PiCore: A Rely-guarantee Framework for Event-based Systems

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

March 17, 2019

Contents

1 Abstract Syntax of PiCore Language

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

type-synonym *'s bexp* = *'s set*

type-synonym *'s guard* = *'s set*

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

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

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

body ev ≡ *snd* (*snd ev*)

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

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

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

2 Some Lemmas of Abstract Syntax

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

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

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

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

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

end

3 Small-step Operational Semantics of PiCore Language

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

3.1 Datatypes for Semantics

datatype *cmd* = *CMP*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

assumes *petran-simps*:
$\Gamma \vdash (a,\ b)\ -pe\rightarrow\ (c,\ d) \Longrightarrow a = c$
assumes *none-no-tran'*: $((\mathit{fin\text{-}com},\ s),(P,t)) \notin \mathit{ptran}\ \Gamma$
assumes *ptran-neq*: $((P,\ s),(P,t)) \notin \mathit{ptran}\ \Gamma$
begin

definition *ptran'* :: $'Env \Rightarrow ('s,'prog)$ *pconf* $\Rightarrow$ $('s,'prog)$ *pconf* $\Rightarrow$ *bool* (- $\vdash$ - $-c\rightarrow$ - [*81*,*81*] *80*)
where $\Gamma \vdash P\ -c\rightarrow\ Q \equiv (P,Q) \in \mathit{ptran}\ \Gamma$

definition *ptrans* :: $'Env \Rightarrow ('s,'prog)$ *pconf* $\Rightarrow$ $('s,'prog)$ *pconf* $\Rightarrow$ *bool* (- $\vdash$ - $-c{*}\rightarrow$ - [*81*,*81*,*81*] *80*)
where $\Gamma \vdash P\ -c{*}\rightarrow\ Q \equiv (P,Q) \in(\mathit{ptran}\ \Gamma)\hat{}{*}$

lemma *none-no-tran*: $\neg(\Gamma \vdash (\mathit{fin\text{-}com},s)\ -c\rightarrow\ (P,t))$
using *none-no-tran'* **by**(*simp add*:*ptran'-def*)

lemma *none-no-tran2*: $\neg(\Gamma \vdash (\mathit{fin\text{-}com},s)\ -c\rightarrow\ Q)$
using *none-no-tran* **by** (*metis prod.collapse*)

lemma *ptran-not-none*: $(\Gamma \vdash (Q,s)\ -c\rightarrow\ (P,t)) \Longrightarrow Q \neq \mathit{fin\text{-}com}$

using *none-no-tran* **apply**(*simp add:ptran′-def*) **using** *not-None-eq* **by** *metis*

3.2 Semantics of Events

inductive *etran* :: *′Env* ⇒ (*′l*,*′k*,*′s*,*′prog*) *econf* ⇒ (*′l*,*′k*,*′s*,*′prog*) *actk* ⇒ (*′l*,*′k*,*′s*,*′prog*) *econf* ⇒ *bool*
(- ⊢ - −*et*−-→ - [*81*,*81*,*81*] *80*)
where
AnonyEvent: Γ ⊢ (*P*, *s*) −*c*→ (*Q*, *t*) ⟹ Γ ⊢ (*AnonyEvent P*, *s*, *x*) −*et*−(*Cmd CMP*)♯*k*→ (*AnonyEvent Q*, *t*, *x*)
| *EventEntry*: ⟦*P* = *body e*; *P* ≠ *fin-com*; *s* ∈ *guard e*; *x′* = *x*(*k*:= *BasicEvent e*)⟧
⟹ Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ ((*AnonyEvent P*), *s*, *x′*)

3.3 Semantics of Event Systems

inductive *estran* :: *′Env* ⇒ (*′l*,*′k*,*′s*,*′prog*) *esconf* ⇒ (*′l*,*′k*,*′s*,*′prog*) *actk* ⇒ (*′l*,*′k*,*′s*,*′prog*) *esconf* ⇒ *bool*
(- ⊢ - −*es*−-→ - [*81*,*81*] *80*)
where
EvtOccur: ⟦*evt*∈ *evts*; Γ ⊢ (*evt*, (*s*, *x*)) −*et*−(*EvtEnt evt*)♯*k*→ (*e*, (*s*, *x′*)) ⟧
⟹ Γ ⊢ (*EvtSys evts*, (*s*, *x*)) −*es*−(*EvtEnt evt*)♯*k*→ (*EvtSeq e* (*EvtSys evts*), (*s*, *x′*))
| *EvtSeq1*: ⟦Γ ⊢ (*e*, *s*, *x*) −*et*−*act*♯*k*→(*e′*, *s′*, *x′*); *e′* ≠ *AnonyEvent fin-com*⟧
⟹ Γ ⊢ (*EvtSeq e es*, *s*, *x*) −*es*−*act*♯*k*→ (*EvtSeq e′ es*, *s′*, *x′*)
| *EvtSeq2*: ⟦Γ ⊢ (*e*, *s*, *x*) −*et*−*act*♯*k*→ (*e′*, *s′*, *x′*); *e′* = *AnonyEvent fin-com*⟧
⟹ Γ ⊢ (*EvtSeq e es*, *s*, *x*) −*es*−*act*♯*k*→ (*es*, *s′*, *x′*)

3.4 Semantics of Parallel Event Systems

inductive
pestran :: *′Env* ⇒ (*′l*,*′k*,*′s*,*′prog*) *pesconf* ⇒ (*′l*,*′k*,*′s*,*′prog*) *actk*
⇒ (*′l*,*′k*,*′s*,*′prog*) *pesconf* ⇒ *bool* (- ⊢ - −*pes*−-→ - [*70*,*70*] *60*)
where
ParES: Γ ⊢ (*pes*(*k*), (*s*, *x*)) −*es*−(*a*♯*k*)→ (*es′*, (*s′*, *x′*)) ⟹ Γ ⊢ (*pes*, (*s*, *x*)) −*pes*−(*a*♯*k*)→ (*pes*(*k*:=*es′*), (*s′*, *x′*))

3.5 Lemmas

3.5.1 programs

lemma *list-eq-if* [*rule-format*]:
∀ *ys*. *xs*=*ys* ⟶ (*length xs* = *length ys*) ⟶ (∀ *i*<*length xs*. *xs*!*i*=*ys*!*i*)
by (*induct xs*) *auto*

lemma *list-eq*: (*length xs* = *length ys* ∧ (∀ *i*<*length xs*. *xs*!*i*=*ys*!*i*)) = (*xs*=*ys*)
apply(*rule iffI*)
apply *clarify*
apply(*erule nth-equalityI*)
apply *simp*+

done

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

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

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

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

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

3.5.2 Events

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

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

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

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

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

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

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

lemma *noevtent-notran*: $ec1 = (BasicEvent\ e,\ s,\ x) \Longrightarrow \neg\ (\exists\, k.\ \Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2)$
            $\Longrightarrow \neg\ (\Gamma \vdash ec1\ -et-t\rightarrow\ ec2)$
  proof −
    assume *p0*: $ec1 = (BasicEvent\ e,\ s,\ x)$ **and**
        p1: $\neg\ (\exists\, k.\ \Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2)$
    then show $\neg\ (\Gamma \vdash ec1\ -et-t\rightarrow\ ec2)$
      proof −
      {
        assume *a0*: $\Gamma \vdash ec1\ -et-t\rightarrow\ ec2$
          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** $\Gamma \vdash ec1\ -et-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow\ ec2$ **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*:$\Gamma \vdash (P,s,x)\ -et-et\rightarrow\ (Q,t,y) \Longrightarrow P \neq Q$
  apply(*erule etran.cases*)
  apply (*simp add*: *prog-not-eq-in-ctran-aux*)
  by *simp*

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

apply *simp*
done

lemma *evt-not-eq-in-tran2* [*simp*]: $\neg(\exists\, et.\ \Gamma \vdash (P,s,x)\ -et-et\rightarrow\ (P,t,y))$ **by** *simp*

3.5.3 Event Systems

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

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

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

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

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

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

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

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

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

lemma *evtseq-no-evtent*:
$[\![\Gamma \vdash (EvtSeq\ e1\ es,\ s1,\ x1)\ -es-t\sharp k\rightarrow (es2,\ s2,\ x2); is\text{-}anonyevt\ e1]\!] \Longrightarrow \neg(\exists\, e.\ t = EvtEnt\ e)$
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*:
$[\![\Gamma \vdash esc1\ -es-t\sharp k\rightarrow esc2;\ getspc\text{-}es\ esc1 = EvtSeq\ e\ esys;\ is\text{-}anonyevt\ e]\!] \Longrightarrow \neg(\exists\, e.\ t = EvtEnt\ e)$
proof −
assume *p0*: $\Gamma \vdash esc1\ -es-t\sharp k\rightarrow esc2$
and *p1*: *getspc-es esc1* = *EvtSeq e esys*
and *p2*: *is-anonyevt e*
then obtain *es1* **and** *s1* **and** *x1* **where** *a1*: *esc1* = (*es1*,*s1*,*x1*)
using *prod-cases3* **by** *blast*
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a2*: *esc2* = (*es2*,*s2*,*x2*)
using *prod-cases3* **by** *blast*
from *p1 a1* **have** *es1* = *EvtSeq e esys* **by** (*simp add*:*getspc-es-def*)
with *p0 p2 a1 a2* **show** *?thesis* **using** *evtseq-no-evtent*[*of* Γ *e esys s1 x1 t k es2 s2 x2*]
by *simp*

qed

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

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

lemma *evtent-is-basicevt*: $\Gamma \vdash$ (*es*, *s*, *x*) $-es-$*EvtEnt e*$\sharp$*k*$\rightarrow$ (*es'*, *s'*, *x'*) $\Longrightarrow$ $\exists$ *e'*. *e* = *BasicEvent e'*
  apply(*rule estran.cases*)
  apply(*simp add*:*get-actk-def*)+
  apply(*rule etran.cases*)
  apply(*simp add*:*get-actk-def*)+
  apply(*rule etran.cases*)
  apply *simp*+
  apply(*rule etran.cases*)
  apply *simp*+
  apply *auto*[*1*]
  apply (*metis ent-spec1 event.exhaust evtseq-no-evtent get-actk-def is-anonyevt.simps*(*1*))+

  done

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

lemma *evtent-is-basicevt-inevtseq2*: $[\![\Gamma \vdash$ *esc1* $-es-$*EvtEnt e1*$\sharp$*k*$\rightarrow$ *esc2*; *getspc-es esc1* = *EvtSeq e es*$]\!]$
    $\Longrightarrow$ *e* = *e1* $\wedge$ ($\exists$ *e'*. *e* = *BasicEvent e'*)
  proof –
    assume *p0*: $\Gamma \vdash$ *esc1* $-es-$*EvtEnt e1*$\sharp$*k*$\rightarrow$ *esc2*
      and *p1*: *getspc-es esc1* = *EvtSeq e es*
    then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: *esc1* = (*es1*,*s1*,*x1*)
      using *prod-cases3* **by** *blast*
    moreover

from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
  using *prod-cases3* **by** *blast*
ultimately show *?thesis*
  using *p0 p1 evtent-is-basicevt-inevtseq*[*of* Γ *e es s1 x1 e1 k es2 s2 x2*] *getspc-es-def*[*of esc1*] **by** *auto*
qed

lemma *evtsysent-evtent0*: Γ ⊢ (*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*) ∧
  Γ ⊢ (*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 Pair-inject esys.inject*(*1*) *esys.inject*(*2*) *select-convs*(*1*))

lemma *evtsysent-evtent*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) ⟹
  s = *s1* ∧ *BasicEvent e* ∈ *es* ∧ Γ ⊢ (*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*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt ev*)♯*k*→ (*esc2*, *s1*,*x1*) ⟹
  s = *s1* ∧ (*ev*∈*es*)
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis ent-spec1*)
apply(*simp*)+
done

lemma *evtsysent-evtent3*: ⟦Γ ⊢ *esc1* −*es*−(*EvtEnt ev*)♯*k*→ *esc2*; *getspc-es esc1* = *EvtSys es*⟧ ⟹
  (*ev*∈*es*)
proof −

assume $p0$: $\Gamma \vdash esc1\ -es-(EvtEnt\ ev)\sharp k\rightarrow\ esc2$
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* Γ *es s1 x1 ev k es2 s2 x2*] **by** *simp*
qed

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

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

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

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

lemma *cmd-enable-impl-notesys2*:
$[\![\Gamma \vdash esc1\ -es-(Cmd\ c)\sharp k\rightarrow\ esc2]\!]$
$\Longrightarrow \neg(\exists\, ess.\ getspc\text{-}es\ esc1 = EvtSys\ ess)$

proof −
assume *p0*: $\Gamma \vdash$ *esc1* −*es*−(*Cmd c*)♯*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* Γ *es1 s1 x1 c k es2 s2 x2*] *getspc-es-def*[*of esc1*]
by *simp*
qed

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

lemma *entevt-notchgstate*: ⟦Γ ⊢ (*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*: ⟦Γ ⊢ (*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*: ⟦Γ ⊢ *esc1* −*es*−(*EvtEnt e*)♯*k*→ *esc2*⟧

$\Longrightarrow (\forall k'. \ k' \neq k \longrightarrow (getx\text{-}es\ esc1)\ k' = (getx\text{-}es\ esc2)\ k')$

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

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

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

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

done

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

lemma *estran-impl-evtentorcmd*: $\llbracket\Gamma \vdash (es, s, x)\ -es-t\rightarrow\ (es', s', x')\rrbracket$
$\Longrightarrow (\exists\, e\ k.\ \Gamma \vdash (es, s, x)\ -es-EvtEnt\ e\sharp k\rightarrow\ (es', s', x')) \vee (\exists\, c\ k.\ \Gamma \vdash (es, s, x)\ -es-Cmd\ c\sharp k\rightarrow\ (es', s', x'))$
apply(*rule estran.cases*)
apply (*simp add*: *get-actk-def*)
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*[*1*]
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)+
apply *auto*[*1*]
apply (*metis get-actk-def*)
apply(*rule etran.cases*)
apply (*simp add*: *get-actk-def*)
apply (*metis get-actk-def*)
apply (*metis get-actk-def*)
done

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

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

lemma *estran-impl-evtentorcmd2'*: $[\![\Gamma \vdash esc1\ -es-t\sharp k\rightarrow\ esc2]\!]$
$\Longrightarrow (\exists\, e.\ \Gamma \vdash esc1\ -es-EvtEnt\ e\sharp k\rightarrow\ esc2) \vee (\exists\, c.\ \Gamma \vdash esc1\ -es-Cmd\ c\sharp k\rightarrow\ esc2)$
proof −
assume *p0*: $\Gamma \vdash esc1\ -es-t\sharp k\rightarrow\ esc2$
then obtain *es1* **and** *s1* **and** *x1* **where** *a0*: $esc1 = (es1,s1,x1)$
using *prod-cases3* **by** *blast*
moreover
from *p0* **obtain** *es2* **and** *s2* **and** *x2* **where** *a1*: $esc2 = (es2,s2,x2)$
using *prod-cases3* **by** *blast*
ultimately show *?thesis* **using** *p0 estran-impl-evtentorcmd'*[*of* Γ *es1 s1 x1 t k es2 s2 x2*] **by** *simp*
qed

3.5.4 Parallel Event Systems

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

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

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

end

end

4 Computations of PiCore Language

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

4.1 Environment transitions

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

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

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

begin

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

using *cpts-p-simps*[*of* [(*P*,*s*)] Γ] **by** *auto*

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

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

4.2 Sequential computations

4.2.1 Sequential computations of programs

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

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

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

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

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

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

4.2.2 Sequential computations of events

inductive-set *cpts-ev* :: *'Env* ⇒ (*'l*,*'k*,*'s*,*'prog*) *econfs set*
for Γ :: *'Env*
where
CptsEvOne: $[(e,s,x)] \in$ *cpts-ev* Γ

| *CptsEvEnv*: $(e,\ t,\ x)\#xs \in$ *cpts-ev* $\Gamma \Longrightarrow (e,\ s,\ y)\#(e,\ t,\ x)\#xs \in$ *cpts-ev* Γ
| *CptsEvComp*: $[\![\Gamma \vdash (e1,s,x)$ $-et-ct\rightarrow$ $(e2,t,y)$; $(e2,t,y)\#xs \in$ *cpts-ev* $\Gamma]\!] \Longrightarrow$ $(e1,s,x)\#(e2,t,y)\#xs \in$ *cpts-ev* Γ

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

4.2.3 Sequential computations of event systems

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

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

4.2.4 Sequential computations of par event systems

inductive-set *cpts-pes* :: *'Env* $\Rightarrow$ *('l,'k,'s,'prog) pesconfs set*
for Γ :: *'Env*
where
  CptsPesOne: $[(pes,s,x)] \in$ *cpts-pes* Γ
| *CptsPesEnv*: $(pes,\ t,\ x)\#xs \in$ *cpts-pes* $\Gamma \Longrightarrow (pes,\ s,\ y)\#(pes,\ t,\ x)\#xs \in$ *cpts-pes* Γ
| *CptsPesComp*: $[\![\Gamma \vdash (pes1,s,x)$ $-pes-ct\rightarrow$ $(pes2,t,y)$; $(pes2,t,y)\#xs \in$ *cpts-pes* $\Gamma]\!] \Longrightarrow (pes1,s,x)\#(pes2,t,y)\#xs \in$ *cpts-pes* Γ

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

4.3 Lemmas

4.3.1 Events

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

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

lemma *eetran-eqconf1*: $\Gamma \vdash ec1 \ -ee\rightarrow \ ec2 \Longrightarrow getspc\text{-}e \ ec1 = getspc\text{-}e \ ec2$
  proof −
    assume *a0*: $\Gamma \vdash ec1 \ -ee\rightarrow \ ec2$
    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*: $e1 = e2 \Longrightarrow \Gamma \vdash (e1, s1, x1) \ -ee\rightarrow \ (e2, s2, x2)$
  by (*simp add*: *eetran.intros*)

lemma *eqconf-eetran*: $getspc\text{-}e \ ec1 = getspc\text{-}e \ ec2 \Longrightarrow \Gamma \vdash ec1 \ -ee\rightarrow \ ec2$
  proof −
    assume $getspc\text{-}e \ ec1 = getspc\text{-}e \ ec2$
    then show *?thesis* **using** *getspc-e-def eetran.EnvE* **by** (*metis eq-fst-iff*)
  qed

lemma *cpts-ev-sub0*: $[\![el \in cpts\text{-}ev \ \Gamma; Suc \ 0 < length \ el]\!] \Longrightarrow drop \ (Suc \ 0) \ el \in cpts\text{-}ev \ \Gamma$
  apply(*rule cpts-ev.cases*)
  apply(*simp*)+
  done

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

```
        qed
      }
      then show ?thesis by auto
      qed
    with p0 p1 show ?thesis by auto
  qed

lemma notran-confeq0: ⟦el ∈ cpts-ev Γ; Suc 0 < length el; ¬ (∃ t. Γ ⊢ 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 Γ; Suc i < length el; ¬ (∃ t. Γ ⊢ el ! i −et−t→
el ! Suc i)⟧
                      ⟹ getspc-e (el ! i) = getspc-e (el ! (Suc i))
  proof −
    assume p0: el ∈ cpts-ev Γ and
           p1: Suc i < length el and
           p2: ¬ (∃ t. Γ ⊢ el ! i −et−t→ el ! Suc i)
    have ∀ el i. el ∈ cpts-ev Γ ∧  Suc i < length el ∧ ¬ (∃ t. Γ ⊢ 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 Γ ∧  Suc i < length el ∧ ¬ (∃ t. Γ ⊢ el ! i −et−t→
el ! Suc i)
        then have getspc-e (el ! i) = getspc-e (el ! (Suc i))
          proof(induct i)
            case 0 then show ?case
              using notran-confeq0 by (metis One-nat-def)
          next
            case (Suc j)
            let ?subel = drop (Suc j) el
            assume b0: el ∈ cpts-ev Γ ∧ Suc (Suc j) < length el ∧ ¬ (∃ t. Γ ⊢ el !
Suc j −et−t→ el ! Suc (Suc j))
         then have b1: ?subel ∈ cpts-ev Γ by (simp add: Suc-lessD b0 cpts-ev-subi)

            from b0 have b2: Suc 0 < length ?subel by auto
            from b0 have b3: ¬ (∃ t. Γ ⊢ ?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 Γ; length el > 0; Γ ⊢ el ! (length el − 1)
−et−t→ ec⟧ ⟹
                        el @ [ec] ∈ cpts-ev Γ
  proof −
    assume p0: el ∈ cpts-ev Γ
      and  p1: length el > 0
      and  p2: Γ ⊢ el ! (length el − 1) −et−t→ ec

     have ∀ el ec t Γ. el ∈ cpts-ev Γ ∧ length el > 0 ∧ Γ ⊢ el ! (length el − 1)
−et−t→ ec ⟶ el @ [ec] ∈ cpts-ev Γ
      proof −
      {
        fix el ec t Γ
        assume a0: el ∈ cpts-ev Γ
          and  a1: length el > 0
          and  a2: Γ ⊢ el ! (length el − 1) −et−t→ ec
        from a0 a1 a2 have el @ [ec] ∈ cpts-ev Γ
          proof(induct el)
            case (CptsEvOne e s x)
            assume b0: Γ ⊢ [(e, s, x)] ! (length [(e, s, x)] − 1) −et−t→ ec
            then have Γ ⊢ (e, s, x) −et−t→ ec by simp
          then show ?case by (metis append-Cons append-Nil cpts-ev.CptsEvComp

                cpts-ev.CptsEvOne surj-pair)
          next
            case (CptsEvEnv e s1 x xs s2 y)
            assume b0: (e, s1, x) # xs ∈ cpts-ev Γ
              and  b1: 0 < length ((e, s1, x) # xs) ⟹
                      Γ ⊢ ((e, s1, x) # xs) ! (length ((e, s1, x) # xs) − 1) −et−t→
ec
                       ⟹ ((e, s1, x) # xs) @ [ec] ∈ cpts-ev Γ
              and  b2: 0 < length ((e, s2, y) # (e, s1, x) # xs)
              and  b3: Γ ⊢ ((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 Γ ⊢ (e, s1, x)−et−t→ ec by simp
                with b1 c0 have ((e, s1, x) # xs) @ [ec] ∈ cpts-ev Γ by simp
                then show ?thesis by (simp add: cpts-ev.CptsEvEnv)
              next
                assume c0: xs ≠ []
```

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

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

lemma *cpts-ev-same*: $[\![$*length el* > *0*; $\forall$ *i*. *i* < *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*$]\!]$ $\Longrightarrow$ *el* $\in$ *cpts-ev* Γ
proof −
assume *p0*: *length el* > *0*
and *p1*: $\forall$ *i*. *i* < *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*
have $\forall$ *el es*. *length el* > *0* $\wedge$ ($\forall$ *i*. *i* < *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*) $\longrightarrow$ *el* $\in$ *cpts-ev* Γ
proof −
{
fix *el es*
assume *a0*: *length* (*el* :: ('*l*,'*k*,'*s*,'*prog*) *econfs*) > *0*
and *a1*: $\forall$ *i*. *i* < *length el* $\longrightarrow$ *getspc-e* (*el*!*i*) = *es*

then have *el* ∈ *cpts-ev* Γ
proof(*induct el*)
case *Nil* **show** *?case* **using** *Nil.prems(1)* **by** *auto*
next
case (*Cons a as*)
assume *b0*: *0* < *length as* ⟹ ∀ *i*<*length as*. *getspc-e* (*as* ! *i*) = *es* ⟹ *as* ∈ *cpts-ev* Γ
and *b1*: *0* < *length* (*a* # *as*)
and *b2*: ∀ *i*<*length* (*a* # *as*). *getspc-e* ((*a* # *as*) ! *i*) = *es*
then show *?case*
proof(*cases as* = [])
assume *c0*: *as* = []
then show *?thesis* **by** (*metis cpts-ev.CptsEvOne old.prod.exhaust*)
next
assume *c0*: ¬(*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-e* (*as* ! *i*) = *es* ⟹ *as* ∈ *cpts-ev* Γ **by** *simp*
with *b2* **have** *as* ∈ *cpts-ev* Γ **by** *force*
moreover from *b2* **have** *getspc-e a* = *es* **by** *auto*
moreover from *b2 c1* **have** *getspc-e b* = *es* **by** *auto*
ultimately show *?thesis* **using** *c1 getspc-e-def* **by** (*metis cpts-ev.CptsEvEnv fst-conv prod-cases3*)
qed
qed
}
then show *?thesis* **by** *auto*
qed

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

4.3.2 Event systems

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

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

lemma *esetran-eqconf1*: Γ ⊢ *esc1* −*ese*→ *esc2* ⟹ *getspc-es esc1* = *getspc-es esc2*
proof −
assume *a0*: Γ ⊢ *esc1* −*ese*→ *esc2*

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

lemma *eqconf-esetran1*: *es1* = *es2* $\Longrightarrow$ $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-ese\rightarrow$ (*es2*, *s2*, *x2*)
by (*simp add*: *esetran.intros*)

lemma *eqconf-esetran*: *getspc-es esc1* = *getspc-es esc2* $\Longrightarrow$ $\Gamma \vdash$ *esc1* $-ese\rightarrow$ *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*: $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-ese\rightarrow$ (*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* $\in$ *cpts-es* Γ; *es1* $\neq$ *es*$]\!]$ $\Longrightarrow$ ($\exists$ *est*. $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-es-est\rightarrow$ (*es*, *s*, *x*))
apply(*rule cpts-es.cases*)
apply(*simp*)+
by *auto*

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

lemma *cpts-es-dropi*: $[\![$*el* $\in$ *cpts-es* Γ; *Suc i* < *length el*$]\!]$ $\Longrightarrow$ *drop* (*Suc i*) *el* $\in$ *cpts-es* Γ
proof −
assume *p0*:*el* $\in$ *cpts-es* Γ **and** *p1*:*Suc i* < *length el*
have $\forall$ *el i*. *el* $\in$ *cpts-es* Γ $\wedge$ *Suc i* < *length el* $\longrightarrow$ *drop* (*Suc i*) *el* $\in$ *cpts-es* Γ
proof −
{
fix *el i*
have *el* $\in$ *cpts-es* Γ $\wedge$ *Suc i* < *length el* $\longrightarrow$ *drop* (*Suc i*) *el* $\in$ *cpts-es* Γ
proof(*induct i*)
case *0* **show** *?case* **by** (*simp add*: *cpts-es-drop0*)

```
        next
          case (Suc j)
           assume b0: el ∈ cpts-es Γ ∧ Suc j < length el ⟶ drop (Suc j) el ∈
cpts-es Γ
          show ?case
            proof
              assume c0: el ∈ cpts-es Γ ∧ Suc (Suc j) < length el
              with b0 have c1: drop (Suc j) el ∈ cpts-es Γ
                by (simp add: c0 Suc-lessD)
              then show drop (Suc (Suc j)) el ∈ cpts-es Γ
                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 Γ; i < length el⟧ ⟹ drop i el ∈ cpts-es Γ
 using cpts-es-dropi by (metis (no-types, hide-lams) drop-0 lessI less-Suc-eq-0-disj)


lemma cpts-es-take0: ⟦el ∈ cpts-es Γ; i < length el; el1 = take (Suc i) el; j <
length el1⟧
                    ⟹ drop (length el1 − Suc j) el1 ∈ cpts-es Γ
 proof −
  assume p0: el ∈ cpts-es Γ
    and p1: i < length el
    and p2: el1 = take (Suc i) el
    and p3: j < length el1
  have ∀ i j. el ∈ cpts-es Γ ∧ i < length el ∧ el1 = take (Suc i) el ∧ j < length
el1
        ⟶ drop (length el1 − Suc j) el1 ∈ cpts-es Γ
    proof −
    {
      fix i j
      assume a0: el ∈ cpts-es Γ
        and a1: i < length el
        and a2: el1 = take (Suc i) el
        and a3: j < length el1
      then have drop (length el1 − Suc j) el1 ∈ cpts-es Γ
        proof(induct j)
          case 0
          have drop (length el1 − Suc 0) el1 = [el ! i]
            by (simp add: a1 a2 take-Suc-conv-app-nth)
          then show ?case by (metis cpts-es.CptsEsOne old.prod.exhaust)
        next
```

case (*Suc jj*)
assume *b0*: *el* ∈ *cpts-es* Γ ⟹ *i* < *length el* ⟹ *el1* = *take* (*Suc i*) *el*
⟹ *jj* < *length el1* ⟹ *drop* (*length el1* − *Suc jj*) *el1* ∈ *cpts-es*
Γ
and *b1*: *el* ∈ *cpts-es* Γ
and *b2*: *i* < *length el*
and *b3*: *el1* = *take* (*Suc i*) *el*
and *b4*: *Suc jj* < *length el1*
then have *b5*: *drop* (*length el1* − *Suc jj*) *el1* ∈ *cpts-es* Γ
using *Suc-lessD* **by** *blast*
let *?el2* = *drop* (*Suc i*) *el*
from *a2* **have** *b6*: *el1* @ *?el2* = *el* **by** *simp*
let *?el1sht* = *drop* (*length el1* − *Suc jj*) *el1*
let *?el1lng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el1*
let *?elsht* = *drop* (*length el1* − *Suc jj*) *el*
let *?ellng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
from *b6* **have** *a7*: *?el1sht* @ *?el2* = *?elsht*
by (*metis diff-is-0-eq diff-le-self drop-0 drop-append*)
from *b6* **have** *a8*: *?el1lng* @ *?el2* = *?ellng*
by (*metis* (*no-types*, *lifting*) *a7 append-eq-append-conv diff-is-0-eq′*
diff-le-self drop-append)
have *a9*: *?ellng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?elsht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc Suc-leI a8*
append-is-Nil-conv b4 diff-diff-cancel drop-all length-drop
list.size(*3*) *not-less old.nat.distinct*(*2*))
from *b1 b4* **have** *a10*: *?elsht* ∈ *cpts-es* Γ
by (*metis a7 append-is-Nil-conv b5 cpts-es-dropi2 drop-all not-less*)
from *b1 b4* **have** *a11*: *?ellng* ∈ *cpts-es* Γ
by (*metis a9 cpts-es-dropi2 drop-all list.simps*(*3*) *not-less*)
have *a12*: *?el1lng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?el1sht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc*
b4 b6 diff-less gr-implies-not0 length-0-conv length-greater-0-conv
nth-append zero-less-Suc)
from *a11* **have** *?el1lng* ∈ *cpts-es* Γ
proof(*induct ?ellng*)
case *CptsEsOne* **show** *?case*
using *CptsEsOne.hyps a7 a9* **by** *auto*
next
case (*CptsEsEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-es* Γ
and *c1*: (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
⟹
drop (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-es* Γ
and *c2*: (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* −
Suc (*Suc jj*)) *el*
from *c0* **have** (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-es* Γ
by (*simp add*: *a11 c2*)
have *c3*: *?el1sht* ! *0* = (*es1*, *t1*, *x1*) **by** (*metis* (*no-types*, *lifting*)
Suc-leI Suc-lessD a7

a9 append-eq-Cons-conv b4 c2 diff-diff-cancel length-drop list.inject
list.size(3) nth-Cons-0 old.nat.distinct(2))
then have *c4*: ∃ *el1sht'*. *?el1sht* = (*es1*, *t1*, *x1*) # *el1sht'* **by** (*metis Cons-nth-drop-Suc b4*
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c5*: *?el1lng* = (*es1*, *s1*, *y1*) # *?el1sht* **using** *a12 a9 c2* **by** *auto*

with *b5 c4* **show** *?case* **using** *cpts-es.CptsEsEnv* **by** *fastforce*
next
case (*CptsEsComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *c0*: Γ ⊢ (*es1*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-es* Γ
and *c2*: (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
⟹ *drop* (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-es* Γ
and *c3*: (*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
have *c4*: *?el1sht* ! *0* = (*es2*, *t1*, *y1*) **by** (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD a7*
a9 append-eq-Cons-conv b4 c3 diff-diff-cancel length-drop list.inject
list.size(3) nth-Cons-0 old.nat.distinct(2))
then have *c5*: ∃ *el1sht'*. *?el1sht* = (*es2*, *t1*, *y1*) # *el1sht'* **by** (*metis Cons-nth-drop-Suc b4*
diff-diff-cancel drop-0 length-drop less-or-eq-imp-le zero-less-Suc)
have *c6*: *?el1lng* = (*es1*, *s1*, *x1*) # *?el1sht* **using** *a12 a9 c3* **by** *auto*
with *b5 c5* **show** *?case* **using** *c0 cpts-es.CptsEsComp* **by** *fastforce*
qed

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

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

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

and *p3*: $m < n$
then have *drop m el* $\in$ *cpts-es* Γ
using *cpts-es-dropi* **by** (*metis* (*no-types*, *lifting*) *drop-0 le-0-eq le-SucE less-le-trans zero-induct*)
then show *?thesis* **using** *cpts-es-take*
by (*metis* (*no-types*, *lifting*) *cpts-es-dropi2 drop-take inc-induct leD le-SucE length-take min.absorb2 p0 p1 p2 p3*)
qed

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

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

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

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

lemma *noevtent-inmid-eq*:
$(\neg\ (\exists\, j.\ j > 0 \wedge \mathit{Suc}\ j < \mathit{length}\ \mathit{esl} \wedge \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ j) = \mathit{EvtSys}\ \mathit{es} \wedge \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ \mathit{Suc}\ j) \neq \mathit{EvtSys}\ \mathit{es}))$
$= (\forall\, j.\ j > 0 \wedge \mathit{Suc}\ j < \mathit{length}\ \mathit{esl} \longrightarrow \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ j) = \mathit{EvtSys}\ \mathit{es} \longrightarrow \mathit{getspc\text{-}es}\ (\mathit{esl}\ !\ \mathit{Suc}\ j) = \mathit{EvtSys}\ \mathit{es})$
by *blast*

lemma *evtseq-next-in-cpts*:
$\mathit{esl}{\in}\mathit{cpts\text{-}es}\ \Gamma \Longrightarrow \forall\, i.\ \mathit{Suc}\ i < \mathit{length}\ \mathit{esl} \wedge \mathit{getspc\text{-}es}\ (\mathit{esl}!i) = \mathit{EvtSeq}\ e\ \mathit{esys}$
$\longrightarrow \mathit{getspc\text{-}es}\ (\mathit{esl}!\mathit{Suc}\ i) = \mathit{esys} \vee (\exists\, e.\ \mathit{getspc\text{-}es}\ (\mathit{esl}!\mathit{Suc}\ i) = \mathit{EvtSeq}\ e\ \mathit{esys})$
proof −
assume *p0*: $\mathit{esl}{\in}\mathit{cpts\text{-}es}\ \Gamma$
then show *?thesis*
proof −
{
fix *i*
assume *a0*: $\mathit{Suc}\ i < \mathit{length}\ \mathit{esl}$
and *a1*: $\mathit{getspc\text{-}es}\ (\mathit{esl}!i) = \mathit{EvtSeq}\ e\ \mathit{esys}$
let $?\mathit{esl1} = \mathit{drop}\ i\ \mathit{esl}$
from *p0 a0* **have** *a2*: $?\mathit{esl1}{\in}\mathit{cpts\text{-}es}\ \Gamma$ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **have** $\mathit{getspc\text{-}es}\ (?\mathit{esl1}!0) = \mathit{EvtSeq}\ e\ \mathit{esys}$ **by** *auto*
then obtain *s1* **and** *x1* **where** *a3*: $?\mathit{esl1}!0 = (\mathit{EvtSeq}\ e\ \mathit{esys}, s1, x1)$
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** $\mathit{getspc\text{-}es}\ (?\mathit{esl1}!1) = \mathit{esys} \vee (\exists\, e.\ \mathit{getspc\text{-}es}\ (?\mathit{esl1}!1) = \mathit{EvtSeq}\ e\ \mathit{esys})$
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*: Γ ⊢ (*es1′*, *s′*, *x′*) −*es*−*et′*→ (*es2′*, *t′*, *y′*)
and *b1*: (*es1′*, *s′*, *x′*) # (*es2′*, *t′*, *y′*) # *xs′* = *drop i esl*
and *b2*: *getspc-es* (*esl* ! *i*) = *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* Γ ⟹ ∀ *i*. *Suc i* < *length esl* ∧ *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
⟶ *getspc-es* (*esl*!*Suc i*) = *esys*
∨ (∃ *e*. *getspc-es* (*esl*!*Suc i*) = *EvtSeq e esys* ∧ *is-anonyevt e*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *Suc i* < *length esl*
and *a1*: *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
let *?esl1* = *drop i esl*

