigcup$ *k*. {(*Core-Init k*, *Core-Init-RGCond k*)} $\cup$
($\bigcup$ *p*.{(*Schedule k p*, *Schedule-RGCond k p*)}) $\cup$
($\bigcup$ (*p*, *m*). {(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}) $\cup$
($\bigcup$ *p*.{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)})
) =
($\bigcup$ *k*. {(*Core-Init k*, *Core-Init-RGCond k*)}) $\cup$
($\bigcup$ *k*. ($\bigcup$ *p*.{(*Schedule k p*, *Schedule-RGCond k p*)})) $\cup$
($\bigcup$ *k*. ($\bigcup$ (*p*, *m*). {(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)})) $\cup$
($\bigcup$ *k*. ($\bigcup$ *p*.{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
by (*metis* (*no-types*) *UN-Un-distrib*)
moreover
have ($\bigcup$ *k*. ($\bigcup$ *p*.{(*Schedule k p*, *Schedule-RGCond k p*)}))
= ($\bigcup$ (*k*, *p*). {(*Schedule k p*, *Schedule-RGCond k p*)}) **by** *blast*
moreover
have ($\bigcup$ *k*. ($\bigcup$ (*p*, *m*). {(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
= ($\bigcup$ (*k*, *p*, *m*). {(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}) **by** *blast*
moreover
have ($\bigcup$ *k*. ($\bigcup$ *p*.{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
= ($\bigcup$ (*k*,*p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}) **by** *blast*

ultimately show *?thesis* **unfolding** *evtrgfset-def* **by** *simp*
qed

definition *evtrgffun* :: (*EventLabel*, *Core*, *State*) *event* ⇒ (*State rgformula*) *option*
where *evtrgffun* ≡ (λ*e*. *Some* (*SOME rg*. (*e*, *rg*)∈*evtrgfset*))

lemma *evtrgffun-exist*: $\forall e \in Domain\ evtrgfset.\ \exists ef \in evtrgfset.\ E_e\ ef = e \wedge evtrgffun\ e = Some\ (snd\ ef)$
by (*metis Domain-iff* E_e*-def evtrgffun-def fst-conv snd-conv someI-ex*)

lemma *diff-e-in-evtrgfset*: $\forall ef1\ ef2.\ ef1 \in evtrgfset \wedge ef2 \in evtrgfset \wedge ef1 \neq ef2 \longrightarrow E_e\ ef1 \neq E_e\ ef2$
apply(*rule allI*)+
apply(*case-tac ef1*∈(⋃*k*.{(*Core-Init k*, *Core-Init-RGCond k*)}))
apply(*case-tac ef2* ∈ (⋃*k*. {(*Core-Init k*, *Core-Init-RGCond k*)}))
apply(*clarify*) **using** *neq-coreinit* **apply** (*simp add*: E_e*-def*) **apply** *force*
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)}))
apply(*clarify*) **using** *neq-coreinit-sched* **apply** (*simp add*:E_e*-def*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
apply(*clarify*) **using** *neq-coreinit-wrtsamp* **apply** (*simp add*:E_e*-def*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
apply(*clarify*) **using** *neq-coreinit-rdsamp* **apply** (*simp add*:E_e*-def*)
apply (*simp add*: *evtrgfset-def*)
apply(*case-tac ef1* ∈ (⋃(*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)}))
apply(*case-tac ef2* ∈ (⋃*k*. {(*Core-Init k*, *Core-Init-RGCond k*)}))
apply(*clarify*) **using** *neq-coreinit-sched* **apply** (*metis* E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)}))
apply(*clarify*) **using** *neq-schedule* **apply** (*metis* E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
apply(*clarify*) **using** *neq-sched-wrtsamp* **apply** (*simp add*: E_e*-def*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
apply(*clarify*) **using** *neq-sched-rdsamp* **apply** (*simp add*: E_e*-def*)
apply (*simp add*: *evtrgfset-def*)
apply(*case-tac ef1* ∈ (⋃(*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
apply(*case-tac ef2* ∈ (⋃*k*. {(*Core-Init k*, *Core-Init-RGCond k*)}))
apply(*clarify*) **using** *neq-coreinit-wrtsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)}))
apply(*clarify*) **using** *neq-sched-wrtsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
apply(*clarify*) **using** *neq-wrt-samp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
apply(*clarify*) **using** *neq-wrtsamp-rdsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply (*simp add*: *evtrgfset-def*)
apply(*case-tac ef1* ∈ (⋃(*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
apply(*case-tac ef2* ∈ (⋃*k*. {(*Core-Init k*, *Core-Init-RGCond k*)}))
apply(*clarify*) **using** *neq-coreinit-rdsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Schedule k p*, *Schedule-RGCond k p*)}))
apply(*clarify*) **using** *neq-sched-rdsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*, *m*).{(*Send-Que-Message k p m*, *Send-Que-Message-RGCond k p m*)}))
apply(*clarify*) **using** *neq-wrtsamp-rdsamp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply(*case-tac ef2* ∈ (⋃(*k*, *p*).{(*Recv-Que-Message k p*, *Recv-Que-Message-RGCond k p*)}))
apply(*clarify*) **using** *neq-rd-samp* **apply** (*metis* (*no-types*, *hide-lams*) E_e*-def fst-conv*)
apply (*simp add*: *evtrgfset-def*)
using *evtrgfset-def* **by** *blast*

lemma *evtrgfset-func*: $\forall ef \in evtrgfset.\ evtrgffun\ (E_e\ ef) = Some\ (snd\ ef)$
proof −
{
fix *ef*
assume *a0*: *ef*∈*evtrgfset*

then have E_e *ef*∈*Domain evtrgfset* **by** (*metis Domain-iff* E_e*-def surjective-pairing*)
then obtain *ef1* **where** *a1*: *ef1*∈*evtrgfset* ∧ E_e *ef1* = E_e *ef* ∧ *evtrgffun* (E_e *ef*) = *Some* (*snd ef1*)
using *evtrgffun-exist*[*rule-format*,*of* E_e *ef*] **by** *auto*
have *evtrgffun* (E_e *ef*) = *Some* (*snd ef*)
proof(*cases ef1* = *ef*)
assume *ef1* = *ef*
with *a1* **show** *?thesis* **by** *simp*
next
assume *b0*: *ef1*≠*ef*
with *diff-e-in-evtrgfset a0 a1* **have** E_e *ef1* ≠ E_e *ef* **by** *blast*
with *a1* **show** *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *all-basic-evts-arinc-help*: ∀ *k*. *ef*∈*all-evts-es* (*fst* (*ARINCXKernel-Spec k*)) ⟶ *is-basicevt* (E_e *ef*)
proof −
{
fix *k*
assume *p0*: *ef*∈*all-evts-es* (*fst* (*ARINCXKernel-Spec k*))
then have *ef*∈*all-evts-es* (*fst* (*EvtSys-on-Core-RGF k*)) **by** (*simp add*:*ARINCXKernel-Spec-def*)
then have *ef*∈*insert* (*Core-Init-RGF k*) (*all-evts-es* (*fst* (*EvtSys1-on-Core-RGF k*)))
by (*simp add*:*EvtSys-on-Core-RGF-def*)
then have *ef* = (*Core-Init-RGF k*) ∨ *ef*∈*all-evts-es* (*fst* (*EvtSys1-on-Core-RGF k*)) **by** *auto*
then have *is-basicevt* (E_e *ef*)
proof
assume *a0*: *ef* = *Core-Init-RGF k*
then show *?thesis*
using *Core-Init-RGF-def Core-Init-def* **unfolding** E_e*-def* **by** *simp*
next
assume *a1*: *ef*∈*all-evts-es* (*fst* (*EvtSys1-on-Core-RGF k*))
then have *ef*∈{*ef*. ∃ *p*. *ef* = *Schedule-RGF k p*} ∪
{*ef*. ∃ *p m*. *ef* = *Send-Que-Message-RGF k p m*} ∪
{*ef*. ∃ *p*. *ef* = *Recv-Que-Message-RGF k p*}
using *all-evts-es-esys EvtSys1-on-Core-RGF-def* **by** *auto*
then have *ef*∈{*ef*. ∃ *p*. *ef* = *Schedule-RGF k p*}
∨ *ef*∈{*ef*. ∃ *p m*. *ef* = *Send-Que-Message-RGF k p m*}
∨ *ef*∈{*ef*. ∃ *p*. *ef* = *Recv-Que-Message-RGF k p*} **by** *auto*
then show *?thesis*
proof
assume *ef*∈{*ef*. ∃ *p*. *ef* = *Schedule-RGF k p*}
then show *?thesis* **unfolding** E_e*-def Schedule-RGF-def Schedule-def* **by** *auto*
next
assume *ef*∈{*ef*. ∃ *p m*. *ef* = *Send-Que-Message-RGF k p m*}
∨ *ef*∈{*ef*. ∃ *p*. *ef* = *Recv-Que-Message-RGF k p*}
then show *?thesis*
proof
assume *ef*∈{*ef*. ∃ *p m*. *ef* = *Send-Que-Message-RGF k p m*}
then have ∃ *p m*. *ef* = *Send-Que-Message-RGF k p m* **by** *auto*
then obtain *p* **and** *m* **where** *ef* = *Send-Que-Message-RGF k p m* **by** *auto*
then show *?thesis* **by** (*simp add*: E_e*-def Send-Que-Message-RGF-def Send-Que-Message-def*)
next
assume *ef*∈{*ef*. ∃ *p*. *ef* = *Recv-Que-Message-RGF k p*}
then have ∃ *p*. *ef* = *Recv-Que-Message-RGF k p* **by** *auto*
then obtain *p* **where** *ef* = *Recv-Que-Message-RGF k p* **by** *auto*
then show *?thesis* **by** (*simp add*: E_e*-def Recv-Que-Message-RGF-def Recv-Que-Message-def*)
qed

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

lemma *all-basic-evts-arinc*: $\forall$ *ef*$\in$*all-evts ARINCXKernel-Spec*. *is-basicevt* (E_e *ef*)
using *all-evts-def*[*of ARINCXKernel-Spec*] *all-basic-evts-arinc-help* **by** *auto*

lemma *bsc-evts-rgfs*: $\forall$ *erg*$\in$*all-evts* (*ARINCXKernel-Spec*). (*evtrgffun* (E_e *erg*)) = *Some* (*snd erg*)
using *evtrgfset-func evtrgfset-eq-allevts-ARINCSpec* **by** *simp*

lemma *Core-Init-SatRG*: $\forall$ *k*. *Core-Init k* $\vdash$ *Core-Init-RGCond k*
apply(*simp add*:*Evt-sat-RG-def*)
apply(*rule allI*)
apply(*simp add*:*Core-Init-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def Core-Init-RGCond-def* Pre_f*-def* $Post_f$*-def*
$Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Basic*)
unfolding *guard-def* **apply** *simp*
apply *simp*
apply *auto*
using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
apply(*simp add*:*stable-def*)+
apply(*simp add*:*Core-Init-RGCond-def* Pre_f*-def* $Post_f$*-def* $Guar_f$*-def*
$Rely_f$*-def getrgformula-def guard-def stable-def*)
apply(*simp add*:*Core-Init-RGCond-def* $Guar_f$*-def getrgformula-def stable-def*)
done

lemma *Sched-SatRG-h2*:
$\vdash$ ´*cur* := ´*cur*(*c2s conf k* $\mapsto$ *p*);;
´*partst* := ´*partst* (*p* := *RUN*)
sat_p [{|*p2s conf p* = *c2s conf k* $\wedge$ ´*cur* (*c2s conf k*) = *None*|} $\cap$ {*V*},
{(*s*, *t*). *s* = *t*}, *UNIV*,
{|(´*cur* = *cur V*(*c2s conf k* $\mapsto$ *p*) $\wedge$
´*partst* = (*partst V*)(*the* (´*cur* (*c2s conf k*)) := *RUN*) $\wedge$
p2s conf p = *c2s conf k* $\vee$
´*cur* = (*cur V*)(*c2s conf k* := *None*) $\wedge$
´*partst* = (*partst V*)(*the* (*cur V* (*c2s conf k*)) := *READY*)) $\wedge$
($\forall$ *c*. *c* $\neq$ *k* $\longrightarrow$ ´*cur* (*c2s conf c*) = *cur V* (*c2s conf c*)) $\wedge$
($\forall$ *c p*. *c* $\neq$ *k* $\wedge$ *p2s conf p* = *c2s conf c* $\longrightarrow$ ´*partst p* = *partst V p*) $\wedge$
´*qbuf* = *qbuf V* $\wedge$ ´*qbufsize* = *qbufsize V* $\vee$
´(*op* = *V*)|}]
apply(*case-tac p2s conf p* = *c2s conf k* $\wedge$ (*cur V*) (*c2s conf k*) = *None*)
apply *simp*
apply(*rule Seq*[**where** *mid*={*s*. *s* = *V* (| *cur* := (*cur V*) (*c2s conf k* := *Some p*)|)}])
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *simp*
apply(*rule disjI1*)
using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]
apply(*rule Seq*[**where** *mid*={}])
apply(*rule Basic*)

apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply(*simp add*:*stable-def*)+ **apply** *auto*[*1*]
done

lemma *Sched-SatRG-h1*:
$\vdash$ *´partst* := *´partst*(*the* (*´cur* (*c2s conf k*)) := *READY*);;
´cur := *´cur* (*c2s conf k* := *None*)
sat_p [{|*p2s conf p* = *c2s conf k* ∧ *´partst p* ≠ *IDLE* ∧ (*´cur* (*c2s conf k*) = *None*
∨ *p2s conf* (*the* (*´cur* (*c2s conf k*))) = *c2s conf k*)|} ∩
{|∃ *y*. *´cur* (*c2s conf k*) = *Some y*|} ∩ {*V*},
{(*s*, *t*). *s* = *t*}, *UNIV*,
{|(*´cur* = *cur V*(*c2s conf k* ↦ *p*) ∧
´partst = (*partst V*)(*the* (*´cur* (*c2s conf k*)) := *RUN*) ∧
p2s conf p = *c2s conf k* ∨
´cur = (*cur V*)(*c2s conf k* := *None*) ∧
´partst = (*partst V*)(*the* (*cur V* (*c2s conf k*)) := *READY*)) ∧
(∀ *c*. *c* ≠ *k* ⟶ *´cur* (*c2s conf c*) = *cur V* (*c2s conf c*)) ∧
(∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ *´partst p* = *partst V p*) ∧
´qbuf = *qbuf V* ∧ *´qbufsize* = *qbufsize V* ∨
´(*op* = *V*)|} ∩
{|*p2s conf p* = *c2s conf k* ∧ *´cur* (*c2s conf k*) = *None*|}]
apply(*case-tac p2s conf p* = *c2s conf k* ∧ *partst V p* ≠ *IDLE*
∧ ((*cur V*) (*c2s conf k*) = *None* ∨ *p2s conf* (*the* ((*cur V*) (*c2s conf k*))) = *c2s conf k*)
∧ (∃ *y*. (*cur V*) (*c2s conf k*) = *Some y*))
apply *simp*
apply(*rule Seq*[**where** *mid*={*s*. *s* = *V* (| *partst* := (*partst V*) (*the* ((*cur V*) (*c2s conf k*)) := *READY*)|)
∧ *p2s conf p* = *c2s conf k*}])
apply *simp*
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply *simp*
apply(*rule disjI1*)
apply(*rule conjI*)
using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
apply(*rule impI*)
apply(*case-tac cur V* (*c2s conf k*) = *None*)
apply *simp*
using *inj-surj-c2s injI surj-def* **apply** (*simp add*: *inj-eq*)
apply(*simp add*:*stable-def*)
apply(*simp add*:*stable-def*)
apply(*simp add*:*stable-def*) **apply** *auto*[*1*]
apply(*rule Seq*[**where** *mid*={}])
apply(*rule Basic*)
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply(*simp add*:*stable-def*)+
apply *auto*[*1*]
done

lemma *Sched-SatRG*: *Schedule k p* $\vdash$ *Schedule-RGCond k p*
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*:*Schedule-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def Schedule-RGCond-def guard-def* Pre_f*-def*
$Post_f$*-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)

apply(*rule Seq*[**where** *mid*=$\{\!|$*p2s conf p* = *c2s conf k* ∧ ´*cur*(*c2s conf k*) = *None* $|\!\}$])
  apply(*rule Cond*)
    apply(*simp add*: *stable-def*)
    apply(*rule Await*)
      apply(*simp add*: *stable-def*)+
      apply(*rule allI*) **apply**(*rule Sched-SatRG-h1*)
      apply(*simp add*: *Skip-def*)
    apply(*rule Basic*)
      apply *auto*[*1*]
      apply *auto*[*1*]
      apply(*simp add*: *stable-def*)+
  apply(*rule Await*)
    apply(*simp add*: *stable-def*)+
    apply(*rule allI*) **apply**(*rule Sched-SatRG-h2*)
      apply(*simp add*: *stable-def Schedule-RGCond-def* Pre_f*-def*
            $Post_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*simp add*: *Schedule-RGCond-def* Pre_f*-def* $Post_f$*-def* $Guar_f$*-def getrgformula-def*)
done

lemma *Send-Que-Message-SatRG-h1*:
  ⊢ ´*qbuf* := ´*qbuf*(*ch-srcqport conf p* := ´*qbuf* (*ch-srcqport conf p*) @ [*m*]);;
    ´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* :=
        Suc (´*qbufsize* (*ch-srcqport conf p*)))
  sat_p [$\{\!|$*is-src-qport conf p* ∧ (∃ *y*. ´*cur* ((*c2s conf*) *k*) = *Some y*)
        ∧ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))$|\!\}$ ∩
      $\{\!|$´*qbufsize* (*ch-srcqport conf p*) < *chmax conf* (*ch-srcqport conf p*)$|\!\}$ ∩ {*V*},
      {(*s*, *t*). *s* = *t*}, *UNIV*,
     $\{\!|$´(*Pair V*) ∈ $\{\!|$^{a}*cur* = ^{o}*cur* ∧
            ^{a}*partst* = ^{o}*partst* ∧
            (^{o}*qbufsize* (*ch-srcqport conf p*) =
             length (^{o}*qbuf* (*ch-srcqport conf p*)) ⟶
             ^{a}*qbufsize* (*ch-srcqport conf p*) =
             length (^{a}*qbuf* (*ch-srcqport conf p*))) ∧
            (∀ *c*. *c* ≠ *ch-srcqport conf p* ⟶ ^{a}*qbuf c* = ^{o}*qbuf c*) ∧
            (∀ *c*. *c* ≠ *ch-srcqport conf p* ⟶ ^{a}*qbufsize c* = ^{o}*qbufsize c*)$|\!\}|\!\}$ ∩ *UNIV*]
apply(*case-tac is-src-qport conf p* ∧ (∃ *y*. (*cur V*) ((*c2s conf*) *k*) = *Some y*)
        ∧ *port-of-part conf p* (*the* ((*cur V*) ((*c2s conf*) *k*)))
        ∧ (*qbufsize V*) (*ch-srcqport conf p*) < *chmax conf* (*ch-srcqport conf p*))
  apply *simp*
  apply(*rule Seq*[**where** *mid*={*s*. *s* = *V*(|*qbuf* := (*qbuf V*)(*ch-srcqport conf p*
                := (*qbuf V*) (*ch-srcqport conf p*) @ [*m*])|)}])
    apply(*rule Basic*)
      apply *auto*[*1*]
      apply(*simp add*: *stable-def*)+
    apply(*rule Basic*)
      apply *auto*[*1*]
      apply(*simp add*: *stable-def*)+
  apply(*rule Seq*[**where** *mid*={}])
    apply(*rule Basic*)
      apply(*simp add*:*stable-def*)+
    apply(*rule Basic*)
      apply(*simp add*:*stable-def*)+
done

lemma *Send-Que-Message-SatRG*:
  Send-Que-Message k p m ⊢ *Send-Que-Message-RGCond k p m*
  apply(*simp add*:*Evt-sat-RG-def*)

apply(*simp add*:*Send-Que-Message-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def Send-Que-Message-RGCond-def guard-def* Pre_f*-def*
$Post_f$*-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Await*)
apply(*simp add*: *stable-def*)
apply(*simp add*: *stable-def*)
apply(*rule allI*) **apply**(*rule Send-Que-Message-SatRG-h1*)
apply(*simp add*: *stable-def Send-Que-Message-RGCond-def* Pre_f*-def* $Rely_f$*-def getrgformula-def*)
apply(*simp add*: *Send-Que-Message-RGCond-def* $Guar_f$*-def getrgformula-def*)
done

lemma *Recv-Que-Message-SatRG-h1*:
$\vdash$ ´*qbuf* := ´*qbuf*(*ch-srcqport conf p* := *tl* (´*qbuf* (*ch-srcqport conf p*)));;
´*qbufsize* := ´*qbufsize* (*ch-srcqport conf p* := ´*qbufsize* (*ch-srcqport conf p*) − *Suc 0*)
sat_p [⦃*is-dest-qport conf p* ∧ (∃ *y*. ´*cur* ((*c2s conf*) *k*) = *Some y*)
∧ *port-of-part conf p* (*the* (´*cur* ((*c2s conf*) *k*)))⦄ ∩
⦃*0* < ´*qbufsize* (*ch-srcqport conf p*)⦄ ∩ {*V*},
{(*s*, *t*). *s* = *t*}, *UNIV*,
⦃´(*Pair V*) ∈ ⦃ᵃ*cur* = ᵒ*cur* ∧ ᵃ*partst* = ᵒ*partst* ∧
(ᵒ*qbufsize* (*ch-srcqport conf p*) = *length* (ᵒ*qbuf* (*ch-srcqport conf p*)) ⟶
ᵃ*qbufsize* (*ch-srcqport conf p*) = *length* (ᵃ*qbuf* (*ch-srcqport conf p*))) ∧
(∀ *c*. *c* ≠ *ch-srcqport conf p* ⟶ ᵃ*qbuf c* = ᵒ*qbuf c*) ∧
(∀ *c*. *c* ≠ *ch-srcqport conf p* ⟶ ᵃ*qbufsize c* = ᵒ*qbufsize c*)⦄⦄ ∩ *UNIV*]
apply(*case-tac is-dest-qport conf p* ∧ (∃ *y*. (*cur V*) ((*c2s conf*) *k*) = *Some y*)
∧ *port-of-part conf p* (*the* ((*cur V*) ((*c2s conf*) *k*)))
∧ *0* < (*qbufsize V*) (*ch-srcqport conf p*))
apply *simp*
apply(*rule Seq*[**where** *mid*={*s*. *s* = *V*⦇*qbuf* := (*qbuf V*)(*ch-srcqport conf p* := *tl* ((*qbuf V*) (*ch-srcqport conf p*)))⦈}])
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*: *stable-def*)+
apply(*rule Basic*)
apply *auto*[*1*]
apply(*simp add*: *stable-def*)+
apply(*rule Seq*[**where** *mid*={}])
apply(*rule Basic*)
apply(*simp add*:*stable-def*)+
apply(*rule Basic*)
apply(*simp add*:*stable-def*)+
done

lemma *Recv-Que-Message-SatRG*: *Recv-Que-Message k p* ⊢ *Recv-Que-Message-RGCond k p*
apply(*simp add*:*Evt-sat-RG-def*)
apply(*simp add*:*Recv-Que-Message-def*)
apply(*rule BasicEvt*)
apply(*simp add*:*body-def Recv-Que-Message-RGCond-def guard-def* Pre_f*-def*
$Post_f$*-def* $Rely_f$*-def* $Guar_f$*-def getrgformula-def*)
apply(*rule Await*)
apply(*simp add*: *stable-def*)
apply(*simp add*: *stable-def*)
apply(*rule allI*) **apply**(*rule Recv-Que-Message-SatRG-h1*)
apply(*simp add*: *stable-def Recv-Que-Message-RGCond-def* Pre_f*-def* $Rely_f$*-def getrgformula-def*)
apply(*simp add*: *Recv-Que-Message-RGCond-def* $Guar_f$*-def getrgformula-def*)
done