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

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

qed

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

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

```
          assume b0: Γ ⊢ (es1', s', x') −es−et'→ (es2', t', y')
            and  b1: (es1', s', x') # (es2', t', y') # xs' = drop i esl
            and  b2: getspc-es (esl ! i) = EvtSys es
          then have b3: es1' = EvtSys es
            by (metis Pair-inject a3 nth-Cons-0)
           from b0 b3 have ∃ e. es2' = EvtSeq e (EvtSys es) using evtsys-evtent
by simp
          then obtain e where es2' = EvtSeq e (EvtSys es) by auto
           with b0 b1 b3 have ∃ e. getspc-es (drop i esl ! 1) = EvtSeq e (EvtSys
es) ∧ is-anonyevt e
            using getspc-es-def  by (metis One-nat-def ent-spec2' evtsysent-evtent0

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


          then show ?case by simp
        qed

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

lemma evtsys-all-es-in-cpts:
  ⟦esl∈cpts-es Γ; 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 Γ
      and  p1: length esl > 0
      and  p2: getspc-es (esl!0) = EvtSys es
    show ?thesis
      proof −
      {
        fix i
        assume a0: i < length esl
         then have getspc-es (esl!i) = EvtSys es ∨ (∃ e. getspc-es (esl!i) = EvtSeq
e (EvtSys es))
        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** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

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

assume *c0*: $\exists$ *e*. *getspc-es* (*esl* ! *j*) = *EvtSeq e* (*EvtSys es*) $\wedge$ *is-anonyevt e*
with *p0 b1* **show** *?thesis* **using** *evtseq-next-in-cpts-anony* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *not-anonyevt-none-in-evtseq*:
$[\![$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSeq e es*,*s1*,*x1*)#(*es*,*s2*,*x2*)#*xs* $]\!] \Longrightarrow$ *e* $\neq$ *AnonyEvent fin-com*
apply(*rule cpts-es.cases*)
apply(*simp*)+
apply (*metis Suc-eq-plus1 add.commute add.right-neutral esys.size*(*3*) *le-add1 lessI not-le*)
apply(*rule estran.cases*)
apply(*simp*)+
apply (*metis Suc-eq-plus1 add.commute add.right-neutral esys.size*(*3*) *le-add1 lessI not-le*)
apply(*rule etran.cases*)
apply(*simp*)+
prefer *2*
apply(*simp*) **using** *ptran-not-none* **apply** *auto*[*1*]
done

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

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

lemma *fst-esys-snd-eseq-exist-evtent2*:
$[\![$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*$]\!] \Longrightarrow$
$\exists$ *e k*. $\Gamma \vdash$ (*EvtSys es*, *s*, *x*) $-es-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
apply(*rule cpts-es.cases*)

apply(*simp*)+
apply *blast*
by (*metis* (*no-types*, *hide-lams*) *cmd-enable-impl-notesys2 estran-impl-evtentorcmd*

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

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

from *a0* **have** *b1*: $2 \leq length\ esl$ **by** *fastforce*
moreover from *b0 b1* **have** $\exists\, s\ x.\ esl!0 = (EvtSys\ es,\ s,\ x)$ **using** *getspc-es-def*
by (*metis eq-fst-iff*)
moreover have $\exists\, ev\ s1\ x1.\ esl!1 = (EvtSeq\ ev\ (EvtSys\ es),\ s1, x1)$ **using** *getspc-es-def*
proof −
from *c1 a0 b0* **have** $\exists\, ev.\ getspc\text{-}es\ (esl!1) = EvtSeq\ ev\ (EvtSys\ es)$
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*:
$[\![esl \in cpts\text{-}es\ \Gamma;\ length\ esl \geq 2;\ \neg\ (\exists\, e\ k.\ \Gamma \vdash esl\ !\ 0\ -es-EvtEnt\ e\sharp k\rightarrow\ esl\ !\ 1)]\!]$
$\Longrightarrow \Gamma \vdash esl\ !\ 0\ -ese\rightarrow\ esl\ !\ 1 \vee (\exists\, c\ k.\ \Gamma \vdash esl\ !\ 0\ -es-Cmd\ c\sharp k\rightarrow\ esl\ !\ 1)$
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*:
$esl \in cpts\text{-}es\ \Gamma \Longrightarrow \forall\, i.\ Suc\ i < length\ esl \wedge (\exists\, e.\ getspc\text{-}es\ (esl!i) = EvtSeq\ e\ esys)$
$\wedge\ getspc\text{-}es\ (esl!Suc\ i) = EvtSeq\ (BasicEvent\ e)\ esys$

$\longrightarrow$ *getspc-es* (*esl*!*i*) = *EvtSeq* (*BasicEvent e*) *esys*

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

then obtain *s1* **and** *x1* **where** *a3*: *?esl1*!*1* = (*EvtSeq* (*BasicEvent e*) *esys*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a1* **have** *getspc-es* (*?esl1*!*0*) = *EvtSeq* (*BasicEvent e*) *esys*
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0*
le-add-diff-inverse2 length-Cons length-drop less-imp-le
list.size(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *drop i esl*
and *b1*: *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*BasicEvent e*) *esys*
then have *es′* = *EvtSeq* (*BasicEvent e*) *esys*
by (*metis One-nat-def a3 nth-Cons-0 nth-Cons-Suc prod.inject*)
with *b0* **show** *?case* **using** *getspc-es-def* **by** (*metis fst-conv nth-Cons-0*)

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

with *b3* **show** *?case* **by** *blast*
qed
}
then show *?thesis* **by** *auto*
qed
qed

lemma *incpts-es-impl-evnorcomptran*:
$esl \in cpts\text{-}es\ \Gamma \Longrightarrow \forall i.\ Suc\ i < length\ esl \longrightarrow \Gamma \vdash esl\ !\ i\ -ese\rightarrow esl\ !\ Suc\ i \vee (\exists et.\ \Gamma \vdash esl\ !\ i\ -es-et\rightarrow esl\ !\ Suc\ i)$
proof −
assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
{
fix *i*
assume *a0*: $Suc\ i < length\ esl$
let $?esl1 = take\ 2\ (drop\ i\ esl)$
from *a0 p0* **have** $take\ (Suc\ (Suc\ i) - i)\ (drop\ i\ esl) \in cpts\text{-}es\ \Gamma$
using *cpts-es-seg*[*of esl* Γ *i Suc* (*Suc i*)] **by** *simp*
then have $?esl1 \in cpts\text{-}es\ \Gamma$ **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 $(esc1, s1, x1)\#[(esc2, s2, x2)] \in cpts\text{-}es\ \Gamma$
by (*metis Cons-nth-drop-Suc Suc-lessD a0 numeral-2-eq-2 take-0 take-Suc-Cons*)
then have $\Gamma \vdash (esc1, s1, x1)\ -ese\rightarrow (esc2, s2, x2) \vee (\exists et.\ \Gamma \vdash (esc1, s1, x1)\ -es-et\rightarrow (esc2, s2, x2))$
apply(*rule cpts-es.cases*)
apply *simp+*
apply (*simp add*: *esetran.intros*)
by *auto*
with *a1 a2* **have** $\Gamma \vdash esl\ !\ i\ -ese\rightarrow esl\ !\ Suc\ i \vee (\exists et.\ \Gamma \vdash esl\ !\ i\ -es-et\rightarrow esl\ !\ Suc\ i)$ **by** *simp*
}
then show *?thesis* **by** *auto*
qed

lemma *incpts-es-eseq-not-evtent*:
$[\![esl \in cpts\text{-}es\ \Gamma;\ Suc\ i < length\ esl;\ \exists e\ esys.\ getspc\text{-}es\ (esl!i) = EvtSeq\ e\ esys \wedge is\text{-}anonyevt\ e]\!]$
$\Longrightarrow \neg(\exists e\ k.\ t = EvtEnt\ e \wedge \Gamma \vdash esl!i\ -es-t\sharp k\rightarrow esl!Suc\ i)$

proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *a0*: *Suc i* < *length esl*
and *a1*: ∃ *e esys*. *getspc-es* (*esl*!*i*) = *EvtSeq e esys* ∧ *is-anonyevt e*
let *?esl1* = *drop i esl*
from *p0 a0* **have** *a2*: *?esl1*∈*cpts-es* Γ **by** (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-lessD*
cpts-es-dropi diff-diff-cancel drop-0 length-drop length-greater-0-conv
less-or-eq-imp-le list.size(*3*))
from *a0 a1* **obtain** *e* **and** *esys* **where** *a3*: *getspc-es* (*?esl1*!*0*) = *EvtSeq e esys* **by** *auto*
then obtain *s1* **and** *x1* **where** *a4*: *?esl1*!*0* = (*EvtSeq e esys*,*s1*,*x1*)
using *getspc-es-def* **by** (*metis fst-conv old.prod.exhaust*)
from *a2 a3* **have** ¬(∃ *e k*. *t* = *EvtEnt e* ∧ Γ ⊢ *?esl1*!*0* −*es*−*t*♯*k*→ *?esl1*!*1*)
proof(*induct ?esl1*)
case (*CptsEsOne es′ s′ x′*)
then show *?case* **by** (*metis One-nat-def Suc-eq-plus1-left Suc-lessD a0*
le-add-diff-inverse2 length-Cons length-drop less-imp-le
list.size(*3*) *not-less-iff-gr-or-eq*)
next
case (*CptsEsEnv es′ t′ x′ xs′ s′ y′*)
assume *b0*: (*es′*, *s′*, *y′*) # (*es′*, *t′*, *x′*) # *xs′* = *?esl1*
and *b1*: *getspc-es* (*?esl1* ! *0*) = *EvtSeq e esys*
then have *es′* = *EvtSeq e esys*
by (*metis Pair-inject a4 nth-Cons-0*)
with *b0* **show** *?case* **using** *getspc-es-def*
by (*metis* (*mono-tags*, *lifting*) *a1 evtseq-no-evtent2 nth-Cons-0 nth-via-drop*)

next
case (*CptsEsComp es1′ s′ x′ et′ es2′ t′ y′ xs′*)
assume *b0*: Γ ⊢ (*es1′*, *s′*, *x′*) −*es*−*et′*→ (*es2′*, *t′*, *y′*)
and *b1*: (*es1′*, *s′*, *x′*) # (*es2′*, *t′*, *y′*) # *xs′* = *drop i esl*
and *b2*: *getspc-es* (*?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*:Γ ⊢ (*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 (*simp add*: *evtseq-ne-es*)

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

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

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

lemma *es-tran-not-etran2*: $\Gamma \vdash$ (*P*,*s*,*x*) $-es-pt\rightarrow$ (*Q*,*t*,*y*) $\Longrightarrow$ $\neg(\Gamma \vdash$ (*P*,*s*,*x*) $-ese\rightarrow$(*Q*,*t*,*y*))
by (*metis esetran.cases evtsys-not-eq-in-tran-aux*)

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

4.3.3 Parallel event systems

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

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

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

lemma *eqconf-pesetran1*: *es1* = *es2* $\Longrightarrow$ $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-pese\rightarrow$ (*es2*, *s2*, *x2*)

by (*simp add*: *pesetran.intros*)

lemma *eqconf-pesetran*: *getspc esc1* = *getspc esc2* $\Longrightarrow$ Γ ⊢ *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*: Γ ⊢ (*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*: ⟦Γ ⊢ *C1* −*pes*−*ct*→ *C2*; *C2#xs* ∈ *cpts-pes* Γ⟧ $\Longrightarrow$ *C1#C2#xs* ∈ *cpts-pes* Γ
proof −
assume *p0*: Γ ⊢ *C1* −*pes*−*ct*→ *C2*
and *p1*: *C2#xs* ∈ *cpts-pes* Γ
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* Γ; (Γ ⊢ *el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *el* ! (*length el* − *1*) −*pese*→ *ec*)⟧ $\Longrightarrow$
el @ [*ec*] ∈ *cpts-pes* Γ
proof −
assume *p0*: *el* ∈ *cpts-pes* Γ
and *p2*: (Γ ⊢ *el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *el* ! (*length el* − *1*) −*pese*→ *ec*)
from *p0* **have** *p1*: *el* ≠ []
using *cpts-pes.simps* **by** *blast*
have ∀ *el ec t*. *el* ∈ *cpts-pes* Γ ∧ ((Γ ⊢ *el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *el* ! (*length el* − *1*) −*pese*→ *ec*))
⟶ *el* @ [*ec*] ∈ *cpts-pes* Γ
proof −
{
fix *el ec t*
assume *a0*: *el* ∈ *cpts-pes* Γ
and *a2*: (Γ ⊢ *el* ! (*length el* − *1*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *el* ! (*length el* − *1*) −*pese*→ *ec*)
then have *a1*: *length el* > *0*

using *cpts-pes.simps* **by** *blast*
from *a0 a1 a2* **have** *el* @ [*ec*] ∈ *cpts-pes* Γ
proof(*induct el*)
case (*CptsPesOne e s x*)
assume *b0*: (Γ ⊢ [(*e*, *s*, *x*)] ! (*length* [(*e*, *s*, *x*)] − *1*) −*pes*−*t*→ *ec*)
∨ Γ ⊢ [(*e*, *s*, *x*)] ! (*length* [(*e*, *s*, *x*)] − *1*) −*pese*→ *ec*
then have (Γ ⊢ (*e*, *s*, *x*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ (*e*, *s*, *x*) −*pese*→ *ec*) **by** *simp*
then show *?case*
proof
assume Γ ⊢ (*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 Γ ⊢ (*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* Γ
and *b1*: *0* < *length* ((*e*, *s1*, *x*) # *xs*) ⟹
(Γ ⊢ ((*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s1*, *x*) # *xs*) − *1*) −*pes*−*t*→ *ec*) ∨
(Γ ⊢ ((*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s1*, *x*) # *xs*) − *1*) −*pese*→ *ec*) ⟹
((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ
and *b2*: *0* < *length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*)
and *b3*: (Γ ⊢ ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) − *1*) −*pes*−*t*→ *ec*) ∨
(Γ ⊢ ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) ! (*length* ((*e*, *s2*, *y*) # (*e*, *s1*, *x*) # *xs*) − *1*) −*pese*→ *ec*)
then show *?case*
proof(*cases xs* = [])
assume *c0*: *xs* = []
with *b3* **have** (Γ ⊢ (*e*, *s1*, *x*) −*pes*−*t*→ *ec*) ∨ (Γ ⊢ (*e*, *s1*, *x*) −*pese*→ *ec*) **by** *simp*
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ **by** *simp*
then show *?thesis* **by** (*simp add*: *cpts-pes.CptsPesEnv*)
next
assume *c0*: *xs* ≠ []
with *b3* **have** (Γ ⊢ *last xs* −*pes*−*t*→ *ec*) ∨ (Γ ⊢ *last xs* −*pese*→ *ec*) **by** (*simp add*: *last-conv-nth*)
with *b1 c0* **have** ((*e*, *s1*, *x*) # *xs*) @ [*ec*] ∈ *cpts-pes* Γ **using** *b3* **by** *auto*
then show *?thesis* **by** (*simp add*: *cpts-pes.CptsPesEnv*)
qed
next

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

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

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

lemma *pes-not-eq-in-tran* [*simp*]: ¬ Γ ⊢ (*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*: Γ ⊢ *pes1* −*pes*−*t*→ *pes2* ⟹ ¬(Γ ⊢ *pes1* −*pese*→*pes2*)
by (*metis pes-not-eq-in-tran pesetranE surj-pair*)

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

lemma *incpts-pes-impl-evnorcomptran*:
  $esl \in cpts\text{-}pes\ \Gamma \Longrightarrow \forall i.\ Suc\ i < length\ esl \longrightarrow \Gamma \vdash esl\ !\ i\ -pese\rightarrow\ esl\ !\ Suc\ i\ \vee$
$(\exists et.\ \Gamma \vdash esl\ !\ i\ -pes-et\rightarrow\ esl\ !\ Suc\ i)$
  proof −
    assume *p0*: $esl \in cpts\text{-}pes\ \Gamma$
    then show *?thesis*
      proof(*induct esl*)
        case (*CptsPesOne*) **show** *?case* **by** *simp*
      next
        case (*CptsPesEnv pes t x xs s y*)
        assume *a0*: $(pes,\ t,\ x)\ \#\ xs \in cpts\text{-}pes\ \Gamma$
          and *a1*: $\forall i.\ Suc\ i < length\ ((pes,\ t,\ x)\ \#\ xs) \longrightarrow$
                $\Gamma \vdash ((pes,\ t,\ x)\ \#\ xs)\ !\ i\ -pese\rightarrow\ ((pes,\ t,\ x)\ \#\ xs)\ !\ Suc\ i\ \vee$
                $(\exists et.\ \Gamma \vdash ((pes,\ t,\ x)\ \#\ xs)\ !\ i\ -pes-et\rightarrow\ ((pes,\ t,\ x)\ \#\ xs)\ !$
$Suc\ i)$
        then show *?case*
          proof −
          {
            fix *i*
            assume *b0*: $Suc\ i < length\ ((pes,\ s,\ y)\ \#\ (pes,\ t,\ x)\ \#\ xs)$
           have $\Gamma \vdash ((pes,\ s,\ y)\ \#\ (pes,\ t,\ x)\ \#\ xs)\ !\ i\ -pese\rightarrow\ ((pes,\ s,\ y)\ \#\ (pes,$
$t,\ x)\ \#\ xs)\ !\ Suc\ i\ \vee$
              $(\exists et.\ \Gamma \vdash ((pes,\ s,\ y)\ \#\ (pes,\ t,\ x)\ \#\ xs)\ !\ i\ -pes-et\rightarrow\ ((pes,\ s,$
$y)\ \#\ (pes,\ t,\ x)\ \#\ xs)\ !\ Suc\ i)$
            proof(*cases* $i = 0$)
              assume *c0*: $i = 0$
              then show *?thesis* **by** (*simp add*: *eqconf-pesetran1 nth-Cons'*)
            next
              assume *c0*: $i \neq 0$
              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*: $\Gamma \vdash (pes1,\ s,\ x)\ -pes-ct\rightarrow\ (pes2,\ t,\ y)$
          and *a1*: $(pes2,\ t,\ y)\ \#\ xs \in cpts\text{-}pes\ \Gamma$
          and *a2*: $\forall i.\ Suc\ i < length\ ((pes2,\ t,\ y)\ \#\ xs) \longrightarrow$
              $\Gamma \vdash ((pes2,\ t,\ y)\ \#\ xs)\ !\ i\ -pese\rightarrow\ ((pes2,\ t,\ y)\ \#\ xs)\ !\ Suc\ i\ \vee$
              $(\exists et.\ \Gamma \vdash ((pes2,\ t,\ y)\ \#\ xs)\ !\ i\ -pes-et\rightarrow\ ((pes2,\ t,\ y)\ \#\ xs)$
$!\ Suc\ i)$
        then show *?case*

```
        proof -
        {
          fix i
          assume b0: Suc i < length ((pes1, s, x) # (pes2, t, y) # xs)
          have Γ ⊢ ((pes1, s, x) # (pes2, t, y) # xs) ! i -pese→ ((pes1, s, x) #
(pes2, t, y) # xs) ! Suc i ∨
               (∃ et. Γ ⊢ ((pes1, s, x) # (pes2, t, y) # xs) ! i -pes-et→ ((pes1,
s, x) # (pes2, t, y) # xs) ! Suc i)
            proof(cases i = 0)
              assume c0: i = 0
              with a0 show ?thesis using nth-Cons-0 nth-Cons-Suc by auto
            next
              assume c0: i ≠ 0
              then have i > 0 by auto
              with a2 b0 show ?thesis using Suc-inject Suc-less-eq2 Suc-pred
                length-Cons nth-Cons-Suc by auto
            qed
        }
        then show ?thesis by auto
        qed
      qed
  qed

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

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

by (*simp add*: *c0 Suc-lessD*)
then show *drop* (*Suc* (*Suc j*)) *el* $\in$ *cpts-pes* Γ
using *c0 cpts-pes-drop0* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p0 p1* **show** *?thesis* **by** *auto*
qed

lemma *cpts-pes-take0*: $[\![$*el* $\in$ *cpts-pes* Γ; *i* $<$ *length el*; *el1* $=$ *take* (*Suc i*) *el*; *j* $<$ *length el1*$]\!]$
$\Longrightarrow$ *drop* (*length el1* $-$ *Suc j*) *el1* $\in$ *cpts-pes* Γ
proof $-$
assume *p0*: *el* $\in$ *cpts-pes* Γ
and *p1*: *i* $<$ *length el*
and *p2*: *el1* $=$ *take* (*Suc i*) *el*
and *p3*: *j* $<$ *length el1*
have $\forall$ *i j*. *el* $\in$ *cpts-pes* Γ $\wedge$ *i* $<$ *length el* $\wedge$ *el1* $=$ *take* (*Suc i*) *el* $\wedge$ *j* $<$ *length el1*
$\longrightarrow$ *drop* (*length el1* $-$ *Suc j*) *el1* $\in$ *cpts-pes* Γ
proof $-$
{
fix *i j*
assume *a0*: *el* $\in$ *cpts-pes* Γ
and *a1*: *i* $<$ *length el*
and *a2*: *el1* $=$ *take* (*Suc i*) *el*
and *a3*: *j* $<$ *length el1*
then have *drop* (*length el1* $-$ *Suc j*) *el1* $\in$ *cpts-pes* Γ
proof(*induct j*)
case *0*
have *drop* (*length el1* $-$ *Suc 0*) *el1* $=$ [*el* ! *i*]
by (*simp add*: *a1 a2 take-Suc-conv-app-nth*)
then show *?case* **by** (*metis cpts-pes.CptsPesOne old.prod.exhaust*)
next
case (*Suc jj*)
assume *b0*: *el* $\in$ *cpts-pes* Γ $\Longrightarrow$ *i* $<$ *length el* $\Longrightarrow$ *el1* $=$ *take* (*Suc i*) *el*
$\Longrightarrow$ *jj* $<$ *length el1* $\Longrightarrow$ *drop* (*length el1* $-$ *Suc jj*) *el1* $\in$ *cpts-pes* Γ
and *b1*: *el* $\in$ *cpts-pes* Γ
and *b2*: *i* $<$ *length el*
and *b3*: *el1* $=$ *take* (*Suc i*) *el*
and *b4*: *Suc jj* $<$ *length el1*
then have *b5*: *drop* (*length el1* $-$ *Suc jj*) *el1* $\in$ *cpts-pes* Γ
using *Suc-lessD* **by** *blast*
let *?el2* $=$ *drop* (*Suc i*) *el*
from *a2* **have** *b6*: *el1* @ *?el2* $=$ *el* **by** *simp*
let *?el1sht* $=$ *drop* (*length el1* $-$ *Suc jj*) *el1*

let *?el1lng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el1*
let *?elsht* = *drop* (*length el1* − *Suc jj*) *el*
let *?ellng* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
from *b6* **have** *a7*: *?el1sht* @ *?el2* = *?elsht*
by (*metis diff-is-0-eq diff-le-self drop-0 drop-append*)
from *b6* **have** *a8*: *?el1lng* @ *?el2* = *?ellng*
by (*metis* (*no-types*, *lifting*) *a7 append-eq-append-conv diff-is-0-eq′ diff-le-self drop-append*)
have *a9*: *?ellng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?elsht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc Suc-leI a8 append-is-Nil-conv b4 diff-diff-cancel drop-all length-drop list.size*(*3*) *not-less old.nat.distinct*(*2*))
from *b1 b4* **have** *a10*: *?elsht* ∈ *cpts-pes* Γ
by (*metis Suc-diff-Suc a7 append-is-Nil-conv b5 cpts-pes-dropi drop-all not-less*)
from *b1 b4* **have** *a11*: *?ellng* ∈ *cpts-pes* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc a9 cpts-pes-dropi diff-is-0-eq drop-0 drop-all leI list.simps*(*3*))
have *a12*: *?el1lng* = (*el* ! (*length el1* − *Suc* (*Suc jj*))) # *?el1sht*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Suc-diff-Suc b4 b6 diff-less gr-implies-not0 length-0-conv length-greater-0-conv nth-append zero-less-Suc*)
from *a11* **have** *?el1lng* ∈ *cpts-pes* Γ
proof(*induct ?ellng*)
case *CptsPesOne* **show** *?case*
using *CptsPesOne.hyps a7 a9* **by** *auto*
next
case (*CptsPesEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-pes* Γ
and *c1*: (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el* ⟹ *drop* (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-pes* Γ
and *c2*: (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
from *c0* **have** (*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1* ∈ *cpts-pes* Γ
by (*simp add*: *a11 c2*)
have *c3*: *?el1sht* ! *0* = (*es1*, *t1*, *x1*) **by** (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD a7 a9 append-eq-Cons-conv b4 c2 diff-diff-cancel length-drop list.inject list.size*(*3*) *nth-Cons-0 old.nat.distinct*(*2*))
then have *c4*: ∃ *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*: Γ ⊢ (*es1*, *s1*, *x1*) −*pes*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-pes* Γ
and *c2*: (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
⟹ *drop* (*length el1* − *Suc* (*Suc jj*)) *el1* ∈ *cpts-pes* Γ
and *c3*: (*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1* = *drop* (*length el1* − *Suc* (*Suc jj*)) *el*
have *c4*: *?el1sht* ! *0* = (*es2*, *t1*, *y1*) **by** (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD a7* *a9 append-eq-Cons-conv b4 c3 diff-diff-cancel length-drop list.inject* *list.size*(*3*) *nth-Cons-0 old.nat.distinct*(*2*))
then have *c5*: ∃ *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* Γ
using *p0 p1 p2 p3* **by** *blast*
qed

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

lemma *cpts-pes-seg*: ⟦*el* ∈ *cpts-pes* Γ; *m* ≤ *length el*; *n* ≤ *length el*; *m* < *n*⟧
⟹ *take* (*n* − *m*) (*drop m el*) ∈ *cpts-pes* Γ
proof −
assume *p0*: *el* ∈ *cpts-pes* Γ
and *p1*: *m* ≤ *length el*
and *p2*: *n* ≤ *length el*
and *p3*: *m* < *n*
then have *drop m el* ∈ *cpts-pes* Γ
using *cpts-pes-dropi* **by** (*metis* (*no-types*, *lifting*) *drop-0 le-0-eq le-SucE less-le-trans zero-induct*)
then show *?thesis* **using** *cpts-pes-take*
by (*smt Suc-diff-Suc diff-diff-cancel diff-less-Suc diff-right-commute length-drop less-le-trans p2 p3*)
qed

lemma *cpts-pes-seg2*: ⟦*el* ∈ *cpts-pes* Γ; *m* ≤ *length el*; *n* ≤ *length el*; *take* (*n* −

$m)$ $(drop\ m\ el) \neq []$]
$\Longrightarrow take\ (n - m)\ (drop\ m\ el) \in cpts\text{-}pes\ \Gamma$

proof −
  assume *p0*: $el \in cpts\text{-}pes\ \Gamma$
    and *p1*: $m \leq length\ el$
    and *p2*: $n \leq length\ el$
    and *p3*: $take\ (n - m)\ (drop\ m\ el) \neq []$
  from *p3* **have** $m < n$ **by** *simp*
  then show *?thesis* **using** *cpts-pes-seg* **using** *p0 p1 p2* **by** *blast*
qed

4.4 Compositionality of the Semantics

4.4.1 Definition of the conjoin operator

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

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

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

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

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

4.4.2 Lemmas of conjoin

lemma *acts-in-conjoin-cpts*: $\Gamma\ c \propto cs \Longrightarrow \forall i.\ Suc\ i < length\ (cs\ k) \longrightarrow \Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ i)$
$\vee (\exists e.\ \Gamma \vdash ((cs\ k)!i)\ -es-(EvtEnt\ e\sharp k)\rightarrow\ ((cs\ k)!\ Suc\ i))$
$\vee (\exists c.\ \Gamma \vdash ((cs\ k)!i)\ -es-(Cmd\ c\sharp k)\rightarrow\ ((cs\ k)!\ Suc\ i))$

proof −
assume *p0*: $\Gamma\ c \propto cs$
{
fix *i*
assume *a0*: *Suc i* $<$ *length* (*cs k*)
from *p0* **have** *a1*: *length c* = *length* (*cs k*) **by** (*simp add*:*conjoin-def same-length-def*)
from *p0* **have** *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
with *a0 a1* **have** (∃ *t k*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ∧
(∀ *k t*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ⟶ (Γ ⊢ *cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) ∧
(∀ *k′*. *k′* ≠ *k* ⟶ (Γ ⊢ *cs k′*!*i* −*ese*→ *cs k′*! *Suc i*))))
∨
((Γ ⊢ (*c*!*i*) −*pese*→ (*c*!*Suc i*)) ∧ (∀ *k*. (Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*))))
by (*simp add*: *compat-tran-def*)
then have Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*)
∨ (∃ *e*. Γ ⊢ ((*cs k*)!*i*) −*es*−(*EvtEnt e*♯*k*)→ ((*cs k*)! *Suc i*))
∨ (∃ *c*. Γ ⊢ ((*cs k*)!*i*) −*es*−(*Cmd c*♯*k*)→ ((*cs k*)! *Suc i*))
proof
assume *b0*: ∃ *t k*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ∧
(∀ *k t*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ⟶ (Γ ⊢ *cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) ∧
(∀ *k′*. *k′* ≠ *k* ⟶ (Γ ⊢ *cs k′*!*i* −*ese*→ *cs k′*! *Suc i*)))
then obtain *t* **and** *k1* **where** *b1*: (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k1*)→ *c*!*Suc i*) ∧
(∀ *k t*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ⟶ (Γ ⊢ *cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) ∧
(∀ *k′*. *k′* ≠ *k* ⟶ (Γ ⊢ *cs k′*!*i* −*ese*→ *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* ≠ *k1*
with *b1* **show** *?thesis* **by** *auto*
qed
next
assume *b0*: (Γ ⊢ (*c*!*i*) −*pese*→ (*c*!*Suc i*)) ∧ (∀ *k*. (Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*)))
then show *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *simp*
qed

lemma *entevt-in-conjoin-cpts*:
⟦Γ *c* ∝ *cs*; *Suc i* < *length* (*cs k*); *getspc-es* ((*cs k*)!*i*) = *EvtSys es*;
getspc-es ((*cs k*)!*Suc i*) ≠ *EvtSys es* ⟧

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

lemma *notentevt-in-conjoin-cpts*:
$[\![\Gamma\ c \propto cs;\ Suc\ i < length\ (cs\ k);\ \neg(getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es \wedge getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es);$
$\forall\, i < length\ (cs\ k).\ getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSys\ es$
$\vee (\exists\, e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))]\!]$
$\Longrightarrow \neg(\exists\, e.\ \Gamma \vdash ((cs\ k)!i)\ {-es-}(EvtEnt\ e\sharp k){\rightarrow}\ ((cs\ k)!\ Suc\ i))$
proof −
assume *p0*: $\Gamma\ c \propto cs$
and *p1*: $Suc\ i < length\ (cs\ k)$
and *p2*: $\neg(getspc\text{-}es\ ((cs\ k)!i) = EvtSys\ es \wedge getspc\text{-}es\ ((cs\ k)!Suc\ i) \neq EvtSys\ es)$
and *p3*: $\forall\, i < length\ (cs\ k).\ getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSys\ es$
$\vee (\exists\, e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))$
from *p2* **have** $getspc\text{-}es\ ((cs\ k)!i) \neq EvtSys\ es \vee getspc\text{-}es\ ((cs\ k)!Suc\ i) = EvtSys\ es$ **by** *simp*
with *p3* **have** $(\exists\, e.\ is\text{-}anonyevt\ e \wedge getspc\text{-}es\ ((cs\ k)\ !\ i) = EvtSeq\ e\ (EvtSys\ es))$
$\vee\ getspc\text{-}es\ ((cs\ k)!Suc\ i) = EvtSys\ es$

using *Suc-lessD p1* **by** *blast*
then show *?thesis*
proof
assume $\exists$ *e. is-anonyevt e* $\wedge$ *getspc-es* ((*cs k*) ! *i*) = *EvtSeq e* (*EvtSys es*)
then obtain *e1* **where** *is-anonyevt e1* $\wedge$ *getspc-es* ((*cs k*) ! *i*) = *EvtSeq e1* (*EvtSys es*) **by** *auto*
then show *?thesis* **using** *evtent-is-basicevt-inevtseq2* **by** *fastforce*
next
assume *getspc-es* ((*cs k*)!*Suc i*) = *EvtSys es*
then show *?thesis* **by** (*metis Suc-lessD evtseq-no-evtent2 evtsys-not-eq-in-tran-aux1 p1 p3*)
qed
qed

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

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

fix *k j*
assume *b0*: $j < length\ c1$
from *p1 p3 a1* **have** *b1*: $cs1\ k = take\ n\ (cs\ k)$ **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < length\ c$
$\longrightarrow (getspc\ (c!j))\ k = getspc\text{-}es\ ((cs\ k)\ !\ j)$
by (*simp add*:*conjoin-def same-spec-def*)
from *p2 b1 b0* **have** $getspc\ (c1!j) = getspc\ (c!j)$
$\wedge\ getspc\text{-}es\ ((cs\ k)\ !\ j) = getspc\text{-}es\ ((cs1\ k)\ !\ j)$
by (*simp add*: *nth-append*)
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* Γ *c1 cs1*
proof −
{
fix *j*
assume *b0*: $Suc\ j < length\ c1$
with *p0 p2* **have** $((\exists\ t\ k.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow\ c!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!j\ -pes-(t\sharp k)\rightarrow\ c!Suc\ j) \longrightarrow (\Gamma \vdash cs\ k!j$
$-es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ -ese\rightarrow\ cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ -pese\rightarrow\ (c!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!j)\ -ese\rightarrow$
$((cs\ k)!\ Suc\ j))))$
by (*simp add*:*conjoin-def compat-tran-def*)
moreover
from *p2 b0* **have** $c!j = c1!j$ **by** *simp*
moreover
from *p2 b0* **have** $c!Suc\ j = c1!Suc\ j$ **by** *simp*
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!j = cs\ k!j$
by (*simp add*: *Suc-lessD*)
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!Suc\ j = cs\ k!Suc\ j$
by (*simp add*: *Suc-lessD*)
ultimately
have $((\exists\ t\ k.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j) \longrightarrow (\Gamma \vdash cs1\ k!j$
$-es-(t\sharp k)\rightarrow\ cs1\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs1\ k'!j\ -ese\rightarrow\ cs1\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c1!j)\ -pese\rightarrow\ (c1!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs1\ k)!j)\ -ese\rightarrow$
$((cs1\ k)!\ Suc\ j))))$ **by** *simp*
}
then show *?thesis* **by** (*simp add*:*compat-tran-def*)
qed

ultimately show *?thesis* **by** (*simp add*:*conjoin-def a0*)
qed

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

have *same-state c1 cs1*
proof −
{
fix *k j*
assume *b0*: $j < length\ c1$
from *p1 p3 a1* **have** *b1*: $cs1\ k = drop\ n\ (cs\ k)$ **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < length\ c$
$\longrightarrow gets\ (c!j) = gets\text{-}es\ ((cs\ k)!j) \wedge getx\ (c!j) = getx\text{-}es\ ((cs\ k)!j)$
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 (*smt a1 add-diff-cancel-left′ drop-eq-Nil length-drop less-diff-conv2 list.size*(*3*)
nat-le-linear nat-neq-iff not-less-zero nth-drop order.asym semiring-normalization-rules(*24*))

}
then show *?thesis* **by** (*simp add*:*same-state-def*)
qed
moreover
have *same-spec c1 cs1*
proof −
{

fix *k j*
assume *b0*: $j < length\ c1$
from *p1 p3 a1* **have** *b1*: $cs1\ k = drop\ n\ (cs\ k)$ **by** *simp*
from *p0* **have** *b2*[*rule-format*]: $\forall k\ j.\ j < length\ c$
$\longrightarrow (getspc\ (c!j))\ k = getspc\text{-}es\ ((cs\ k)\ !\ j)$
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* Γ *c1 cs1*
proof −
{
fix *j*
assume *b0*: $Suc\ j < length\ c1$
with *p0 p2* **have** $((\exists t\ k.\ (\Gamma \vdash c!(n+j)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (n+j)))\ \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!(n+j)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (n+j)) \longrightarrow (\Gamma \vdash cs$
$k!(n+j)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (n+j))\ \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(n+j)\ -ese\rightarrow\ cs\ k'!\ Suc$
$(n+j)))))$
$\vee$
$((\Gamma \vdash (c!(n+j))\ -pese\rightarrow\ (c!Suc\ (n+j)))\ \wedge\ (\forall k.\ (\Gamma \vdash ((cs$
$k)!(n+j))\ -ese\rightarrow\ ((cs\ k)!\ Suc\ (n+j)))))$
by (*simp add:conjoin-def compat-tran-def*)
moreover
from *p2 b0* **have** $c!(n+j) = c1!j$ **by** *simp*
moreover
from *p2 b0* **have** $c!Suc\ (n+j) = c1!Suc\ j$ **by** *simp*
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!j = cs\ k!(n+j)$
by (*metis drop-eq-Nil length-greater-0-conv less-imp-Suc-add linear nth-drop zero-less-Suc*)
moreover
from *p1 p2 p3 a1 b0* **have** $\forall k.\ cs1\ k!Suc\ j = cs\ k!Suc\ (n+j)$
by (*smt Suc-lessE add-Suc-right drop-eq-Nil length-greater-0-conv linear nth-drop zero-less-Suc*)
ultimately

have $((\exists\, t\ k.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c1!j\ -pes-(t\sharp k)\rightarrow\ c1!Suc\ j) \longrightarrow (\Gamma \vdash cs1\ k!j$
$-es-(t\sharp k)\rightarrow\ cs1\ k!\ Suc\ j) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs1\ k'!j\ -ese\rightarrow\ cs1\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c1!j)\ -pese\rightarrow\ (c1!Suc\ j)) \wedge (\forall\, k.\ (\Gamma \vdash ((cs1\ k)!j)\ -ese\rightarrow$
$((cs1\ k)!\ Suc\ j))))$ **by** *simp*
}
then show *?thesis* **by** (*simp add*:*compat-tran-def*)
qed
ultimately show *?thesis* **by** (*simp add*:*conjoin-def a0*)
qed

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

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

let *?cs* = (λ*k*. *drop 1* (*cs k*))
from *b1* **have** Γ *?esl* ∝ *?cs* **using** *drop-n-conjoin*[*of* Γ *?esllon cs 1 ?esl ?cs*] **by** *auto*
with *a0 a1*[*of ?cs*] **have** *b2*: *?cs k* ∈ *cpts-es* Γ **by** *simp*
from *b1* **have** *b3*: *cs k* ! *0* = (*pes k*, *s*, *y*)
using *conjoin-def*[*of* Γ *?esllon cs*] *same-state-def*[*of ?esllon cs*] *same-spec-def*[*of ?esllon cs*]
by (*metis esconf-trip gets-def getspc-def getx-def length-greater-0-conv list.simps*(*3*) *nth-Cons-0 prod.sel*(*1*) *prod.sel*(*2*))

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

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

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

from *b1* **have** *compat-tran* Γ *?esllon cs* **by** (*simp add*:*conjoin-def*)
then have $((\exists t\ k.\ (\Gamma \vdash ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0) \longrightarrow (\Gamma \vdash cs\ k!0\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ 0) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!0\ -ese\rightarrow\ cs\ k'!\ Suc\ 0))))$
$\vee$
$((\Gamma \vdash (?esllon!0)\ -pese\rightarrow\ (?esllon!Suc\ 0)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!0)\ -ese\rightarrow\ ((cs\ k)!\ Suc\ 0))))$
using *compat-tran-def*[*of* Γ *?esllon cs*] **by** *fastforce*
then have *cs k* ∈ *cpts-es* Γ
proof
assume *c0*: $(\exists t\ k.\ (\Gamma \vdash ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash ?esllon!0\ -pes-(t\sharp k)\rightarrow\ ?esllon!Suc\ 0) \longrightarrow (\Gamma \vdash cs\ k!0\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ 0) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!0\ -ese\rightarrow\ cs\ k'!\ Suc\ 0)))$
then obtain *t1* **and** *k1* **where** *c1*: $(\Gamma \vdash ?esllon!0\ -pes-(t1\sharp k1)\rightarrow\ ?esllon!Suc\ 0)$ **by** *auto*
with *c0* **have** *c2*: $(\Gamma \vdash cs\ k1!0\ -es-(t1\sharp k1)\rightarrow\ cs\ k1!\ Suc\ 0) \wedge$
$(\forall k'.\ k' \neq k1 \longrightarrow (\Gamma \vdash cs\ k'!0\ -ese\rightarrow\ cs\ k'!\ Suc\ 0))$ **by** *auto*
show *?thesis*
proof(*cases k* = *k1*)
assume *d0*: *k* = *k1*
with *c2* **have** $(\Gamma \vdash cs\ k!0\ -es-(t1\sharp k)\rightarrow\ cs\ k!\ Suc\ 0)$ **by** *auto*
with *b2 b3 b4* **show** *?thesis* **using** *CptsEsComp*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD cpts-es-not-empty drop-0 drop-eq-Nil not-le*)
next
assume *d0*: *k* ≠ *k1*
with *c2* **have** $\Gamma \vdash cs\ k!0\ -ese\rightarrow\ cs\ k!\ Suc\ 0$ **by** *auto*
with *b2 b3 b4* **show** *?thesis* **using** *CptsEsEnv*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-lessD cpts-es-not-empty*

drop-0 drop-eq-Nil esetran-eqconf not-le)

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

lemma *conjoin-imp-cptses-k*:
$[\![c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ \Gamma\ c \propto cs]\!]$
$\Longrightarrow cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
proof −
assume *p0*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p1*: $\Gamma\ c \propto cs$
from *p0* **have** *a1*: $c \in cpts\text{-}pes\ \Gamma \wedge c!0 = (pes,s,x)$ **by** (*simp add*:*cpts-of-pes-def*)
from *a1 p1* **have** $cs\ k \in cpts\text{-}es\ \Gamma$ **using** *conjoin-imp-cptses-k-help* **by** *auto*
moreover
from *p0 p1* **have** $cs\ k\ !\ 0 = (pes\ k,s,x)$
by (*metis a1 conjoin-def cpts-pes-not-empty esconf-trip fst-conv gets-def getspc-def getx-def length-greater-0-conv same-spec-def same-state-def snd-conv*)

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

4.4.3 Semantics is Compositional

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

from *p1* **obtain** *k* **and** *p* **where** $pes\ k = p$ **by** *auto*
with *a0* **have** *a2*: $(cs\ k)!0=(pes\ k,s,x) \wedge (cs\ k) \in cpts\text{-}es\ \Gamma$ **by** *auto*
then have $(cs\ k) \neq []$ **by** *auto*
moreover

from *a0* **have** *same-length c cs* **by** (*simp add*:*conjoin-def*)
ultimately have *a3*: $c \neq []$ **using** *same-length-def* **by** *force*

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

from *a0* **have** *same-state c cs* **by** (*simp add*:*conjoin-def*)
with *a3* **have** *gets* (*c*!*0*) = *gets-es* ((*cs k*)!*0*) $\wedge$ *getx* (*c*!*0*) = *getx-es* ((*cs k*)!*0*)
by (*simp add*:*same-state-def*)
with *a2* **have** *gets* (*c*!*0*) = *s* $\wedge$ *getx* (*c*!*0*) = *x*
by (*simp add*:*gets-def getx-def gets-es-def getx-es-def*)
with *b3* **show** *?thesis* **using** *gets-def getx-def getspc-def* **by** (*metis prod.collapse*)
qed
have $\forall i.$ $i > 0 \wedge i \leq$ *length c* $\longrightarrow$ *take i c* $\in$ *cpts-pes* Γ
proof −
{
fix *i*
assume *b0*: $i > 0 \wedge i \leq$ *length c*
then have *take i c* $\in$ *cpts-pes* Γ
proof(*induct i*)
case *0* **show** *?case* **using** *0.prems* **by** *auto*
next
case (*Suc j*)
assume *c0*: $0 < j \wedge j \leq$ *length c* $\Longrightarrow$ *take j c* $\in$ *cpts-pes* Γ
and *c1*: $0 <$ *Suc j* $\wedge$ *Suc j* $\leq$ *length c*
show *?case*
proof(*cases* $j = 0$)
assume *d0*: $j = 0$
with *c0* **show** *?case* **by** (*simp add*: *a3 cpts-pes.CptsPesOne g0 hd-conv-nth take-Suc*)
next
assume *d0*: $j \neq 0$
from *a0* **have** *d1*: *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
then have *d2*: $\forall j.$ *Suc j* < *length c* $\longrightarrow$
($\exists t\ k.$ ($\Gamma \vdash$ *c*!*j* −*pes*−(*t*♯*k*)→ *c*!*Suc j*) $\wedge$
($\forall k\ t.$ ($\Gamma \vdash$ *c*!*j* −*pes*−(*t*♯*k*)→ *c*!*Suc j*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*j* −*es*−(*t*♯*k*)→ *cs k*! *Suc j*) $\wedge$
($\forall k'.$ $k' \neq k$ $\longrightarrow$ ($\Gamma \vdash$ *cs k'*!*j* −*ese*→ *cs k'*! *Suc j*))))
$\vee$
(($\Gamma \vdash$ (*c*!*j*) −*pese*→ (*c*!*Suc j*)) $\wedge$ ($\forall k.$ ($\Gamma \vdash$ ((*cs k*)!*j*) −*ese*→ ((*cs k*)! *Suc j*))))
by (*simp add*:*compat-tran-def*)

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

with *d3* **have** *d4*: $(\exists\, t\ k.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \longrightarrow (\Gamma \vdash$
$cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (j-1)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(j-1)\ -ese\rightarrow\ cs\ k'!$
$Suc\ (j-1)))))$
$\vee$
$((\Gamma \vdash (c!(j-1))\ -pese\rightarrow\ (c!Suc\ (j-1))) \wedge (\forall\, k.\ (\Gamma \vdash ((cs$
$k)!(j-1))\ -ese\rightarrow\ ((cs\ k)!Suc\ (j-1)))))$
using *d2* **by** *auto*
from *c0 c1 d0* **have** *d5*: *take j c* $\in$ *cpts-pes* Γ **by** *auto*
from *d4* **show** *?case*
proof
assume $(\exists\, t\ k.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \wedge$
$(\forall\, k\ t.\ (\Gamma \vdash c!(j-1)\ -pes-(t\sharp k)\rightarrow\ c!Suc\ (j-1)) \longrightarrow (\Gamma \vdash$
$cs\ k!(j-1)\ -es-(t\sharp k)\rightarrow\ cs\ k!\ Suc\ (j-1)) \wedge$
$(\forall\, k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!(j-1)\ -ese\rightarrow\ cs\ k'!$
$Suc\ (j-1)))))$
then obtain *t* **and** *k* **where** *e0*: $(\Gamma \vdash (c!(j-1))\ -pes-(t\sharp k)\rightarrow$
$(c!Suc\ (j-1)))$ **by** *auto*
then have $\Gamma \vdash ((take\ j\ c)\ !\ (length\ (take\ j\ c)\ -\ 1))\ -pes-(t\sharp k)\rightarrow$
$(c!Suc\ (j-1))$
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leD Suc-lessD*
d6 butlast-take c1 d0 length-butlast neq0-conv nth-append-length
take-Suc-conv-app-nth)
with *d5* **have** (*take j c*) @ [$c!Suc\ (j-1)$] $\in$ *cpts-pes* Γ **using**
cpts-pes-onemore **by** *blast*
then show *?thesis* **using** *d0 d6 take-Suc-conv-app-nth* **by** *fastforce*
next
assume $(\Gamma \vdash (c!(j-1))\ -pese\rightarrow\ (c!Suc\ (j-1))) \wedge (\forall\, k.\ (\Gamma \vdash ((cs$
$k)!(j-1))\ -ese\rightarrow\ ((cs\ k)!Suc\ (j-1))))$
then have $\Gamma \vdash ((take\ j\ c)\ !\ (length\ (take\ j\ c)\ -\ 1))\ -pese\rightarrow$
$(c!Suc\ (j-1))$
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-leD Suc-lessD*
d6 butlast-take c1 d0 length-butlast neq0-conv nth-append-length
take-Suc-conv-app-nth)
with *d5* **have** (*take j c*) @ [$c!Suc\ (j-1)$] $\in$ *cpts-pes* Γ **using**
cpts-pes-onemore **by** *blast*
then show *?thesis* **using** *d0 d6 take-Suc-conv-app-nth* **by** *fastforce*
qed

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

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

lemma *comp-tran-env*: $[\![(\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ t1\ x1);\ c = (pes,\ t1,\ x1)$ $\#\ xs;\ c{\in}cpts\text{-}pes\ \Gamma;$
$\Gamma\ c \propto cs;\ c' = (pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs]\!] \Longrightarrow$
$compat\text{-}tran\ \Gamma\ c'\ (\lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k)$
proof −
let $?cs' = \lambda k.\ (pes\ k,\ s1,\ y1)\ \#\ cs\ k$
assume *p0*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ t1\ x1$
and *p1*: $c \in cpts\text{-}pes\ \Gamma$
and *p2*: $\Gamma\ c \propto cs$
and *p3*: $c' = (pes,\ s1,\ y1)\ \#\ (pes,\ t1,\ x1)\ \#\ xs$
and *p4*: $c = (pes,\ t1,\ x1)\ \#\ xs$
from *p0* **have** *b3*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma \wedge (cs\ k)!0 = (pes\ k,t1,x1)$ **by** (*simp add*:*cpts-of-es-def*)
show *compat-tran* $\Gamma\ c'\ ?cs'$
proof −
{
fix *j*
assume *dd0*: $Suc\ j < length\ c'$
have $(\exists t\ k.\ (\Gamma \vdash (c'!j)\ {-pes-}(t\sharp k){\rightarrow}\ (c'!Suc\ j)) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c'!j\ {-pes-}(t\sharp k){\rightarrow}\ c'!Suc\ j) \longrightarrow (\Gamma \vdash ?cs'\ k!j$ ${-es-}(t\sharp k){\rightarrow}\ ?cs'\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash ?cs'\ k'!j\ {-ese}{\rightarrow}\ ?cs'\ k'!\ Suc$ $j))))$
$\vee$
$((\Gamma \vdash (c'!j)\ {-pese}{\rightarrow}\ (c'!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((?cs'\ k)!j)\ {-ese}{\rightarrow}$ $((?cs'\ k)!\ Suc\ j))))$
proof(*cases* $j = 0$)
assume *d0*: $j = 0$
from *p3* **have** $(\Gamma \vdash (c'!0)\ {-pese}{\rightarrow}\ (c'!1))$
by (*simp add*: *pesetran.intros*)
moreover
have $\forall k.\ (\Gamma \vdash ((?cs'\ k)!0)\ {-ese}{\rightarrow}\ ((?cs'\ k)!1))$
by (*simp add*: *b3 esetran.intros*)
ultimately show *?thesis* **using** *d0* **by** *simp*
next
assume *d0*: $j \neq 0$
then have *d0-1*: $j > 0$ **by** *simp*
from *p2* **have** *compat-tran* $\Gamma\ c\ cs$ **by** (*simp add*:*conjoin-def*)
then have *d1*: $\forall j.\ Suc\ j < length\ c \longrightarrow$
$(\exists t\ k.\ (\Gamma \vdash c!j\ {-pes-}(t\sharp k){\rightarrow}\ c!Suc\ j) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!j\ {-pes-}(t\sharp k){\rightarrow}\ c!Suc\ j) \longrightarrow (\Gamma \vdash cs\ k!j$ ${-es-}(t\sharp k){\rightarrow}\ cs\ k!\ Suc\ j) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!j\ {-ese}{\rightarrow}\ cs\ k'!\ Suc\ j))))$
$\vee$
$((\Gamma \vdash (c!j)\ {-pese}{\rightarrow}\ (c!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!j)$

$-ese\rightarrow$ ((*cs k*)! *Suc j*))))
by (*simp add*:*compat-tran-def*)
from *p3 p4 dd0 d0* **have** *d2*: *Suc* (*j*−*1*) < *length c* **by** *auto*
let *?j1* = *j* − *1*
from *d1 d2* **have** *d3*: (∃ *t k*. (Γ ⊢ *c*!(*j*−*1*) −*pes*−(*t*♯*k*)→ *c*!*Suc* (*j*−*1*)) ∧
(∀ *k t*. (Γ ⊢ *c*!(*j*−*1*) −*pes*−(*t*♯*k*)→ *c*!*Suc* (*j*−*1*)) ⟶ (Γ ⊢ *cs k*!(*j*−*1*) −*es*−(*t*♯*k*)→ *cs k*! *Suc* (*j*−*1*)) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (Γ ⊢ *cs k*′!(*j*−*1*) −*ese*→ *cs k*′! *Suc* (*j*−*1*)))))
∨
((Γ ⊢ (*c*!(*j*−*1*)) −*pese*→ (*c*!*Suc* (*j*−*1*))) ∧ (∀ *k*. (Γ ⊢ ((*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* Γ (*pes2 k*) *t1 x1*); *c* = (*pes2*, *t1*, *x1*) # *xs*; *c*∈*cpts-pes* Γ;
Γ *c* ∝ *cs*; *c*′ = (*pes1*, *s1*, *y1*) # (*pes2*, *t1*, *x1*) # *xs*; Γ ⊢ (*pes1*, *s1*, *y1*) −*pes*−*ct*→ (*pes2*, *t1*, *x1*)⟧
⟹ *compat-tran* Γ *c*′ (λ*k*. (*pes1 k*, *s1*, *y1*) # *cs k*)
proof −
let *?cs*′ = λ*k*. (*pes1 k*, *s1*, *y1*) # *cs k*
assume *p0*: ∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes2 k*) *t1 x1*
and *p1*: *c* ∈ *cpts-pes* Γ
and *p2*: Γ *c* ∝ *cs*
and *p3*: *c*′ = (*pes1*, *s1*, *y1*) # (*pes2*, *t1*, *x1*) # *xs*
and *p4*: *c* = (*pes2*, *t1*, *x1*) # *xs*
and *p5*: Γ ⊢ (*pes1*, *s1*, *y1*) −*pes*−*ct*→ (*pes2*, *t1*, *x1*)
from *p0* **have** *b3*: ∀ *k*. *cs k* ∈ *cpts-es* Γ ∧ (*cs k*)!*0* = (*pes2 k*,*t1*,*x1*) **by** (*simp add*:*cpts-of-es-def*)
show *compat-tran* Γ *c*′ *?cs*′
proof −
{
fix *j*
assume *dd0*: *Suc j* < *length c*′
have (∃ *t k*. (Γ ⊢ (*c*′!*j*) −*pes*−(*t*♯*k*)→ (*c*′!*Suc j*)) ∧
(∀ *k t*. (Γ ⊢ *c*′!*j* −*pes*−(*t*♯*k*)→ *c*′!*Suc j*) ⟶ (Γ ⊢ *?cs*′ *k*!*j* −*es*−(*t*♯*k*)→ *?cs*′ *k*! *Suc j*) ∧

($\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash ?cs'\ k'!j\ -ese\rightarrow\ ?cs'\ k'!\ Suc$ $j))))$
$\vee$
$((\Gamma \vdash (c'!j)\ -pese\rightarrow\ (c'!Suc\ j)) \wedge (\forall k.\ (\Gamma \vdash ((?cs'\ k)!j)\ -ese\rightarrow$ $((?cs'\ k)!\ Suc\ j))))$
proof(*cases* $j = 0$)
assume *d0*: $j = 0$
from *p5* **obtain** *k* **and** *aa* **where** *c0*: $ct = (aa\sharp k)$ **using** *get-actk-def* **by** (*metis cases*)
with *p5* **have** $\exists es'.\ (\Gamma \vdash (pes1\ k,\ s1,\ y1)\ -es-(aa\sharp k)\rightarrow\ (es',\ t1,\ x1))$ $\wedge\ pes2 = pes1(k:=es')$
using *pestran-estran* **by** *auto*
then obtain *es'* **where** *c1*: $(\Gamma \vdash (pes1\ k,\ s1,\ y1)\ -es-(aa\sharp k)\rightarrow\ (es',$ $t1,\ x1)) \wedge pes2 = pes1(k:=es')$
by *auto*
from *b3* **have** *c2*: $cs\ k \in cpts\text{-}es\ \Gamma \wedge (cs\ k)!0 = (pes2\ k,t1,x1)$ **by** *auto*
then obtain *xs1* **where** *c4*: $(cs\ k) = (pes2\ k,t1,x1)\#xs1$
by (*metis cpts-es-not-empty neq-Nil-conv nth-Cons-0*)
then have *c3*: $?cs'\ k = (pes1\ k,\ s1,\ y1)\ \#\ (pes2\ k,t1,x1)\#xs1$ **by** *simp*

from *p3 p5 c0* **have** *g0*: $\Gamma \vdash (c'!0)\ -pes-(aa\sharp k)\rightarrow\ (c'!Suc\ 0)$ **by** *auto*
moreover
have $\forall k1\ t1.\ (\Gamma \vdash c'!0\ -pes-(t1\sharp k1)\rightarrow\ c'!Suc\ 0) \longrightarrow (\Gamma \vdash\ ?cs'\ k1!0$ $-es-(t1\sharp k1)\rightarrow\ ?cs'\ k1!\ Suc\ 0) \wedge$
$(\forall k'.\ k' \neq k1 \longrightarrow (\Gamma \vdash\ ?cs'\ k'!0\ -ese\rightarrow\ ?cs'\ k'!$ $Suc\ 0))$
proof −
{
fix *k1 t1*
assume *d0*: $\Gamma \vdash c'!0\ -pes-(t1\sharp k1)\rightarrow\ c'!Suc\ 0$
with *p3* **have** $\Gamma \vdash\ ?cs'\ k1!0\ -es-(t1\sharp k1)\rightarrow\ ?cs'\ k1!\ Suc\ 0$
using *b3 fun-upd-apply nth-Cons-0 nth-Cons-Suc pestran-estran* **by** *fastforce*
moreover
from *d0* **have** $\forall k'.\ k' \neq k1 \longrightarrow (\Gamma \vdash\ ?cs'\ k'!0\ -ese\rightarrow\ ?cs'\ k'!\ Suc$ $0)$
using *b3 esetran.intros fun-upd-apply nth-Cons-0 nth-Cons-Suc p3 pestran-estran* **by** *fastforce*
ultimately have $(\Gamma \vdash c'!0\ -pes-(t1\sharp k1)\rightarrow\ c'!Suc\ 0) \longrightarrow (\Gamma \vdash\ ?cs'$ $k1!0\ -es-(t1\sharp k1)\rightarrow\ ?cs'\ k1!\ Suc\ 0) \wedge$
$(\forall k'.\ k' \neq k1 \longrightarrow (\Gamma \vdash\ ?cs'\ k'!0\ -ese\rightarrow\ ?cs'\ k'!$ $Suc\ 0))$ **by** *simp*
}
then show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **using** *d0* **by** *auto*
next
assume *d0*: $j \neq 0$
then have *d0-1*: $j > 0$ **by** *simp*

from *p2* **have** *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
then have *d1*: ∀ *j*. *Suc j* < *length c* ⟶
(∃ *t k*. (Γ ⊢ *c*!*j* −*pes*−(*t*♯*k*)→ *c*!*Suc j*) ∧
(∀ *k t*. (Γ ⊢ *c*!*j* −*pes*−(*t*♯*k*)→ *c*!*Suc j*) ⟶ (Γ ⊢ *cs k*!*j* −*es*−(*t*♯*k*)→ *cs k*! *Suc j*) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (Γ ⊢ *cs k*′!*j* −*ese*→ *cs k*′! *Suc j*))))
∨
((Γ ⊢ (*c*!*j*) −*pese*→ (*c*!*Suc j*)) ∧ (∀ *k*. (Γ ⊢ ((*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*. (Γ ⊢ *c*!(*j*−*1*) −*pes*−(*t*♯*k*)→ *c*!*Suc* (*j*−*1*)) ∧
(∀ *k t*. (Γ ⊢ *c*!(*j*−*1*) −*pes*−(*t*♯*k*)→ *c*!*Suc* (*j*−*1*)) ⟶ (Γ ⊢ *cs k*!(*j*−*1*) −*es*−(*t*♯*k*)→ *cs k*! *Suc* (*j*−*1*)) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (Γ ⊢ *cs k*′!(*j*−*1*) −*ese*→ *cs k*′! *Suc* (*j*−*1*)))))
∨
((Γ ⊢ (*c*!(*j*−*1*)) −*pese*→ (*c*!*Suc* (*j*−*1*))) ∧ (∀ *k*. (Γ ⊢ ((*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* Γ ⟶
(∃ *cs*. (∀ *k*. (*cs k*) ∈ *cpts-of-es* Γ ((*getspc* (*c*!*0*)) *k*) (*gets* (*c*!*0*)) (*getx* (*c*!*0*))) ∧ Γ *c* ∝ *cs*)
proof −
{
fix *c*
assume *p0*: *c* ∈ *cpts-pes* Γ
then have ∃ *cs*. (∀ *k*. (*cs k*) ∈ *cpts-of-es* Γ ((*getspc* (*c*!*0*)) *k*) (*gets* (*c*!*0*)) (*getx* (*c*!*0*))) ∧ Γ *c* ∝ *cs*
proof(*induct c*)
case (*CptsPesOne pes1 s1 x1*)
let *?cs* = λ*k*. [(*pes1 k*, *s1*,*x1*)]
let *?c* = [(*pes1*, *s1*, *x1*)]
have ∀ *k*. *?cs k* ∈ *cpts-of-es* Γ (*getspc* (*?c* ! *0*) *k*) (*gets* (*?c* ! *0*)) (*getx* (*?c*

! *0*))
proof –
{
fix *k*
have *?cs k* = [(*pes1 k*,*s1*,*x1*)] by *simp*
moreover
have *?cs k* ∈ *cpts-es* Γ by (*simp add*: *cpts-es.CptsEsOne*)
ultimately have *?cs k* ∈ *cpts-of-es* Γ (*pes1 k*) *s1 x1* by (*simp add*: *cpts-of-es-def*)
}
then show *?thesis* by (*simp add*: *gets-def getspc-def getx-def*)
qed
moreover
have Γ *?c* ∝ *?cs*
proof –
have *same-length ?c ?cs* by (*simp add*: *same-length-def*)
moreover
have *same-state ?c ?cs* using *same-state-def gets-def gets-es-def getx-def getx-es-def*
by (*smt length-Cons less-Suc0 list.size*(*3*) *nth-Cons-0 snd-conv*)
moreover
have *same-spec ?c ?cs* using *same-spec-def getspc-def getspc-es-def*
by (*metis* (*mono-tags*, *lifting*) *fst-conv length-Cons less-Suc0 list.size*(*3*) *nth-Cons-0*)
moreover
have *compat-tran* Γ *?c ?cs* by (*simp add*: *compat-tran-def*)
ultimately show *?thesis* by (*simp add*:*conjoin-def*)
qed
ultimately show *?case* by *auto*
next
case (*CptsPesEnv pes1 t1 x1 xs s1 y1*)
let *?c* = (*pes1*, *t1*, *x1*) # *xs*
assume *b0*: *?c*∈ *cpts-pes* Γ
and *b1*: ∃ *cs*. (∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*getspc* (*?c* ! *0*) *k*) (*gets* (*?c* ! *0*)) (*getx* (*?c* ! *0*))) ∧ Γ *?c* ∝ *cs*
then obtain *cs* where *b2*: (∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes1 k*) *t1 x1*) ∧ Γ *?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* Γ ∧ (*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'* = λ*k*. (*pes1 k*,*s1*,*y1*)#(*cs k*)
have *g0*: ∀ *k*. *?cs' k* ∈ *cpts-of-es* Γ (*getspc* (*?c'* ! *0*) *k*) (*gets* (*?c'* ! *0*)) (*getx* (*?c'* ! *0*))
proof –
{
fix *k*
from *b3* have *c0*: *cs k* ∈*cpts-es* Γ ∧ (*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 Γ by (simp add: cpts-es.CptsEsEnv)
           then have ?cs' k ∈ cpts-of-es Γ (getspc (?c' ! 0) k)  (gets (?c' ! 0))
(getx (?c' ! 0))
            by (simp add: cpts-of-es-def gets-def getspc-def getx-def)
        }
        then show ?thesis by auto
        qed
      from b2 have b4: Γ ?c ∝ cs by simp
      from b1 have g1: Γ ?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-Suc0 list.size(*4*) *not-less nth-Cons'*)
qed
}
then show *?thesis* **by** (*simp add*: *same-spec-def*)
qed
moreover
from *b0 b2 b4* **have** *compat-tran* Γ *?c' ?cs'*
using *comp-tran-env* [*of cs* Γ *pes1 t1 x1 ?c xs ?c' s1 y1*] **by** *simp*
ultimately show *?thesis* **by** (*simp add*:*conjoin-def*)
qed
from *g0 g1* **show** *?case* **by** *auto*
next
case (*CptsPesComp pes1 s1 y1 ct pes2 t1 x1 xs*)
let *?c* = (*pes2*, *t1*, *x1*) # *xs*
assume *b0*: *?c*∈ *cpts-pes* Γ
and *b1*: ∃ *cs*. (∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*getspc* (*?c* ! *0*) *k*) (*gets* (*?c* ! *0*)) (*getx* (*?c* ! *0*))) ∧ Γ *?c* ∝ *cs*
and *b00*: Γ ⊢ (*pes1*, *s1*, *y1*) −*pes*−*ct*→ (*pes2*, *t1*, *x1*)
then obtain *cs* **where** *b2*: (∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes2 k*) *t1 x1*) ∧ Γ *?c* ∝ *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* Γ ∧ (*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'* = λ*k*. (*pes1 k*,*s1*,*y1*)#(*cs k*)
have *g0*: ∀ *k*. *?cs' k* ∈ *cpts-of-es* Γ (*getspc* (*?c'* ! *0*) *k*) (*gets* (*?c'* ! *0*)) (*getx* (*?c'* ! *0*))
proof −
{
fix *k*
obtain *ka* **and** *aa* **where** *c0*: *ct* = (*aa*♯*ka*) **using** *get-actk-def* **by** (*metis cases*)
with *b00* **have** ∃ *es'*. (Γ ⊢ (*pes1 ka*, *s1*, *y1*) −*es*−(*aa*♯*ka*)→ (*es'*, *t1*, *x1*)) ∧ *pes2* = *pes1*(*ka*:=*es'*)
using *pestran-estran* **by** *auto*
then obtain *es'* **where** *c1*: (Γ ⊢ (*pes1 ka*, *s1*, *y1*) −*es*−(*aa*♯*ka*)→ (*es'*, *t1*, *x1*)) ∧ *pes2* = *pes1*(*ka*:=*es'*)
by *auto*
from *b3* **have** *c2*: *cs k* ∈*cpts-es* Γ ∧ (*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* Γ (*getspc* (*?c'* ! *0*) *k*) (*gets* (*?c'* ! *0*)) (*getx* (*?c'* ! *0*))
proof(*cases k* = *ka*)

assume *d0*: $k = ka$
with *c1* **have** $\Gamma \vdash (pes1\ k,\ s1,\ y1)\ -es-(aa\sharp k)\rightarrow\ (pes2\ k,\ t1,\ x1)$ **by** *auto*
with *c2 c3 d0* **have** *?cs′ k* $\in$ *cpts-es* Γ
using *cpts-es.CptsEsComp* **by** *fastforce*
then show *?thesis* **by** (*simp add*: *cpts-of-es-def gets-def getspc-def getx-def*)
next
assume *d0*: $k \neq ka$
with *c1* **have** *pes1 k* = *pes2 k* **by** *simp*
with *c2 c3* **have** *d1*: *?cs′ k* $\in$ *cpts-es* Γ
by (*simp add*: *cpts-es.CptsEsEnv*)
then show *?thesis* **by** (*simp add*: *cpts-of-es-def gets-def getspc-def getx-def*)
qed
}
then show *?thesis* **by** *auto*
qed
from *b2* **have** *b4*: Γ *?c* $\propto$ *cs* **by** *simp*
from *b1* **have** *g1*: Γ *?c′* $\propto$ *?cs′*
proof −
from *b4* **have** *same-length ?c′ ?cs′*
by (*simp add*: *conjoin-def same-length-def*)
moreover
have *same-state ?c′ ?cs′*
proof −
{
fix *k′ j*
assume *c0*: *j* < *length ?c′*
have *gets* (*?c′!j*) = *gets-es* ((*?cs′ k′*)*!j*) $\wedge$ *getx* (*?c′!j*) = *getx-es* ((*?cs′ k′*)*!j*)
proof(*cases j* = *0*)
assume *d0*: *j* = *0*
then show *?thesis* **by** (*simp add:gets-def gets-es-def getx-def getx-es-def*)
next
assume *d0*: $j \neq 0$
with *b4* **show** *?thesis* **using** *same-state-def gets-def gets-es-def getx-def getx-es-def*
using *c0 conjoin-def length-Cons less-Suc-eq-0-disj nth-Cons-Suc* **by** *fastforce*
qed
}
then show *?thesis* **by** (*simp add*: *same-state-def*)
qed

moreover
have *same-spec ?c′ ?cs′*
proof −

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

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

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


lemma cpt-imp-exist-conjoin-cs: c ∈ cpts-of-pes Γ pes s x
              ⟹ ∃ cs. (∀ k. (cs k) ∈ cpts-of-es Γ (pes k) s x) ∧ Γ c ∝ cs
  proof −
    assume p0: c ∈ cpts-of-pes Γ pes s x
    then have c!0=(pes,s,x) ∧ c ∈ cpts-pes Γ by (simp add:cpts-of-pes-def)
    then show ?thesis
      using cpt-imp-exist-conjoin-cs0 getspc-def gets-def getx-def
        by (metis fst-conv snd-conv)
  qed