lemma *EvtSys1-on-core-SatRG*:
  $\forall k. \vdash$ *fst* (*EvtSys1-on-Core-RGF k*) sat_s
    [Pre_f (*snd* (*EvtSys1-on-Core-RGF k*)),
     $Rely_f$ (*snd* (*EvtSys1-on-Core-RGF k*)),
     $Guar_f$ (*snd* (*EvtSys1-on-Core-RGF k*)),
     $Post_f$ (*snd* (*EvtSys1-on-Core-RGF k*))]
  apply(*rule allI*)
  apply(*simp add*:*EvtSys1-on-Core-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def getrgformula-def*)
  apply(*rule EvtSys-h*)
  apply(*clarify*)
  apply(*case-tac* (a,b)$\in$ {(*Schedule-RGF k x*)})
  using *Sched-SatRG Schedule-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 singletonD*)
  apply(*case-tac* (a,b)$\in$($\bigcup$(p, m). {*Send-Que-Message-RGF k p m*}))
  apply(*clarify*)
  using *Send-Que-Message-SatRG Send-Que-Message-RGF-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 Evt-sat-RG-def*
  apply (*smt Abs-unit-cases empty-iff singletonD*)
  apply(*case-tac* (a,b)$\in$($\bigcup$$p$. {*Recv-Que-Message-RGF k p*}))
  apply(*clarify*)
  using *Recv-Que-Message-SatRG Recv-Que-Message-RGF-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 Evt-sat-RG-def*
  apply (*smt Abs-unit-cases empty-iff singletonD*)
  apply *blast*

  apply(*clarify*)
  apply(*case-tac* (a,b)$\in$ {(*Schedule-RGF k x*)})
  apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def* Pre_e*-def getrgformula-def*)
  apply(*case-tac* (a,b)$\in$($\bigcup$(p, m). {*Send-Que-Message-RGF k p m*}))
  apply *clarify*
  apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def* Pre_e*-def getrgformula-def*)
  apply(*case-tac* (a,b)$\in$($\bigcup$$p$. {*Recv-Que-Message-RGF k p*}))
  apply(*clarify*)
  apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def* Pre_e*-def getrgformula-def*)
  apply *blast*

  apply(*clarify*)
  apply(*case-tac* (a,b)$\in$ {(*Schedule-RGF k x*)})
  apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def* $Rely_e$*-def getrgformula-def*)
  apply(*case-tac* (a,b)$\in$($\bigcup$(p, m). {*Send-Que-Message-RGF k p m*}))
  apply *clarify*
  apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def* $Rely_e$*-def getrgformula-def*)
  apply(*case-tac* (a,b)$\in$($\bigcup$$p$. {*Recv-Que-Message-RGF k p*}))
  apply(*clarify*)
  apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def* $Rely_e$*-def getrgformula-def*)
  apply *blast*

  apply(*clarify*)
  apply(*case-tac* (a,b)$\in$ {(*Schedule-RGF k x*)})
  apply(*simp add*:*Schedule-RGF-def Schedule-RGCond-def getrgformula-def* $Guar_e$*-def*)
   apply *auto*[*1*]
  apply(*case-tac* (a,b)$\in$($\bigcup$(p, m). {*Send-Que-Message-RGF k p m*}))
  apply(*simp add*:*Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
   apply *auto*[*1*]
  apply(*case-tac* (a,b)$\in$($\bigcup$$p$. {*Recv-Que-Message-RGF k p*}))
  apply(*simp add*:*Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def* $Guar_e$*-def*)
   apply *auto*[*1*]

```
apply blast

apply(clarify)
apply(case-tac (a,b)∈ {(Schedule-RGF k x)})
apply(simp add:Schedule-RGF-def Schedule-RGCond-def getrgformula-def Guar_e-def)
apply(case-tac (a,b)∈(⋃(p, m). {Send-Que-Message-RGF k p m}))
apply(simp add:Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def Guar_e-def)
apply(case-tac (a,b)∈(⋃p. {Recv-Que-Message-RGF k p}))
apply(simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def Guar_e-def)
apply blast

apply(clarify)
apply(case-tac (a,b)∈ {(Schedule-RGF k xa)})
  apply(case-tac (aa,ba)∈ {(Schedule-RGF k xb)})
  apply(simp add:Schedule-RGF-def Schedule-RGCond-def getrgformula-def Pre_e-def)
  apply(case-tac (aa,ba)∈(⋃(p, m). {Send-Que-Message-RGF k p m}))
  apply(simp add:Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply(case-tac (aa,ba)∈(⋃p. {Recv-Que-Message-RGF k p}))
  apply(simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply blast

apply(case-tac (a,b)∈(⋃(p, m). {Send-Que-Message-RGF k p m}))
  apply(case-tac (aa,ba)∈ {(Schedule-RGF k xb)})
  apply(simp add:Schedule-RGF-def Schedule-RGCond-def getrgformula-def Pre_e-def)
  apply(case-tac (aa,ba)∈(⋃(p, m). {Send-Que-Message-RGF k p m}))
  apply(simp add:Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply(case-tac (aa,ba)∈(⋃p. {Recv-Que-Message-RGF k p}))
  apply(simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply blast
apply(case-tac (a,b)∈(⋃p. {Recv-Que-Message-RGF k p}))
  apply(case-tac (aa,ba)∈ {(Schedule-RGF k xb)})
  apply(simp add:Schedule-RGF-def Schedule-RGCond-def getrgformula-def Pre_e-def)
  apply(case-tac (aa,ba)∈(⋃(p, m). {Send-Que-Message-RGF k p m}))
  apply(simp add:Send-Que-Message-RGF-def Send-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply(case-tac (aa,ba)∈(⋃p. {Recv-Que-Message-RGF k p}))
  apply(simp add:Recv-Que-Message-RGF-def Recv-Que-Message-RGCond-def getrgformula-def Pre_e-def)
    apply auto[1]
  apply blast
apply blast
apply (simp add:stable-def)
by simp

lemma EvtSys-on-core-SatRG:
 ∀k. ⊢ fst (EvtSys-on-Core-RGF k) sat_s
          [Pre_f (snd (EvtSys-on-Core-RGF k)),
           Rely_f (snd (EvtSys-on-Core-RGF k)),
           Guar_f (snd (EvtSys-on-Core-RGF k)),
           Post_f (snd (EvtSys-on-Core-RGF k))]
 apply(rule allI)
 apply(simp add:EvtSys-on-Core-RGF-def Pre_f-def Rely_f-def
          Guar_f-def Post_f-def getrgformula-def)
 apply(rule EvtSeq-h)
 apply(simp add:E_e-def Core-Init-RGF-def Pre_e-def Rely_e-def Guar_e-def Post_e-def)
```

using *Core-Init-SatRG getrgformula-def*
apply (*simp add*: *Evt-sat-RG-def* $Guar_f$*-def* $Post_f$*-def* Pre_f*-def* $Rely_f$*-def*)
using *EvtSys1-on-core-SatRG* **apply** *simp*
apply(*simp add*:*Core-Init-RGF-def Core-Init-RGCond-def* Pre_e*-def getrgformula-def*)
apply(*simp add*:*EvtSys1-on-Core-RGF-def* $Post_f$*-def getrgformula-def*)
apply(*simp add*:*Core-Init-RGF-def Core-Init-RGCond-def* $Rely_e$*-def getrgformula-def*)
apply *auto*[*1*]
apply(*simp add*:*EvtSys1-on-Core-RGF-def* $Rely_f$*-def getrgformula-def*)

apply(*simp add*:*Core-Init-RGF-def Core-Init-RGCond-def* $Guar_e$*-def* $Guar_f$*-def getrgformula-def EvtSys1-on-Core-RGF-def*)
apply *auto*[*1*]
apply(*simp add*:*EvtSys1-on-Core-RGF-def Core-Init-RGCond-def* $Guar_f$*-def getrgformula-def*)
by (*simp add*:*EvtSys1-on-Core-RGF-def Core-Init-RGF-def Core-Init-RGCond-def* $Post_e$*-def* Pre_f*-def getrgformula-def*)

lemma *spec-sat-rg*: $\vdash$ *ARINCXKernel-Spec SAT* [{*s0*}, {}, *UNIV*, *UNIV*]
apply (*rule ParallelESys*)
apply(*simp add*:*ARINCXKernel-Spec-def*) **using** *EvtSys-on-core-SatRG*
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(*simp add*:*ARINCXKernel-Spec-def EvtSys-on-Core-RGF-def* Pre_{es}*-def getrgformula-def*)
apply(*simp add*:*s0-def System-Init-def*)
apply *simp*
apply(*rule allI*)+
apply(*simp add*:*ARINCXKernel-Spec-def EvtSys-on-Core-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply(*rule impI*)
apply(*rule conjI*)
apply *auto*[*1*]
apply *metis*
apply *metis*
apply(*rule conjI*)
apply *auto*[*1*]
apply(*rule conjI*)
apply *auto*[*1*]
apply *auto*[*1*] **apply** *force*

apply (*simp add*: *Collect-mono Id-fstsnd-eq*)
apply *simp*+
done

11.4 Invariant proof

definition *cur-part-cond* :: *State* $\Rightarrow$ *bool*
where *cur-part-cond s* $\equiv \forall$ *sched p*. (*cur s*) *sched* = *Some p* $\longrightarrow$ *sched* = (*p2s conf*) *p*

definition *cur-part-inv* :: (*State*) *invariant*
where *cur-part-inv* $\equiv$ {*s*. *cur-part-cond s*}

definition *cur-part-mode-cond* :: *State* $\Rightarrow$ *bool*
where *cur-part-mode-cond s* $\equiv$
$\forall$ *sched p*. *p2s conf p* = *sched* $\wedge$ (*cur s*) *sched* = *Some p* $\longrightarrow$ (*partst s*) *p* = *RUN*

definition *cur-part-mode-inv* :: (*State*) *invariant*
where *cur-part-mode-inv* $\equiv$ {*s*. *cur-part-mode-cond s*}

definition *qbuf-size-cond* :: *State* $\Rightarrow$ *bool*

where *qbuf-size-cond s* ≡ ∀ *c*. (*qbufsize s*) *c* = *length* ((*qbuf s*) *c*)

definition *qbuf-size-inv* :: (*State*) *invariant*
where *qbuf-size-inv* ≡ {*s*. *qbuf-size-cond s*}

definition *invariant* ≡ *cur-part-inv* ∩ *cur-part-mode-inv* ∩ *qbuf-size-inv*

lemma *init-sat-inv*: {*s0*}⊆ *invariant*
by(*simp add*:*s0-init System-Init-def invariant-def cur-part-inv-def cur-part-cond-def cur-part-mode-inv-def cur-part-mode-cond-def qbuf-size-inv-def qbuf-size-cond-def*)

lemma *stb-guar-coreinit*: *stable invariant* ({|ª*cur*= º*cur* ∧ ª*qbuf*= º*qbuf* ∧ ª*qbufsize*= º*qbufsize*
∧ (∀ *p*. *p2s conf p* = *c2s conf k* ⟶ º*partst p* = *IDLE* ∧ ª*partst p* = *READY*)
∧ (∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ ª*partst p* = º*partst p*)|} ∪ *Id*)
unfolding *stable-def invariant-def cur-part-inv-def cur-part-cond-def cur-part-mode-inv-def cur-part-mode-cond-def qbuf-size-inv-def qbuf-size-cond-def*
apply *clarify*
apply *simp*
apply (*rule conjI*)
apply(*rule allI*)+ **apply**(*rule impI*) **apply** *presburger*
apply(*rule conjI*)
apply(*rule allI*)+ **apply**(*rule impI*)
apply(*case-tac x* = *y*)
apply *blast*
apply(*case-tac p2s conf p* = *c2s conf k*)
apply (*metis PartMode.distinct*(*3*))
apply (*metis* (*no-types*, *lifting*) *inj-surj-c2s surj-def*)
apply(*rule allI*) **by** *metis*