theorem par-evtsys-semantics-comp:
  cpts-of-pes Γ pes s x = {c. ∃ cs. (∀ k. (cs k) ∈ cpts-of-es Γ (pes k) s x) ∧ Γ c ∝
cs}
  proof −
    have ∀ c. c∈cpts-of-pes Γ pes s x ⟶ (∃ cs. (∀ k. (cs k) ∈ cpts-of-es Γ (pes k)
s x) ∧ Γ c ∝ cs)
      proof −
```

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

end

end

5 Rely-guarnatee Validity of Picore Computations

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

5.1 Definitions Correctness Formulas

locale *event-validity = event-comp ptran petran fin-com cpts-p cpts-of-p*
for *ptran* :: $'Env \Rightarrow (('prog \times 's) \times 'prog \times 's)\ set$
and *petran* :: $'Env \Rightarrow ('s,'prog)\ pconf \Rightarrow ('s,'prog)\ pconf \Rightarrow bool$ (- $\vdash$ - $-pe\rightarrow$ - [81,81,81] 80)
and *fin-com* :: $'prog$
and *cpts-p* :: $'Env \Rightarrow ('s,'prog)\ pconfs\ set$
and *cpts-of-p* :: $'Env \Rightarrow 'prog \Rightarrow 's \Rightarrow (('s,'prog)\ pconfs)\ set$
+
fixes *prog-validity* :: $'Env \Rightarrow 'prog \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
(- $\models$ - sat_p [-, -, -, -] [60,60,0,0,0,0] 45)
fixes *assume-p* :: $'Env \Rightarrow ('s\ set \times ('s \times 's)\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$
fixes *commit-p* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('s,'prog)\ pconfs)\ set$

assumes *prog-validity-def*: $\Gamma \models P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow$
$\forall s.\ cpts\text{-}of\text{-}p\ \Gamma\ P\ s \cap assume\text{-}p\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}p\ \Gamma\ (guar,\ post)$

assumes *assume-p-def*: $gets\text{-}p\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$\Gamma \vdash c!i\ {-pe\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in rely)$
$\Longrightarrow c \in assume\text{-}p\ \Gamma\ (pre,\ rely)$

assumes *commit-p-def*: $c \in commit\text{-}p\ \Gamma\ (guar,\ post) \Longrightarrow (\forall i.\ Suc\ i{<}length\ c \longrightarrow$

$\Gamma \vdash c!i\ {-c\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\text{-}p\ (c!i),\ gets\text{-}p\ (c!Suc\ i)) \in guar) \wedge$
$(getspc\text{-}p\ (last\ c) = fin\text{-}com \longrightarrow gets\text{-}p\ (last\ c) \in post)$
begin

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

definition *commit-e* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('l,'k,'s,'prog)\ econfs)$ *set* **where**
$commit\text{-}e\ \Gamma \equiv \lambda(guar,\ post).\ \{c.\ (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$(\exists t.\ \Gamma \vdash c!i\ {-et{-}t\rightarrow}\ c!(Suc\ i)) \longrightarrow (gets\text{-}e\ (c!i),\ gets\text{-}e\ (c!Suc\ i)) \in$ $guar) \wedge$
$(getspc\text{-}e\ (last\ c) = AnonyEvent\ fin\text{-}com \longrightarrow gets\text{-}e\ (last\ c) \in post)\}$

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

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

definition *commit-es* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('l,'k,'s,'prog)\ esconfs)$ *set* **where**
$commit\text{-}es\ \Gamma \equiv \lambda(guar,\ post).\ \{c.\ (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$(\exists t.\ \Gamma \vdash c!i\ {-es{-}t\rightarrow}\ c!(Suc\ i)) \longrightarrow (gets\text{-}es\ (c!i),\ gets\text{-}es\ (c!Suc\ i))$ $\in guar)\ \}$

definition *es-validity* :: $'Env \Rightarrow ('l,'k,'s,'prog)\ esys \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
$(\text{-} \models \text{-}\ sat_s\ [\text{-},\ \text{-},\ \text{-},\ \text{-}]\ [60,60,0,0,0,0]\ 45)$ **where**
$\Gamma \models es\ sat_s\ [pre,\ rely,\ guar,\ post] \equiv$
$\forall s\ x.\ (cpts\text{-}of\text{-}es\ \Gamma\ es\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}es\ \Gamma\ (guar,\ post)$

definition *assume-pes* :: $'Env \Rightarrow ('s\ set \times ('s \times 's)\ set) \Rightarrow (('l,'k,'s,'prog)\ pesconfs)$

set **where**
  assume-pes $\Gamma \equiv \lambda(pre,\ rely).\ \{c.\ gets\ (c!0) \in pre \wedge (\forall\, i.\ Suc\ i{<}length\ c \longrightarrow$
  $\Gamma \vdash c!i\ {-pese\rightarrow}\ c!(Suc\ i) \longrightarrow (gets\ (c!i),\ gets\ (c!Suc\ i)) \in rely)\}$

definition *commit-pes* :: $'Env \Rightarrow (('s \times 's)\ set \times 's\ set) \Rightarrow (('l,'k,'s,'prog)\ pesconfs)$ *set* **where**
  commit-pes $\Gamma \equiv \lambda(guar,\ post).\ \{c.\ (\forall\, i.\ Suc\ i{<}length\ c \longrightarrow$
  $(\exists\, t.\ \Gamma \vdash c!i\ {-pes{-}t\rightarrow}\ c!(Suc\ i)) \longrightarrow (gets\ (c!i),\ gets\ (c!Suc\ i)) \in guar)\}$

definition *pes-validity* :: $'Env \Rightarrow ('l,'k,'s,'prog)\ paresys \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
  (- $\models$ - *SAT* [-, -, -, -] [60,60,0,0,0,0] 45) **where**
  $\Gamma \models pes\ SAT\ [pre,\ rely,\ guar,\ post] \equiv$
  $\forall\, s\ x.\ (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}pes\ \Gamma\ (guar,\ post)$

5.2 Lemmas of Correctness Formulas

lemma *assume-es-one-more*:
  $[\![esl{\in}cpts\text{-}es\ \Gamma;\ m > 0;\ m < length\ esl;\ take\ m\ esl{\in}assume\text{-}es\ \Gamma\ (pre,\ rely);\ \neg(\Gamma \vdash esl!(m{-}1)\ {-ese\rightarrow}esl!m)]\!]$
  $\Longrightarrow take\ (Suc\ m)\ esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$
  proof −
    assume *p0*: $esl{\in}cpts\text{-}es\ \Gamma$
      and *p1*: $m > 0$
      and *p2*: $m < length\ esl$
      and *p3*: $take\ m\ esl{\in}assume\text{-}es\ \Gamma\ (pre,\ rely)$
      and *p4*: $\neg(\Gamma \vdash esl!(m{-}1)\ {-ese\rightarrow}esl!m)$
    let $?esl1 = take\ (Suc\ m)\ esl$
    let $?esl = take\ m\ esl$
    have $gets\text{-}es\ (?esl1!0) \in pre \wedge (\forall\, i.\ Suc\ i{<}length\ ?esl1 \longrightarrow$
      $\Gamma \vdash ?esl1!i\ {-ese\rightarrow}\ ?esl1!(Suc\ i) \longrightarrow (gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in rely)$
      proof
        from *p1 p2 p3* **show** $gets\text{-}es\ (?esl1!0) \in pre$ **by** (*simp add*:*assume-es-def*)
      next
        show $\forall\, i.\ Suc\ i{<}length\ ?esl1 \longrightarrow$
          $\Gamma \vdash ?esl1!i\ {-ese\rightarrow}\ ?esl1!(Suc\ i) \longrightarrow (gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in rely$
          proof −
          {
            fix *i*
            assume *a0*: $Suc\ i{<}length\ ?esl1$
              and *a1*: $\Gamma \vdash ?esl1!i\ {-ese\rightarrow}\ ?esl1!(Suc\ i)$
            have $(gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in rely$
              proof(*cases* $i < m - 1$)
                assume *b0*: $i < m - 1$
                with *p1* **have** *b1*: $gets\text{-}es\ (?esl1!i) = gets\text{-}es\ (?esl!i)$ **by** *simp*

from $b0$ $p1$ **have** $b2$: *gets-es* (*?esl1*!*Suc* i) = *gets-es* (*?esl*!*Suc* i) **by** *simp*
from $p3$ **have** $\forall i$. *Suc* i<*length* *?esl* $\longrightarrow$
$\Gamma \vdash$ *?esl*!i −*ese*→ *?esl*!(*Suc* i) $\longrightarrow$
(*gets-es* (*?esl*!i), *gets-es* (*?esl*!*Suc* i)) $\in$ *rely*
by (*simp add*:*assume-es-def*)
with $b0$ **have** (*gets-es* (*?esl*!i), *gets-es* (*?esl*!*Suc* i)) $\in$ *rely*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-mono Suc-pred a1 length-take less-SucI less-imp-le-nat min.absorb2 nth-take p1 p2*)
with $b1$ $b2$ **show** *?thesis* **by** *simp*
next
assume $\neg$(i < m − *1*)
with $a0$ **have** $b0$: i = m − *1* **by** (*simp add*: *less-antisym p1*)
with *p1 p4 a1* **show** *?thesis* **by** *simp*
qed
} **then show** *?thesis* **by** *auto* **qed**
qed
then show *?thesis* **by** (*simp add*:*assume-es-def*)
qed

lemma *assume-es-take-n*:
⟦m > *0*; $m \le$ *length esl*; *esl*$\in$*assume-es* Γ (*pre*, *rely*)⟧
$\Longrightarrow$ *take m esl* $\in$ *assume-es* Γ (*pre*, *rely*)
proof −
assume *p1*: m > *0*
and *p2*: $m \le$ *length esl*
and *p3*: *esl*$\in$*assume-es* Γ (*pre*, *rely*)
let *?esl1* = *take m esl*
from *p3* **have** *gets-es* (*esl*!*0*)$\in$*pre* **by** (*simp add*:*assume-es-def*)
with *p1 p2 p3* **have** *gets-es* (*?esl1*!*0*) $\in$ *pre* **by** *simp*
moreover
have $\forall i$. *Suc* i<*length* *?esl1* $\longrightarrow$
$\Gamma \vdash$ *?esl1*!i −*ese*→ *?esl1*!(*Suc* i) $\longrightarrow$ (*gets-es* (*?esl1*!i), *gets-es* (*?esl1*!*Suc* i)) $\in$ *rely*
proof −
{
fix i
assume *a0*: *Suc* i<*length* *?esl1*
and *a1*: $\Gamma \vdash$ *?esl1*!i −*ese*→ *?esl1*!(*Suc* i)
with *p3* **have** (*gets-es* (*esl*!i), *gets-es* (*esl*!*Suc* i)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
with *p1 p2 a0* **have** (*gets-es* (*?esl1*!i), *gets-es* (*?esl1*!*Suc* i)) $\in$ *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*:
  $[\![m < length\ esl;\ esl \in assume\text{-}es\ \Gamma\ (pre,\ rely);\ gets\text{-}es\ (esl!m) \in pre1]\!]$
  $\Longrightarrow drop\ m\ esl \in assume\text{-}es\ \Gamma\ (pre1,\ rely)$

  proof −
    assume *p1*: $m < length\ esl$
      and *p3*: $esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$
      and *p2*: $gets\text{-}es\ (esl!m) \in pre1$
    let $?esl1 = drop\ m\ esl$
    from *p1 p2 p3* **have** $gets\text{-}es\ (?esl1!0) \in pre1$
      by (*simp add*: *hd-conv-nth hd-drop-conv-nth not-less*)
    moreover
    have $\forall i.\ Suc\ i<length\ ?esl1 \longrightarrow$
        $\Gamma \vdash ?esl1!i\ -ese\rightarrow\ ?esl1!(Suc\ i) \longrightarrow (gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc$
$i)) \in rely$
      proof −
      {
        fix *i*
        assume *a0*: $Suc\ i<length\ ?esl1$
          and *a1*: $\Gamma \vdash ?esl1!i\ -ese\rightarrow\ ?esl1!(Suc\ i)$
        with *p1 p3* **have** $(gets\text{-}es\ (esl!(m+i)),\ gets\text{-}es\ (esl!Suc\ (m+i))) \in rely$ **by**
(*simp add*: *assume-es-def*)
        with *p1 p2 a0* **have** $(gets\text{-}es\ (?esl1!i),\ gets\text{-}es\ (?esl1!Suc\ i)) \in rely$
          using *Suc-lessD length-take min.absorb2 nth-take* **by** *auto*
      }
      then show *?thesis* **by** *auto* **qed**
    ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
  qed

lemma *commit-es-take-n*:
  $[\![m > 0;\ m \leq length\ esl;\ esl \in commit\text{-}es\ \Gamma\ (guar,\ post)]\!]$
  $\Longrightarrow take\ m\ esl \in commit\text{-}es\ \Gamma\ (guar,\ post)$

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

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

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

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

lemma *commit-es-imp*: $[\![guar1 \subseteq guar;\ post1 \subseteq post;\ c \in commit\text{-}es\ \Gamma\ (guar1,post1)]\!] \Longrightarrow c \in commit\text{-}es\ \Gamma\ (guar,post)$

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

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

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

Suc i)
$\longrightarrow$ (*gets* (*c* ! *i*), *gets* (*c* ! *Suc i*)) $\in$ *guar*
by *auto*
qed
qed

lemma *assume-pes-take-n*:
$[\![$*m* > *0*; *m* $\leq$ *length esl*; *esl*$\in$*assume-pes* Γ (*pre*, *rely*)$]\!]$
$\Longrightarrow$ *take m esl* $\in$ *assume-pes* Γ (*pre*, *rely*)
proof −
assume *p1*: *m* > *0*
and *p2*: *m* $\leq$ *length esl*
and *p3*: *esl*$\in$*assume-pes* Γ (*pre*, *rely*)
let *?esl1* = *take m esl*
from *p3* **have** *gets* (*esl*!*0*)$\in$*pre* **by** (*simp add*:*assume-pes-def*)
with *p1 p2 p3* **have** *gets* (*?esl1*!*0*) $\in$ *pre* **by** *simp*
moreover
have $\forall$ *i*. *Suc i*<*length ?esl1* $\longrightarrow$
$\Gamma \vdash$ *?esl1*!*i* −*pese*$\rightarrow$ *?esl1*!(*Suc i*) $\longrightarrow$ (*gets* (*?esl1*!*i*), *gets* (*?esl1*!*Suc i*))
$\in$ *rely*
proof −
{
fix *i*
assume *a0*: *Suc i*<*length ?esl1*
and *a1*: $\Gamma \vdash$ *?esl1*!*i* −*pese*$\rightarrow$ *?esl1*!(*Suc i*)
with *p3* **have** (*gets* (*esl*!*i*), *gets* (*esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-pes-def*)
with *p1 p2 a0* **have** (*gets* (*?esl1*!*i*), *gets* (*?esl1*!*Suc i*)) $\in$ *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*$\in$*assume-pes* Γ (*pre*, *rely*); *gets* (*esl*!*m*) $\in$ *pre1*$]\!]$
$\Longrightarrow$ *drop m esl* $\in$ *assume-pes* Γ (*pre1*, *rely*)
proof −
assume *p1*: *m* < *length esl*
and *p3*: *esl*$\in$*assume-pes* Γ (*pre*, *rely*)
and *p2*: *gets* (*esl*!*m*) $\in$ *pre1*
let *?esl1* = *drop m esl*
from *p1 p2 p3* **have** *gets* (*?esl1*!*0*) $\in$ *pre1*
by (*simp add*: *hd-conv-nth hd-drop-conv-nth not-less*)
moreover
have $\forall$ *i*. *Suc i*<*length ?esl1* $\longrightarrow$
$\Gamma \vdash$ *?esl1*!*i* −*pese*$\rightarrow$ *?esl1*!(*Suc i*) $\longrightarrow$ (*gets* (*?esl1*!*i*), *gets* (*?esl1*!*Suc i*))
$\in$ *rely*
proof −
{

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

end

end

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

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

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

6.1 Proof System for Programs

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

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

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

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

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

lemma *stable-e-int2*: *stable-e s r* ⟹ *stable-e t r* ⟹ *stable-e* (*s* ∩ *t*) *r*
by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-e-def*)

lemma *stable-e-int3*: *stable-e k r* ⟹ *stable-e s r* ⟹ *stable-e t r* ⟹ *stable-e* (*k* ∩ *s* ∩ *t*) *r*

by (*metis* (*full-types*) *IntD1 IntD2 IntI stable-e-def*)

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

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* ⇒ (*'s* × *'s*) *set*
where $Rely_f$ *rg* = *rely-rgf rg*

definition $Guar_f$ :: *'s rgformula* ⇒ (*'s* × *'s*) *set*
where $Guar_f$ *rg* = *guar-rgf rg*

definition $Post_f$ :: *'s rgformula* ⇒ *'s set*
where $Post_f$ *rg* = *post-rgf rg*

type-synonym (*'l*,*'k*,*'s*,*'prog*) *rgformula-e* = (*'l*,*'k*,*'s*,*'prog*) *event* × *'s rgformula*

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

definition *get-int-rely* :: (*'l*,*'k*,*'s*,*'prog*) *rgformula-e set* ⇒ (*'s* × *'s*) *set*
where *get-int-rely S* ≡ $\{s.\ \forall f \in S.\ s \in Rely_f\ (snd\ f)\}$

definition *get-un-guar* :: (*'l*,*'k*,*'s*,*'prog*) *rgformula-e set* ⇒ (*'s* × *'s*) *set*
where *get-un-guar S* ≡ $\{s.\ \exists f \in S.\ s \in Guar_f\ (snd\ f)\}$

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

datatype $('l,'k,'s,'prog)$ *rgformula-ess* =
  rgf-EvtSeq $('l,'k,'s,'prog)$ *rgformula-e* $('l,'k,'s,'prog)$ *rgformula-ess* $\times$ $'s$ *rgformula*
  | *rgf-EvtSys* $('l,'k,'s,'prog)$ *rgformula-e set*

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

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

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

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

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

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

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

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

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

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

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

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

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

locale *event-hoare* = *event-validity ptran petran fin-com cpts-p cpts-of-p prog-validity assume-p commit-p*

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

6.3 Proof System for Events

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

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

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

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

6.4 Proof System for Event Systems

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

$rely \subseteq Rely_e\ ef;\ rely \subseteq Rely_f\ (snd\ esf);$
$Guar_e\ ef \subseteq guar;\ Guar_f\ (snd\ esf) \subseteq guar;$
$Post_e\ ef \subseteq Pre_f\ (snd\ esf)]\!]$
$\Longrightarrow \Gamma \vdash (rgf\text{-}EvtSeq\ ef\ esf)\ sat_s\ [pre,\ rely,\ guar,\ post]$

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

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

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

6.5 Proof System for Parallel Event Systems

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

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

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

7 Soundness

7.1 Some previous lemmas

7.1.1 event

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

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

7.1.2 event system

lemma *concat-i-lm*[*rule-format*]: $\forall ls\ l.\ concat\ ls = l \wedge (\forall i < length\ ls.\ ls!i \neq []) \longrightarrow (\forall i.\ Suc\ i < length\ ls \longrightarrow$

$(\exists\, m\ n.\ m \leq \mathit{length}\ l \wedge n \leq \mathit{length}\ l \wedge m \leq n \wedge \mathit{ls}!i@[(\mathit{ls}!\mathit{Suc}$
$i)!0] = \mathit{take}\ (n - m)\ (\mathit{drop}\ m\ l)))$
proof $-$
{
fix *ls*
have $\forall\, l.\ \mathit{concat}\ \mathit{ls} = l \wedge (\forall\, i{<}\mathit{length}\ \mathit{ls}.\ \mathit{ls}!i \neq []) \longrightarrow (\forall\, i.\ \mathit{Suc}\ i < \mathit{length}\ \mathit{ls}$
$\longrightarrow$
$(\exists\, m\ n.\ m \leq \mathit{length}\ l \wedge n \leq \mathit{length}\ l \wedge m \leq n \wedge \mathit{ls}!i@[(\mathit{ls}!\mathit{Suc}$
$i)!0] = \mathit{take}\ (n - m)\ (\mathit{drop}\ m\ l)))$
proof(*induct ls*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons x xs*)
assume *a0*: $\forall\, l.\ \mathit{concat}\ \mathit{xs} = l \wedge (\forall\, i{<}\mathit{length}\ \mathit{xs}.\ \mathit{xs}\ !\ i \neq []) \longrightarrow$
$(\forall\, i.\ \mathit{Suc}\ i < \mathit{length}\ \mathit{xs} \longrightarrow (\exists\, m\ n.\ m \leq \mathit{length}\ l \wedge n \leq \mathit{length}$
$l \wedge$
$m \leq n \wedge \mathit{xs}\ !\ i\ @\ [\mathit{xs}\ !\ \mathit{Suc}\ i\ !\ 0] = \mathit{take}\ (n - m)\ (\mathit{drop}$
$m\ l)))$
show *?case*
proof $-$
{
fix *l*
assume *b0*: $\mathit{concat}\ (x\ \#\ \mathit{xs}) = l$
and *b1*: $\forall\, i{<}\mathit{length}\ (x\ \#\ \mathit{xs}).\ (x\ \#\ \mathit{xs})\ !\ i \neq []$
let $?l' = \mathit{concat}\ \mathit{xs}$
from *b0* **have** *b2*: $l = x@?l'$ **by** *simp*
have $\forall\, i.\ \mathit{Suc}\ i < \mathit{length}\ (x\ \#\ \mathit{xs}) \longrightarrow (\exists\, m\ n.\ m \leq \mathit{length}\ l \wedge n \leq \mathit{length}$
$l \wedge$
$m \leq n \wedge (x\ \#\ \mathit{xs})\ !\ i\ @\ [(x\ \#\ \mathit{xs})\ !\ \mathit{Suc}\ i\ !\ 0] = \mathit{take}\ (n - m)$
$(\mathit{drop}\ m\ l))$
proof $-$
{
fix *i*
assume *c0*: $\mathit{Suc}\ i < \mathit{length}\ (x\ \#\ \mathit{xs})$
then have *c1*: $\mathit{length}\ \mathit{xs} > 0$ **by** *auto*
have $\exists\, m\ n.\ m \leq \mathit{length}\ l \wedge n \leq \mathit{length}\ l \wedge m \leq n \wedge$
$(x\ \#\ \mathit{xs})\ !\ i\ @\ [(x\ \#\ \mathit{xs})\ !\ \mathit{Suc}\ i\ !\ 0] = \mathit{take}\ (n - m)\ (\mathit{drop}\ m\ l)$
proof($\mathit{cases}\ i = 0$)
assume *d0*: $i = 0$
from *b1 c1* **have** *d1*: $(x\ \#\ \mathit{xs})\ !\ 1 \neq []$ **by** (*metis One-nat-def c0*
d0)
with *b0* **have** *d2*: $x\ @\ [\mathit{xs}!0\ !\ 0] = \mathit{take}\ (\mathit{length}\ x + 1)\ (\mathit{drop}\ 0\ l)$
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*: $(x\ \#\ \mathit{xs})\ !\ 0\ @\ [(x\ \#\ \mathit{xs})\ !\ 1\ !\ 0] = \mathit{take}\ (\mathit{length}\ x$

$+ 1)$ $(drop\ 0\ l)$
by *simp*
moreover
have $0 \leq length\ l$ **using** *calculation* **by** *auto*
moreover
from *b0 d1* **have** $length\ x + 1 \leq length\ l$
by (*metis Suc-eq-plus1 d2 drop-0 length-append-singleton linear take-all*)
ultimately show *?thesis* **using** *d0* **by** *force*
next
assume *d0*: $i \neq 0$
moreover
from *b1* **have** *d1*: $\forall i<length\ xs.\ xs\ !\ i \neq []$ **by** *auto*
moreover
from *c0* **have** $Suc\ (i - 1) < length\ xs$ **using** *d0* **by** *auto*
ultimately have $\exists m\ n.\ m \leq length\ ?l' \wedge n \leq length\ ?l' \wedge$
$m \leq n \wedge xs\ !\ (i - 1)\ @\ [xs\ !\ Suc\ (i - 1)\ !\ 0] = take\ (n - m)\ (drop\ m\ ?l')$
using *a0 d0* **by** *blast*
then obtain *m* **and** *n* **where** *d2*: $m \leq length\ ?l' \wedge n \leq length\ ?l' \wedge$
$m \leq n \wedge xs\ !\ (i - 1)\ @\ [xs\ !\ Suc\ (i - 1)\ !\ 0] = take\ (n - m)\ (drop\ m\ ?l')$
by *auto*
let $?m' = m + length\ x$
let $?n' = n + length\ x$
from *b0 d2* **have** $?m' \leq length\ l$ **by** *auto*
moreover
from *b0 d2* **have** $?n' \leq length\ l$ **by** *auto*
moreover
from *d2* **have** $?m' \leq ?n'$ **by** *auto*
moreover
have $(x\ \#\ xs)\ !\ i\ @\ [(x\ \#\ xs)\ !\ Suc\ i\ !\ 0] = take\ (?n' - ?m')\ (drop\ ?m'\ l)$
using *b2 d0 d2* **by** *auto*
ultimately have $?m' \leq length\ l \wedge ?n' \leq length\ l \wedge ?m' \leq ?n' \wedge$
$(x\ \#\ xs)\ !\ i\ @\ [(x\ \#\ xs)\ !\ Suc\ i\ !\ 0] = take\ (?n' - ?m')\ (drop\ ?m'\ l)$ **by** *simp*
then show *?thesis* **by** *blast*
qed
}
then show *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *blast*

```
  qed

lemma concat-last-lm: ∀ 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*: $i \leq$ *length l*
from *a0* **have** $\exists\, k\, j.\ k <$ *length lt* $\wedge\ j \leq$ *length* (*lt*!*k*) $\wedge$
drop i l = (*drop j* (*lt*!*k*)) @ *concat* (*drop* (*Suc k*) *lt*)
proof(*induct i*)
case *0*
assume *b0*: $0 \leq$ *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* $\longrightarrow \Gamma \vdash$ *l!i* $-ese\rightarrow$ *l!*(*Suc i*)
  $\longrightarrow$ (*gets-es* (*l!i*), *gets-es* (*l!Suc i*)) $\in$ *rely* $\Longrightarrow$
  $\forall$ *m subl. m* $\leq$ *length l* $\wedge$ *subl* = *take m l* $\longrightarrow$ ($\forall i.$ *Suc i<length subl* $\longrightarrow \Gamma$
$\vdash$ *subl!i* $-ese\rightarrow$ *subl!*(*Suc i*)
  $\longrightarrow$ (*gets-es* (*subl!i*), *gets-es* (*subl!Suc i*)) $\in$ *rely*)
  proof −
  assume *p0*: $\forall i.$ *Suc i<length l* $\longrightarrow \Gamma \vdash$ *l!i* $-ese\rightarrow$ *l!*(*Suc i*)
  $\longrightarrow$ (*gets-es* (*l!i*), *gets-es* (*l!Suc i*)) $\in$ *rely*
  show *?thesis*
  proof −
  {
  fix *m*
  have $\forall$ *subl. m* $\leq$ *length l* $\wedge$ *subl* = *take m l* $\longrightarrow$ ($\forall i.$ *Suc i<length subl* $\longrightarrow$
$\Gamma \vdash$ *subl!i* $-ese\rightarrow$ *subl!*(*Suc i*)
  $\longrightarrow$ (*gets-es* (*subl!i*), *gets-es* (*subl!Suc i*)) $\in$ *rely*)
  proof(*induct m*)
  case *0* **show** *?case* **by** *simp*
  next
  case (*Suc n*)
  assume *a0*: $\forall$ *subl. n* $\leq$ *length l* $\wedge$ *subl* = *take n l* $\longrightarrow$
  ($\forall i.$ *Suc i* < *length subl* $\longrightarrow \Gamma \vdash$ *subl* ! *i* $-ese\rightarrow$ *subl* ! *Suc i*
$\longrightarrow$
  (*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) $\in$ *rely*)
  show *?case*
  proof −
  {
  fix *subl*
  assume *b0*: *Suc n* $\leq$ *length l*
  and *b1*: *subl* = *take* (*Suc n*) *l*
  with *a0* **have** $\forall i.$ *Suc i* < *length subl* $\longrightarrow \Gamma \vdash$ *subl* ! *i* $-ese\rightarrow$ *subl* !
Suc i $\longrightarrow$
  (*gets-es* (*subl* ! *i*), *gets-es* (*subl* ! *Suc i*)) $\in$ *rely*
  using *p0* **by** *auto*
  }
  then show *?thesis* **by** *auto*
  qed
  qed
  }
  then show *?thesis* **by** *auto*

    qed
  qed

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

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

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

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

lemma *pre-trans*: $[\![esl \in assume\text{-}es\ \Gamma\ (pre,\ rely);\ \forall i<length\ esl.\ getspc\text{-}es\ (esl!i) = es;\ stable\text{-}e\ pre\ rely]\!]$
$\Longrightarrow \forall i<length\ esl.\ gets\text{-}es\ (esl!i) \in pre$
proof −
assume *p0*: $esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$
and *p2*: $\forall i<length\ esl.\ getspc\text{-}es\ (esl!i) = es$
and *p3*: *stable-e pre rely*
then show *?thesis*
proof −
{
fix *i*
assume *a0*: *i<length esl*
then have $gets\text{-}es\ (esl!i) \in pre$
proof(*induct i*)
case *0* **from** *p0* **show** *?case* **by** (*simp add*:*assume-es-def*)
next
case (*Suc j*)
assume *b0*: $j < length\ esl \Longrightarrow gets\text{-}es\ (esl\ !\ j) \in pre$
and *b1*: $Suc\ j < length\ esl$
then have *b2*: $gets\text{-}es\ (esl\ !\ j) \in pre$ **by** *auto*

from *p2 b1* **have** $getspc\text{-}es\ (esl\ !\ j) = es$ **by** *auto*
moreover
from *p2 b1* **have** $getspc\text{-}es\ (esl\ !\ Suc\ j) = es$ **by** *auto*
ultimately have $\Gamma \vdash esl\ !\ j\ -ese\rightarrow\ esl\ !\ Suc\ j$ **by** (*simp add*: *eqconf-esetran*)
with *p0 b1* **have** $(gets\text{-}es\ (esl!j),\ gets\text{-}es\ (esl!Suc\ j)) \in rely$ **by** (*simp add*:*assume-es-def*)
with *p3 b2* **show** *?case* **by** (*simp add*:*stable-e-def*)
qed

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

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

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

7.1.3 parallel event system

7.2 State trace equivalence

7.2.1 trace equivalence of program and anonymous event

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

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

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

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

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

lemma *lower-evts-same-s*: *ps* = *lower-evts* (*es*::('l,'k,'s,'prog) *econfs*) ⟹ ∀ *i*<*length ps*. *gets-p* (*ps*!*i*) = *gets-e* (*es*!*i*)
proof(*induct ps arbitrary*:*es*)
case *Nil*
then show *?case* **by**(*simp add*:*lower-evts-def lower-anonyevt1-def*)
next
case (*Cons a ps*)
assume *p*: (⋀*es*. *ps* = *lower-evts* (*es*::('l,'k,'s,'prog) *econfs*) ⟹ ∀ *i*<*length ps*. *gets-p* (*ps* ! *i*) = *gets-e* (*es* ! *i*))
and *p1*: *a* # *ps* = *lower-evts es*
{
fix *i*
assume *i*: *i*<*length* (*a* # *ps*)
then have *gets-p* ((*a* # *ps*) ! *i*) = *gets-e* (*es* ! *i*)
proof(*induct i*)
case *0*
then show *?case* **apply** (*simp add*:*gets-p-def gets-e-def*) **using** *p1* **apply**(*case-tac getspc-e* (*es*!*0*))
apply (*simp add*:*lower-evts-def lower-anonyevt1-def getspc-e-def*)
apply (*metis AnonyEv gets-e-def getspc-e-def lower-anonyevt1-def map-eq-Cons-D nth-Cons-0 sndI*)
apply (*simp add*:*lower-evts-def lower-anonyevt1-def getspc-e-def*)

by (*metis BasicEv gets-e-def getspc-e-def lower-anonyevt1-def map-eq-Cons-D nth-Cons-0 sndI*)
next
case (*Suc j*)
assume *a0*: *Suc j* < *length* (*a* # *ps*)
hence *a1*: *j* < *length ps* **by** *auto*
from *p1* **have** *ps* = *lower-evts* (*tl es*) **apply** (*simp add*:*lower-evts-def lower-anonyevt1-def*) **by** *auto*
moreover
have *gets-p* ((*a* # *ps*) ! *Suc j*) = *gets-p* (*ps* ! *j*) **by**(*simp add*: *gets-p-def*)
moreover
from *p1* **have** *gets-e* (*es* ! *Suc j*) = *gets-e* (*tl es* ! *j*) **using** *lower-evts-same-len*[*of a* # *ps es*] **apply**(*simp add*: *gets-e-def*)
by (*metis length-0-conv list.simps*(*3*) *local.nth-tl nth-Cons-Suc*)
ultimately show *?case*
using *lower-evts-same-len*[*of ps tl es*] *p*[*rule-format*,*of tl es j*] *a1* **by** *auto*
qed

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

lemma *equiv-lower-evts0* : $\llbracket\exists P.$ *getspc-e* (*es* ! *0*) = *AnonyEvent P*; *es* ∈ *cpts-ev* Γ$\rrbracket \Longrightarrow$ *lower-evts es* ∈*cpts-p* Γ
proof−
assume *a0*: *es* ∈ *cpts-ev* Γ **and** *a1*: ∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*
have ∀ *es P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P* ∧ *es* ∈ *cpts-ev* Γ ⟶ *lower-evts es* ∈*cpts-p* Γ
proof −
{
fix *es*
assume *b0*: ∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P* **and**
b1: *es* ∈ *cpts-ev* Γ
from *b1 b0* **have** *lower-evts es* ∈*cpts-p* Γ
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*: *CptsPOne*)
next
case (*CptsEvEnv e′ t′ x′ xs′ s′ y′*)
assume *c0*: (*e′*, *t′*, *x′*) # *xs′* ∈ *cpts-ev* Γ **and**

c1: $\exists P$. *getspc-e* $(((e', t', x') \# xs') ! 0) = AnonyEvent\ P \Longrightarrow$ *lower-evts* $((e', t', x') \# xs') \in$ *cpts-p* Γ **and**
c2: $\exists P$. *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* $((e', t', x') \# xs') \in$ *cpts-p* Γ **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*: *CptsPEnv*)
next
case (*CptsEvComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *c0*: $\Gamma \vdash (e1, s1, x1) -et\rightarrow (e2, t1, y1)$ **and**
c1: $(e2, t1, y1) \# xs1 \in$ *cpts-ev* Γ **and**
c2: $\exists P$. *getspc-e* $(((e2, t1, y1) \# xs1) ! 0) = AnonyEvent\ P$
$\Longrightarrow$ *lower-evts* $((e2, t1, y1) \# xs1) \in$ *cpts-p* Γ **and**
c3: $\exists P$. *getspc-e* $(((e1, s1, x1) \# (e2, t1, y1) \# xs1) ! 0) =$ $AnonyEvent\ P$
from *c3* **obtain** *P* **where** *c-*:*getspc-e* $(((e1, s1, x1) \# (e2, t1, y1) \#$ $xs1) ! 0) = AnonyEvent\ P$ **by** *auto*
then have *c4*: $e1 = AnonyEvent\ P$ **by** (*simp add*:*getspc-e-def*)
with *c0* **have** $\exists Q$. $e2 = AnonyEvent\ Q$
apply(*clarify*)
apply(*rule etran.cases*)
apply(*simp-all*)+
done
then obtain *Q* **where** *c5*: $e2 = AnonyEvent\ Q$ **by** *auto*
with *c2* **have** *c6*:*lower-evts* $((e2, t1, y1) \# xs1) \in$ *cpts-p* Γ **by** (*simp add*: *getspc-e-def*)
have *c7*: *lower-evts* $((e1, s1, x1) \# (e2, t1, y1) \# xs1) =$
(*lower-anonyevt1* $(e1, s1, x1)$) $\#$ *lower-evts* $((e2, t1, y1) \# xs1)$
by (*simp add*: *lower-evts-def*)
have *c7-*: *lower-evts* $((e2, t1, y1) \# xs1) =$ *lower-anonyevt1* $(e2, t1,$ $y1) \#$ *lower-evts* $xs1$
by (*simp add*: *lower-evts-def*)

with *c6* **have** *c8*: *lower-anonyevt1* (*e2*, *t1*, *y1*) # *lower-evts xs1* ∈ *cpts-p* Γ **by** *simp*
from *c4* **have** *c9*: *lower-anonyevt1* (*e1*, *s1*, *x1*) = (*P*, *s1*)
by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
from *c5* **have** *c10*: *lower-anonyevt1* (*e2*, *t1*, *y1*) = (*Q*, *t1*)
by (*simp add*: *gets-e-def getspc-e-def lower-anonyevt1-def*)
from *c0 c4 c5* **have** *c11*: Γ ⊢ (*AnonyEvent P*, *s1*, *x1*) −*et*−*et*→ (*AnonyEvent Q*, *t1*, *y1*) **by** *simp*
then have Γ ⊢ (*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* Γ
using *CptsPComp* **by** *simp*
with *c7 c7-* **show** *?case* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed
with *a0 a1* **show** *?thesis* **by** *blast*
qed

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

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

7.2.2 trace between of basic and anonymous events

lemma *evtent-in-cpts1*: $el \in cpts\text{-}ev\ \Gamma \wedge el\ !\ 0 = (BasicEvent\ ev,\ s,\ x) \Longrightarrow$
$Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ {-et-}(EvtEnt\ (BasicEvent\ ev))\sharp k{\to}\ el\ !\ (Suc\ i) \Longrightarrow$
$(\forall j.\ Suc\ j \le i \longrightarrow getspc\text{-}e\ (el\ !\ j) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ j\ {-ee\to}\ el\ !\ (Suc\ j))$

proof −
assume *p0*: $el \in cpts\text{-}ev\ \Gamma \wedge el\ !\ 0 = (BasicEvent\ ev,\ s,\ x)$
assume *p1*: $Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ {-et-}(EvtEnt\ (BasicEvent\ ev))\sharp k{\to}\ el\ !\ (Suc\ i)$
from *p0* **have** *p01*: $el \in cpts\text{-}ev\ \Gamma$ **and**
p02: $el\ !\ 0 = (BasicEvent\ ev,\ s,\ x)$ **by** *auto*
from *p1* **have** *p3*: $getspc\text{-}e\ (el\ !\ i) = BasicEvent\ ev$ **by** (*meson ent-spec*)
show $\forall j.\ Suc\ j \le i \longrightarrow getspc\text{-}e\ (el\ !\ j) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ j\ {-ee\to}\ el\ !\ (Suc\ j)$
proof −
{
fix *j*
assume *a0*: $Suc\ j \le i$
have $\forall k.\ k < i \longrightarrow getspc\text{-}e\ (el\ !\ (i - k - 1)) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ (i - k - 1){-ee\to}\ el\ !\ (i - k)$
proof −
{
fix *k*
assume $k < i$
then have $getspc\text{-}e\ (el\ !\ (i - k - 1)) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ (i - k - 1){-ee\to}\ el\ !\ (i - k)$
proof(*induct k*)
case *0*
from *p3* **have** *b0*: $\neg(\exists t\ ec1.\ \Gamma \vdash ec1{-et-}t{\to}(el\ !\ i))$
using *no-tran2basic getspc-e-def* **by** (*metis prod.collapse*)
with *p1 p01* **have** *b1*: $getspc\text{-}e\ (el\ !\ (i - 1)) = getspc\text{-}e\ (el\ !\ i)$ **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*: $m < i \Longrightarrow getspc\text{-}e\ (el\ !\ (i - m - 1)) = BasicEvent\ ev$
$\wedge\ \Gamma \vdash el\ !\ (i - m - 1)\ {-ee\to}\ el\ !\ (i - m)$ **and**
b1: $Suc\ m < i$
then have *b2*: $getspc\text{-}e\ (el\ !\ (i - m - 1)) = BasicEvent\ ev$ **and**
b3: $\Gamma \vdash el\ !\ (i - m - 1)\ {-ee\to}\ el\ !\ (i - m)$
using *Suc-lessD* **apply** *blast*
using *Suc-lessD b0 b1* **by** *blast*
have *b4*: $Suc\ m = m + 1$ **by** *auto*
with *b2* **have** $\neg(\exists t\ ec1.\ \Gamma \vdash ec1{-et-}t{\to}(el\ !\ (i - Suc\ m)))$
using *no-tran2basic getspc-e-def* **by** (*metis diff-diff-left prod.collapse*)

with *p1 p02* **have** *b5*: $getspc\text{-}e\ (el\ !\ ((i - Suc\ m - 1))) = getspc\text{-}e$

$(el\ !\ (i - Suc\ m))$
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** $\Gamma \vdash el\ !\ (i - Suc\ m - 1)\ -ee\rightarrow\ el\ !\ (i - Suc\ m)$ **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*: $el \in cpts\text{-}ev\ \Gamma \wedge el\ !\ 0 = (BasicEvent\ ev,\ s,\ x) \Longrightarrow$
$Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow\ el\ !\ (Suc\ i) \Longrightarrow$
$(gets\text{-}e\ (el\ !\ i) \in guard\ ev \wedge drop\ (Suc\ i)\ el \in$
$cpts\text{-}of\text{-}ev\ \Gamma\ (AnonyEvent\ (body\ ev))\ (gets\text{-}e\ (el\ !\ (Suc\ i)))\ ((getx\text{-}e\ (el\ !\ i))\ (k := BasicEvent\ ev)))$
proof −
assume *p0*: $el \in cpts\text{-}ev\ \Gamma \wedge el\ !\ 0 = (BasicEvent\ ev,\ s,\ x)$
assume *p1*: $Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow\ el\ !\ (Suc\ i)$
then have *a2*: $gets\text{-}e\ (el\ !\ i) \in guard\ ev \wedge gets\text{-}e\ (el\ !\ i) = gets\text{-}e\ (el\ !\ (Suc\ i))$
$\wedge\ getspc\text{-}e\ (el\ !\ (Suc\ i)) = AnonyEvent\ (body\ ev)$
$\wedge\ getx\text{-}e\ (el\ !\ (Suc\ i)) = (getx\text{-}e\ (el\ !\ i))\ (k := BasicEvent\ ev)$
by (*meson ent-spec2*)

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

qed

lemma *no-evtent-in-cpts*: $el \in cpts\text{-}ev\ \Gamma \Longrightarrow el\ !\ 0 = (BasicEvent\ ev,\ s,\ x) \Longrightarrow$
$(\neg\ (\exists\, i\ k.\ Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow$
$el\ !\ (Suc\ i))\ ) \Longrightarrow$
$(\forall\, j.\ Suc\ j < length\ el \longrightarrow getspc\text{-}e\ (el\ !\ j) = BasicEvent\ ev$
$\wedge\ \Gamma \vdash el\ !\ j\ -ee\rightarrow el\ !\ (Suc\ j)$
$\wedge\ getspc\text{-}e\ (el\ !\ (Suc\ j)) = BasicEvent\ ev)$
proof $-$
assume *p0*: $el \in cpts\text{-}ev\ \Gamma$ **and**
p1: $el\ !\ 0 = (BasicEvent\ ev,\ s,\ x)$ **and**
p2: $\neg\ (\exists\, i\ k.\ Suc\ i < length\ el \wedge \Gamma \vdash el\ !\ i\ -et-(EvtEnt\ (BasicEvent$
$ev))\sharp k\rightarrow el\ !\ (Suc\ i))$
show *?thesis*
proof $-$
{
fix *j*
assume $Suc\ j < length\ el$
then have $getspc\text{-}e\ (el\ !\ j) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ j\ -ee\rightarrow el\ !\ (Suc$
$j)$
$\wedge\ getspc\text{-}e\ (el\ !\ (Suc\ j)) = BasicEvent\ ev$
proof(*induct j*)
case *0*
assume *a0*: $Suc\ 0 < length\ el$
from *p1* **have** *a00*: $getspc\text{-}e\ (el\ !\ 0) = BasicEvent\ ev$ **by** (*simp*
add:getspc-e-def)
from *a0 p2* **have** $\neg\ (\exists\, k.\ \Gamma \vdash el\ !\ 0\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow$
$el\ !\ (Suc\ 0))$ **by** *simp*
with *p0 p1* **have** $\neg\ (\exists\, t.\ \Gamma \vdash el\ !\ 0\ -et-t\rightarrow el\ !\ (Suc\ 0))$ **by** (*metis*
noevtent-notran)
with *p0 a0* **have** *a1*: $getspc\text{-}e\ (el\ !\ 0) = getspc\text{-}e\ (el\ !\ (Suc\ 0))$
using *notran-confeqi* **by** *blast*

with *a00* **have** *a2*: $getspc\text{-}e\ (el\ !\ (Suc\ 0)) = BasicEvent\ ev$ **by** *simp*
from *a1* **have** $\Gamma \vdash el\ !\ 0\ -ee\rightarrow el\ !\ Suc\ 0$ **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*: $Suc\ m < length\ el \Longrightarrow getspc\text{-}e\ (el\ !\ m) = BasicEvent\ ev$
$\wedge\ \Gamma \vdash el\ !\ m\ -ee\rightarrow el\ !\ Suc\ m$
$\wedge\ getspc\text{-}e\ (el\ !\ Suc\ m) = BasicEvent\ ev$
assume *a1*: $Suc\ (Suc\ m) < length\ el$
with *a0* **have** *a2*: $getspc\text{-}e\ (el\ !\ m) = BasicEvent\ ev \wedge \Gamma \vdash el\ !\ m\ -ee\rightarrow$
$el\ !\ Suc\ m$ **by** *simp*
then have *a3*: $getspc\text{-}e\ (el\ !\ Suc\ m) = BasicEvent\ ev$ **using** *getspc-e-def*
by (*metis eetranE fstI*)

then have *a4*: $\exists s\ x.\ el\ !\ Suc\ m = (BasicEvent\ ev,\ s,\ x)$ **unfolding** *getspc-e-def*
by (*metis fst-conv surj-pair*)
from *a0 a1 p2* **have** $\neg (\exists k.\ \Gamma \vdash el\ !\ (Suc\ m)\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow\ el\ !\ (Suc\ (Suc\ m)\ ))$ **by** *simp*
with *a4* **have** *a5*: $\neg (\exists t.\ \Gamma \vdash el\ !\ (Suc\ m)\ -et-t\rightarrow\ el\ !\ (Suc\ (Suc\ m)))$
using *noevtent-notran* **by** *metis*

with *p0 a0 a1* **have** *a6*: $getspc\text{-}e\ (el\ !\ (Suc\ m)) = getspc\text{-}e\ (el\ !\ (Suc\ (Suc\ m)))$
using *notran-confeqi* **by** *blast*
with *a3* **have** *a7*: $getspc\text{-}e\ (el\ !\ (Suc\ (Suc\ m))) = BasicEvent\ ev$ **by** *simp*
from *a6* **have** $\Gamma \vdash el\ !\ Suc\ m\ -ee\rightarrow\ el\ !\ Suc\ (Suc\ m)$ **using** *getspc-e-def eetran.EnvE*
by (*metis eq-fst-iff*)

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

7.2.3 trace between of event and event system

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

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

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

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

lemma *e-eqv-einevtseq-s* : $\llbracket e\text{-}eqv\text{-}einevtseq\ esl\ el\ es;\ gets\text{-}e\ e1 = gets\text{-}es\ es1;\ getx\text{-}e\ e1 = getx\text{-}es\ es1;$
$getspc\text{-}es\ es1 = EvtSeq\ (getspc\text{-}e\ e1)\ es\rrbracket \Longrightarrow e\text{-}eqv\text{-}einevtseq$

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

lemma *rm-evtsys-same-sx*: *same-s-x esl* (*rm-evtsys esl*)
  proof(*induct esl*)
    case *Nil*
    show *?case* **by** (*simp add*:*rm-evtsys-def same-s-x-def*)
  next
    case (*Cons ec1 esl1*)
    assume *a0*: *same-s-x esl1* (*rm-evtsys esl1*)
    have *a1*: *rm-evtsys* (*ec1* # *esl1*) = *rm-evtsys1 ec1* # *rm-evtsys esl1* **by** (*simp add*:*rm-evtsys-def*)
     obtain *es* **and** *s* **and** *x* **where** *a2*: *ec1* = (*es*, *s*, *x*) **using** *prod-cases3* **by** *blast*
    then show *?case*
      proof(*induct es*)
        case (*EvtSeq x1 es1*)
        assume *b0*: *ec1* = (*EvtSeq x1 es1*, *s*, *x*)
         then have *b1*: *rm-evtsys1 ec1* # *rm-evtsys esl1* = (*x1*, *s*, *x*) # *rm-evtsys esl1*
          by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
        have *length* (*ec1* # *esl1*) = *length* (*rm-evtsys* (*ec1* # *esl1*)) **by** (*simp add*: *rm-evtsys-def*)

moreover have $\forall\, i.\ Suc\ i \leq length\ (rm\text{-}evtsys\ (ec1\ \#\ esl1)) \longrightarrow$
$gets\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = gets\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
$\wedge\ getx\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = getx\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
proof –
{
fix i
assume $c0$: $Suc\ i \leq length\ (rm\text{-}evtsys\ (ec1\ \#\ esl1))$
have $gets\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = gets\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
$\wedge\ getx\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = getx\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
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 $(rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i = (rm\text{-}evtsys\ esl1)\ !\ (i - 1)$
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 $b1$: $rm\text{-}evtsys1\ ec1\ \#\ rm\text{-}evtsys\ esl1 = (AnonyEvent\ fin\text{-}com,\ s,\ x)\ \#\ rm\text{-}evtsys\ esl1$
by (*simp add*:*rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
have $length\ (ec1\ \#\ esl1) = length\ (rm\text{-}evtsys\ (ec1\ \#\ esl1))$ **by** (*simp add*: *rm-evtsys-def*)
moreover have $\forall\, i.\ Suc\ i \leq length\ (rm\text{-}evtsys\ (ec1\ \#\ esl1)) \longrightarrow$
$gets\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = gets\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
$\wedge\ getx\text{-}e\ ((rm\text{-}evtsys\ (ec1\ \#\ esl1))\ !\ i) = getx\text{-}es\ ((ec1\ \#\ esl1)\ !\ i)$
proof –
{
fix i
assume $c0$: $Suc\ i \leq length\ (rm\text{-}evtsys\ (ec1\ \#\ esl1))$

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*
qed
qed

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

7.3 Soundness of Events

lemma *anony-cfgs0* : ⟦∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*; *es* ∈ *cpts-ev* Γ⟧
⟹ ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof −
assume *a0*: *es* ∈ *cpts-ev* Γ **and** *a1*: ∃ *P*. *getspc-e* (*es* ! *0*) = *AnonyEvent P*
from *a0 a1* **show** ∀ *i*. (*i* < *length es* ⟶ (∃ *Q*. *getspc-e* (*es*!*i*) = *AnonyEvent Q*))
proof(*induct es*)

case $(CptsEvOne\ e\ s\ x)$
assume $b0$: $\exists P.\ getspc\text{-}e\ ([(e,\ s,\ x)]\ !\ 0) = AnonyEvent\ P$
show $?case$ **using** $b0$ **by** $auto$
next
case $(CptsEvEnv\ e'\ t'\ x'\ xs'\ s'\ y')$
assume $b0$: $(e',\ t',\ x')\ \#\ xs' \in cpts\text{-}ev\ \Gamma$ **and**
$b1$: $\exists P.\ getspc\text{-}e\ (((e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P \Longrightarrow$
$\forall i{<}length\ ((e',\ t',\ x')\ \#\ xs').\ \exists Q.\ getspc\text{-}e\ (((e',\ t',\ x')\ \#\ xs')\ !\ i) = AnonyEvent\ Q$ **and**
$b2$: $\exists P.\ getspc\text{-}e\ (((e',\ s',\ y')\ \#\ (e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P$
from $b2$ **obtain** $P1$ **where** $b3$: $getspc\text{-}e\ (((e',\ s',\ y')\ \#\ (e',\ t',\ x')\ \#\ xs')\ !\ 0) = AnonyEvent\ P1$ **by** $auto$
then have $b4$: $e' = AnonyEvent\ P1$ **by** $(simp\ add\text{:}\ getspc\text{-}e\text{-}def)$
with $b1$ **have** $\forall i{<}length\ ((e',\ t',\ x')\ \#\ xs').\ \exists Q.\ getspc\text{-}e\ (((e',\ t',\ x')\ \#\ xs')\ !\ i) = AnonyEvent\ Q$
by $(simp\ add\text{:}\ getspc\text{-}e\text{-}def)$
with $b4$ **show** $?case$ **by** $(metis\ (no\text{-}types,\ hide\text{-}lams)\ Ex\text{-}list\text{-}of\text{-}length\ b3\ gr0\text{-}conv\text{-}Suc$
$length\text{-}Cons\ length\text{-}tl\ list.sel(3)\ not\text{-}less\text{-}eq\ nth\text{-}non\text{-}equal\text{-}first\text{-}eq)$
next
case $(CptsEvComp\ e1\ s1\ x1\ et\ e2\ t1\ y1\ xs1)$
assume $b0$: $\Gamma \vdash (e1,\ s1,\ x1)\ {-et{-}et{\rightarrow}}\ (e2,\ t1,\ y1)$ **and**
$b1$: $(e2,\ t1,\ y1)\ \#\ xs1 \in cpts\text{-}ev\ \Gamma$ **and**
$b2$: $\exists P.\ getspc\text{-}e\ (((e2,\ t1,\ y1)\ \#\ xs1)\ !\ 0) = AnonyEvent\ P \Longrightarrow$
$\forall i{<}length\ ((e2,\ t1,\ y1)\ \#\ xs1).\ \exists Q.\ getspc\text{-}e\ (((e2,\ t1,\ y1)\ \#\ xs1)\ !\ i) = AnonyEvent\ Q$ **and**
$b3$: $\exists P.\ getspc\text{-}e\ (((e1,\ s1,\ x1)\ \#\ (e2,\ t1,\ y1)\ \#\ xs1)\ !\ 0) = AnonyEvent\ P$
from $b3$ **obtain** $P1$ **where** $b4$: $getspc\text{-}e\ (((e1,\ s1,\ x1)\ \#\ (e2,\ t1,\ y1)\ \#\ xs1)\ !\ 0) = AnonyEvent\ P1$ **by** $auto$
then have $b5$: $e1 = AnonyEvent\ P1$ **by** $(simp\ add\text{:}\ getspc\text{-}e\text{-}def)$
with $b0$ **have** $\exists Q.\ e2 = AnonyEvent\ Q$
apply$(clarify)$
apply$(rule\ etran.cases)$
apply$(simp\text{-}all)+$
done
then have $\exists P.\ getspc\text{-}e\ (((e2,\ t1,\ y1)\ \#\ xs1)\ !\ 0) = AnonyEvent\ P$ **by** $(simp\ add\text{:}getspc\text{-}e\text{-}def)$
with $b2$ **have** $b6$: $\forall i{<}length\ ((e2,\ t1,\ y1)\ \#\ xs1).\ \exists Q.\ getspc\text{-}e\ (((e2,\ t1,\ y1)\ \#\ xs1)\ !\ i) = AnonyEvent\ Q$ **by** $auto$
with $b5$ **show** $?case$ **by** $(metis\ (no\text{-}types,\ hide\text{-}lams)\ Ex\text{-}list\text{-}of\text{-}length\ b3\ gr0\text{-}conv\text{-}Suc$
$length\text{-}Cons\ length\text{-}tl\ list.sel(3)\ not\text{-}less\text{-}eq\ nth\text{-}non\text{-}equal\text{-}first\text{-}eq)$
qed
qed

lemma $anony\text{-}cfgs$: $es \in cpts\text{-}of\text{-}ev\ \Gamma\ (AnonyEvent\ P)\ s\ x \Longrightarrow \forall i.\ (i < length\ es \longrightarrow (\exists Q.\ getspc\text{-}e\ (es!i) = AnonyEvent\ Q)\ )$

proof −
assume *a0*: $es \in$ *cpts-of-ev* Γ (*AnonyEvent P*) *s x*
then have *a1*: *es*!*0*=(*AnonyEvent P*,(*s*,*x*)) $\wedge$ *es* $\in$ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)
then have $\exists$ *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*: $\Gamma \models P\ sat_p$ [*pre*, *rely*, *guar*, *post*] $\Longrightarrow \Gamma \models$ *AnonyEvent P* sat_e [*pre*, *rely*, *guar*, *post*]
proof −
assume *a0*: $\Gamma \models P\ sat_p$ [*pre*, *rely*, *guar*, *post*]
then have *a1*: $\forall$ *s*. *cpts-of-p* Γ *P s* $\cap$ *assume-p* Γ (*pre*, *rely*) $\subseteq$ *commit-p* Γ (*guar*, *post*)
using *prog-validity-def* **by** *simp*
then have $\forall$ *s x*. (*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\subseteq$ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *s x*
have $\forall$ *el*. *el*$\in$(*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\longrightarrow$ *el*$\in$ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *el*
assume *b0*: *el*$\in$(*cpts-of-ev* Γ (*AnonyEvent P*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
then obtain *pl* **where** *b1*: *pl* = *lower-evts el* **by** *simp*
with *b0* **have** *b2*: *pl* $\in$ *cpts-of-p* Γ *P s* **using** *equiv-lower-evts* **by** *auto*
from *b0 b1* **have** *b21*: *pl* $\in$ *cpts-p* Γ $\wedge$ *pl*!*0* = (*P*,*s*) **using** *equiv-lower-evts2*[*of el* Γ *P s x*] **by** *auto*
from *b0* **have** *b3*: *el*!*0*=(*AnonyEvent P*,(*s*,*x*)) **and** *b4*: *el* $\in$ *cpts-ev* Γ
by (*simp add*:*cpts-of-ev-def*)+
from *b0* **have** *b5*: *el* $\in$ *assume-e* Γ (*pre*, *rely*) **by** *simp*
hence *b51*: *gets-e* (*el*!*0*) $\in$ *pre* **by** (*simp add*:*assume-e-def*)
from *b1 b21 b3 b51* **have** *b6*: *gets-p* (*pl*!*0*) $\in$ *pre* **by** (*simp add*:*gets-p-def gets-e-def*)

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

```
from b1 have c5: lower-anonyevt1 (el!Suc i) = pl!Suc i
  by (simp add: Suc-lessD c2 lower-evts-def)

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

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

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

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

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

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

  with a1 b2 have b9: pl ∈ commit-p Γ (guar, post) by auto
  then have b10: (∀ i. Suc i<length el ⟶
    (∃ t. Γ ⊢ el!i −et−t→ el!(Suc i)) ⟶ (gets-e (el!i), gets-e (el!Suc i))
∈ guar)
    proof −
    {
```

fix *i*
assume *c0*: *Suc i<length el*
assume *c1*: $\exists$ *t*. $\Gamma \vdash$ *el*!*i* $-$*et*$-$*t*$\rightarrow$ *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*: $\exists$ *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*: $\exists$ *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*: $\Gamma \vdash$ *pl*!*i* $-$*c*$\rightarrow$ *pl*!(*Suc i*)
proof $-$
from *c1* **obtain** *t* **where** *d0*: $\Gamma \vdash$ *el*!*i* $-$*et*$-$*t*$\rightarrow$ *el*!(*Suc i*) **by** *auto*
obtain *s1* **and** *x1* **where** *d1*: *s1* = *gets-e* (*el* ! *i*) $\wedge$ *x1* = *getx-e* (*el* ! *i*) **by** *simp*
obtain *s2* **and** *x2* **where** *d2*: *s2* = *gets-e* (*el* ! (*Suc i*)) $\wedge$ *x2* = *getx-e* (*el* ! (*Suc i*)) **by** *simp*
with *d1 c71 c81* **have** *d21*: *el* ! *i* = (*AnonyEvent Q1*, *s1*, *x1*)
$\wedge$ *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*: $\Gamma \vdash$ (*AnonyEvent Q1*, *s1*, *x1*) $-$*et*$-$*t*$\rightarrow$ (*AnonyEvent Q2*, *s2*, *x2*) **by** *simp*
then have $\exists$ *k*. *t* = ((*Cmd CMP*)$\sharp$*k*)
apply(*rule etran.cases*)
apply *simp-all*
by *auto*
then obtain *k* **where** *t* = ((*Cmd CMP*)$\sharp$*k*) **by** *auto*
with *d3* **have** *d4*: $\Gamma \vdash$ (*Q1*,*s1*) $-$*c*$\rightarrow$ (*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*)) $\in$ *guar* **by** (*simp add*:*commit-p-def*)

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

have *b11*: (*getspc-e* (*last el*) = *AnonyEvent fin-com* ⟶ *gets-e* (*last el*) ∈ *post*)
proof
assume *c0*: *getspc-e* (*last el*) = *AnonyEvent fin-com*
from *b1* **have** *c1*: *last pl* = *lower-anonyevt1* (*last el*)
by (*metis b4 cpts-e-not-empty last-map lower-evts-def*)
from *b9* **have** *c2*: *getspc-p* (*last pl*) = *fin-com* ⟶ *gets-p* (*last pl*) ∈ *post* **by** (*simp add*:*commit-p-def*)
from *c0 c1* **have** *c3*: *getspc-p* (*last pl*) = *fin-com*
by (*simp add*: *getspc-p-def lower-anonyevt1-def*)
with *c2* **have** *c4*: *gets-p* (*last pl*) ∈ *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* Γ (*guar*, *post*) **by** (*simp add*:*commit-e-def*)

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

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

lemma *BasicEvt-sound*:
⟦ Γ ⊨ (*body ev*) sat_p [*pre* ∩ (*guard ev*), *rely*, *guar*, *post*];
stable-e pre rely; ∀ *s*. (*s*, *s*)∈*guar*⟧
⟹ Γ ⊨ ((*BasicEvent ev*)::('*l*,'*k*,'*s*,'*prog*) *event*) sat_e [*pre*, *rely*, *guar*, *post*]
proof −
assume *p0*: Γ ⊨ (*body ev*) sat_p [*pre* ∩ (*guard ev*), *rely*, *guar*, *post*]
assume *p1*: ∀ *s*. (*s*, *s*)∈*guar*

assume *p2*: *stable-e pre rely*
have $\forall$ *s x*. (*cpts-of-ev* Γ ((*BasicEvent ev*)::('*l*,'*k*,'*s*,'*prog*) *event*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
$\subseteq$ *commit-e* Γ (*guar*, *post*)
proof –
{
fix *s x*
have $\forall$ *el*. *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\longrightarrow$ *el*$\in$ *commit-e* Γ (*guar*, *post*)
proof –
{
fix *el*
assume *b0*: *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
then have *b0-1*: *el*$\in$(*cpts-of-ev* Γ (*BasicEvent ev*) *s x*) **and**
b0-2: *el* $\in$ *assume-e* Γ (*pre*, *rely*) **by** *auto*
from *b0-1* **have** *b1*: *el* ! *0* = (*BasicEvent ev*, (*s*, *x*)) **and**
b2: *el* $\in$ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)+
from *b0-2* **have** *b3*: *gets-e* (*el*!*0*) $\in$ *pre* **and**
b4: ($\forall$ *i*. *Suc i*<*length el* $\longrightarrow$ $\Gamma \vdash$ *el*!*i* $-ee\rightarrow$ *el*!(*Suc i*) $\longrightarrow$
(*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *rely*) **by** (*simp add*: *assume-e-def*)+
have *el*$\in$ *commit-e* Γ (*guar*, *post*)
proof(*cases* $\exists$ *i k*. *Suc i* < *length el* $\wedge$ $\Gamma \vdash$ *el* ! *i* $-et-$(*EvtEnt* (*BasicEvent ev*))$\sharp$*k*$\rightarrow$ *el* ! (*Suc i*))
assume *c0*: $\exists$ *i k*. *Suc i* < *length el* $\wedge$ $\Gamma \vdash$ *el* ! *i* $-et-$(*EvtEnt* (*BasicEvent ev*))$\sharp$*k*$\rightarrow$ *el* ! (*Suc i*)
then obtain *m* **and** *k* **where** *c1*: *Suc m* < *length el* $\wedge$ $\Gamma \vdash$ *el* ! *m* $-et-$(*EvtEnt* (*BasicEvent ev*))$\sharp$*k*$\rightarrow$ *el* ! (*Suc m*)
by *auto*
with *b1 b2* **have** *c2*: $\forall$ *j*. *Suc j* $\leq$ *m* $\longrightarrow$ *getspc-e* (*el* ! *j*) = *BasicEvent ev* $\wedge$ $\Gamma \vdash$ *el* ! *j* $-ee\rightarrow$ *el* ! (*Suc j*)
by (*meson evtent-in-cpts1*)
from *b1 b2 c1* **have** *c4*: *gets-e* (*el* ! *m*) $\in$ *guard ev* **and**
c6: *drop* (*Suc m*) *el* $\in$ *cpts-of-ev* Γ (*AnonyEvent* (*body ev*)) (*gets-e* (*el* ! (*Suc m*))) ((*getx-e* (*el* ! *m*)) (*k* := *BasicEvent ev*))
using *evtent-in-cpts2*[*of el* Γ *ev s x m k*] **by** *auto*

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

from *b4 c1 c2* **have** *c8*:$\forall$ *j*. *Suc j* $\leq$ *m* $\longrightarrow$ (*gets-e* (*el* ! *j*), *gets-e* (*el* ! (*Suc j*))) $\in$ *rely* **by** *auto*
with *p2 b3* **have** *c9*: $\forall$ *j*. *j* $\leq$ *m* $\longrightarrow$ *gets-e* (*el* ! *j*) $\in$ *pre*
proof –
{
fix *j*

assume *d0*: $j \leq m$
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-e-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* Γ ((*AnonyEvent* (*body*
ev))::(*'l*,*'k*,*'s*,*'prog*) *event*) *s x*) ∩
assume-e Γ (*pre* ∩ (*guard ev*), *rely*) ⊆ *commit-e* Γ (*guar*, *post*) **by**
(*simp add*:*evt-validity-def*)

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

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

have *c14*: ∀ *i*. *Suc i* < *length el* ⟶ (∃ *t*. Γ ⊢ *el* ! *i* −*et*−*t*→ *el* ! *Suc*
i)
⟶ (*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *guar*
proof −
{
fix *i*
assume *d0*: *Suc i* < *length el* **and**

d1: $(\exists t.\ \Gamma \vdash el\ !\ i\ -et-t\rightarrow\ el\ !\ Suc\ i)$
then have $(gets\text{-}e\ (el\ !\ i),\ gets\text{-}e\ (el\ !\ Suc\ i)) \in guar$
proof(*cases* $Suc\ i \leq m$)
assume *e0*: $Suc\ i \leq m$
with *c2* **have** $\Gamma \vdash el\ !\ i\ -ee\rightarrow\ el\ !\ (Suc\ i)$ **by** *auto*
then have $\neg(\exists t.\ \Gamma \vdash el\ !\ i\ -et-t\rightarrow\ el\ !\ Suc\ i)$
by (*metis eetranE evt-not-eq-in-tran prod.collapse*)
with *d1* **show** *?thesis* **by** *simp*
next
assume *e0*: $\neg\ Suc\ i \leq m$
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** $\Gamma \vdash el\ !\ i\ -et-(EvtEnt\ (BasicEvent\ ev))\sharp k\rightarrow$ $el\ !\ (Suc\ i)$ **by** *simp*
then have $gets\text{-}e\ (el\ !\ i) = gets\text{-}e\ (el\ !\ (Suc\ i))$ **by** (*meson ent-spec2*)
with *p1* **show** *?thesis* **by** *auto*
next
assume *f0*: $\neg\ Suc\ i = m + 1$
with *e1* **have** *f1*: $Suc\ i > Suc\ m$ **by** *auto*
from *c13* **have** *f2*: $\forall i.\ Suc\ i < length\ (drop\ (Suc\ m)\ el)$ $\longrightarrow$
$(\exists t.\ \Gamma \vdash (drop\ (Suc\ m)\ el)\ !\ i\ -et-t\rightarrow\ (drop\ (Suc\ m)\ el)\ !\ Suc\ i) \longrightarrow$
$(gets\text{-}e\ ((drop\ (Suc\ m)\ el)\ !\ i),\ gets\text{-}e\ ((drop\ (Suc\ m)\ el)\ !\ Suc\ i)) \in guar$
by (*simp add*:*commit-e-def*)
with *d0 d1 f1* **have** $(gets\text{-}e\ (drop\ (Suc\ m)\ el\ !\ (i - Suc\ m)),\ gets\text{-}e\ (drop\ (Suc\ m)\ el\ !\ Suc\ (i - Suc\ m))) \in guar$
proof −
from *d0 f1* **have** *g0*: $Suc\ (i - Suc\ m) < length\ (drop\ (Suc\ m)\ el)$ **by** *auto*
from *d1 f1* **have** $(\exists t.\ \Gamma \vdash drop\ (Suc\ m)\ el\ !\ (i - Suc\ m)\ -et-t\rightarrow\ drop\ (Suc\ m)\ el\ !\ Suc\ (i - Suc\ m))$
using *d0* **by** *auto*
with *g0 f2* **show** *?thesis* **by** *simp*
qed
then show *?thesis*
using *c1 f1* **by** *auto*
qed
qed
}
then show *?thesis* **by** *auto*
qed

from *c13* **have** *c15*: *getspc-e* (*last el*) = *AnonyEvent fin-com* ⟶ *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* ∧ Γ ⊢ *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*
∧ Γ ⊢ *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*. Γ ⊢ *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*: ∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*)
with *c1* **have** Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i* **by** *auto*
then have ¬ (∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*))
by (*metis eetranE evt-not-eq-in-tran2 prod.collapse*)
with *d0* **have** *False* **by** *simp*
}
then show *?thesis* **by** *auto*
qed
from *b1 b2* **have** *el* ≠ [] **using** *cpts-e-not-empty* **by** *auto*
with *b1 b2* **obtain** *els* **where** *el* = (*BasicEvent ev*, *s*, *x*) # *els*
by (*metis hd-Cons-tl hd-conv-nth*)
then have *getspc-e* (*last el*) = *BasicEvent ev*
proof(*induct els*)
case *Nil*
assume *el* = [(*BasicEvent ev*, *s*, *x*)]
then have *last el* = (*BasicEvent ev*, *s*, *x*) **by** *simp*
then show *?case* **by** (*simp add*:*getspc-e-def*)
next
case (*Cons els1 elsr*)
assume *d0*: *el* = (*BasicEvent ev*, *s*, *x*) # *els1* # *elsr*
then have *d1*: *length el* > *1* **by** *simp*
with *d0* **obtain** *mm* **where** *d2*: *Suc mm* = *length el* **by** *simp*
with *d1* **obtain** *jj* **where** *d3*: *Suc jj* = *mm* **using** *d0* **by** *auto*
with *d2* **have** *d4*: *last el* = *el* ! *mm*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc drop-eq-Nil last-ConsL last-drop le-eq-less-or-eq lessI*)

with *c1* **have** *getspc-e* (*el* ! (*Suc jj*)) = *BasicEvent ev* **using** *d2 d3* **by** *auto*
with *d3 d4* **show** *?case* **by** *simp*
qed

then have *c3*: *getspc-e* (*last el*) = *AnonyEvent fin-com* $\longrightarrow$ *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*:
$[\![pre \subseteq pre'; rely \subseteq rely'; guar' \subseteq guar; post' \subseteq post;$
$\Gamma \models ev\ sat_e\ [pre', rely', guar', post']]\!]$
$\Longrightarrow \Gamma \models ev\ sat_e\ [pre, rely, guar, post]$
proof −
assume *p0*: $pre \subseteq pre'$
and *p1*: $rely \subseteq rely'$
and *p2*: $guar' \subseteq guar$
and *p3*: $post' \subseteq post$
and *p4*: $\Gamma \models ev\ sat_e\ [pre', rely', guar', post']$
from *p4* **have** *p5*: $\forall s\ x.$ (*cpts-of-ev* Γ *ev s x*) $\cap$ *assume-e* Γ (*pre'*, *rely'*) $\subseteq$ *commit-e* Γ (*guar'*, *post'*)
by (*simp add*: *evt-validity-def*)
have $\forall s\ x.$ (*cpts-of-ev* Γ *ev s x*) $\cap$ *assume-e* Γ (*pre*, *rely*) $\subseteq$ *commit-e* Γ (*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: *c*$\in$(*cpts-of-ev* Γ *ev s x*) $\cap$ *assume-e* Γ (*pre*, *rely*)
then have *c*$\in$(*cpts-of-ev* Γ *ev s x*) $\wedge$ *c*$\in$*assume-e* Γ (*pre*, *rely*) **by** *simp*
with *p0 p1* **have** *c*$\in$(*cpts-of-ev* Γ *ev s x*) $\wedge$ *c*$\in$*assume-e* Γ (*pre'*, *rely'*)
using *assume-e-imp*[*of pre pre' rely rely' c*] **by** *simp*
with *p5* **have** *c*$\in$*commit-e* Γ (*guar'*, *post'*) **by** *auto*
with *p2 p3* **have** *c*$\in$*commit-e* Γ (*guar*, *post*)
using *commit-e-imp*[*of guar' guar post' post c*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*evt-validity-def*)

qed

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

7.4 Soundness of Event Systems

lemma *evtseq-nfin-samelower*: $[\![$*esl* $\in$ *cpts-of-es* Γ (*EvtSeq e es*) *s x*; $\forall$ *i*. *Suc i* $\leq$ *length esl* $\longrightarrow$ *getspc-es* (*esl* ! *i*) $\neq$ *es*$]\!]$
$\Longrightarrow$ ($\exists$ *el*. (*el* $\in$ *cpts-of-ev* Γ *e s x* $\wedge$ *length esl* = *length el* $\wedge$ *e-eqv-einevtseq esl el es*))
proof −
assume *p0*: *esl* $\in$ *cpts-of-es* Γ (*EvtSeq e es*) *s x*
and *p1*: $\forall$ *i*. *Suc i* $\leq$ *length esl* $\longrightarrow$ *getspc-es* (*esl* ! *i*) $\neq$ *es*
from *p0* **have** *p01*: *esl* ! *0* = (*EvtSeq e es*, *s*, *x*) $\wedge$ *esl* $\in$ *cpts-es* Γ **by** (*simp add*: *cpts-of-es-def*)
then have *p01-1*: *esl* ! *0* = (*EvtSeq e es*, *s*, *x*) **by** *simp*
then have *p2*: $\exists$ *e*. *getspc-es* (*esl* ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
from *p01* **have** *p01-2*: *esl* $\in$ *cpts-es* Γ **by** *simp*
let *?el* = *rm-evtsys esl*
have *a1*: *length esl* = *length ?el* **by** (*simp add*: *rm-evtsys-def*)
moreover have *?el* $\in$ *cpts-of-ev* Γ *e s x*
proof −
from *p01-2 p1 p2* **have** *b1*: *?el* $\in$ *cpts-ev* Γ
proof(*induct esl*)
case (*CptsEsOne es1 s1 x1*)
assume *c0*: $\exists$ *e*. *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e es*
then obtain *e1* **where** *c1*: *getspc-es* ([(*es1*, *s1*, *x1*)] ! *0*) = *EvtSeq e1 es* **by** *auto*
then have *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
then have *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
then have *rm-evtsys* [(*es1*, *s1*, *x1*)] = [(*e1*, *s1*, *x1*)] **by** (*simp add*:*rm-evtsys-def*)
then show *?case* **by** (*simp add*: *cpts-ev.CptsEvOne*)
next
case (*CptsEsEnv es1 t1 x1 xs1 s1 y1*)
assume *c0*: (*es1*, *t1*, *x1*) # *xs1* $\in$ *cpts-es* Γ
and *c1*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *t1*, *x1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *i*) $\neq$ *es*
$\Longrightarrow\exists$ *e*. *getspc-es* (((*es1*, *t1*, *x1*) # *xs1*) ! *0*) = *EvtSeq e es*
$\Longrightarrow$ *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) $\in$ *cpts-ev* Γ
and *c11*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*)
$\longrightarrow$ *getspc-es* (((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) !