lemma *stb-guar-sched*: *stable invariant*
({|(ª*cur* = º*cur*(*c2s conf k* := *Some p*) ∧ ª*partst* = º*partst*(*the* (ª*cur*(*c2s conf k*)) := *RUN*)
∧ *p2s conf p* = *c2s conf k*
∨ (ª*cur* = º*cur*(*c2s conf k* := *None*) ∧ ª*partst* = º*partst*(*the* (º*cur* (*c2s conf k*)) := *READY*)))
∧ (∀ *c*. *c* ≠ *k* ⟶ ª*cur* (*c2s conf c*) = º*cur* (*c2s conf c*))
∧ (∀ *c p*. *c* ≠ *k* ∧ *p2s conf p* = *c2s conf c* ⟶ ª*partst p* = º*partst p*)
∧ ª*qbuf*= º*qbuf*
∧ ª*qbufsize*= º*qbufsize*|} ∪ *Id*)
apply(*simp add*:*stable-def invariant-def cur-part-inv-def cur-part-cond-def cur-part-mode-inv-def cur-part-mode-cond-def qbuf-size-inv-def qbuf-size-cond-def*)
apply(*rule allI*)
apply(*rule impI*)
apply(*rule allI*)
apply(*rule conjI*)
apply(*rule impI*)
apply(*rule conjI*)
apply(*rule allI*)+
apply(*rule impI*)
apply *auto*[*1*]
apply (*metis option.sel*)
apply (*metis option.discI*)
apply(*rule allI*)+
apply(*rule impI*)
apply(*case-tac p2s conf pa* = *c2s conf k*)
apply *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *inj-surj-c2s surj-def*)
apply(*rule impI*)
apply(*rule conjI*)

apply *blast*
apply(*rule allI*)
by *simp*

lemma *stb-guar-sndmsg*:
stable invariant
($\{\!\!\{^{a}cur = {}^{o}cur \wedge {}^{a}partst = {}^{o}partst \wedge$
$({}^{o}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{o}qbuf\ (ch\text{-}srcqport\ conf\ p))$
$\longrightarrow {}^{a}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{a}qbuf\ (ch\text{-}srcqport\ conf\ p))) \wedge$
$(\forall\ c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbuf\ c = {}^{o}qbuf\ c) \wedge$
$(\forall\ c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbufsize\ c = {}^{o}qbufsize\ c)\}\!\!\}$)
apply(*simp add*:*stable-def invariant-def cur-part-inv-def cur-part-cond-def cur-part-mode-inv-def qbuf-size-inv-def qbuf-size-cond-def*)
apply(*simp add*:*cur-part-mode-cond-def*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **by** *metis*

lemma *stb-guar-recvmsg*:
stable invariant
($\{\!\!\{^{a}cur = {}^{o}cur \wedge {}^{a}partst = {}^{o}partst \wedge$
$({}^{o}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{o}qbuf\ (ch\text{-}srcqport\ conf\ p))$
$\longrightarrow {}^{a}qbufsize\ (ch\text{-}srcqport\ conf\ p) = length\ ({}^{a}qbuf\ (ch\text{-}srcqport\ conf\ p))) \wedge$
$(\forall\ c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbuf\ c = {}^{o}qbuf\ c) \wedge$
$(\forall\ c.\ c \neq ch\text{-}srcqport\ conf\ p \longrightarrow {}^{a}qbufsize\ c = {}^{o}qbufsize\ c)\}\!\!\}$)
apply(*simp add*:*stable-def invariant-def cur-part-inv-def cur-part-cond-def cur-part-mode-inv-def qbuf-size-inv-def qbuf-size-cond-def*)
apply(*simp add*:*cur-part-mode-cond-def*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **apply**(*rule impI*)
apply(*rule allI*) **by** *metis*

lemma *evts-stb-invar*: $\forall\ ef \in evtrgfset.$ *stable invariant* ($Guar_e\ ef$)
unfolding *evtrgfset-def*
apply(*clarify*)
apply(*case-tac* $(a,\ b) \in (\bigcup k.\ \{(Core\text{-}Init\ k,\ Core\text{-}Init\text{-}RGCond\ k)\})$)
apply(*simp add*:*Core-Init-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-coreinit rgformula.select-convs*(*3*) **apply** *auto*[*1*]
apply(*case-tac* $(a,\ b) \in (\bigcup (k,\ p).\ \{(Schedule\ k\ p,\ Schedule\text{-}RGCond\ k\ p)\})$)
apply(*simp add*:*Schedule-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-sched rgformula.select-convs*(*3*) **apply** *auto*[*1*]
apply(*case-tac* $(a,\ b) \in (\bigcup (k,\ p,\ m).\ \{(Send\text{-}Que\text{-}Message\ k\ p\ m,\ Send\text{-}Que\text{-}Message\text{-}RGCond\ k\ p\ m)\})$)
apply(*simp add*:*Send-Que-Message-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-sndmsg rgformula.select-convs*(*3*) **apply** *auto*[*1*]
apply(*case-tac* $(a,\ b) \in (\bigcup (k,\ p).\ \{(Recv\text{-}Que\text{-}Message\ k\ p,\ Recv\text{-}Que\text{-}Message\text{-}RGCond\ k\ p)\})$)
apply(*simp add*:*Recv-Que-Message-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-recvmsg rgformula.select-convs*(*3*) **apply** *auto*[*1*]
by *blast*

theorem *ARINC-invariant-theorem*:
invariant-of-pares (*paresys-spec ARINCXKernel-Spec*) {*s0*} *invariant*
using *invariant-theorem*[*of ARINCXKernel-Spec* {*s0*} *evtrgffun invariant*]
spec-sat-rg evts-stb-invar evtrgfset-eq-allevts-ARINCSpec
all-basic-evts-arinc evts-stb-invar init-sat-inv bsc-evts-rgfs **by** *auto*

end

12 Formal Specification and Reasoning of an Interruptable Controller for Stepper Motor

theory *IRQStepperMotor*
  imports *PiCore-Syntax PiCore-RG-Invariant*
begin

12.1 functional specification

datatype *Device* = *Ctrl* | *Radar* | *PIC*

datatype *Irq* = *C* | *R*

record *State* = *stack* :: *Irq list*
iflag :: *bool*
car-pos :: *int*
obstacle-pos :: *int list*
i :: *int*
pos-aux :: *int*
obst-pos-aux :: *int list*

datatype *EL* = *ForwardH* | *BackwardH* | *ObstacleH* | *IRQsE*

datatype *Parameter* = *Irq Irq* | *Integer int* | *Str string* | *Natural nat*

type-synonym *EventLabel* = *EL* × (*Parameter list* × *Device*)

definition *get-evt-label* :: *EL* ⇒ *Parameter list* ⇒ *Device* ⇒ *EventLabel* (- - @ - [*0*,*0*,*0*] *20*)
  where *get-evt-label el ps k* ≡ (*el*,(*ps*,*k*))

definition *iret* :: *State prog*
  where *iret* ≡ ´*stack* := *tl* ´*stack*

definition *push* :: *Irq* ⇒ *State prog*
  where *push d* ≡ ´*stack* := *d* # (´*stack*)

definition *cli* :: *State prog*
  where *cli* ≡ ´*iflag* := *False*

definition *sti* :: *State prog*
  where *sti* ≡ ´*iflag* := *True*

definition *stm* :: *Irq* ⇒ *State prog* ⇒ *State prog* (- ▶ -)
  where *stm d p* ≡ *AWAIT hd* ´*stack* = *d THEN p END*

definition *willcollide* :: *int* ⇒ *int* ⇒ *int list* ⇒ *bool*
  where *willcollide s t l* ≡ *find* ($\lambda x.\ s \leq x \wedge x \leq t$) *l* = *None*

definition *collide* :: *'a* ⇒ *'a list* ⇒ *bool*
  where *collide pos l* ≡ *find* ($\lambda x.\ x = pos$) *l* ≠ *None*

definition *IRQs* :: *Irq* ⇒ (*EventLabel*, *Device*, *State*) *event*
  where *IRQs d* ≡
    EVENT IRQsE [*Irq d*] @ *PIC*
    THEN
      ATOMIC

(*the interrupt is the one being handled have to be delayed(skipped)
the interrupt should not be the PIC *)
IF hd ´stack ≠ d THEN push d FI
END
END

definition *forward* :: *nat* ⇒ (*EventLabel*, *Device*, *State*) *event*
where *forward v* ≡
EVENT ForwardH [*Natural v*] @ *Ctrl*
THEN
(*C* ▶ ´*i* := *0*);;
(*C* ▶ ´*pos-aux* := ´*car-pos*);;
WHILE ´*i* ≠ *int v* ∧ ¬*collide* (´*car-pos* + *1*) ´*obstacle-pos DO*
(*C* ▶ *ATOMIC*
IF ¬*collide* (´*car-pos* + *1*) ´*obstacle-pos THEN*
´*car-pos* := ´*car-pos* + *1*
FI
END);;
(*C* ▶ ´*i* := ´*i* + *1*)
OD;;
(*C* ▶ *iret*)
END

definition *backward* :: *nat* ⇒ (*EventLabel*, *Device*, *State*) *event*
where *backward v* ≡
EVENT BackwardH [*Natural v*] @ *Ctrl*
THEN
(*C* ▶ ´*i* := *0*);;
(*C* ▶ ´*pos-aux* := ´*car-pos*);;
WHILE ´*i* ≠ *int v* ∧ ¬*collide* (´*car-pos* − *1*) ´*obstacle-pos DO*
(*C* ▶ *ATOMIC*
IF ¬*collide* (´*car-pos* − *1*) ´*obstacle-pos THEN*
´*car-pos* := ´*car-pos* − *1*
FI
END);;
(*C* ▶ ´*i* := ´*i* + *1*)
OD;;
(*C* ▶ *iret*)
END

definition *obstacle* :: *int* ⇒ (*EventLabel*, *Device*, *State*) *event*
where *obstacle v* ≡
EVENT ObstacleH [*Integer v*] @ *Radar*
THEN
(*R* ▶ ´*obst-pos-aux* := ´*obstacle-pos*);;
(*R* ▶ *IF v* ≠ ´*car-pos* ∧ *v* ≠ ´*car-pos* + *1* ∧ *v* ≠ ´*car-pos* − *1 THEN*
´*obstacle-pos* := *v* # ´*obstacle-pos*
FI);;
(*R* ▶ *iret*)
END

12.2 Rely-guarantee condition of events

definition *forward-RGCond* :: *nat* ⇒ (*State*) *rgformula*
where *forward-RGCond v* ≡

$RG[\{\!|True|\!\},$

$(\{\!|{}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux$
$\wedge (hd\ {}^{o}stack \neq C \longrightarrow (({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos$
$\vee {}^{a}stack = C \# {}^{o}stack) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge (hd\ {}^{o}stack = C \longrightarrow {}^{o}obstacle\text{-}pos = {}^{a}obstacle\text{-}pos \wedge {}^{a}stack = R \# {}^{o}stack$
$\wedge {}^{o}obst\text{-}pos\text{-}aux = {}^{a}obst\text{-}pos\text{-}aux)|\!\} \cup Id),$

$(\{\!|hd\ {}^{o}stack = C \wedge ((({}^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack) \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos) \vee$
$(\neg collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos + 1))$
$\wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux|\!\} \cup Id),$

$\{\!|\acute{}car\text{-}pos = \acute{}pos\text{-}aux + \acute{}i \wedge$
$(\acute{}i = int\ v \vee collide\ (\acute{}pos\text{-}aux + \acute{}i + 1)\ \acute{}obstacle\text{-}pos)\ |\!\}]$

definition $backward\text{-}RGCond :: nat \Rightarrow (State)\ rgformula$
where $backward\text{-}RGCond\ v \equiv$
$RG[\{\!|True|\!\},$

$(\{\!|{}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux$
$\wedge (hd\ {}^{o}stack \neq C \longrightarrow (({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos$
$\vee {}^{a}stack = C \# {}^{o}stack) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge (hd\ {}^{o}stack = C \longrightarrow {}^{o}obstacle\text{-}pos = {}^{a}obstacle\text{-}pos \wedge {}^{a}stack = R \# {}^{o}stack$
$\wedge {}^{o}obst\text{-}pos\text{-}aux = {}^{a}obst\text{-}pos\text{-}aux)|\!\} \cup Id),$

$(\{\!|hd\ {}^{o}stack = C \wedge ((({}^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack) \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos) \vee$
$(\neg collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos - 1))$
$\wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux|\!\} \cup Id),$