i) ≠ *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* Γ **by** (*simp add*:*getspc-es-def*)
have *c6*: *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) = (*rm-evtsys1* (*es1*, *t1*, *x1*)) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)
have *c7*: *rm-evtsys* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) =
(*rm-evtsys1* (*es1*, *s1*, *y1*)) # (*rm-evtsys1* (*es1*, *t1*, *x1*)) # (*rm-evtsys xs1*)
by (*simp add*: *rm-evtsys-def*)
from *c4* **have** *c8*: *rm-evtsys1* (*es1*, *s1*, *y1*) = (*e1*, *s1*, *y1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
from *c4* **have** *c9*: *rm-evtsys1* (*es1*, *t1*, *x1*) = (*e1*, *t1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
have *c10*: *rm-evtsys* ((*es1*, *s1*, *y1*) # (*es1*, *t1*, *x1*) # *xs1*) = (*e1*, *s1*, *y1*) # (*e1*, *t1*, *x1*) # *rm-evtsys xs1*
by (*simp add*: *c7 c8 c9*)
have *rm-evtsys* ((*es1*, *t1*, *x1*) # *xs1*) = (*e1*, *t1*, *x1*) # *rm-evtsys xs1*
by (*simp add*: *c6 c9*)
with *c5 c10* **show** *?case* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
next
case (*CptsEsComp es1 s1 x1 et es2 t1 y1 xs1*)
assume *c0*: Γ ⊢ (*es1*, *s1*, *x1*) −*es*−*et*→ (*es2*, *t1*, *y1*)
and *c1*: (*es2*, *t1*, *y1*) # *xs1* ∈ *cpts-es* Γ
and *c2*: ∀ *i*. *Suc i* ≤ *length* ((*es2*, *t1*, *y1*) # *xs1*) ⟶ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) ≠ *es*
⟹ ∃ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
⟹ *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*) ∈ *cpts-ev* Γ
and *c3*: ∀ *i*. *Suc i* ≤ *length* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*)
⟶ *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *i*) ≠ *es*
and *c4*: ∃ *e*. *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *c4* **obtain** *e1* **where** *c41*: *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e1 es*
by *auto*
then have *c5*: *es1* = *EvtSeq e1 es* **by** (*simp add*:*getspc-es-def*)
from *c3* **have** *getspc-es* (*es2*, *t1*, *y1*) ≠ *es* **by** *auto*
then have *c6*: *es2* ≠ *es* **by** (*simp add*:*getspc-es-def*)

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

then obtain *e2* where *c7*: *es2* = *EvtSeq e2 es* by *auto*
with *c0 c5* have $\exists t$. $\Gamma \vdash$ (*e1*,*s1*,*x1*) $-et-t\rightarrow$ (*e2*,*t1*,*y1*) by (*simp add*: *evtseq-tran-exist-etran*)
then obtain *t* where *c71*: $\Gamma \vdash$ (*e1*,*s1*,*x1*) $-et-t\rightarrow$ (*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*: $\forall i$. *Suc i* $\leq$ *length* ((*es2*, *t1*, *y1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es*
by *auto*
from *c7* have $\exists e$. *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*) $\in$ *cpts-ev* Γ by *auto*
from *c5* have *c12*: *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
from *c7* have *c13*: *rm-evtsys1* (*es2*, *t1*, *y1*) = (*e2*, *t1*, *y1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
with *c71 c8 c9 c11 c12* show *?case* using *cpts-ev.CptsEvComp* by *fastforce*
qed
moreover have *?el* ! *0*=(*e*,(*s*,*x*))
proof −
from *p01* have *rm-evtsys1* (*esl* ! *0*) = (*e*, *s*, *x*)
by (*simp add*: *gets-es-def getspc-es-def getx-es-def rm-evtsys1-def*)
moreover from *a1 b1* have *?el* ! *0* = *rm-evtsys1* (*esl* ! *0*) using *rm-evtsys-def*
by (*metis cpts-e-not-empty length-greater-0-conv nth-map*)
ultimately show *?thesis* by *simp*
qed
ultimately have *?el* ! *0*=(*e*,(*s*,*x*)) $\wedge$ *?el* $\in$ *cpts-ev* Γ by *auto*
then show *?thesis* by (*simp add*: *cpts-of-ev-def*)
qed
moreover from *p01-2 p1 p2* have *e-eqv-einevtseq esl ?el es*
proof(*induct esl*)
case (*CptsEsOne es1 s1 x1*)
assume *a0*: $\exists e$. *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* Γ
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*: $\Gamma \vdash$ (*es1*, *s1*, *x1*) $-es-et\rightarrow$ (*es2*, *t1*, *y1*)
and *a1*: (*es2*, *t1*, *y1*) # *xs1* $\in$ *cpts-es* Γ
and *a2*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es2*, *t1*, *y1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es* $\Longrightarrow$
$\exists$ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es* $\Longrightarrow$
e-eqv-einevtseq ((*es2*, *t1*, *y1*) # *xs1*) (*rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)) *es*
and *a3*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*)
$\longrightarrow$ *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es*
and *a4*: $\exists$ *e*. *getspc-es* (((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es*
from *a3* **have** *a5*: $\forall$ *i*. *Suc i* $\leq$ *length* ((*es2*, *t1*, *y1*) # *xs1*) $\longrightarrow$ *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *i*) $\neq$ *es*
by *auto*
from *a4* **obtain** *e1* **where** *a6*: *es1* = *EvtSeq e1 es* **using** *getspc-es-def* **by** (*metis fst-conv nth-Cons-0*)
from *a3* **have** *getspc-es* (*es2*, *t1*, *y1*) $\neq$ *es* **by** *auto*
then have *a7*: *es2* $\neq$ *es* **by** (*simp add*:*getspc-es-def*)
with *a0 a6* **have** $\exists$ *e2*. *es2* = *EvtSeq e2 es* **by** (*meson evtseq-tran-evtsys*)
then obtain *e2* **where** *a8*: *es2* = *EvtSeq e2 es* **by** *auto*
then have *a9*: $\exists$ *e*. *getspc-es* (((*es2*, *t1*, *y1*) # *xs1*) ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
with *a2 a5* **have** *a10*: *e-eqv-einevtseq* ((*es2*, *t1*, *y1*) # *xs1*) (*rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)) *es* **by** *simp*
have *a11*: *rm-evtsys* ((*es1*, *s1*, *x1*) # (*es2*, *t1*, *y1*) # *xs1*) = *rm-evtsys1* (*es1*, *s1*, *x1*) # *rm-evtsys* ((*es2*, *t1*, *y1*) # *xs1*)
by (*simp add*:*rm-evtsys-def*)
from *a6* **have** *a12*: *rm-evtsys1* (*es1*, *s1*, *x1*) = (*e1*, *s1*, *x1*)
by (*simp add*: *gets-es-def getspc-es-def rm-evtsys1-def getx-es-def*)
with *a6 a11 a10* **show** *?case* **using** *gets-e-def gets-es-def getx-e-def getx-es-def*

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

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

lemma *evtseq-fst-finish*:
$[\![$*esl* $\in$ *cpts-es* Γ; *getspc-es* (*esl* ! *0*) = *EvtSeq e es*; *Suc m* $\leq$ *length esl*;
$\exists$ *i*. *i* $\leq$ *m* $\wedge$ *getspc-es* (*esl* ! *i*) = *es*$]\!]$ $\Longrightarrow$

$\exists\, i.\ (i \le m \wedge \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}) \wedge (\forall j.\ j < i \longrightarrow \textit{getspc-es}\ (\textit{esl} ! j) \neq \textit{es})$

proof −

assume *p0*: $\textit{esl} \in \textit{cpts-es}\ \Gamma$

and *p1*: $\textit{getspc-es}\ (\textit{esl} ! 0) = \textit{EvtSeq}\ e\ \textit{es}$

and *p2*: $\textit{Suc}\ m \le \textit{length}\ \textit{esl}$

and *p3*: $\exists\, i.\ i \le m \wedge \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}$

have $\forall\, m.\ \textit{esl} \in \textit{cpts-es}\ \Gamma \wedge \textit{getspc-es}\ (\textit{esl} ! 0) = \textit{EvtSeq}\ e\ \textit{es} \wedge \textit{Suc}\ m \le \textit{length}\ \textit{esl} \wedge$
$(\exists\, i.\ i \le m \wedge \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}) \longrightarrow$
$(\exists\, i.\ (i \le m \wedge \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}) \wedge (\forall j.\ j < i \longrightarrow \textit{getspc-es}\ (\textit{esl} ! j) \neq \textit{es}))$

proof −

{

fix *m*

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

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

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

and *a3*: $(\exists\, i.\ i \le m \wedge \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es})$

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

proof(*induct m*)

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

next

case (*Suc n*)

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

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

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

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

and *b4*: $\exists\, i{\le}\textit{Suc}\ n.\ \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}$

show *?case*

proof(*cases* $\exists\, i{\le}n.\ \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}$)

assume *c0*: $\exists\, i{\le}n.\ \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es}$

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

using *Suc-leD* **by** *blast*

then show *?case* **using** *le-Suc-eq* **by** *blast*

next

assume *c0*: $\neg\ (\exists\, i{\le}n.\ \textit{getspc-es}\ (\textit{esl} ! i) = \textit{es})$

with *b4* **have** $\textit{getspc-es}\ (\textit{esl} ! (\textit{Suc}\ n)) = \textit{es}$ **using** *le-SucE* **by** *auto*

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

ultimately show *?case* **by** *blast*

qed

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

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

lemma *EventSeq-sound* :
$[\![\Gamma \models e\ sat_e\ [pre,\ rely1,\ guar1,\ post1]; \Gamma \models es\ sat_s\ [pre2,\ rely2,\ guar2,\ post];$
$rely \subseteq rely1; rely \subseteq rely2; guar1 \subseteq guar; guar2 \subseteq guar; post1 \subseteq pre2]\!]$
$\Longrightarrow \Gamma \models EvtSeq\ e\ es\ sat_s\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: $\Gamma \models e\ sat_e\ [pre,\ rely1,\ guar1,\ post1]$
and *p1*: $\Gamma \models es\ sat_s\ [pre2,\ rely2,\ guar2,\ post]$
and *p2*: $rely \subseteq rely1$
and *p3*: $rely \subseteq rely2$
and *p4*: $guar1 \subseteq guar$
and *p5*: $guar2 \subseteq guar$
and *p6*: $post1 \subseteq pre2$
then have $\forall s\ x.\ (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}es\ \Gamma\ (guar,\ post)$
proof −
{
fix *s x*
have $\forall esl.\ esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely) \longrightarrow esl \in commit\text{-}es\ \Gamma\ (guar,\ post)$
proof −
{
fix *esl*
assume *a0*: $esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x) \cap assume\text{-}es\ \Gamma\ (pre,\ rely)$
then have *a01*: $esl \in cpts\text{-}of\text{-}es\ \Gamma\ (EvtSeq\ e\ es)\ s\ x$ **by** *simp*
from *a0* **have** *a02*: $esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$ **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*: $esl \in cpts\text{-}es\ \Gamma$ **by** (*simp add*: *cpts-of-es-def*)

have $esl \in commit\text{-}es\ \Gamma\ (guar,\ post)$
proof(*cases* $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) \neq es$)
assume *b0*: $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) \neq es$
with *a01* **have** $\exists el.\ (el \in cpts\text{-}of\text{-}ev\ \Gamma\ e\ s\ x \wedge length\ esl = length\ el \wedge e\text{-}eqv\text{-}einevtseq\ esl\ el\ es)$
by (*simp add*: *evtseq-nfin-samelower*)
then obtain *el* **where** *b1*: $el \in cpts\text{-}of\text{-}ev\ \Gamma\ e\ s\ x \wedge length\ esl = length\ el \wedge e\text{-}eqv\text{-}einevtseq\ esl\ el\ es$
by *auto*
have $el \in assume\text{-}e\ \Gamma\ (pre,\ rely1)$

proof(*simp add*:*assume-e-def*, *rule conjI*)
from *a02* **have** *c0*: *gets-es* (*esl* ! *0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
moreover
from *b1* **have** *gets-e* (*el* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-e-def*)
moreover
from *a01-1* **have** *gets-es* (*esl* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-es-def*)
ultimately show *gets-e* (*el* ! *0*) ∈ *pre* **by** *simp*
next
show ∀ *i*. *Suc i* < *length el* ⟶ Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i* ⟶
(*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely1*
proof −
{
fix *i*
assume *c0*:*Suc i* < *length el*
and *c1*: Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i*
then have *c2*: *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *eetran-eqconf1*)
moreover from *b1 c0* **have** *getspc-es* (*esl* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
moreover from *b1 c0* **have** *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
ultimately have *c3*: *getspc-es* (*esl* ! *i*) = *getspc-es* (*esl* ! *Suc i*) **by** *simp*

then have Γ ⊢ *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* Γ (*guar1*, *post1*)
by (*meson IntI contra-subsetD evt-validity-def*)
then have ∀ *i*. *Suc i*<*length el* ⟶ (∃ *t*. Γ ⊢ *el*!*i* −*et*−*t*→ *el*!(*Suc i*))

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

from *b1 c0* **have** *c3*: *getspc-es* (*esl* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
from *c1* **have** *getspc-es* (*esl* ! *i*) $\neq$ *getspc-es* (*esl* ! *Suc i*)
using *evtsys-not-eq-in-tran-aux getspc-es-def* **by** (*metis surjective-pairing*)
with *c2 c3* **have** *getspc-e* (*el* ! *i*) $\neq$ *getspc-e* (*el* ! *Suc i*) **by** *simp*
then have $\exists$ *t*. $\Gamma \vdash$ (*el* ! *i*) $-et-t\rightarrow$ (*el* ! *Suc i*)
using *b1 c0 cpts-of-ev-def notran-confeqi* **by** *fastforce*
with *b2* **have** (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) $\in$ *guar*
using *b1 c0* **by** *auto*
moreover have *gets-e* (*el*!*i*) = *gets-es* (*esl* ! *i*)
using *b1 c0 e-eqv-einevtseq-def less-imp-le* **by** *fastforce*
moreover have *gets-e* (*el*!*Suc i*) = *gets-es* (*esl* ! *Suc i*)
using *Suc-leI b1 c0 e-eqv-einevtseq-def* **by** *fastforce*
ultimately have (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *guar* **by** *simp*
}
then show *?thesis* **by** *auto*
qed
qed
next
assume *b0*: $\neg$ ($\forall$ *i*. *Suc i* $\leq$ *length esl* $\longrightarrow$ *getspc-es* (*esl* ! *i*) $\neq$ *es*)
from *a01-1* **have** *b00*: *getspc-es* (*esl* ! *0*) = *EvtSeq e es* **by** (*simp add*:*getspc-es-def*)
from *b0* **have** $\exists$ *m*. *Suc m* $\leq$ *length esl* $\wedge$ *getspc-es* (*esl* ! *m*) = *es* **by** *auto*
then obtain *m* **where** *b1*: *Suc m* $\leq$ *length esl* $\wedge$ *getspc-es* (*esl* ! *m*) = *es* **by** *auto*

then have $\exists\, i.\ i \leq m \wedge$ *getspc-es* (*esl* ! *i*) = *es* **by** *auto*
with *a01-1 a01-2 b00 b1* **have** *b2*: $\exists\, i.$ ($i \leq m \wedge$ *getspc-es* (*esl* ! *i*) = *es*) $\wedge$ ($\forall\, j.\ j < i \longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
using *evtseq-fst-finish* **by** *blast*
then obtain *n* **where** *b3*: ($n \leq m \wedge$ *getspc-es* (*esl* ! *n*) = *es*) $\wedge$ ($\forall\, j.\ j < n \longrightarrow$ *getspc-es* (*esl* ! *j*) $\neq$ *es*)
by *auto*
with *b00* **have** *b41*: $n \neq 0$ **by** (*metis* (*no-types*, *hide-lams*) *add.commute add.right-neutral*
add-Suc dual-order.irrefl esys.size(*3*) *le-add1 le-imp-less-Suc*)
then have *b4*: $n > 0$ **by** *auto*
then obtain *esl0* **where** *b5*: *esl0* = *take n esl* **by** *simp*
then have *b5-1*: *length esl0* = *n* **using** *b1 b3 less-le-trans* **by** *auto*
obtain *esl1* **where** *b6*: *esl1* = *drop n esl* **by** *simp*
with *b5* **have** *b7*: *esl0* @ *esl1* = *esl* **by** *simp*
from *a01-2 b1 b3 b4 b5* **have** *b8*: *esl0* $\in$ *cpts-es* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-le-lessD cpts-es-take less-trans*)
from *a01-2 b1 b3 b4 b5 b6* **have** *b9*: *esl1* $\in$ *cpts-es* Γ
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-le-lessD cpts-es-dropi le-neq-implies-less less-trans*)
have *b10*: *esl0* ! *0* = (*EvtSeq e es*, *s*, *x*) **by** (*simp add*: *a01-1 b4 b5*)
have *b11*: *getspc-es* (*esl1* ! *0*) = *es* **using** *b1 b3 b6* **by** *auto*

from *b3 b5* **have** *b11-1*: $\forall\, i.\ i <$ *length esl0* $\longrightarrow$ *getspc-es* (*esl0* ! *i*) $\neq$ *es* **by** *auto*
moreover from *b8 b10* **have** *esl0* $\in$ *cpts-of-es* Γ (*EvtSeq e es*) *s x* **by** (*simp add*:*cpts-of-es-def*)
ultimately have *b12*: $\exists\, el.$ (*el* $\in$ *cpts-of-ev* Γ *e s x* $\wedge$ *length esl0* = *length el* $\wedge$ *e-eqv-einevtseq esl0 el es*)
by (*simp add*: *evtseq-nfin-samelower*)
then obtain *el* **where** *b12-1*: *el* $\in$ *cpts-of-ev* Γ *e s x* $\wedge$ *length esl0* = *length el* $\wedge$ *e-eqv-einevtseq esl0 el es*
by *auto*
then have *b12-2*: *el* $\in$ *cpts-ev* Γ **by** (*simp add*:*cpts-of-ev-def*)

from *a02* **have** *b13*: *gets-es* (*esl*!*0*) $\in$ *pre* $\wedge$ ($\forall\, i.$ *Suc i*<*length esl* $\longrightarrow$
$\Gamma \vdash$ *esl*!*i* $-ese\rightarrow$ *esl*!(*Suc i*) $\longrightarrow$ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely*)
by (*simp add*:*assume-es-def*)
have *b14*: *esl0* $\in$ *assume-es* Γ (*pre*, *rely*)
proof(*simp add*:*assume-es-def*, *rule conjI*)
show *gets-es* (*esl0* ! *0*) $\in$ *pre* **using** *a01-1 b10 b13* **by** *auto*
next
from *b5 b13* **show** $\forall\, i.$ *Suc i* < *length esl0* $\longrightarrow \Gamma \vdash$ *esl0* ! *i* $-ese\rightarrow$ *esl0* ! *Suc i*
$\longrightarrow$ (*gets-es* (*esl0* ! *i*), *gets-es* (*esl0* ! *Suc i*)) $\in$ *rely* **by** *auto*

qed
with *p2* **have** *b15*: *esl0* ∈ *assume-es* Γ (*pre*, *rely1*)
by (*simp add*: *assume-es-def subset-iff*)

have *b16*: *el* ∈ *assume-e* Γ (*pre*, *rely1*)
proof(*simp add*:*assume-e-def*, *rule conjI*)
from *a02* **have** *c0*: *gets-es* (*esl* ! *0*) ∈ *pre* **by** (*simp add*:*assume-es-def*)
moreover
from *b12-1* **have** *gets-e* (*el* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-e-def*)
moreover
from *a01-1* **have** *gets-es* (*esl* ! *0*) = *s* **by** (*simp add*:*cpts-of-ev-def gets-es-def*)
ultimately show *gets-e* (*el* ! *0*) ∈ *pre* **by** *simp*
next
show ∀ *i*. *Suc i* < *length el* ⟶ Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i* ⟶
(*gets-e* (*el* ! *i*), *gets-e* (*el* ! *Suc i*)) ∈ *rely1*
proof −
{
fix *i*
assume *c0*:*Suc i* < *length el*
and *c1*: Γ ⊢ *el* ! *i* −*ee*→ *el* ! *Suc i*
then have *c2*: *getspc-e* (*el* ! *i*) = *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *eetran-eqconf1*)
moreover from *b12-1 c0* **have** *getspc-es* (*esl0* ! *i*) = *EvtSeq* (*getspc-e* (*el* ! *i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
moreover from *b12-1 c0* **have** *getspc-es* (*esl0* ! *Suc i*) = *EvtSeq* (*getspc-e* (*el* ! *Suc i*)) *es*
by (*simp add*: *e-eqv-einevtseq-def*)
ultimately have *c3*: *getspc-es* (*esl0* ! *i*) = *getspc-es* (*esl0* ! *Suc i*) **by** *simp*

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

moreover have *gets-es* (*esl0*!*i*) = *gets-e* (*el* ! *i*)
by (*metis b12-1 c0 e-eqv-einevtseq-def less-imp-le-nat*)

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

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

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

have ∀ *i*. *Suc i*<*length esl* ⟶ (∃ *t*. Γ ⊢ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*))
⟶ (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
proof −
{
fix *i*
assume *c0*: *Suc i*<*length esl*
and *c1*: ∃ *t*. Γ ⊢ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*)
have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
proof(*cases 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*) $\neq$ *getspc-e* (*el* ! *Suc i*)
by (*simp add*: *Suc-lessD b5 d0*)
then have $\exists$ *t*. $\Gamma \vdash$ (*el* ! *i*) $-et-t\rightarrow$ (*el* ! *Suc i*)
using *b12-1 b5-1 cpts-of-ev-def d0 notran-confeqi* **by** *fastforce*

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

then show *?thesis* **by** (*simp add*: *Suc-lessD b5 d0*)
next
assume *d0*: $\neg$ (*Suc i* < *n*)
from *b5-1 b12-1* **have** *d1*: *getspc-es* (*esl0* ! (*n*−*1*)) = *EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*
by (*simp add*: *b12-1 e-eqv-einevtseq-def b4*)
with *b5* **have** *d1-1*: *getspc-es* (*esl* ! (*n*−*1*)) = *EvtSeq* (*getspc-e* (*el* ! (*n*−*1*))) *es*
by (*simp add*: *b4*)
then have $\exists$ *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*)) $\wedge$
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* $\in$ *cpts-es* Γ **using** *cpts-es-dropi*
by (*metis* (*no-types*, *hide-lams*) *Suc-diff-1 Suc-le-lessD b5 b5-1*

drop-0 less-or-eq-imp-le neq0-conv not-le take-all zero-less-diff)

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

from *d8* **have** *d9*: *?el1* ∈ *cpts-of-ev* Γ *e s x*
by (*metis* (*no-types*, *lifting*) *append-Cons b12-1 b3 b4 b5-1*
cpts-of-ev-def list.*size*(*3*) *mem-Collect-eq neq-Nil-conv nth-Cons-0*)
moreover from *b16 d7* **have** *?el1* ∈ *assume-e* Γ (*pre*, *rely1*)
proof −
have *gets-e* (*?el1*!*0*) ∈ *pre*
proof −
from *b16* **have** *gets-e* (*el*!*0*) ∈ *pre* **by** (*simp add*:*assume-e-def*)
then show *?thesis* **by** (*metis b12-1 b4 b5-1 nth-append*)
qed
moreover
have ∀ *i*. *Suc i*<*length ?el1* ⟶ Γ ⊢ *?el1*!*i* −*ee*→ *?el1*!(*Suc i*) ⟶
(*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *rely1*
proof −
{
fix *i*
assume *e0*: *Suc i*<*length ?el1*
and *e1*: Γ ⊢*?el1*!*i* −*ee*→ *?el1*!(*Suc i*)
from *b16* **have** *e2*: ∀ *i*. *Suc i*<*length el* ⟶ Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*) ⟶
(*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely1* **by** (*simp add*:*assume-e-def*)
have (*gets-e* (*?el1*!*i*), *gets-e* (*?el1*!*Suc i*)) ∈ *rely1*
proof(*cases Suc i* < *length ?el1* − *1*)
assume *f0*: *Suc i* < *length ?el1* − *1*
with *e0 e2* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-diff-1*
Suc-less-eq Suc-mono e1 length-append-singleton nth-append zero-less-Suc)
next

assume ¬ (*Suc i* < *length ?el1* − *1*)
then have *f0*: *Suc i* ≥ *length ?el1* − *1* **by** *simp*
with *e0* **have** *f1*: *Suc i* = *length ?el1* − *1* **by** *simp*
then have *f2*: *?el1*!(*Suc i*) = (*AnonyEvent fin-com*, *sn*, *xn*) **by** *simp*
from *f1* **have** *f3*: *?el1*!*i* = (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*)
by (*metis b12-1 b5-1 d3 diff-Suc-1 length-append-singleton lessI nth-append*)

with *d7 f2* **have** *getspc-e* (*?el1*!*i*) ≠ *getspc-e* (*?el1*!(*Suc i*))
using *evt-not-eq-in-tran-aux* **by** (*metis e1 eetran.cases*)
moreover from *e1* **have** *getspc-e* (*?el1*!*i*) = *getspc-e* (*?el1*!(*Suc i*))
using *eetran-eqconf1* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
}
then show *?thesis* **by** *auto*
qed

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

have *d11*: *getspc-e* (*last ?el1*) = *AnonyEvent fin-com* **by** (*simp add*:*getspc-e-def*)
with *d10* **have** *d12*: *gets-e* (*last ?el1*) ∈ *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*. Γ ⊢*?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 fin-com*, *sn*, *xn*)
using *b12-1 b5-1 g0* **by** *auto*
moreover from *g0 b5-1 b12-1* **have** *?el1*!*i* = (*getspc-e* (*el* ! (*n*−*1*)), *sn1*, *xn1*)

by (*metis b12-1 b5-1 d3 diff-Suc-1 lessI nth-append*)
ultimately have (*sn1*,*sn*) ∈ *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* Γ *es sn xn* **by** (*simp add*:*cpts-of-es-def*)

have *esl1* ∈ *assume-es* Γ (*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* ⟶ Γ ⊢ *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*: Γ ⊢ *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*: Γ ⊢ *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* Γ (*guar2*,*post*)
by (*meson Int-iff es-validity-def subsetCE*)

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

have *g7*: *Suc* (*i* − *n*) < *length esl1* **using** *b6 c0 g1* **by** *auto*
from *g4* **have** ∀ *i*. *Suc i*<*length esl1* ⟶ (∃ *t*. Γ ⊢ *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*,′*prog*) *esconfs* ⇒(′*l*,′*k*,′*s*,′*prog*) *event set* ⇒
((′*l*,′*k*,′*s*,′*prog*) *esconfs*) *list* ⇒ ((′*l*,′*k*,′*s*,′*prog*) *esconfs*) *list*
where *parse-es-cpts-i2* [] *es rlst* = *rlst* |
parse-es-cpts-i2 (*x*#*xs*) *es rlst* =
(*if getspc-es x* = *EvtSys es* ∧ *length xs* > *0*
∧ (*getspc-es* (*xs*!*0*) ≠ *EvtSys es*) *then*
parse-es-cpts-i2 xs es (*rlst*@[[*x*]])

else
parse-es-cpts-i2 xs es (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*x*])))

lemma *concat-list-lemma-take-n* [*rule-format*]:
$[\![esl = concat\ lst;\ i \leq length\ lst]\!] \Longrightarrow$
$\exists\, k.\ k \leq length\ esl \wedge take\ k\ esl = concat\ (take\ i\ lst)$
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*]:
$[\![esl = concat\ lst;\ i \leq length\ lst]\!] \Longrightarrow$
$\exists\, k.\ k \leq length\ esl \wedge k = length\ (concat\ (take\ i\ lst)) \wedge take\ k\ esl = concat\ (take\ i\ lst)$
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*

$\Longrightarrow \exists\, k \le length\ esl.\ k = length\ (concat\ (take\ ii\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ ii\ lst)$
and $a1$: $esl = concat\ lst$
and $a2$: $Suc\ ii \le length\ lst$
then have $\exists\, k \le length\ esl.\ k = length\ (concat\ (take\ ii\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ ii\ lst)$
using *Suc-leD* **by** *blast*
then obtain k **where** $a3$: $k \le length\ esl \wedge k = length\ (concat\ (take\ ii\ lst))$
$\wedge\ take\ k\ esl = concat\ (take\ ii\ lst)$
by *auto*
from *a2* **have** $a4$: $concat\ (take\ (Suc\ ii)\ lst) = concat\ (take\ ii\ lst)\ @\ lst!ii$
by (*simp add*: *take-Suc-conv-app-nth*)
with *a3* **have** $concat\ (take\ (Suc\ ii)\ lst) = take\ (k + length\ (lst!ii))\ esl$
by (*metis Cons-nth-drop-Suc Suc-le-lessD a2 append-eq-conv-conj append-take-drop-id concat.simps*(*2*) *concat-append p0 take-add*)
then show *?case* **by** (*metis a2 concat-list-lemma-take-n length-take min.absorb2 p0*)
qed
qed

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

esl ! $Suc\ i = lt$! k ! $Suc\ j$))
{
fix *esl*
assume $b0$: $esl = concat\ (l \# lt)$
and $b1$: $\forall i{<}length\ (l \# lt).\ 0 < length\ ((l \# lt)\ !\ i)$