$\{\!|\acute{}car\text{-}pos = \acute{}pos\text{-}aux - \acute{}i \wedge$
$(\acute{}i = int\ v \vee collide\ (\acute{}pos\text{-}aux - \acute{}i - 1)\ \acute{}obstacle\text{-}pos)\ |\!\}]$

definition $obstacle\text{-}RGCond :: int \Rightarrow (State)\ rgformula$
where $obstacle\text{-}RGCond\ v \equiv$
$RG[\{\!|True|\!\},$

$(\{\!|{}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux \wedge$
$(hd\ {}^{o}stack \neq R \longrightarrow {}^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack$
$\vee (\neg collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos + 1)$
$\vee (\neg collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos - 1)$
$\vee {}^{a}stack = R \# {}^{o}stack) \wedge$
$(hd\ {}^{o}stack = R \longrightarrow {}^{o}car\text{-}pos = {}^{a}car\text{-}pos \wedge {}^{o}i = {}^{a}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux$
$\wedge {}^{a}stack = C \# {}^{o}stack)\ |\!\} \cup Id),$

$(\{\!|hd\ {}^{o}stack = R \wedge ((({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ {}^{o}car\text{-}pos\ {}^{o}obstacle\text{-}pos = collide\ {}^{a}car\text{-}pos\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux\ |\!\} \cup Id),$
$\{\!|\acute{}obstacle\text{-}pos = v \# \acute{}obst\text{-}pos\text{-}aux \vee \acute{}obstacle\text{-}pos = \acute{}obst\text{-}pos\text{-}aux|\!\}]$

definition $IRQs\text{-}RGCond :: Irq \Rightarrow (State)\ rgformula$
where $IRQs\text{-}RGCond\ d \equiv$

$RG[\{\!|True|\!\}$,
$\{\!|True|\!\}$,
$(\{\!|hd\ {}^{o}stack \neq d \wedge {}^{a}stack = d \# {}^{o}stack \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos$
$\wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos$
$\wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux|\!\} \cup Id)$,
$\{\!|True|\!\}]$

definition *forward-RGF* :: *nat* ⇒ (*EventLabel*, *Device*, *State*) *rgformula-e*
where *forward-RGF v* ≡ (*forward v*, *forward-RGCond v*)

definition *backward-RGF* :: *nat* ⇒ (*EventLabel*, *Device*, *State*) *rgformula-e*
where *backward-RGF v* ≡ (*backward v*, *backward-RGCond v*)

definition *obstacle-RGF* :: *int* ⇒ (*EventLabel*, *Device*, *State*) *rgformula-e*
where *obstacle-RGF v* ≡ (*obstacle v*, *obstacle-RGCond v*)

definition *IRQs-RGF* :: *Irq* ⇒ (*EventLabel*, *Device*, *State*) *rgformula-e*
where *IRQs-RGF r* ≡ (*IRQs r*, *IRQs-RGCond r*)

definition *EvtSys-on-Motor-RGF* :: (*EventLabel*, *Device*, *State*) *rgformula-es*
where *EvtSys-on-Motor-RGF* ≡
$(rgf\text{-}EvtSys\ ((\bigcup v.\{forward\text{-}RGF\ v\}) \cup$
$(\bigcup v.\ \{backward\text{-}RGF\ v\}))$,
$RG[\{\!|True|\!\}$,
$(\{\!|{}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux$
$\wedge (hd\ {}^{o}stack \neq C \longrightarrow (({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos$
$\vee {}^{a}stack = C \# {}^{o}stack) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos$
$\wedge collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge (hd\ {}^{o}stack = C \longrightarrow {}^{o}obstacle\text{-}pos = {}^{a}obstacle\text{-}pos \wedge {}^{a}stack = R \# {}^{o}stack$
$\wedge {}^{o}obst\text{-}pos\text{-}aux = {}^{a}obst\text{-}pos\text{-}aux)|\!\} \cup Id)$,

$(\{\!|hd\ {}^{o}stack = C \wedge ({}^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack \vee$
$(\neg collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos + 1) \vee$
$(\neg collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos - 1))$
$\wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux|\!\} \cup Id)$,

$(\bigcup v.\ \{\!|\acute{}car\text{-}pos = \acute{}pos\text{-}aux + \acute{}i \wedge (\acute{}i = int\ v \vee collide\ (\acute{}car\text{-}pos + 1)\ \acute{}obstacle\text{-}pos) \vee$
$\acute{}car\text{-}pos = \acute{}pos\text{-}aux - \acute{}i \wedge (\acute{}i = int\ v \vee collide\ (\acute{}car\text{-}pos - 1)\ \acute{}obstacle\text{-}pos)\ |\!\})])$

definition *EvtSys-on-Radar-RGF* :: (*EventLabel*, *Device*, *State*) *rgformula-es*
where *EvtSys-on-Radar-RGF* ≡
$(rgf\text{-}EvtSys\ (\bigcup v.\{obstacle\text{-}RGF\ v\})$,
$RG[\{\!|True|\!\}$,

$(\{\!|{}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux \wedge$
$(hd\ {}^{o}stack \neq R \longrightarrow {}^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack$
$\vee (\neg collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos + 1)$
$\vee (\neg collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos - 1)$
$\vee {}^{a}stack = R \# {}^{o}stack) \wedge$
$(hd\ {}^{o}stack = R \longrightarrow {}^{o}car\text{-}pos = {}^{a}car\text{-}pos \wedge {}^{o}i = {}^{a}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux$
$\wedge {}^{a}stack = C \# {}^{o}stack)\ |\!\} \cup Id)$,

$(\{\!|hd\ {}^{o}stack = R \wedge ((({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$

$\vee$ (*set* ${}^{\mathrm{o}}$*obstacle-pos* $\subseteq$ *set* ${}^{\mathrm{a}}$*obstacle-pos*
$\wedge$ *collide* (${}^{\mathrm{o}}$*car-pos* $-$ *1*) ${}^{\mathrm{o}}$*obstacle-pos* $=$ *collide* (${}^{\mathrm{a}}$*car-pos* $-$ *1*) ${}^{\mathrm{a}}$*obstacle-pos*
$\wedge$ *collide* ${}^{\mathrm{o}}$*car-pos* ${}^{\mathrm{o}}$*obstacle-pos* $=$ *collide* ${}^{\mathrm{a}}$*car-pos* ${}^{\mathrm{a}}$*obstacle-pos*
$\wedge$ *collide* (${}^{\mathrm{o}}$*car-pos* $+$ *1*) ${}^{\mathrm{o}}$*obstacle-pos* $=$ *collide* (${}^{\mathrm{a}}$*car-pos* $+$ *1*) ${}^{\mathrm{a}}$*obstacle-pos*))
$\wedge$ ${}^{\mathrm{a}}$*car-pos* $=$ ${}^{\mathrm{o}}$*car-pos* $\wedge$ ${}^{\mathrm{a}}$*i* $=$ ${}^{\mathrm{o}}$*i* $\wedge$ ${}^{\mathrm{a}}$*pos-aux* $=$ ${}^{\mathrm{o}}$*pos-aux* $\}\!\!\} \cup$ *Id*),

($\bigcup$ *v*. $\{\!\!\{$ ´*obstacle-pos* $=$ *v* # ´*obst-pos-aux* $\vee$ ´*obstacle-pos* $=$ ´*obst-pos-aux*$\}\!\!\}$)])

definition *EvtSys-on-PIC-RGF* :: (*EventLabel*, *Device*, *State*) *rgformula-es*
where *EvtSys-on-PIC-RGF* $\equiv$
(*rgf-EvtSys* ($\bigcup$ *d*. {*IRQs-RGF d*}),
RG[$\{\!\!\{$*True*$\}\!\!\}$,
$\{\!\!\{$*True*$\}\!\!\}$,
($\bigcup$ *d*. ($\{\!\!\{$*hd* ${}^{\mathrm{o}}$*stack* $\neq$ *d* $\wedge$ ${}^{\mathrm{a}}$*stack* $=$ *d* # ${}^{\mathrm{o}}$*stack* $\wedge$ ${}^{\mathrm{a}}$*car-pos* $=$ ${}^{\mathrm{o}}$*car-pos*
$\wedge$ ${}^{\mathrm{a}}$*i* $=$ ${}^{\mathrm{o}}$*i* $\wedge$ ${}^{\mathrm{a}}$*pos-aux* $=$ ${}^{\mathrm{o}}$*pos-aux* $\wedge$ ${}^{\mathrm{a}}$*obstacle-pos* $=$ ${}^{\mathrm{o}}$*obstacle-pos*
$\wedge$ ${}^{\mathrm{a}}$*obst-pos-aux* $=$ ${}^{\mathrm{o}}$*obst-pos-aux*$\}\!\!\}$) $\cup$ *Id*),
$\{\!\!\{$*True*$\}\!\!\}$])

definition *Carsystem-Spec* :: (*EventLabel*, *Device*, *State*) *rgformula-par*
where *Carsystem-Spec k* $\equiv$ *case k of Ctrl* $\Rightarrow$ *EvtSys-on-Motor-RGF*
| *Radar* $\Rightarrow$ *EvtSys-on-Radar-RGF*
| *PIC* $\Rightarrow$ *EvtSys-on-PIC-RGF*

12.3 Functional correctness by rely guarantee proof

definition *init* :: *State*
where *init* $\equiv$ (|*stack* $=$ [], *iflag* $=$ *True*, *car-pos* $=$ *0*,
obstacle-pos $=$ [], *i* $=$ *0*,
pos-aux $=$ *0*, *obst-pos-aux* $=$ []|)

consts *s0*::*State*
definition *s0-witness*::*State*
where *s0-witness* $\equiv$ *init*

specification (*s0*)
s0-init: *s0* $\equiv$ *init*
by *simp*

lemma *all-basic-evts-arinc-help*: $\forall$ *k*. *ef*$\in$*all-evts-es* (*fst* (*Carsystem-Spec k*)) $\longrightarrow$ *is-basicevt* (E_e *ef*)
apply(*rule allI*) **apply**(*rule impI*)
unfolding *Carsystem-Spec-def*
apply(*case-tac k* $=$ *Ctrl*)
apply *auto*[*1*]
apply(*simp add*: *EvtSys-on-Motor-RGF-def forward-RGF-def backward-RGF-def* E_e*-def*
forward-def backward-def)
using *is-basicevt.simps* **apply** *auto*[*1*]
apply(*case-tac k* $=$ *Radar*)
apply *auto*[*1*]
apply(*simp add*: *EvtSys-on-Radar-RGF-def obstacle-RGCond-def obstacle-RGF-def* E_e*-def*
obstacle-def)
using *is-basicevt.simps* **apply** *auto*[*1*]
apply(*case-tac k* $=$ *PIC*)
apply *auto*[*1*]
apply(*simp add*: *EvtSys-on-PIC-RGF-def IRQs-RGCond-def IRQs-RGF-def* E_e*-def*
IRQs-def)
using *is-basicevt.simps* **apply** *auto*[*1*]

using *Device.exhaust* **by** *blast*

lemma *all-basic-evts-arinc*: $\forall$ *ef*$\in$*all-evts Carsystem-Spec. is-basicevt* (E_e *ef*)
using *all-evts-def* [*of Carsystem-Spec*] *all-basic-evts-arinc-help* **by** *auto*

definition *evtrgfset* :: ((*EventLabel*, *Device*, *State*) *event* $\times$ (*State rgformula*)) *set*
where *evtrgfset* $\equiv$ ($\bigcup$ *v*.{(*forward v*, *forward-RGCond v*)})
$\cup$ ($\bigcup$ *v*.{(*backward v*, *backward-RGCond v*)})
$\cup$ ($\bigcup$ *v*.{(*obstacle v*, *obstacle-RGCond v*)})
$\cup$ ($\bigcup$ *d*.{(*IRQs d*, *IRQs-RGCond d*)})

definition *evtrgffun* :: (*EventLabel*, *Device*, *State*) *event* $\Rightarrow$ (*State rgformula*) *option*
where *evtrgffun* $\equiv$ (λ*e*. *Some* (*SOME rg*. (*e*, *rg*)$\in$*evtrgfset*))

lemma *evtrgffun-exist*: $\forall$ *e*$\in$*Domain evtrgfset*. $\exists$ *ef*$\in$*evtrgfset*. E_e *ef* = *e* $\wedge$ *evtrgffun e* = *Some* (*snd ef*)
by (*metis Domain-iff* E_e*-def evtrgffun-def fst-conv snd-conv someI-ex*)

lemma *evtrgfset-eq-allevts-Spec*: *all-evts Carsystem-Spec* = *evtrgfset*
proof −
have *all-evts Carsystem-Spec* = ($\bigcup$ *k*. *all-evts-es* (*fst* (*Carsystem-Spec k*)))
by (*simp add*:*all-evts-def*)
then have *all-evts Carsystem-Spec* = *all-evts-es* (*fst EvtSys-on-Motor-RGF*) $\cup$
all-evts-es (*fst EvtSys-on-Radar-RGF*) $\cup$
all-evts-es (*fst EvtSys-on-PIC-RGF*)
apply(*simp add*:*Carsystem-Spec-def*)
apply *auto*
apply (*metis* (*no-types*, *lifting*) *Device.case*(*1*) *Device.case*(*2*) *Device.exhaust Device.simps*(*9*))
apply (*metis Device.case*(*1*))
apply (*metis Device.case*(*2*))
by (*metis Device.simps*(*9*))
then show *?thesis*
unfolding *evtrgfset-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def IRQs-RGF-def forward-RGF-def backward-RGF-def obstacle-RGF-def*
by *simp*
qed

lemma *diff-e-in-evtrgfset*: $\forall$ *ef1 ef2*. *ef1*$\in$*evtrgfset* $\wedge$ *ef2*$\in$*evtrgfset* $\wedge$ *ef1*$\neq$*ef2* $\longrightarrow$ E_e *ef1* $\neq$ E_e *ef2*
apply(*rule allI*)+
apply(*rule impI*)
apply(*case-tac ef1*$\in$($\bigcup$ *v*.{(*forward v*, *forward-RGCond v*)}))
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*backward v*, *backward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def backward-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*obstacle v*, *obstacle-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def obstacle-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *d*.{(*IRQs d*, *IRQs-RGCond d*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def IRQs-def get-evt-label-def*)
apply (*simp add*: E_e*-def forward-def backward-def obstacle-def IRQs-def get-evt-label-def evtrgfset-def*)
apply *force*
apply(*case-tac ef1*$\in$($\bigcup$ *v*.{(*backward v*, *backward-RGCond v*)}))
apply(*case-tac ef2*$\in$($\bigcup$ *v*.{(*forward v*, *forward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def backward-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*obstacle v*, *obstacle-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def backward-def obstacle-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *d*.{(*IRQs d*, *IRQs-RGCond d*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def backward-def IRQs-def get-evt-label-def*)
apply (*simp add*: E_e*-def forward-def backward-def obstacle-def*

IRQs-def get-evt-label-def evtrgfset-def)
apply *force*
apply(*case-tac ef1* $\in$ ($\bigcup$ *v*.{(*obstacle v*, *obstacle-RGCond v*)}))
apply(*case-tac ef2*$\in$($\bigcup$ *v*.{(*forward v*, *forward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def obstacle-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*backward v*, *backward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def obstacle-def backward-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *d*.{(*IRQs d*, *IRQs-RGCond d*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def obstacle-def IRQs-def get-evt-label-def*)
apply (*simp add*: E_e*-def forward-def backward-def obstacle-def*
IRQs-def get-evt-label-def evtrgfset-def)
apply *force*
apply(*case-tac ef1* $\in$ ($\bigcup$ *d*.{(*IRQs d*, *IRQs-RGCond d*)}))
apply(*case-tac ef2*$\in$($\bigcup$ *v*.{(*forward v*, *forward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def forward-def IRQs-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*backward v*, *backward-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def IRQs-def backward-def get-evt-label-def*)
apply(*case-tac ef2* $\in$ ($\bigcup$ *v*.{(*obstacle v*, *obstacle-RGCond v*)}))
apply(*clarify*) **apply** (*simp add*: E_e*-def IRQs-def obstacle-def get-evt-label-def*)
apply (*simp add*: E_e*-def forward-def backward-def obstacle-def*
IRQs-def get-evt-label-def evtrgfset-def)
apply *force*
using *evtrgfset-def* **by** *blast*

lemma *evtrgfset-func*: $\forall$ *ef*$\in$*evtrgfset*. *evtrgffun* (E_e *ef*) = *Some* (*snd ef*)
proof −
{
fix *ef*
assume *a0*: *ef*$\in$*evtrgfset*
then have E_e *ef*$\in$*Domain evtrgfset* **by** (*metis Domain-iff* E_e*-def surjective-pairing*)
then obtain *ef1* **where** *a1*: *ef1*$\in$*evtrgfset* $\wedge$ E_e *ef1* = E_e *ef* $\wedge$ *evtrgffun* (E_e *ef*) = *Some* (*snd ef1*)
using *evtrgffun-exist*[*rule-format*,*of* E_e *ef*] **by** *auto*
have *evtrgffun* (E_e *ef*) = *Some* (*snd ef*)
proof(*cases ef1* = *ef*)
assume *ef1* = *ef*
with *a1* **show** *?thesis* **by** *simp*
next
assume *b0*: *ef1*$\neq$*ef*
with *diff-e-in-evtrgfset a0 a1* **have** E_e *ef1* $\neq$ E_e *ef* **by** *blast*
with *a1* **show** *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *bsc-evts-rgfs*: $\forall$ *erg*$\in$*all-evts* (*Carsystem-Spec*). (*evtrgffun* (E_e *erg*)) = *Some* (*snd erg*)
using *evtrgfset-func evtrgfset-eq-allevts-Spec* **by** *simp*

lemma *id-belong*[*simp*]: *Id* $\subseteq$⦃^{a}x= ^{o}x⦄
by (*simp add*: *Collect-mono Id-fstsnd-eq*)

lemma *collide-subset*: *set a* $\subseteq$ *set b* $\Longrightarrow$ *collide x a* $\Longrightarrow$ *collide x b*
unfolding *collide-def* **by** (*simp add*: *find-None-iff subset-eq*)

lemma *forward-satRG*: *forward v* $\vdash$ *forward-RGCond v*
apply(*simp add*:*Evt-sat-RG-def*)
apply (*simp add*: *forward-def forward-RGCond-def*)
apply(*rule BasicEvt*)

```
apply(simp add:body-def Pre_f-def Post_f-def guard-def
        Rely_f-def Guar_f-def getrgformula-def)

apply(rule Seq[where mid={|´car-pos = ´pos-aux + ´i ∧ (int v = ´i ∨ collide (´car-pos + 1) ´obstacle-pos) |}])
 apply(rule Seq[where mid={|´car-pos = ´pos-aux + ´i|}])
  apply(rule Seq[where mid={|´i = 0|}])

    apply(simp add:stm-def)
    apply(rule Await)
      apply(simp add:stable-def)+
      apply(rule allI)
      apply(rule Basic)
        apply auto[1]
        apply(simp add:stable-def)
        apply(simp add:stable-def)
        apply(simp add:stable-def) apply auto[1]

    apply(simp add:stm-def)
    apply(rule Await)
      apply(simp add:stable-def)+
      apply(rule allI)
      apply(rule Basic)
        apply auto[1]
        apply(simp add:stable-def)
        apply(simp add:stable-def)
        apply(simp add:stable-def) apply auto[1]

    apply(rule While)
      apply(simp add:stable-def)
      apply(simp add: collide-def) apply auto[1]
      apply(simp add:stable-def) apply(rule allI) apply(rule impI)+ apply(rule allI)
        apply(case-tac int v = i x)
          apply auto[1]
          apply simp apply (metis collide-subset)

      apply(rule Seq[where mid={|´car-pos = ´pos-aux + ´i + 1|}])
        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def) apply metis
          apply(simp add:stable-def)
            apply(rule allI)
            apply(rule Await)
              apply(simp add:stable-def) apply auto[1]
              apply(simp add:stable-def) apply auto[1]
              apply(rule allI)
              apply(rule Cond)
                apply(simp add:stable-def) apply auto[1]
                  apply(case-tac V = Va)
                    apply simp
                    apply(rule Basic)
                      apply auto[1]
                      apply(simp add:stable-def)
                      apply(simp add:stable-def) apply auto[1]
                      apply(simp add:stable-def) apply auto[1]
                    apply simp
                    apply(rule Basic)
                      apply simp+
                      apply(simp add:stable-def)+ apply auto[1]
```

```
              apply (simp add:Skip-def)
              apply(rule Basic)
                apply auto[1]
                apply simp
                apply(simp add:stable-def) apply auto[1]
                apply(simp add:stable-def) apply auto[1]
                apply simp
          apply(simp add:stm-def)
          apply(rule Await)
            apply(simp add:stable-def)+
            apply(rule allI)
            apply(rule Basic)
              apply auto[1]
              apply(simp add:stable-def)+ apply auto[1]
          apply simp
          apply(simp add:stm-def)
          apply(rule Await)
            apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
              apply(case-tac int v = i x)
                apply simp apply (metis collide-subset)

            apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
              apply(case-tac int v = i x)
                apply simp
                apply simp apply (metis collide-subset)
            apply(rule allI)
            apply(simp add:iret-def)
            apply(rule Basic)
              apply auto[1]
              apply(simp add:stable-def)+ apply auto[1]
              apply(simp add:stable-def) apply auto[1]
      apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
      apply(simp add: Guar_f-def getrgformula-def)
  done

lemma backward-satRG: backward v ⊢ backward-RGCond v
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: backward-def backward-RGCond-def)
    apply(rule BasicEvt)
      apply(simp add:body-def Pre_f-def Post_f-def guard-def
                Rely_f-def Guar_f-def getrgformula-def)

      apply(rule Seq[where mid={´car-pos = ´pos-aux − ´i ∧ (int v = ´i ∨ collide (´car-pos − 1) ´obstacle-pos) }])
       apply(rule Seq[where mid={´car-pos = ´pos-aux − ´i}])
        apply(rule Seq[where mid={´i = 0}])

          apply(simp add:stm-def)
          apply(rule Await)
            apply(simp add:stable-def)+
            apply(rule allI)
            apply(rule Basic)
              apply auto[1]
              apply(simp add:stable-def)
              apply(simp add:stable-def)
              apply(simp add:stable-def) apply auto[1]

          apply(simp add:stm-def)
          apply(rule Await)
```

```
  apply(simp add:stable-def)+
  apply(rule allI)
  apply(rule Basic)
    apply auto[1]
    apply(simp add:stable-def)
    apply(simp add:stable-def)
    apply(simp add:stable-def) apply auto[1]

apply(rule While)
  apply(simp add:stable-def)
  apply(simp add: collide-def) apply auto[1]
  apply(simp add:stable-def) apply(rule allI) apply(rule impI)+ apply(rule allI)
    apply(case-tac int v = i x)
      apply auto[1]
      apply simp apply (metis collide-subset)

  apply(rule Seq[where mid={´car-pos = ´pos-aux − ´i − 1}])
    apply(simp add:stm-def)
    apply(rule Await)
      apply(simp add:stable-def) apply metis
      apply(simp add:stable-def)
        apply(rule allI)
        apply(rule Await)
          apply(simp add:stable-def) apply auto[1]
          apply(simp add:stable-def) apply auto[1]
          apply(rule allI)
          apply(rule Cond)
            apply(simp add:stable-def) apply auto[1]
              apply(case-tac V = Va)
                apply simp
                apply(rule Basic)
                  apply auto[1]
                  apply(simp add:stable-def)
                  apply(simp add:stable-def) apply auto[1]
                  apply(simp add:stable-def) apply auto[1]
                apply simp
                apply(rule Basic)
                  apply simp+
                  apply(simp add:stable-def)+ apply auto[1]
            apply (simp add:Skip-def)
            apply(rule Basic)
              apply auto[1]
              apply simp
              apply(simp add:stable-def) apply auto[1]
              apply(simp add:stable-def) apply auto[1]
              apply simp
    apply(simp add:stm-def)
    apply(rule Await)
      apply(simp add:stable-def)+
      apply(rule allI)
      apply(rule Basic)
        apply auto[1]
        apply(simp add:stable-def)+ apply auto[1]
    apply simp
    apply(simp add:stm-def)
    apply(rule Await)
      apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
        apply(case-tac int v = i x)
```

```
          apply simp apply (metis collide-subset)

        apply(simp add:stable-def) apply(rule allI) apply(rule impI)+
          apply(case-tac int v = i x)
            apply simp
            apply simp apply (metis collide-subset)
        apply(rule allI)
        apply(simp add:iret-def)
        apply(rule Basic)
          apply auto[1]
          apply(simp add:stable-def)+ apply auto[1]
          apply(simp add:stable-def) apply auto[1]
    apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
    apply(simp add: Guar_f-def getrgformula-def)
  done

lemma obstacle-satRG: obstacle v ⊢ obstacle-RGCond v
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: obstacle-def obstacle-RGCond-def)
  apply(rule BasicEvt)
    apply(simp add:body-def Pre_f-def Post_f-def guard-def
              Rely_f-def Guar_f-def getrgformula-def)
    apply(rule Seq[where mid={´obstacle-pos = v # ´obst-pos-aux ∨ ´obstacle-pos = ´obst-pos-aux}])
      apply(rule Seq[where mid={´obst-pos-aux = ´obstacle-pos}])

        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(case-tac hd (stack V) = R)
            apply simp
            apply(rule Basic)
              apply simp+
              apply(simp add:stable-def)+ apply auto[1]
            apply simp
            apply(rule Basic)
              apply(simp add:stable-def)+

        apply(simp add:stm-def)
        apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(rule Cond)
            apply(simp add:stable-def)
            apply(case-tac obst-pos-aux V = obstacle-pos V ∧ hd (stack V) = R ∧ v ≠ car-pos V ∧
                        v ≠ car-pos V + 1 ∧
                        v ≠ car-pos V −1)
              apply simp
              apply(rule Basic)
                apply(simp add:collide-def)
                apply auto[1]
                apply auto[1]
                apply(simp add:stable-def)+
              apply(rule Basic)
                apply(simp add:collide-def)
                apply auto[1]
                apply auto[1]
                apply(simp add:stable-def)+
```

```
          apply(simp add:Skip-def)
          apply(rule Basic)
            apply auto[1]
            apply(simp add:stable-def)+
        apply(simp add:stm-def)
          apply(rule Await)
          apply(simp add:stable-def)+
          apply(rule allI)
          apply(simp add:iret-def)
          apply(rule Basic)
            apply auto[1]
            apply simp
            apply(simp add:stable-def)
            apply(simp add:stable-def)
    apply(simp add: stable-def Pre_f-def getrgformula-def Rely_f-def)
    apply(simp add: Guar_f-def getrgformula-def)
  done

lemma Interrupt-satRG: IRQs d ⊢ IRQs-RGCond d
  apply(simp add:Evt-sat-RG-def)
  apply (simp add: IRQs-def IRQs-RGCond-def)
  apply(rule BasicEvt)
    apply(simp add:body-def Pre_f-def Post_f-def guard-def
                 Rely_f-def Guar_f-def getrgformula-def)
    apply(rule Await)
      apply(simp add:stable-def)+
      apply(rule allI)
      apply(rule Cond)
        apply(simp add:stable-def)
        apply(simp add:push-def)
        apply(rule Basic)
          apply auto[1]
          apply(simp add:stable-def)+

        apply(simp add:Skip-def)
        apply(rule Basic)
          apply auto[1]
          apply(simp add:stable-def)
          apply(simp add:stable-def)
          apply(simp add:stable-def)
          apply simp

    apply(simp add:stable-def Pre_f-def Rely_f-def getrgformula-def)
    by(simp add:Guar_f-def getrgformula-def)

lemma EvtSys-on-Motor-SatRG:
  ⊢ fst (EvtSys-on-Motor-RGF) sat_s
             [Pre_f (snd (EvtSys-on-Motor-RGF)),
              Rely_f (snd (EvtSys-on-Motor-RGF)),
              Guar_f (snd (EvtSys-on-Motor-RGF)),
              Post_f (snd (EvtSys-on-Motor-RGF))]
    apply(simp add:EvtSys-on-Motor-RGF-def Pre_f-def Rely_f-def
               Guar_f-def Post_f-def getrgformula-def)
    apply(rule EvtSys-h)

  apply clarify
  apply(case-tac (a,b)∈ (⋃ v. {forward-RGF v}))
```