{
fix *i*
assume $c0$: $Suc\ i < length\ esl$
then have $\exists k\ j.\ Suc\ k < length\ (l \# lt) \wedge$
$Suc\ j < length\ ((l \# lt)\ !\ k\ @\ [(l \# lt)\ !\ Suc\ k\ !\ 0]) \wedge$
$esl\ !\ i = ((l \# lt)\ !\ k\ @\ [(l \# lt)\ !\ Suc\ k\ !\ 0])\ !\ j \wedge$
$esl\ !\ Suc\ i = ((l \# lt)\ !\ k\ @\ [(l \# lt)\ !\ Suc\ k\ !\ 0])\ !\ Suc\ j \vee$
$Suc\ k = length\ (l \# lt) \wedge$
$Suc\ j < length\ ((l \# lt)\ !\ k) \wedge esl\ !\ i = (l \# lt)\ !\ k\ !\ j \wedge esl\ !\ Suc$
$i = (l \# lt)\ !\ k\ !\ Suc\ j$
proof(*cases* $lt = []$)
assume $d0$: $lt = []$
with $b0$ **have** $esl = l$ **by** *auto*
with $b0\ c0$ **have** $Suc\ 0 = length\ (l \# []) \wedge$
$Suc\ i < length\ ((l \# [])\ !\ 0) \wedge esl\ !\ i = (l \# [])\ !\ 0\ !\ i \wedge esl\ !\ Suc$
$i = (l \# [])\ !\ 0\ !\ Suc\ i$
by *simp*
with $d0$ **show** *?thesis* **by** *auto*
next
assume $d0$: $lt \neq []$
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 $e00$: $\neg(Suc\ i < length\ (l@[(l \# lt)\ !\ Suc\ 0!0]))$
then have $e0$: $Suc\ i \geq length\ (l@[(l \# lt)\ !\ Suc\ 0!0])$ **by** *simp*
from $b0$ **have** $\exists esl1.\ esl = l@esl1 \wedge esl1 = concat\ lt$ **by** *simp*
then obtain *esl1* **where** $e1$: $esl = l@esl1 \wedge esl1 = concat\ lt$ **by**
auto
with $a0\ b1$ **have** $e2$: $\forall i.\ Suc\ i < length\ esl1 \longrightarrow$
$(\exists k\ j.\ Suc\ k < length\ lt \wedge$
$Suc\ j < length\ (lt\ !\ k\ @\ [lt\ !\ Suc\ k\ !\ 0]) \wedge$
$esl1\ !\ i = (lt\ !\ k\ @\ [lt\ !\ Suc\ k\ !\ 0])\ !\ j \wedge esl1\ !\ Suc\ i = (lt$
$!\ k\ @\ [lt\ !\ Suc\ k\ !\ 0])\ !\ Suc\ j \vee$
$Suc\ k = length\ lt \wedge Suc\ j < length\ (lt\ !\ k) \wedge esl1\ !\ i = lt$
$!\ k\ !\ j \wedge esl1\ !\ Suc\ i = lt\ !\ k\ !\ Suc\ j)$
by *auto*

from *c0 e0 e00 e1* **have** *e3*: *esl*!*i* = *esl1*!(*i*−*length l*)
  by (*simp add*: *length-append-singleton nth-append*)
from *c0 e0 e00 e1* **have** *e4*: *esl*!*Suc i* = *esl1*!(*Suc i* − *length l*)
  by (*simp add*: *length-append-singleton less-Suc-eq nth-append*)
from *c0 e0 e00 e1* **have** *e5*: *Suc* (*i*−*length l*) < *length esl1*
  using *Suc-le-mono add.commute le-SucI length-append*
  length-append-singleton less-diff-conv2 **by** *auto*
with *e2* **have** ∃ *k j*. *Suc k* < *length lt* ∧
  Suc j < *length* (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ∧
  esl1 ! (*i*−*length l*) = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *j* ∧ *esl1* !
Suc (*i*−*length l*) = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *Suc j* ∨
  Suc k = *length lt* ∧ *Suc j* < *length* (*lt* ! *k*) ∧ *esl1* !
(*i*−*length l*) = *lt* ! *k* ! *j* ∧ *esl1* ! *Suc* (*i*−*length l*) = *lt* ! *k* ! *Suc j*
  by *auto*
then obtain *k* **and** *j* **where** *Suc k* < *length lt* ∧
  Suc j < *length* (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ∧
  esl1 ! (*i*−*length l*) = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *j* ∧ *esl1* !
Suc (*i*−*length l*) = (*lt* ! *k* @ [*lt* ! *Suc k* ! *0*]) ! *Suc j* ∨
  Suc k = *length lt* ∧ *Suc j* < *length* (*lt* ! *k*) ∧ *esl1* !
(*i*−*length l*) = *lt* ! *k* ! *j* ∧ *esl1* ! *Suc* (*i*−*length l*) = *lt* ! *k* ! *Suc j*
  by *auto*

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

lemma *concat-list-lemma2* [*rule-format*]:
∀ *esl lst*. *esl* = *concat lst* ⟶
  (∀ *i* < *length lst*. (*take* (*length* (*lst*!*i*)) (*drop* (*length* (*concat* (*take i lst*)))
esl) = *lst* ! *i*))
proof −
{
fix *lst*
have ∀ *esl*. *esl* = *concat lst* ⟶
  (∀ *i* < *length lst*. (*take* (*length* (*lst*!*i*)) (*drop* (*length* (*concat* (*take i lst*)))
esl) = *lst* ! *i*))
proof(*induct lst*)
  case *Nil* **then show** *?case* **by** *simp*
next
  case (*Cons l lt*)

assume *a0*[*rule-format*]: ∀ *esl*. *esl* = *concat lt* ⟶
(∀ *i*<*length lt*. *take* (*length* (*lt* ! *i*)) (*drop* (*length* (*concat* (*take i lt*))) *esl*) = *lt* ! *i*)
{
fix *esl*
assume *b0*: *esl* = *concat* (*l* # *lt*)
let *?esl* = *concat lt*
from *b0* **have** *b1*: *esl* = *l* @ *?esl* **by** *auto*
{
fix *i*
assume *c0*: *i*<*length* (*l* # *lt*)
have *take* (*length* ((*l* # *lt*) ! *i*)) (*drop* (*length* (*concat* (*take i* (*l* # *lt*)))) *esl*) = (*l* # *lt*) ! *i*
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
then show *?thesis* **by** (*simp add*: *b0 d0*)
next
assume *d0*: *i* ≠ *0*
with *c0* **have** *take* (*length* (*lt* ! (*i*−*1*))) (*drop* (*length* (*concat* (*take* (*i*−*1*) *lt*))) *?esl*) = *lt* ! (*i*−*1*)
using *a0*[*of ?esl i*−*1*] **by** (*metis One-nat-def leI less-Suc0 less-diff-conv2 list.size*(*4*))
moreover
from *d0 c0* **have** *lt* ! (*i* − *1*) = (*l* # *lt*) ! *i* **by** (*simp add*: *nth-Cons'*)
moreover
from *b0 b1 d0 c0* **have** *drop* (*length* (*concat* (*take* (*i*−*1*) *lt*))) *?esl*
= *drop* (*length* (*concat* (*take i* (*l* # *lt*)))) *esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append drop-Cons'*)
ultimately show *?thesis* **by** *simp*
qed
}
}
then show *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *concat-list-lemma3* [*rule-format*]:
⟦*esl* = *concat lst*; *i* < *length lst*; *length* (*lst*!*i*) > *1*⟧ ⟹
∃ *k j*. *k* = *length* (*concat* (*take i lst*)) ∧ *j* = *length* (*concat* (*take* (*Suc i*) *lst*)) ∧
k ≤ *length esl* ∧ *j* ≤ *length esl* ∧ *k* < *j* ∧ *drop k* (*take j esl*) = *lst* ! *i*
proof −
assume *p0*: *esl* = *concat lst*
and *p1*: *i* < *length lst*
and *p2*: *length* (*lst*!*i*) > *1*
then have *a1*: *take* (*length* (*lst*!*i*)) (*drop* (*length* (*concat* (*take i lst*))) *esl*) =

lst ! *i*
using *concat-list-lemma2* **by** *auto*
let *?k* = *length* (*concat* (*take i lst*))
let *?j* = *length* (*concat* (*take* (*Suc i*) *lst*))
from *p0 p1 p2* **have** *a10*: *drop ?k* (*take ?j esl*) = *lst* ! *i*
proof −
have *length* (*lst* ! *i*) + *length* (*concat* (*take i lst*)) = *length* (*concat* (*take* (*Suc i*) *lst*))
by (*simp add*: *p1 take-Suc-conv-app-nth*)
then show *?thesis*
by (*metis* (*full-types*) *a1 take-drop*)
qed
have *a2*: *?j* − *?k* = *length* (*lst*!*i*) **by** (*simp add*: *p1 take-Suc-conv-app-nth*)
have *a3*: *?j* = *?k* + *length* (*lst*!*i*) **by** (*simp add*: *p1 take-Suc-conv-app-nth*)
moreover
from *p0 p1* **have** *?k* ≤ *length esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append nat-le-linear take-all*)
moreover
from *p0 p1* **have** *?j* ≤ *length esl*
by (*metis append-eq-conv-conj append-take-drop-id concat-append nat-le-linear take-all*)
moreover
from *a3 p2* **have** *?k* < *?j* **using** *a2 diff-is-0-eq leI not-less0* **by** *linarith*
ultimately have *?k* ≤ *length esl* ∧ *?j* ≤ *length esl* ∧ *?k* < *?j* ∧ *drop ?k* (*take ?j esl*) = *lst* ! *i*
using *a10* **by** *simp*
then show *?thesis* **by** *blast*
qed

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

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

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

from *p0 p1 p2* **have** $\exists k.\ k \leq$ *length esl* $\wedge$ *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* $\leq$ *length esl* $\wedge$ *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 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*⟧ $\Longrightarrow$
$\exists k\ j.\ k$ = *length* (*concat* (*take i lst*)) $\wedge$ *j* = *Suc* (*length* (*concat* (*take* (*Suc i*) *lst*))) $\wedge$
k $\leq$ *length esl* $\wedge$ *j* $\leq$ *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*) > *0*
then have $\exists k.\ k \leq$ *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* $\leq$ *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** $\exists k.\ k \leq$ *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* $\leq$ *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* (*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** $\exists k.\ k \leq$ *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* $\leq$ *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 $\leq$ *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 $\exists k.\ k \leq$ *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* $\leq$ *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** $\exists k.\ k \leq$ *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* $\leq$ *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* $\leq$ *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* $\leq$ *length esl* $\wedge$ *k1* = *length* (*concat* (*take i lst*))

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

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

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

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

qed

lemma *parse-es-cpts-i2-concat*:
$\forall$ *esl rlst es*. *esl*$\in$*cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
$\longrightarrow$ *concat* (*parse-es-cpts-i2 esl es rlst*) = *concat rlst* @ *esl*
proof −
{
fix *esl*
have $\forall$ *rlst es*. *esl*$\in$*cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ [] $\longrightarrow$ *concat* (*parse-es-cpts-i2 esl es rlst*) = *concat rlst* @ *esl*
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *rlst es*. *esl1* $\in$ *cpts-es* Γ $\wedge$ *rlst* $\neq$ [] $\longrightarrow$ *concat* (*parse-es-cpts-i2 esl1 es rlst*) = *concat rlst* @ *esl1*
then show *?case*
proof −
{
fix *rlst es*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ $\wedge$ (*rlst*::(('*l*,'*k*,'*s*,'*prog*) *esconfs*) *list*) $\neq$ []
have *concat* (*parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*) = *concat rlst* @ (*esc* # *esl1*)
proof(*cases getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*)
assume *c0*: *getspc-es esc* = *EvtSys es* $\wedge$ *length esl1* > *0* $\wedge$ *getspc-es*

$(esl1!0) \neq EvtSys\ es$
then have *c1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])
by *simp*
from *b0* **have** *c2*: *rlst*@[[*esc*]] ≠ [] **by** *simp*
from *b0 c0* **have** *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
with *a0 c2* **have** *c3*: *concat* (*parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])) = *concat* (*rlst*@[[*esc*]]) @ *esl1* **by** *simp*
have *concat rlst* @ (*esc* # *esl1*) = *concat* (*rlst*@[[*esc*]]) @ *esl1* **by** *auto*
with *c1 c3* **show** *?thesis* **by** *presburger*
next
assume *c0*: ¬(*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* Γ **using** *cpts-es-dropi* **by** *force*
from *b0* **have** *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) ≠ [] **by** *simp*
with *a0 d1* **have** *d2*: *concat* (*parse-es-cpts-i2 esl1 es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))) =
concat (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) @ *esl1* **by** *auto*
from *b0* **have** *d3*: *concat rlst* @ (*esc* # *esl1*) = *concat* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) @ *esl1*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI append-assoc append-butlast-last-id concat.simps*(*2*) *concat-append length-butlast list-update-length self-append-conv2*)

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

lemma *parse-es-cpts-i2-concat1*:
$esl \in cpts\text{-}es\ \Gamma \Longrightarrow concat\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]) = esl$
by (*simp add*: *parse-es-cpts-i2-concat*)

lemma *parse-es-cpts-i2-lst0*:
$\forall\, esl\ l1\ l2\ es.\ esl \in cpts\text{-}es\ \Gamma \wedge (l2{::}(('l,'k,'s,'prog)\ esconfs)\ list) \neq []$
$\longrightarrow parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ (l1@l2) = l1@(parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ l2)$
proof −
{
fix *esl*
have $\forall\, l1\ l2\ es.\ esl \in cpts\text{-}es\ \Gamma \wedge (l2{::}(('l,'k,'s,'prog)\ esconfs)\ list) \neq []$
$\longrightarrow parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ (l1@l2) = l1@(parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es$
$l2)$
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall\, l1\ l2\ es.\ esl1 \in cpts\text{-}es\ \Gamma \wedge (l2{::}(('l,'k,'s,'prog)\ esconfs)\ list)$
$\neq []$
$\longrightarrow parse\text{-}es\text{-}cpts\text{-}i2\ esl1\ es\ (l1\ @l2) = l1\ @\ parse\text{-}es\text{-}cpts\text{-}i2$
$esl1\ es\ l2$
show *?case*
proof −
{
fix *l1 l2 es*
assume *b0*: $esc\ \#\ esl1 \in cpts\text{-}es\ \Gamma$
and *b1*: $(l2{::}(('l,'k,'s,'prog)\ esconfs)\ list) \neq []$
have $parse\text{-}es\text{-}cpts\text{-}i2\ (esc\ \#\ esl1)\ es\ (l1\ @\ l2) = l1\ @\ parse\text{-}es\text{-}cpts\text{-}i2$
$(esc\ \#\ esl1)\ es\ l2$
proof(*cases* $esl1 = []$)
assume *c0*: $esl1 = []$
then have $parse\text{-}es\text{-}cpts\text{-}i2\ (esc\ \#\ [])\ es\ (l1\ @\ l2) =$
$parse\text{-}es\text{-}cpts\text{-}i2\ []\ es\ (list\text{-}update\ (l1\ @\ l2)\ (length\ (l1\ @\ l2)$
$-\ 1)\ (last\ (l1\ @\ l2)\ @\ [esc]))$
by *simp*
then have *c1*: $parse\text{-}es\text{-}cpts\text{-}i2\ (esc\ \#\ [])\ es\ (l1\ @\ l2) =$
$list\text{-}update\ (l1\ @\ l2)\ (length\ (l1\ @\ l2) - 1)\ (last\ (l1\ @\ l2)$
$@\ [esc])$

by *simp*
with *b1* **have** *c2*: *parse-es-cpts-i2* (*esc* # []) *es* (*l1* @ *l2*) =
l1 @ (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*]))
by (*smt append-butlast-last-id append-is-Nil-conv butlast-append butlast-list-update last-appendR last-list-update list-update-nonempty*)
have *l1* @ *parse-es-cpts-i2* (*esc* # []) *es l2* =
l1 @ *parse-es-cpts-i2* [] *es* (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*])) **by** *simp*
then have *l1* @ *parse-es-cpts-i2* (*esc* # []) *es l2* =
l1 @ (*list-update l2* (*length l2* − *1*) (*last l2* @ [*esc*])) **by** *simp*
with *c0 c2* **show** *?thesis* **by** *simp*
next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* ∈ *cpts-es* Γ **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*:
$\forall$ *esl l1 l2 es. esl*$\in$*cpts-es* Γ $\wedge$ (*l2*::(('l,'k,'s,'prog) *esconfs*) *list*) $\neq$ []
$\longrightarrow$ *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*: $\forall$ *esl elst rlst es l. esl*$\in$*cpts-es* Γ $\wedge$ *rlst* = [*l*] $\wedge$ *elst* = *parse-es-cpts-i2 esl es rlst*
$\longrightarrow$ ($\exists$ *i*$\leq$*length* (*elst*!*0*). *take i* (*elst*!*0*) = *l*)
proof –
{
fix *esl*
have $\forall$ *elst rlst es l. esl*$\in$*cpts-es* Γ $\wedge$ *rlst* = [*l*] $\wedge$ *elst* = *parse-es-cpts-i2 esl es rlst*
$\longrightarrow$ ($\exists$ *i*$\leq$*length* (*elst*!*0*). *take i* (*elst*!*0*) = *l*)
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *elst rlst es l. esl1* $\in$ *cpts-es* Γ $\wedge$ *rlst* = [*l*] $\wedge$ *elst* = *parse-es-cpts-i2 esl1 es rlst*
$\longrightarrow$ ($\exists$ *i*$\leq$*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l*)
show *?case*
proof –
{
fix *elst rlst es l*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ
and *b1*: *rlst* = [*l*]
and *b2*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*
have $\exists$ *i*$\leq$*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l*
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
with *b2* **have** *c1*: *elst* = *parse-es-cpts-i2* [] *es* (*list-update rlst* (*length rlst* – *1*) (*last rlst* @ [*esc*]))
by *simp*
then have *elst* = *list-update rlst* (*length rlst* – *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b1* **have** *c2*: *elst* = [*l*@[*esc*]] **by** *simp*
then show *?thesis* **by** (*metis butlast-conv-take butlast-snoc linear*

nth-Cons-0 take-all)

next

assume *c0*: ¬(*esl1* = [])

with *b0* **have** *c1*: *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*

from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*

by (*meson neq-Nil-conv*)

show *?thesis*

proof(*cases getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)

assume *d0*: *getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*

with *c2* **have** *d01*: *getspc-es ec1* ≠ *EvtSys es* **by** *simp*

from *d0* **have** *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* = *parse-es-cpts-i2 esl1 es* (*rlst*@[[*esc*]])

by *simp*

with *b1 b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) **by** *simp*

from *c1* **have** *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) = [*l*]@*parse-es-cpts-i2 esl1 es* ([[*esc*]])

using *parse-es-cpts-i2-lst* **by** *blast*

with *d2* **have** *elst* = [*l*] @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*

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

next

assume *d0*: ¬(*getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)

then have *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst* =
parse-es-cpts-i2 esl1 es (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *auto*

with *b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))

by *simp*

with *b1* **have** *elst* = *parse-es-cpts-i2 esl1 es* ([*l* @ [*esc*]]) **by** *simp*

with *a0 c1* **have** ∃ *i*≤*length* (*elst* ! *0*). *take i* (*elst* ! *0*) = *l* @ [*esc*] **by** *simp*

then obtain *i* **where** *i*≤*length* (*elst* ! *0*) ∧ *take i* (*elst* ! *0*) = *l* @ [*esc*] **by** *auto*

then show *?thesis* **by** (*metis* (*no-types*, *lifting*) *butlast-snoc butlast-take diff-le-self dual-order.trans*)

qed

qed

}

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

qed

qed

}

then show *?thesis* **by** *blast*

qed

lemma *parse-es-cpts-i2-start-withlen* [*simp*]:
$\forall$ *esl elst rlst es l. esl*$\in$*cpts-es* $\Gamma \wedge$ *rlst* $\neq$ [] $\wedge$ *elst* = *parse-es-cpts-i2 esl es rlst* $\longrightarrow$
($\forall$ *i. i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$
length (*elst*!*i*) $\geq$ *2* $\wedge$ *getspc-es* (*elst*!*i*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*1*) $\neq$ *EvtSys es*)
proof −
{
fix *esl*
have $\forall$ *elst rlst es l. esl*$\in$*cpts-es* $\Gamma \wedge$ *rlst* $\neq$ [] $\wedge$ *elst* = *parse-es-cpts-i2 esl es rlst* $\longrightarrow$
($\forall$ *i. i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$
length (*elst*!*i*) $\geq$ *2* $\wedge$ *getspc-es* (*elst*!*i*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*1*) $\neq$ *EvtSys es*)
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *elst rlst es l. esl1* $\in$ *cpts-es* $\Gamma \wedge$ *rlst* $\neq$ [] $\wedge$ *elst* = *parse-es-cpts-i2 esl1 es rlst* $\longrightarrow$
($\forall$ *i. i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$
length (*elst*!*i*) $\geq$ *2* $\wedge$ *getspc-es* (*elst* ! *i* ! *0*) = *EvtSys es*
$\wedge$ *getspc-es* (*elst* ! *i* ! *1*) $\neq$ *EvtSys es*)
then show *?case*
proof −
{
fix *elst rlst es l*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ
and *b1*: *rlst* $\neq$ []
and *b2*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es rlst*
have $\forall$ *i. i* $\geq$ *length rlst* $\wedge$ *i* < *length elst* $\longrightarrow$ *length* (*elst*!*i*) $\geq$ *2* $\wedge$ *getspc-es* (*elst* ! *i* ! *0*) = *EvtSys es*
$\wedge$ *getspc-es* (*elst* ! *i* ! *1*) $\neq$ *EvtSys es*
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
then have *c1*: *parse-es-cpts-i2* (*esc* # []) *es rlst* =
parse-es-cpts-i2 [] *es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*])) **by** *simp*
have *c2*: *parse-es-cpts-i2* [] *es* (*list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]))
= *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b2 c0 c1* **have** *elst* = *list-update rlst* (*length rlst* − *1*) (*last rlst* @ [*esc*]) **by** *simp*
with *b1* **show** *?thesis* **by** *auto*
next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* $\in$ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*

by (*meson neq-Nil-conv*)
show *?thesis*
proof(*cases getspc-es esc = EvtSys es* ∧ *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** *blast*
with *d2* **have** *d3*: *elst* = *rlst* @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*
show *?thesis*
proof(*cases esl2* = [])
assume *e0*: *esl2* = []
with *c2* **have** *e1*: *elst* = *rlst* @ *parse-es-cpts-i2* [] *es* (*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]] @ [*ec1*]))
using *b2 d1* **by** *auto*
then have *elst* = *rlst* @ (*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]] @ [*ec1*]))
by *simp*
then have *elst* = *rlst* @ ([[*esc*] @ [*ec1*]]) **by** *simp*
with *d0 d01* **show** *?thesis* **using** *leD le-eq-less-or-eq* **by** *auto*
next
assume *e0*: ¬(*esl2* = [])

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

from *c2 d01 d3* **have** *elst* = *rlst* @ *parse-es-cpts-i2 esl2 es* (*list-update* [[*esc*]] (*length* [[*esc*]] − *1*) (*last* [[*esc*]] @ [*ec1*])) **by** *simp*
then have *e2*: *elst* = *rlst* @ *parse-es-cpts-i2 esl2 es* [[*esc*]@[*ec1*]] **by** *simp*
with *d3* **have** *e3*: *?elst2* = *parse-es-cpts-i2 esl2 es* [[*esc*]@[*ec1*]] **by** *simp*
from *c1 c2 e0* **have** *esl2*∈*cpts-es* Γ **using** *cpts-es-dropi* **by** *force*

```
                    with e3 have e4: ∃ i≤length (?elst2!0). take i (?elst2!0) =
[esc]@[ec1]
                    using parse-es-cpts-i2-fst by blast
                   with d0 d01 e1 e2 e3 show ?thesis
                    proof -
                    {
                     fix i
                     assume f0: length rlst ≤ i ∧ i < length elst
                    have length (elst ! i) ≥ 2 ∧ getspc-es (elst ! i ! 0) = EvtSys
es
                            ∧ getspc-es (elst ! i ! 1) ≠ EvtSys es
                       proof(cases length rlst = i)
                         assume g0: length rlst = i
                             then have elst ! i = ?elst2!0 by (simp add: e2 e3
nth-append)
                         with e4 show ?thesis
                                  by (metis (no-types, lifting) One-nat-def Suc-1
butlast-snoc
                                         butlast-take c2 d0 diff-Suc-1 length-Cons
length-append-singleton
                             length-take lessI list.size(3) min.absorb2 nth-Cons-0
                               nth-append-length nth-take)
                       next
                         assume g0: ¬ (length rlst = i)
                         with f0 have length rlst < i ∧ 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: ¬(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 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*]:
$[\![esl \in cpts\text{-}es\ \Gamma;\ rlst \neq [];\ elst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ rlst]\!] \Longrightarrow$
$\forall i.\ i \geq length\ rlst \wedge i < length\ elst \longrightarrow length\ (elst!i) \geq 2$
$\wedge\ getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \neq EvtSys\ es$
using *parse-es-cpts-i2-start-withlen* **by** *fastforce*

lemma *parse-es-cpts-i2-fstempty*: $[\![esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;\ esl \in cpts\text{-}es\ \Gamma;$
$rlst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]]\!] \Longrightarrow rlst!0 = []$
proof −
assume *p0*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p1*: $esl \in cpts\text{-}es\ \Gamma$
and *p2*: $rlst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]$
then have $rlst = parse\text{-}es\text{-}cpts\text{-}i2\ ((EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs)\ es\ ([[]]@[[(EvtSys\ es,\ s,\ x)]])$
by (*simp add:getspc-es-def*)
moreover from *p0 p1* **have** $(EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs \in cpts\text{-}es\ \Gamma$
using *cpts-es-dropi* **by** *force*
ultimately have $rlst = [[]]@\ parse\text{-}es\text{-}cpts\text{-}i2\ ((EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs)\ es\ ([[(EvtSys\ es,\ s,\ x)]])$
using *parse-es-cpts-i2-lst0* **by** *blast*
then show *?thesis* **by** *simp*
qed

lemma *parse-es-cpts-i2-concat3*: $[\![esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs;\ esl \in cpts\text{-}es\ \Gamma;$
$rlst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]]]\!] \Longrightarrow concat\ (tl\ rlst) = esl$
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*:
$\forall esl\ elst\ l\ es.\ esl \in cpts\text{-}es\ \Gamma \wedge elst = parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [l] \longrightarrow$
$\neg(length\ l > 1 \wedge getspc\text{-}es\ (last\ l) = EvtSys\ es \wedge getspc\text{-}es\ (esl!0) \neq EvtSys\ es) \longrightarrow$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ l \wedge$
$getspc\text{-}es\ (l!j) = EvtSys\ es \wedge getspc\text{-}es\ (l!Suc\ j) \neq EvtSys\ es) \longrightarrow$
$(\forall i.\ i < length\ elst \longrightarrow \neg(\exists j.\ j > 0 \wedge Suc\ j < length\ (elst!i) \wedge$
$getspc\text{-}es\ (elst!i!j) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!Suc\ j) \neq EvtSys\ es))$
proof −

{
fix *esl*
have $\forall$ *elst l es. esl*$\in$*cpts-es* $\Gamma \wedge$ *elst* = *parse-es-cpts-i2 esl es* [*l*] $\longrightarrow$
$\neg$(*length l* > *1* $\wedge$ *getspc-es* (*last l*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*0*) $\neq$ *EvtSys es*) $\longrightarrow$
$\neg$($\exists$*j. j* > *0* $\wedge$ *Suc j* < *length l* $\wedge$
getspc-es (*l*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*l*!*Suc j*) $\neq$ *EvtSys es*) $\longrightarrow$
($\forall$*i. i* < *length elst* $\longrightarrow$ $\neg$($\exists$*j. j* > *0* $\wedge$ *Suc j* < *length* (*elst*!*i*) $\wedge$
getspc-es (*elst*!*i*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*Suc j*) $\neq$ *EvtSys es*))
proof(*induct esl*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons esc esl1*)
assume *a0*: $\forall$ *elst l es. esl1*$\in$*cpts-es* $\Gamma \wedge$ *elst* = *parse-es-cpts-i2 esl1 es* [*l*] $\longrightarrow$
$\neg$(*length l* > *1* $\wedge$ *getspc-es* (*last l*) = *EvtSys es* $\wedge$ *getspc-es* (*esl1*!*0*) $\neq$ *EvtSys es*) $\longrightarrow$
$\neg$($\exists$*j. j* > *0* $\wedge$ *Suc j* < *length l* $\wedge$
getspc-es (*l*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*l*!*Suc j*) $\neq$ *EvtSys es*) $\longrightarrow$
($\forall$*i. i* < *length elst* $\longrightarrow$ $\neg$($\exists$*j. j* > *0* $\wedge$ *Suc j* < *length* (*elst*!*i*) $\wedge$
getspc-es (*elst*!*i*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*Suc j*) $\neq$ *EvtSys es*))
then show *?case*
proof –
{
fix *elst l es*
assume *b0*: *esc* # *esl1* $\in$ *cpts-es* Γ
and *b1*: *elst* = *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*]
and *b2*: $\neg$ (*length l* > *1* $\wedge$ *getspc-es* (*last l*) = *EvtSys es* $\wedge$ *getspc-es* ((*esc* # *esl1*) ! *0*) $\neq$ *EvtSys es*)
and *b3*: $\neg$ ($\exists$*j*>*0*. *Suc j* < *length l* $\wedge$ *getspc-es* (*l* ! *j*) = *EvtSys es* $\wedge$ *getspc-es* (*l* ! *Suc j*) $\neq$ *EvtSys es*)
have ($\forall$*i. i* < *length elst* $\longrightarrow$ $\neg$ ($\exists$*j*>*0*. *Suc j* < *length* (*elst* ! *i*) $\wedge$
getspc-es (*elst* ! *i* ! *j*) = *EvtSys es* $\wedge$ *getspc-es* (*elst* ! *i* ! *Suc j*) $\neq$ *EvtSys es*))
proof(*cases esl1* = [])
assume *c0*: *esl1* = []
then have *c1*: *parse-es-cpts-i2* (*esc* # []) *es* [*l*] =
parse-es-cpts-i2 [] *es* (*list-update* [*l*] (*length* [*l*] – *1*) (*last* [*l*] @ [*esc*])) **by** *simp*
have *c2*: *parse-es-cpts-i2* [] *es* (*list-update* [*l*] (*length* [*l*] – *1*) (*last* [*l*] @ [*esc*]))
= *list-update* [*l*] (*length* [*l*] – *1*) (*last* [*l*] @ [*esc*]) **by** *simp*
with *b1 c0 c1* **have** *elst* = *list-update* [*l*] (*length* [*l*] – *1*) (*last* [*l*] @

$[esc]$) **by** *simp*
then have *elst* = [*l* @ [*esc*]] **by** *simp*
with *b2 b3* **show** *?thesis* **by** (*smt Suc-eq-plus1-left Suc-lessD Suc-lessI diff-Suc-1*
dual-order.strict-trans last-conv-nth length-Cons length-append-singleton

less-antisym less-one list.size(*3*) *nat-neq-iff nth-Cons-0 nth-append nth-append-length*)

next
assume *c0*: ¬(*esl1* = [])
with *b0* **have** *c1*: *esl1* ∈ *cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *c0* **obtain** *esl2* **and** *ec1* **where** *c2*: *esl1* = *ec1* # *esl2*
by (*meson neq-Nil-conv*)
show *?thesis*
proof(*cases getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
assume *d0*: *getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*
with *c2* **have** *d01*: *getspc-es ec1* ≠ *EvtSys es* **by** *simp*
from *d0* **have** *d1*: *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*] = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]])
by *simp*
with *b1 b2* **have** *d2*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) **by** *simp*
from *c1* **have** *d4*: *parse-es-cpts-i2 esl1 es* ([*l*]@[[*esc*]]) = [*l*]@*parse-es-cpts-i2 esl1 es* ([[*esc*]])
using *parse-es-cpts-i2-lst0* **by** *blast*
with *d2* **have** *d3*: *elst* = [*l*] @ *parse-es-cpts-i2 esl1 es* ([[*esc*]]) **by** *simp*
let *?elst1* = *parse-es-cpts-i2 esl1 es* ([[*esc*]])
have ¬(*length* [*esc*] > *1* ∧ *getspc-es* (*last* [*esc*]) = *EvtSys es* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)
by *simp*
moreover have ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length* [*esc*] ∧
getspc-es ([*esc*]!*j*) = *EvtSys es* ∧ *getspc-es* ([*esc*]!*Suc j*) ≠ *EvtSys es*) **by** *simp*
ultimately have ∀ *i*. *i* < *length ?elst1* ⟶ ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length* (*?elst1*!*i*) ∧
getspc-es (*?elst1*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*?elst1*!*i*!*Suc j*) ≠ *EvtSys es*)
using *a0 c1* **by** *simp*
with *b3 d3* **show** *?thesis* **by** (*smt Nil-is-append-conv Nitpick.size-list-simp*(*2*)
One-nat-def Suc-diff-Suc Suc-less-eq append-Cons append-Nil
diff-Suc-1 diff-Suc-Suc list.sel(*3*) *not-gr0 nth-Cons'*)
next
assume *d0*: ¬(*getspc-es esc* = *EvtSys es* ∧ *length esl1* > *0* ∧ *getspc-es* (*esl1*!*0*) ≠ *EvtSys es*)

then have *parse-es-cpts-i2* (*esc* # *esl1*) *es* [*l*] =
parse-es-cpts-i2 esl1 es (*list-update* [*l*] (*length* [*l*] − *1*)
(*last* [*l*] @ [*esc*]))
by *auto*
with *b1* **have** *d1*: *elst* = *parse-es-cpts-i2 esl1 es* ([*l*@[*esc*]]) **by**
simp
show *?thesis*
proof(*cases length esl1* = *0*)
assume *e0*: *length esl1* = *0*
then have *e1*: *esl1* = [] **by** *simp*
with *d1* **have** *elst* = [*l*@[*esc*]] **by** *simp*
with *b2* **show** *?thesis* **using** *e1 c0* **by** *linarith*
next
assume *e0*: ¬(*length esl1* = *0*)
then have *length esl1* > *0* **by** *simp*
with *d0* **have** *e1*: ¬(*getspc-es esc* = *EvtSys es* ∧ *getspc-es*
(*esl1*!*0*) ≠ *EvtSys es*) **by** *simp*
then have ¬ (*1* < *length* (*l*@[*esc*]) ∧ *getspc-es* (*last* (*l*@[*esc*]))
= *EvtSys es*
∧ *getspc-es* (*esl1* ! *0*) ≠ *EvtSys es*) **by** *auto*
moreover from *b2 b3* **have** ¬ (∃ *j*>*0*. *Suc j* < *length* (*l*@[*esc*])
∧ *getspc-es* ((*l*@[*esc*]) ! *j*) = *EvtSys es* ∧
getspc-es ((*l*@[*esc*]) ! *Suc j*) ≠ *EvtSys es*)
by (*metis* (*no-types*, *hide-lams*) *Suc-neq-Zero diff-Suc-1*
last-conv-nth
length-append-singleton less-antisym list.size(*3*) *not-gr0*
not-less-eq
nth-Cons-0 nth-append zero-less-diff)
ultimately show *?thesis* **using** *a0 d1 c1* **by** *blast*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed
qed
}
then show *?thesis* **by** *blast*
qed

lemma *parse-es-cpts-i2-noent-mid*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *parse-es-cpts-i2 esl es* [[]]⟧ ⟹ ∀ *i*. *i* < *length* (*tl elst*) ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* ((*tl elst*)!*i*) ∧
getspc-es ((*tl elst*)!*i*!*j*) = *EvtSys es* ∧ *getspc-es* ((*tl elst*)!*i*!*Suc*
j) ≠ *EvtSys es*)
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ

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

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

```
    qed
  qed

lemma parse-es-cpts-i2-noent-mid-i:
    ⟦esl = (EvtSys es, s, x) # (EvtSeq e (EvtSys es), s1,x1) # xs; esl∈cpts-es Γ;
     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 Γ
      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 Γ 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  Γ
          parse-es-cpts-i2 esl es [[]]] by blast
    ultimately show ?thesis by (simp add: nth-append p4)
  qed

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

cat.simps(2))
from *c0 c1* **have** *concat* (*drop j elst*) ∈ *cpts-es* Γ **by** *simp*
with *c1 c2* **show** *?case*
using *cpts-es-dropi2*[*of concat* (*drop j elst*) Γ *length* (*elst* ! *j*)]
by (*smt List.nth-tl Suc-leI Suc-lessE concat-last-lm diff-Suc-1 drop.simps*(*1*)

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

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

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

lemma *parse-es-cpts-i2-in-cptes-i*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
∀ *i*. *Suc i* < *length elst* ⟶ (*elst*!*i*)@[*elst*!*Suc i*!*0*] ∈*cpts-es* Γ
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
then have *p3*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
from *p0 p1 p2* **have** *p4*: ∀ *i*. *i* < *length elst* ⟶ *length* (*elst*!*i*) ≥ *2*
using *parse-es-cpts-i2-start-aux*[*of esl es s x e s1 x1 xs* Γ *parse-es-cpts-i2 esl es* [[]]]
by *simp*

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

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

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

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

lemma *parse-es-cpts-i2-in-cptes-last*:
⟦*esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*; *esl*∈*cpts-es* Γ;
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧ ⟹
last elst ∈*cpts-es* Γ
proof −
assume *p0*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p1*: *esl*∈*cpts-es* Γ
and *p2*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
then have ∀ *i*. *i* < *length elst* ⟶ *concat* (*drop i elst*)∈*cpts-es* Γ
using *parse-es-cpts-i2-drop-cptes*[*of esl es s x e s1 x1 xs* Γ *elst*] **by** *fastforce*
then show *?thesis*

by (*metis* (*no-types*, *lifting*) *append-butlast-last-id append-eq-conv-conj concat.simps*(*1*) *concat.simps*(*2*) *diff-less length-butlast length-greater-0-conv*