using *forward-satRG forward-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* $(a,b) \in (\bigcup v.\ \{backward\text{-}RGF\ v\})$)
using *backward-satRG backward-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 *blast*

apply *clarify*
apply(*case-tac* $(a,b) \in (\bigcup v.\ \{forward\text{-}RGF\ v\})$)
apply(*simp add*: *forward-RGF-def* E_e*-def* Pre_e*-def forward-RGCond-def*)
apply (*smt UNIV-I getrgformula-def rgformula.simps*(*1*))
apply(*case-tac* $(a,b) \in (\bigcup v.\ \{backward\text{-}RGF\ v\})$)
apply(*simp add*: *backward-RGF-def* E_e*-def* Pre_e*-def backward-RGCond-def*)
apply (*smt UNIV-I getrgformula-def rgformula.simps*(*1*))
apply *blast*

unfolding *Ball-def* **apply**(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup v.\ \{forward\text{-}RGF\ v\})$)
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Rely_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Rely_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]

apply(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup v.\ \{forward\text{-}RGF\ v\})$)
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Guar_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Guar_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]

apply(*rule allI*) **apply**(*rule impI*)
apply(*case-tac* $x \in (\bigcup v.\ \{forward\text{-}RGF\ v\})$)
apply (*simp add*:*forward-RGF-def forward-RGCond-def* $Post_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]
apply (*simp add*:*backward-RGF-def backward-RGCond-def* $Post_e$*-def getrgformula-def*)
apply (*erule exE*) **apply** *auto*[*1*]

apply *auto*[*1*]
apply (*simp add*:*forward-RGF-def forward-RGCond-def backward-RGF-def*
backward-RGCond-def Pre_e*-def* $Post_e$*-def getrgformula-def*)+

apply(*simp add*:*stable-def*)
by *simp*

lemma *EvtSys-on-Radar-SatRG*:
$\vdash$ *fst* (*EvtSys-on-Radar-RGF*) sat_s
[Pre_f (*snd* (*EvtSys-on-Radar-RGF*)),
$Rely_f$ (*snd* (*EvtSys-on-Radar-RGF*)),
$Guar_f$ (*snd* (*EvtSys-on-Radar-RGF*)),
$Post_f$ (*snd* (*EvtSys-on-Radar-RGF*))]
apply(*simp add*:*EvtSys-on-Radar-RGF-def* Pre_f*-def* $Rely_f$*-def*
$Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply *auto*[*1*]
apply(*simp add*:E_e*-def obstacle-RGF-def*)

using *obstacle-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 obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply(*simp add*: $Rely_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply(*simp add*: $Guar_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*) **apply** *auto*[*1*]
apply(*simp add*: $Post_e$*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*) **apply** *auto*[*1*]
apply(*simp add*: $Post_e$*-def* Pre_e*-def obstacle-RGF-def obstacle-RGCond-def getrgformula-def*)
apply(*simp add*:*stable-def*)
by *simp*

lemma *EvtSys-on-PIC-SatRG*:
$\vdash$ *fst* (*EvtSys-on-PIC-RGF*) sat_s
[Pre_f (*snd* (*EvtSys-on-PIC-RGF*)),
$Rely_f$ (*snd* (*EvtSys-on-PIC-RGF*)),
$Guar_f$ (*snd* (*EvtSys-on-PIC-RGF*)),
$Post_f$ (*snd* (*EvtSys-on-PIC-RGF*))]
apply(*simp add*:*EvtSys-on-PIC-RGF-def* Pre_f*-def* $Rely_f$*-def* $Guar_f$*-def* $Post_f$*-def getrgformula-def*)
apply(*rule EvtSys-h*)

apply *auto*[*1*]
apply(*simp add*:E_e*-def IRQs-RGF-def*)
using *Interrupt-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 IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply(*simp add*: $Rely_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply(*simp add*: $Guar_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply(*rule allI*) **apply** *auto*[*1*]
apply(*simp add*: $Post_e$*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*) **apply** *auto*[*1*]
apply(*simp add*: $Post_e$*-def* Pre_e*-def IRQs-RGF-def IRQs-RGCond-def getrgformula-def*)
apply(*simp add*:*stable-def*)
by *simp*

lemma *functional-correctness*: $\vdash$ *Carsystem-Spec SAT*
[{| *True* |},
{| ª*car-pos* = º*car-pos* ∧ ª*i* = º*i* ∧ ª*pos-aux* = º*pos-aux* ∧ ª*obstacle-pos* = º*obstacle-pos*
∧ ª*obst-pos-aux* = º*obst-pos-aux*
∧ (*hd* º*stack* ≠ *C* ⟶ ((ª*stack* = *tl* º*stack* ∨ ª*obst-pos-aux* = º*obstacle-pos*
∨ ª*stack* = *C* # º*stack*) ∧ ª*obstacle-pos* = º*obstacle-pos*)
∨ (*set* º*obstacle-pos* ⊆ *set* ª*obstacle-pos*
∧ *collide* (º*car-pos* − *1*) º*obstacle-pos* = *collide* (ª*car-pos* − *1*) ª*obstacle-pos*
∧ *collide* (º*car-pos* + *1*) º*obstacle-pos* = *collide* (ª*car-pos* + *1*) ª*obstacle-pos*))
∧ (*hd* º*stack* = *C* ⟶ º*obstacle-pos* = ª*obstacle-pos* ∧ ª*stack* = *R* # º*stack*
∧ º*obst-pos-aux* = ª*obst-pos-aux*)
∧ (*hd* º*stack* ≠ *R* ⟶ ª*i* = *0* ∨ ª*i* = º*i* + *1* ∨ ª*stack* = *tl* º*stack*
∨ (¬*collide* (º*car-pos* + *1*) º*obstacle-pos* ∧ ª*car-pos* = º*car-pos* + *1*)
∨ (¬*collide* (º*car-pos* − *1*) º*obstacle-pos* ∧ ª*car-pos* = º*car-pos* − *1*)
∨ ª*stack* = *R* # º*stack*)
∧ (*hd* º*stack* = *R* ⟶ º*car-pos* = ª*car-pos* ∧ º*i* = ª*i* ∧ ª*pos-aux* = º*pos-aux*
∧ ª*stack* = *C* # º*stack*) |} ∪ *Id*,
{| *hd* º*stack* = *C* ∧ (ª*i* = *0* ∨ ª*i* = º*i* + *1* ∨ ª*stack* = *tl* º*stack* ∨
(¬*collide* (º*car-pos* + *1*) º*obstacle-pos* ∧ ª*car-pos* = º*car-pos* + *1*) ∨
(¬*collide* (º*car-pos* − *1*) º*obstacle-pos* ∧ ª*car-pos* = º*car-pos* − *1*))
∧ ª*obstacle-pos* = º*obstacle-pos* ∧ ª*obst-pos-aux* = º*obst-pos-aux* |}

$\cup$ $\{\!\!\{ hd\ {}^{o}stack = R \wedge (((^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ {}^{o}car\text{-}pos\ {}^{o}obstacle\text{-}pos = collide\ {}^{a}car\text{-}pos\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge\ {}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux\ \}\!\!\}$
$\cup\ (\bigcup d.\ (\{\!\!\{ hd\ {}^{o}stack \neq d \wedge {}^{a}stack = d \# {}^{o}stack \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos$
$\wedge\ {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos$
$\wedge\ {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux \}\!\!\})) \cup Id,$
$\{\!\!\{ True \}\!\!\}]$

apply (*rule ParallelESys*)
apply(*simp add:Carsystem-Spec-def*)
apply(*rule allI*)
using *EvtSys-on-Motor-SatRG EvtSys-on-Radar-SatRG EvtSys-on-PIC-SatRG*
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 Device.case(1) Device.case(2) Device.exhaust Device.simps(9)*)

apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* Pre_{es}*-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac k = Ctrl*)
apply (*simp add:EvtSys-on-Motor-RGF-def getrgformula-def*)
apply(*case-tac k = Radar*)
apply (*simp add:EvtSys-on-Radar-RGF-def getrgformula-def*)
apply(*case-tac k = PIC*)
apply (*simp add:EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*

apply *simp*
apply(*rule allI*)
apply(*rule conjI*)
apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply(*case-tac k = Ctrl*)
apply(*simp add: EvtSys-on-Motor-RGF-def getrgformula-def*) **apply** *auto[1]*
apply(*case-tac k = Radar*)
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*
apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply(*case-tac k = Ctrl*)
apply(*simp add: EvtSys-on-Motor-RGF-def getrgformula-def*) **apply** *auto[1]*
apply(*case-tac k = Radar*)
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def*)
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*

apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac j = Ctrl*)
apply(*case-tac k = Ctrl*)
apply *simp*

```
      apply(case-tac k = Radar)
        apply auto[1]
        apply(simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def)
        apply auto[1]
        apply(case-tac k = PIC)
          apply(simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def)
          using Device.exhaust apply blast
  apply(case-tac j = Radar)
    apply(case-tac k = Radar)
      apply simp
        apply(case-tac k = Ctrl)
          apply auto[1]
          apply(simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def)
            apply auto[1]
        apply(case-tac k = PIC)
          apply(simp add: EvtSys-on-PIC-RGF-def getrgformula-def)
          using Device.exhaust apply blast
  apply(case-tac j = PIC)
    apply(case-tac k = PIC)
      apply simp
        apply(case-tac k = Ctrl)
          apply auto[1]
          apply(simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def)
          apply auto[1]
          using Irq.exhaust apply blast
        apply(case-tac k = Radar)
          apply auto[1]
          apply(simp add: EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def)
            apply(case-tac a = b)
            apply simp
            apply simp
            apply (erule exE)
            apply(case-tac x = R)
              using Irq.exhaust apply auto[1]
              using Irq.exhaust apply auto[1]
        using Device.exhaust apply blast
  using Device.exhaust apply blast

 apply(simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def
       EvtSys-on-PIC-RGF-def Guar_es-def Rely_es-def getrgformula-def)
 apply(rule allI)
 apply(case-tac k = PIC)
  apply(simp add:  EvtSys-on-PIC-RGF-def getrgformula-def) apply auto[1]
  apply(case-tac k = Radar)
   apply(simp add:  EvtSys-on-Radar-RGF-def getrgformula-def) apply auto[1]
   apply(case-tac k = Ctrl)
    apply(simp add:  EvtSys-on-Motor-RGF-def getrgformula-def) apply auto[1]
 using Device.exhaust apply blast

 by(simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def
       EvtSys-on-PIC-RGF-def Post_es-def getrgformula-def)

lemma functional-correctness2: ⊢ Carsystem-Spec SAT
  [{True},
   {},
  {hd ºstack = C ∧ (ªi = 0 ∨ ªi = ºi + 1 ∨ ªstack = tl ºstack ∨
    (¬collide (ºcar-pos + 1) ºobstacle-pos ∧ ªcar-pos = ºcar-pos + 1) ∨
    (¬collide (ºcar-pos − 1) ºobstacle-pos ∧ ªcar-pos = ºcar-pos − 1))
```

$\wedge\ {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux\}$
$\cup\ \{hd\ {}^{o}stack = R \wedge ((({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee\ (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ {}^{o}car\text{-}pos\ {}^{o}obstacle\text{-}pos = collide\ {}^{a}car\text{-}pos\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge\ {}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux\ \}$
$\cup\ (\bigcup d.\ (\{hd\ {}^{o}stack \neq d \wedge {}^{a}stack = d\ \#\ {}^{o}stack \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos$
$\wedge\ {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos$
$\wedge\ {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux\})) \cup Id,$
$\{True\}]$

apply (*rule ParallelESys*)
apply(*simp add:Carsystem-Spec-def*)
apply(*rule allI*)
using *EvtSys-on-Motor-SatRG EvtSys-on-Radar-SatRG EvtSys-on-PIC-SatRG*
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 Device.case(1) Device.case(2) Device.exhaust Device.simps(9)*)

apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* Pre_{es}*-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac k = Ctrl*)
apply (*simp add:EvtSys-on-Motor-RGF-def getrgformula-def*)
apply(*case-tac k = Radar*)
apply (*simp add:EvtSys-on-Radar-RGF-def getrgformula-def*)
apply(*case-tac k = PIC*)
apply (*simp add:EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*

apply *simp*

apply(*simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def* $Guar_{es}$*-def* $Rely_{es}$*-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac j = Ctrl*)
apply(*case-tac k = Ctrl*)
apply *simp*
apply(*case-tac k = Radar*)
apply *auto[1]*
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*
apply(*case-tac j = Radar*)
apply(*case-tac k = Radar*)
apply *simp*
apply(*case-tac k = Ctrl*)
apply *auto[1]*
apply(*simp add: EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def getrgformula-def*)
apply *auto[1]*
apply(*case-tac k = PIC*)
apply(*simp add: EvtSys-on-PIC-RGF-def getrgformula-def*)
using *Device.exhaust* **apply** *blast*
apply(*case-tac j = PIC*)
apply(*case-tac k = PIC*)
apply *simp*

```
        apply(case-tac k = Ctrl)
          apply auto[1]
          apply(simp add: EvtSys-on-Motor-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def)
          apply auto[1]
          using Irq.exhaust apply blast
        apply(case-tac k = Radar)
          apply auto[1]
          apply(simp add: EvtSys-on-Radar-RGF-def EvtSys-on-PIC-RGF-def getrgformula-def)
            apply(case-tac a = b)
            apply simp
            apply simp
            apply (erule exE)
            apply(case-tac x = R)
              using Irq.exhaust apply auto[1]
              using Irq.exhaust apply auto[1]
        using Device.exhaust apply blast
    using Device.exhaust apply blast

  apply(simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def
        EvtSys-on-PIC-RGF-def Guar_es-def Rely_es-def getrgformula-def)
  apply(rule allI)
  apply(case-tac k = PIC)
    apply(simp add: EvtSys-on-PIC-RGF-def getrgformula-def) apply auto[1]
    apply(case-tac k = Radar)
      apply(simp add: EvtSys-on-Radar-RGF-def getrgformula-def) apply auto[1]
      apply(case-tac k = Ctrl)
        apply(simp add: EvtSys-on-Motor-RGF-def getrgformula-def) apply auto[1]
  using Device.exhaust apply blast

  by(simp add:Carsystem-Spec-def EvtSys-on-Motor-RGF-def EvtSys-on-Radar-RGF-def
        EvtSys-on-PIC-RGF-def Post_es-def getrgformula-def)
```

12.4 Invariant proof

```
lemma spec-sat-rg: ⊢ Carsystem-Spec SAT [{s0}, {}, UNIV, UNIV]
  using functional-correctness ParallelESys-conseq
    [where pre={s0} and pre'=UNIV and rely={} and guar=UNIV and post=UNIV and pesf=Carsystem-Spec]
      by simp


definition invariant :: (State) invariant
  where invariant ≡ {s. ¬ collide (car-pos s) (obstacle-pos s)}

lemma init-sat-inv: {s0}⊆ invariant
  by(simp add:s0-init init-def invariant-def collide-def)

lemma stb-guar-interrupt: stable invariant ({|hd ºstack ≠ d ∧ ªstack = d # ºstack ∧ ªcar-pos = ºcar-pos
            ∧ ªi = ºi ∧ ªpos-aux = ºpos-aux ∧ ªobstacle-pos = ºobstacle-pos
            ∧ ªobst-pos-aux = ºobst-pos-aux|} ∪ Id)
  unfolding stable-def invariant-def collide-def
  apply clarify
  apply simp
  by auto

lemma stb-guar-forward: stable invariant
    ({|hd ºstack = M ∧ (((ªi = 0 ∨ ªi = ºi + 1 ∨ ªstack = tl ºstack) ∧ ªcar-pos = ºcar-pos) ∨
        (¬collide (ºcar-pos + 1) ºobstacle-pos ∧ ªcar-pos = ºcar-pos + 1))
        ∧ ªobstacle-pos = ºobstacle-pos ∧ ªobst-pos-aux = ºobst-pos-aux|} ∪ Id)
```

unfolding *stable-def invariant-def collide-def*
apply *clarify*
apply *simp*
by *auto*

lemma *stb-guar-backward*: *stable invariant*
$(\{\!|hd\ {}^{o}stack = M \wedge (((^{a}i = 0 \vee {}^{a}i = {}^{o}i + 1 \vee {}^{a}stack = tl\ {}^{o}stack) \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos) \vee$
$(\neg collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos \wedge {}^{a}car\text{-}pos = {}^{o}car\text{-}pos - 1))$
$\wedge\ {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos \wedge {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obst\text{-}pos\text{-}aux|\!\} \cup Id)$
unfolding *stable-def invariant-def collide-def*
apply *clarify*
apply *simp*
by *auto*

lemma *stb-guar-obstacle*: *stable invariant*
$(\{\!|hd\ {}^{o}stack = R \wedge ((({}^{a}stack = tl\ {}^{o}stack \vee {}^{a}obst\text{-}pos\text{-}aux = {}^{o}obstacle\text{-}pos) \wedge {}^{a}obstacle\text{-}pos = {}^{o}obstacle\text{-}pos)$
$\vee\ (set\ {}^{o}obstacle\text{-}pos \subseteq set\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos - 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos - 1)\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ {}^{o}car\text{-}pos\ {}^{o}obstacle\text{-}pos = collide\ {}^{a}car\text{-}pos\ {}^{a}obstacle\text{-}pos$
$\wedge\ collide\ ({}^{o}car\text{-}pos + 1)\ {}^{o}obstacle\text{-}pos = collide\ ({}^{a}car\text{-}pos + 1)\ {}^{a}obstacle\text{-}pos))$
$\wedge\ {}^{a}car\text{-}pos = {}^{o}car\text{-}pos \wedge {}^{a}i = {}^{o}i \wedge {}^{a}pos\text{-}aux = {}^{o}pos\text{-}aux\ |\!\} \cup Id)$
unfolding *stable-def invariant-def collide-def*
apply *clarify*
apply *simp*
by *auto*

lemma *evts-stb-invar*: $\forall\ ef \in evtrgfset.\ stable\ invariant\ (Guar_e\ ef)$
unfolding *evtrgfset-def*
apply(*clarify*)
apply$(case\text{-}tac\ (a,\ b) \in (\bigcup k.\ \{(forward\ k,\ forward\text{-}RGCond\ k)\}))$
apply(*simp add*:*forward-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-forward rgformula.select-convs*(*3*) **apply** *auto*[*1*]

apply$(case\text{-}tac\ (a,\ b) \in (\bigcup k.\ \{(backward\ k,\ backward\text{-}RGCond\ k)\}))$
apply(*simp add*:*backward-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-backward rgformula.select-convs*(*3*) **apply** *auto*[*1*]

apply$(case\text{-}tac\ (a,\ b) \in (\bigcup k.\ \{(obstacle\ k,\ obstacle\text{-}RGCond\ k)\}))$
apply(*simp add*:*obstacle-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-obstacle rgformula.select-convs*(*3*) **apply** *auto*[*1*]

apply$(case\text{-}tac\ (a,\ b) \in (\bigcup k.\ \{(IRQs\ k,\ IRQs\text{-}RGCond\ k)\}))$
apply(*simp add*:*IRQs-RGCond-def* $Guar_e$*-def getrgformula-def*)
using *stb-guar-interrupt rgformula.select-convs*(*3*) **apply** *auto*[*1*]

by *blast*

theorem *Carsystem-invariant-theorem*:
invariant-of-pares (*paresys-spec Carsystem-Spec*) {*s0*} *invariant*
using *invariant-theorem*[*of Carsystem-Spec* {*s0*} *evtrgffun invariant*]
spec-sat-rg evts-stb-invar evtrgfset-eq-allevts-Spec
all-basic-evts-arinc evts-stb-invar init-sat-inv bsc-evts-rgfs **by** *auto*

end