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

lemma *evtsys-fst-ent*:
$[\![esl \in cpts\text{-}es\ \Gamma;\ getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es;\ Suc\ m \leq length\ esl;\ \exists\, i.\ i \leq m \wedge getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es]\!]$
$\Longrightarrow \exists\, i.\ (i < m \wedge getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es \wedge getspc\text{-}es\ (esl\ !\ Suc\ i) \neq EvtSys\ es)$
$\wedge (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es)$
proof −
assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
and *p1*: $getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es$
and *p2*: $Suc\ m \leq length\ esl$
and *p3*: $\exists\, i.\ i \leq m \wedge getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es$
have $\forall m.\ esl \in cpts\text{-}es\ \Gamma \wedge getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es \wedge Suc\ m \leq length\ esl$
$\wedge (\exists\, i.\ i \leq m \wedge getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es)$
$\longrightarrow (\exists\, i.\ (i < m \wedge getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es \wedge getspc\text{-}es\ (esl\ !\ Suc\ i) \neq EvtSys\ es)$
$\wedge (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es))$
proof −
{
fix *m*
assume *a0*: $esl \in cpts\text{-}es\ \Gamma$
and *a1*: $getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es$
and *a2*: $Suc\ m \leq length\ esl$
and *a3*: $\exists\, i.\ i \leq m \wedge getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es$
then have $\exists\, i.\ (i < m \wedge getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$
$\wedge getspc\text{-}es\ (esl\ !\ Suc\ i) \neq EvtSys\ es)$
$\wedge (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es)$
proof(*induct m*)
case *0* **show** *?case* **using** *0.prems*(*4*) *p1* **by** *auto*
next
case (*Suc n*)
assume *b0*: $esl \in cpts\text{-}es\ \Gamma \Longrightarrow$
$getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es \Longrightarrow$
$Suc\ n \leq length\ esl \Longrightarrow$
$\exists\, i{\leq}n.\ getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es \Longrightarrow$
$\exists\, i.\ (i < n \wedge getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$
$\wedge getspc\text{-}es\ (esl\ !\ Suc\ i) \neq EvtSys\ es)$
$\wedge (\forall j{<}i.\ getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es)$
and *b1*: $esl \in cpts\text{-}es\ \Gamma$
and *b2*: $getspc\text{-}es\ (esl\ !\ 0) = EvtSys\ es$
and *b3*: $Suc\ (Suc\ n) \leq length\ esl$
and *b4*: $\exists\, i{\leq}Suc\ n.\ getspc\text{-}es\ (esl\ !\ i) \neq EvtSys\ es$
show *?case*

proof(*cases* $\exists\, i{\le}n.$ *getspc-es* (*esl* ! *i*) $\neq$ *EvtSys es*)
  assume *c0*: $\exists\, i{\le}n.$ *getspc-es* (*esl* ! *i*) $\neq$ *EvtSys es*
  with *b0 b1 b2 b3* **have** $\exists\, i.$ ($i < n$ $\wedge$ *getspc-es* (*esl* ! *i*) = *EvtSys es*
    $\wedge$ *getspc-es* (*esl* ! *Suc i*) $\neq$ *EvtSys es*)
    $\wedge$ ($\forall\, j{<}i.$ *getspc-es* (*esl* ! *j*) = *EvtSys es*) **by** *simp*
  then show *?thesis* **using** *less-Suc-eq* **by** *auto*
next
  assume *c0*: $\neg$($\exists\, i{\le}n.$ *getspc-es* (*esl* ! *i*) $\neq$ *EvtSys es*)
  with *b4* **have** *getspc-es* (*esl* ! *Suc n*) $\neq$ *EvtSys es*
   using *le-SucE* **by** *auto*
  moreover from *c0* **have** $\forall\, j{<}n.$ *getspc-es* (*esl* ! *j*) = *EvtSys es* **by**
auto
  moreover from *c0* **have** *getspc-es* (*esl* ! *n*) = *EvtSys es* **by** *auto*
  ultimately show *?thesis* **by** *blast*
qed
qed
}
then show *?thesis* **by** *auto*
qed

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

lemma *rm-evtsys-in-cptse0*:
  $[\![$*esl*$\in$*cpts-es* Γ; *length esl* > *0*; $\exists\, e.$ *getspc-es* (*esl*!*0*) = *EvtSeq e* (*EvtSys es*);
  $\neg$($\exists\, j.$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc*
j) $\neq$ *EvtSys es*) $]\!]$
   $\Longrightarrow$ *rm-evtsys esl* $\in$ *cpts-ev* Γ
proof –
 assume *p0*: *esl*$\in$*cpts-es* Γ
  and *p1*: *length esl* > *0*
  and *p2*: $\exists\, e.$ *getspc-es* (*esl*!*0*) = *EvtSeq e* (*EvtSys es*)
  and *p3*: $\neg$($\exists\, j.$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es*
(*esl*!*Suc j*) $\neq$ *EvtSys es*)
 have $\forall$ *esl e es* .*esl*$\in$*cpts-es* Γ $\wedge$ *length esl* > *0* $\wedge$ ($\exists\, e.$ *getspc-es* (*esl*!*0*) =
EvtSeq e (*EvtSys es*)) $\wedge$
  $\neg$($\exists\, j.$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc*
j) $\neq$ *EvtSys es*)
   $\longrightarrow$ *rm-evtsys esl* $\in$ *cpts-ev* Γ
  proof –
  {
   fix *esl e es*
   assume *a0*: *esl*$\in$*cpts-es* Γ
    and *a1*: *length esl* > *0*
    and *a2*: $\exists\, e.$ *getspc-es* (*esl*!*0*) = *EvtSeq e* (*EvtSys es*)
     and *a3*: $\neg$($\exists\, j.$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$
getspc-es (*esl*!*Suc j*) $\neq$ *EvtSys es*)
   from *a0 a1 a2 a3* **have** *rm-evtsys esl* $\in$ *cpts-ev* Γ

proof(*induct esl*)
case (*CptsEsOne es1 s x*)
show *?case*
proof(*induct es1*)
case (*EvtSeq x1 es1*)
have *rm-evtsys* [(*EvtSeq x1 es1*, *s*, *x*)] = [(*x1*, *s*, *x*)]
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
then show *?case* **by** (*simp add*: *cpts-ev*.*CptsEvOne*)
next
case (*EvtSys xa*)
have *rm-evtsys* [(*EvtSys xa*, *s*, *x*)] = [(*AnonyEvent fin-com*, *s*, *x*)]
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
then show *?case* **by** (*simp add*: *cpts-ev*.*CptsEvOne*)
qed
next
case (*CptsEsEnv es1 t x xs s y*)
assume *b0*: (*es1*, *t*, *x*) # *xs* ∈ *cpts-es* Γ
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* Γ
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* Γ **using** *b1 b3* **by** *blast*

then have *b4*: *rm-evtsys1* (*es1*, *t*, *x*) # *rm-evtsys xs* ∈ *cpts-ev* Γ **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* Γ
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* Γ
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
ultimately show *?case* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
next
case (*EvtSys xa*)
assume *c0*: *rm-evtsys1* (*EvtSys xa*, *t*, *x*) # *rm-evtsys xs* ∈ *cpts-ev* Γ
have *rm-evtsys* ((*EvtSys xa*, *s*, *y*) # (*EvtSys xa*, *t*, *x*) # *xs*) =
(*AnonyEvent fin-com*, *s*, *y*) # (*AnonyEvent fin-com*, *t*, *x*) # *rm-evtsys xs*
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
moreover from *c0* **have** (*AnonyEvent fin-com*,*t*, *x*) # *rm-evtsys xs* ∈ *cpts-ev* Γ
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
ultimately show *?case* **by** (*simp add*: *cpts-ev.CptsEvEnv*)
qed
next
case (*CptsEsComp e1 s1 x1 et e2 t1 y1 xs1*)
assume *b0*: Γ ⊢ (*e1*, *s1*, *x1*) −*es*−*et*→ (*e2*, *t1*, *y1*)
and *b1*: (*e2*, *t1*, *y1*) # *xs1* ∈ *cpts-es* Γ
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* Γ
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* $\wedge$
getspc-es $(((e1, s1, x1) \# (e2, t1, y1) \# xs1) ! Suc\ j) \neq$ *EvtSys es*)
have *b6*: *rm-evtsys* $((e1, s1, x1) \# (e2, t1, y1) \# xs1) =$
rm-evtsys1 $(e1, s1, x1) \#$ *rm-evtsys1* $(e2, t1, y1) \#$ *rm-evtsys xs1*
by (*simp add*:*rm-evtsys-def*)
from *b4* **obtain** e' **where** *getspc-es* $(((e1, s1, x1) \# (e2, t1, y1) \# xs1) ! 0) =$ *EvtSeq* e' (*EvtSys es*)
by *auto*
then have *b7*: $e1 =$ *EvtSeq* e' (*EvtSys es*) **by** (*simp add*:*getspc-es-def*)
show *?case*
proof(*cases* $\exists e.\ e2 =$ *EvtSeq e* (*EvtSys es*))
assume *c0*: $\exists e.\ e2 =$ *EvtSeq e* (*EvtSys es*)
then obtain *e* **where** *c1*: $e2 =$ *EvtSeq e* (*EvtSys es*) **by** *auto*
then have *c2*: $\exists e.$ *getspc-es* $(((e2, t1, y1) \# xs1) ! 0) =$ *EvtSeq e* (*EvtSys es*)
by (*simp add*:*getspc-es-def*)
moreover from *b5* **have** $\neg$ ($\exists j.$ *Suc j* $<$ *length* $((e2, t1, y1) \# xs1)$ $\wedge$
getspc-es $(((e2, t1, y1) \# xs1) ! j) =$ *EvtSys es* $\wedge$
getspc-es $(((e2, t1, y1) \# xs1) ! Suc\ j) \neq$ *EvtSys es*) **by** *force*
ultimately have *c3*: *rm-evtsys* $((e2, t1, y1) \# xs1) \in$ *cpts-ev* Γ **using** *b2* **by** *blast*
then have *c5*: *rm-evtsys1* $(e2, t1, y1) \#$ *rm-evtsys xs1* $\in$ *cpts-ev* Γ **by** (*simp add*:*rm-evtsys-def*)

from *b0 c1 b7* **have** $\exists t.\ \Gamma \vdash (e', s1, x1) -et-t\rightarrow (e, t1, y1)$
using *evtseq-tran-exist-etran* **by** *simp*
then obtain *t* **where** *c8*: $\Gamma \vdash (e', s1, x1) -et-t\rightarrow (e, t1, y1)$ **by** *auto*
from *b7* **have** *rm-evtsys1* $(e1, s1, x1) = (e', s1, x1)$
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
moreover from *c1* **have** *rm-evtsys1* $(e2, t1, y1) = (e, t1, y1)$
by (*simp add*:*rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
ultimately show *?thesis* **using** *b6 c8 c5* **using** *cpts-ev.CptsEvComp* **by** *fastforce*
next
assume *c0*: $\neg(\exists e.\ e2 =$ *EvtSeq e* (*EvtSys es*))
with *b0 b7* **have** *c1*: $e2 =$ *EvtSys es* **by** (*meson evtseq-tran-evtseq*)
then have *c11*: *rm-evtsys1* $(e2, t1, y1) \#$ *rm-evtsys xs1* $\in$ *cpts-ev* Γ
proof $-$
from *b5* **have** *d0*: $\neg$ ($\exists j.$ *Suc j* $<$ *length* $((e2, t1, y1) \# xs1) \wedge$
getspc-es $(((e2, t1, y1) \# xs1) ! j) =$ *EvtSys es* $\wedge$
getspc-es $(((e2, t1, y1) \# xs1) ! Suc\ j) \neq$ *EvtSys es*) **by**

force

have *d00*: $\forall j.\ j < length\ xs1 \longrightarrow getspc\text{-}es\ (xs1!j) = EvtSys\ es$
proof −
{
fix *j*
assume *e0*: $j < length\ xs1$
then have $getspc\text{-}es\ (xs1!j) = EvtSys\ es$
proof(*induct j*)
case *0* **from** *b1 c1 d0* **show** *?case*
using *getspc-es-def* **by** (*metis One-nat-def e0 fst-conv length-Cons less-one not-less-eq nth-Cons-0 nth-Cons-Suc*)
next
case (*Suc m*)
assume *f0*: $m < length\ xs1 \Longrightarrow getspc\text{-}es\ (xs1\ !\ m) = EvtSys\ es$
and *f1*: $Suc\ m < length\ xs1$
with *d0* **show** *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed
then have *d1*: $\forall j.\ j < length\ (rm\text{-}evtsys\ xs1) \longrightarrow getspc\text{-}e\ ((rm\text{-}evtsys\ xs1)!j) = AnonyEvent\ fin\text{-}com$
by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def getspc-e-def*)
from *c1* **have** *d2*: $rm\text{-}evtsys1\ (e2, t1, y1) = (AnonyEvent\ fin\text{-}com, t1, y1)$
by (*simp add:rm-evtsys1-def getspc-es-def gets-es-def getx-es-def getspc-e-def*)
with *d1* **have** $\forall i.\ i < length\ (rm\text{-}evtsys1\ (e2, t1, y1)\ \#\ rm\text{-}evtsys\ xs1) \longrightarrow getspc\text{-}e\ ((rm\text{-}evtsys1\ (e2, t1, y1)\ \#\ rm\text{-}evtsys\ xs1)!i) = AnonyEvent\ fin\text{-}com$
using *getspc-e-def less-Suc-eq-0-disj* **by** *force*
moreover have $length\ (rm\text{-}evtsys1\ (e2, t1, y1)\ \#\ rm\text{-}evtsys\ xs1) > 0$ **by** *simp*
ultimately show *?thesis* **using** *cpts-ev-same* **by** *blast*

qed
from *b7* **have** *c2*: $rm\text{-}evtsys1\ (e1, s1, x1) = (e', s1, x1)$
by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
from *c1* **have** *c3*: $rm\text{-}evtsys1\ (e2, t1, y1) = (AnonyEvent\ fin\text{-}com, t1, y1)$
by (*simp add:rm-evtsys-def rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)
from *b0 b7 c1* **have** $\exists t.\ \Gamma \vdash (e', s1, x1)\ {-et-t\rightarrow}\ (AnonyEvent\ fin\text{-}com, t1, y1)$

using *evtseq-tran-0-exist-etran* **by** *simp*
then obtain *t* **where** $\Gamma \vdash (e', s1, x1) -et-t\rightarrow (AnonyEvent\ fin\text{-}com, t1, y1)$ **by** *auto*
with *b6 c2 c3 c11* **show** *?thesis* **using** *cpts-ev.CptsEvComp* **by** *fastforce*
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p0 p1 p2 p3* **show** *?thesis* **by** *force*
qed

lemma *rm-evtsys-in-cptse*:
$[\![esl \in cpts\text{-}es\ \Gamma;\ esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs;$
$\Gamma \vdash (EvtSys\ es,\ s,\ x) -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1);$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es);$
$el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs)$
$]\!] \Longrightarrow$
$el \in cpts\text{-}ev\ \Gamma$
proof −
assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
and *p1*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p2*: $\Gamma \vdash (EvtSys\ es,\ s,\ x) -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)$
and *p3*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
and *p4*: $el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs)$
let $?esl1 = (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
from *p0 p1* **have** *a1*: $?esl1 \in cpts\text{-}es\ \Gamma$ **using** *cpts-es-dropi* **by** *force*
moreover have *a2*: $length\ ?esl1 > 0$ **by** *simp*
moreover have *a3*: $\exists e.\ getspc\text{-}es\ (?esl1\ !\ 0) = EvtSeq\ e\ (EvtSys\ es)$ **by** (*simp add:getspc-es-def*)
moreover from *p1 p3* **have** *a4*: $\neg\ (\exists j.\ Suc\ j < length\ ?esl1 \wedge getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (?esl1\ !\ Suc\ j) \neq EvtSys\ es)$ **by** *force*
ultimately have $?esl1 \in cpts\text{-}es\ \Gamma$ **using** *rm-evtsys-in-cptse0* **by** *blast*

with *a1 a2 a3 a4* **have** *a5*: $rm\text{-}evtsys\ ?esl1 \in cpts\text{-}ev\ \Gamma$ **using** *rm-evtsys-in-cptse0* **by** *blast*
have $rm\text{-}evtsys\ ?esl1 = rm\text{-}evtsys1\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ rm\text{-}evtsys\ xs$
by (*simp add:rm-evtsys-def*)
then have *a6*: $rm\text{-}evtsys\ ?esl1 = (ev,s1,x1)\ \#\ rm\text{-}evtsys\ xs$
by (*simp add:rm-evtsys1-def getspc-es-def gets-es-def getx-es-def*)

from *p2* **have** $\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) $-et-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*ev*, *s1*, *x1*)
using *evtsysent-evtent*[*of* Γ *es s x e k ev s1 x1*] **by** *auto*
with *p4 a6* **show** *?thesis* **using** *a5 cpts-ev.CptsEvComp* **by** *fastforce*
qed

lemma *fstent-nomident-e-sim-es-aux*:
$[\![$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
$\neg$($\exists$*j*. *j* > *0* $\wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*); *el*$\in$*cpts-ev* $\Gamma]\!]$ $\Longrightarrow$
$\forall$*i*. *i* > *0* $\wedge$ *i* < *length el* $\longrightarrow$
(*getspc-es* (*esl*!*i*) = *EvtSys es* $\wedge$ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
$\vee$ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*))
proof –
assume *p0*: *esl*$\in$*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: $\neg$($\exists$*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 *p3*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *p4*: *el*$\in$*cpts-ev* Γ
let *?el1* = *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
have *a1*: *length ?esl1* = *length ?el1* **using** *rm-evtsys-same-sx same-s-x-def* **by** *blast*
from *p0 p1* **have** *a2*: *?esl1*$\in$*cpts-es* Γ **using** *cpts-es-dropi* **by** *force*
from *p2* **have** *p2-1*: $\forall$*j*. *j* > *0* $\wedge$ *Suc j* < *length esl* $\longrightarrow$
getspc-es (*esl* ! *j*) = *EvtSys es* $\longrightarrow$ *getspc-es* (*esl* ! *Suc j*) = *EvtSys es*
using *noevtent-inmid-eq* **by** *auto*
have $\forall$*i*. *i* < *length ?el1* $\longrightarrow$
(*getspc-es* (*?esl1*!*i*) = *EvtSys es* $\wedge$ *getspc-e* (*?el1*!*i*) = *AnonyEvent fin-com*)

$\vee$ (*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* $\wedge$ *getspc-e* (*?el1*!*i*) = *AnonyEvent fin-com*)
$\vee$ (*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*: $j < length\ ?el1 \Longrightarrow getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es\ \wedge$
$getspc\text{-}e\ (?el1\ !\ j) = AnonyEvent\ fin\text{-}com\ \vee$
$getspc\text{-}es\ (?esl1\ !\ j) =$
$EvtSeq\ (getspc\text{-}e\ (?el1\ !\ j))\ (EvtSys\ es)$
and *c1*: $Suc\ j < length\ ?el1$
then have *c2*: $getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es\ \wedge$
$getspc\text{-}e\ (?el1\ !\ j) = AnonyEvent\ fin\text{-}com\ \vee$
$getspc\text{-}es\ (?esl1\ !\ j) =$
$EvtSeq\ (getspc\text{-}e\ (?el1\ !\ j))\ (EvtSys\ es)$ **by** *simp*
show *?case*
proof(*cases* $getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es\ \wedge$
$getspc\text{-}e\ (?el1\ !\ j) = AnonyEvent\ fin\text{-}com$)
assume *d0*: $getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es\ \wedge$
$getspc\text{-}e\ (?el1\ !\ j) = AnonyEvent\ fin\text{-}com$
with *p1 p2-1 a1* **have** *d1*: $getspc\text{-}es\ (?esl1\ !\ Suc\ j) = EvtSys\ es$
proof −
from *p1 d0* **have** $getspc\text{-}es\ (esl\ !\ Suc\ j) = EvtSys\ es$ **by** *simp*
moreover
from *p1 c1* **have** $0 < Suc\ j \wedge Suc\ (Suc\ j) < length\ esl$
using *a1* **by** *auto*
ultimately have $getspc\text{-}es\ (esl\ !\ Suc\ (Suc\ j)) = EvtSys\ es$
using *p2-1* **by** *simp*
with *p1* **show** *?thesis* **by** *simp*
qed
with *a1 c1* **have** *d2*: $getspc\text{-}e\ (?el1\ !\ Suc\ j) = AnonyEvent\ fin\text{-}com$
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSysrm **by** (*smt fst-conv nth-map*)
with *d1* **show** *?case* **by** *simp*
next
assume $\neg(getspc\text{-}es\ (?esl1\ !\ j) = EvtSys\ es\ \wedge$
$getspc\text{-}e\ (?el1\ !\ j) = AnonyEvent\ fin\text{-}com)$
with *c2* **have** *d0*: $getspc\text{-}es\ (?esl1\ !\ j) =$
$EvtSeq\ (getspc\text{-}e\ (?el1\ !\ j))\ (EvtSys\ es)$
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** $getspc\text{-}es\ (?esl1\ !\ j) = EvtSeq\ e\ (EvtSys\ es)$
by (*simp add:getspc-es-def getspc-e-def*)
moreover
from *e1* **have** $gets\text{-}e\ (?el1\ !\ j) = gets\text{-}es\ (?esl1\ !\ j)$
by (*simp add*: *Suc.prems less-or-eq-imp-le same-s-x-def*)
moreover
from *e1* **have** $getx\text{-}e\ (?el1\ !\ j) = getx\text{-}es\ (?esl1\ !\ j)$
by (*simp add*: *Suc.prems less-or-eq-imp-le same-s-x-def*)

ultimately show *?thesis*
using *d1 getspc-es-def gets-es-def getx-es-def gets-e-def getx-e-def*
by (*metis prod.collapse snd-conv*)
qed
then show *?case*
proof(*cases getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es*)
assume *e0*: *getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es*
then obtain *s2* **and** *x2* **where** *e1*: *?esl1* ! *Suc j* = (*EvtSys es*, *s2*,*x2*)
using *getspc-es-def* **by** (*metis fst-conv surj-pair*)
then have *e2*: *?el1* ! *Suc j* = (*AnonyEvent fin-com*, *s2*,*x2*)
using *getspc-es-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSysrm **by** (*metis Suc.prems a1 fst-conv nth-map snd-conv*)
with *e1* **have** *getspc-es* (*?esl1* ! *Suc j*) = *EvtSys es* $\wedge$
getspc-e (*?el1* ! *Suc j*) = *AnonyEvent fin-com*
using *getspc-es-def getspc-e-def* **by** (*metis fst-conv*)
then show *?thesis* **by** *simp*
next
assume *e0*: *getspc-es* (*?esl1* ! *Suc j*) $\neq$ *EvtSys es*
with *a1 a2 c1 d2* **have** $\exists$ *e1*. *getspc-es* (*?esl1* ! *Suc j*) = *EvtSeq e1* (*EvtSys es*)
using *evtseq-next-in-cpts getspc-es-def* **by** *fastforce*
then obtain *e1* **where** *e1*:*getspc-es* (*?esl1* ! *Suc j*) = *EvtSeq e1* (*EvtSys es*) **by** *auto*
with *a1 c1* **have** *getspc-e* (*?el1* ! *Suc j*) = *e1*
using *getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def*
gets-es-def getx-es-def EvtSeqrm **by** (*smt fstI nth-map*)
with *e1* **have** *getspc-es* (*?esl1* ! *Suc j*) =
EvtSeq (*getspc-e* (*?el1* ! *Suc j*)) (*EvtSys es*) **by** *simp*
then show *?thesis* **by** *simp*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed
with *p1 p2 p3 p4* **show** *?thesis* **by** (*metis* (*no-types*, *lifting*) *Suc-diff-1*
Suc-less-SucD length-Cons nth-Cons-pos)
qed

lemma *fstent-nomident-e-sim-es*:
$[\![$*esl*$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
$\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length esl* $\wedge$ *getspc-es* (*esl*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*esl*!*Suc j*) $\neq$ *EvtSys es*)$]\!]$ $\Longrightarrow$
$\exists$ *el e s x*. *el*$\in$*cpts-of-ev* Γ (*BasicEvent e*) *s x* $\wedge$ *e-sim-es esl el es e*
proof –
assume *p0*: *esl*$\in$*cpts-es* Γ

and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p3*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
from *p1* **have** ∃ *t*. Γ ⊢ (*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*: Γ ⊢ (*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*) ∧
Γ ⊢ (*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*) ∧
Γ ⊢ (*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** Γ ⊢ (*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* Γ
using *rm-evtsys-in-cptse* [*of esl* Γ *es s x ev s1 x1 xs*]
by (*metis estran.EvtOccur evtsysent-evtent0 noevtent-notran0*)
moreover have *e-sim-es esl ?el es e*
proof −
from *a3* **have** *b1*: *length esl* = *length ?el* **by** (*simp add*:*rm-evtsys-def*)
moreover
from *p1* **have** *b2*: *getspc-es* (*esl* ! *0*) = *EvtSys es* **by** (*simp add*:*getspc-es-def*)
moreover
have *b3*: *getspc-e* (*?el* ! *0*) = *BasicEvent e* **by** (*simp add*:*getspc-e-def*)
moreover
from *a3 b1* **have** *b4*: ∀ *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-es* (*esl*!*i*) = *EvtSys es* $\wedge$ *getspc-e* (*?el*!*i*) = *AnonyEvent fin-com*)
$\vee$ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*)) (*EvtSys es*))
using *p0 p1 p3 a4* **by** (*meson fstent-nomident-e-sim-es-aux*)
ultimately show *?thesis* **by** (*simp add*:*e-sim-es-def*)
qed
ultimately show *?thesis* **using** *cpts-of-ev-def* **by** (*smt mem-Collect-eq nth-Cons'*)

qed

lemma *fstent-nomident-e-sim-es2*:
⟦*esl*∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*); *el*∈*cpts-ev* Γ⟧ ⟹
e-sim-es esl el es e
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es*
∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *p5*: *el*∈*cpts-ev* Γ
from *p2* **have** *a2*: Γ ⊢ (*BasicEvent e*, *s*, *x*) −*et*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*ev*, *s1*, *x1*)
using *evtsysent-evtent*[*of* Γ *es s x e k ev s1 x1*] **by** *auto*
let *?esl1* = (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
let *?el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys ?esl1*
let *?el1* = *rm-evtsys ?esl1*

have *a5*: *?el* = (*BasicEvent e*, *s*, *x*) # *?el1* **by** *simp*
from *p1* **have** *a3*: *esl* = (*EvtSys es*, *s*, *x*) # *?esl1* **by** *simp*
from *p0 p1 p2 p3 p4 a2* **have** *a4*: *?el* ∈ *cpts-ev* Γ
  using *rm-evtsys-in-cptse* **by** *metis*
show *?thesis*
  proof −
    from *a3* **have** *b1*: *length esl* = *length ?el* **by** (*simp add:rm-evtsys-def*)
    moreover
   from *p1* **have** *b2*: *getspc-es* (*esl* ! *0*) = *EvtSys es* **by** (*simp add:getspc-es-def*)
    moreover
    have *b3*: *getspc-e* (*?el* ! *0*) = *BasicEvent e* **by** (*simp add:getspc-e-def*)
    moreover
    from *a3 b1* **have** *b4*: ∀ *i*. *i* < *length ?el* ⟶
          gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
          getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
      proof −
        have *c1*: *same-s-x ?esl1* (*rm-evtsys ?esl1*) **using** *rm-evtsys-same-sx* **by**
auto
        show *?thesis*
          proof −
          {
            fix *i*
            have *i* < *length ?el* ⟶
              gets-e (*?el* ! *i*) = *gets-es* (*esl* ! *i*) ∧
              getx-e (*?el* ! *i*) = *getx-es* (*esl* ! *i*)
              proof(*cases i* = *0*)
                assume *i* = *0*
                with *p1* **show** *?thesis* **using** *gets-e-def getx-e-def gets-es-def*
                  getx-es-def **by** (*metis nth-Cons-0 snd-conv*)
              next
                assume *i*≠*0*
                with *p1 p3 a3 c1* **show** *?thesis* **by** (*simp add*: *same-s-x-def*)
              qed
          }
          then show *?thesis* **by** *auto*
          qed
      qed
    moreover
    have ∀ *i*. *i* > *0* ∧ *i* < *length ?el* ⟶
           (*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*?el*!*i*) = *AnonyEvent*
fin-com)
         ∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*)) (*EvtSys es*))
      using *p0 p1 p3 a4* **by** (*meson fstent-nomident-e-sim-es-aux*)
    ultimately show *?thesis* **using** *e-sim-es-def* **using** *p4* **by** *blast*
  qed

qed

lemma *e-sim-es-same-assume*:

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

e-sim-es esl el es e; *esl*∈*assume-es* Γ (*pre*,*rely*)$]\!]$
⟹ *el*∈*assume-e* Γ (*pre*,*rely*)

proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p2*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
and *p3*: ¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *p4*: *el* = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
and *a1*: *e-sim-es esl el es e*
and *b0*: *esl*∈*assume-es* Γ (*pre*,*rely*)
from *p3* **have** *p3-1*: ∀*j*. *j* > *0* ∧ *Suc j* < *length esl* ⟶ *getspc-es* (*esl* ! *j*) = *EvtSys es*
⟶ *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* ⟶
Γ ⊢ *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* ⟶ Γ ⊢ *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*: Γ ⊢ *el*!*i* −*ee*→ *el*!(*Suc i*)
with *a2* **have** ¬(Γ ⊢ *el*!*0* −*ee*→ *el*!*1*)
by (*metis* (*no-types*, *lifting*) *One-nat-def eetran-eqconf evtsysent-evtent0*

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

with *c1* **have** *c2*: $i \neq 0$ **by** (*metis One-nat-def*)
with *a1* **have** *c3*: (*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*))
using *e-sim-es-def Suc-lessD c0* **by** *blast*
from *c1* **have** *c4*: *getspc-e* (*el*!*i*) = *getspc-e* (*el*!*Suc i*)
by (*simp add*: *eetran-eqconf1*)
from *a1 c0 a3* **have** *c5*: *gets-es* (*esl*!*i*) = *gets-e* (*el*!*i*)
∧ *gets-es* (*esl*!*Suc i*) = *gets-e* (*el*!*Suc i*) **by** (*simp add*:*e-sim-es-def*)
from *a1 c0 a3* **have** *c6*:
(*getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*Suc i*)) (*EvtSys es*))
using *e-sim-es-def* **by** *blast*
have (*gets-e* (*el*!*i*), *gets-e* (*el*!*Suc i*)) ∈ *rely*
proof(*cases getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
assume *d0*: *getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*
with *c2 p3-1 c0 a3* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSys es* **by** *auto*
with *d0* **have** Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)

with *b1 c0 a3* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely* **by** *auto*
then show *?thesis* **using** *c5* **by** *simp*
next
assume ¬(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*el*!*i*) = *AnonyEvent fin-com*)
with *c3* **have** *d0*: *getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*)
by *simp*
let *?ei* = *getspc-e* (*el*!*i*)
show *?thesis*
proof(*cases ?ei* = *AnonyEvent fin-com*)
assume *e0*: *?ei* = *AnonyEvent fin-com*
with *c1* **have** *e1*: *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*
using *eetran-eqconf1* **by** *fastforce*
show *?thesis*
proof(*cases getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
assume *f0*: *getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*
with *d0* **have** *getspc-e* (*el*!*i*) ≠ *AnonyEvent fin-com*

proof −
let *?esl′* = *drop i esl*
from *p0* **have** *?esl′*∈*cpts-es* Γ
by (*metis Suc-lessD a3 c0 c2 cpts-es-dropi old.nat.exhaust*)
moreover
from *c0 a3* **have** *length ?esl′* > *1*
by *auto*
moreover
from *d0* **have** *getspc-es* (*?esl′*!*0*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*)
using *a3 c0* **by** *auto*
moreover
from *f0* **have** *getspc-es* (*?esl′*!*1*) = *EvtSys es*
using *a3 c0* **by** *fastforce*
ultimately show *?thesis* **using** *not-anonyevt-none-in-evtseq1* **by** *blast*
qed
with *e0* **show** *?thesis* **by** *simp*
next
assume ¬(*getspc-es* (*esl*!*Suc i*) = *EvtSys es* ∧ *getspc-e* (*el*!*Suc i*) = *AnonyEvent fin-com*)
with *c6* **have** *f0*: *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*Suc i*)) (*EvtSys es*)
by *simp*
with *c4* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*i*)) (*EvtSys es*) **by** *simp*
with *d0* **have** *getspc-es* (*esl*!*Suc i*) = *getspc-es* (*esl*!*i*) **by** *simp*
then have Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)
with *b1* **have** (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *rely*
by (*simp add*: *a3 c0*)
with *c5* **show** *?thesis* **by** *simp*
qed
next
assume *e0*: *?ei* ≠ *AnonyEvent fin-com*
with *c4 c6* **have** *getspc-es* (*esl*!*Suc i*) = *EvtSeq* (*getspc-e* (*el*!*Suc i*)) (*EvtSys es*)
by *simp*
with *c4 d0* **have** *getspc-es* (*esl*!*Suc i*) = *getspc-es* (*esl*!*i*) **by** *simp*
then have Γ ⊢ *esl*!*i* −*ese*→ *esl*!*Suc i* **by** (*simp add*: *eqconf-esetran*)

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

ultimately show *?thesis* **by** (*simp add*:*assume-e-def*)
qed
qed

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

$e\text{-}sim\text{-}es\ esl\ el\ es\ e;\ el \in commit\text{-}e\ \Gamma\ (guar,post)]\!]$
$\Longrightarrow esl \in commit\text{-}es\ \Gamma\ (guar,post)$
proof −
assume *p0*: $esl \in cpts\text{-}es\ \Gamma$
and *p1*: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and *p2*: $\Gamma \vdash (EvtSys\ es,\ s,\ x)\ -es-(EvtEnt\ (BasicEvent\ e))\sharp k\rightarrow (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)$
and *p3*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
and *p4*: $el = (BasicEvent\ e,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs)$
and *a1*: $e\text{-}sim\text{-}es\ esl\ el\ es\ e$
and *b3*: $el \in commit\text{-}e\ \Gamma\ (guar,post)$
from *p3* **have** *p3-1*: $\forall j.\ j > 0 \wedge Suc\ j < length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es$
$\longrightarrow getspc\text{-}es\ (esl\ !\ Suc\ j) = EvtSys\ es$ **using** *noevtent-inmid-eq* **by** *auto*
from *p0 p1 p2 p3 p4* **have** *a0*: $el \in cpts\text{-}ev\ \Gamma$ **using** *rm-evtsys-in-cptse* **by** *metis*
let $?esl1 = (EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
let $?el1 = rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1,x1)\ \#\ xs)$
from *p4* **have** *a2*: $el = (BasicEvent\ e,\ s,\ x)\ \#\ (ev,s1,x1)\ \#\ rm\text{-}evtsys\ xs$
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*: $\forall i.\ Suc\ i<length\ el \longrightarrow$
$(\exists t.\ \Gamma \vdash el!i\ -et-t\rightarrow el!(Suc\ i)) \longrightarrow (gets\text{-}e\ (el!i),\ gets\text{-}e\ (el!Suc\ i)) \in guar$
by (*simp add*:*commit-e-def*)
then show $esl \in commit\text{-}es\ \Gamma\ (guar,post)$
proof −
have $\forall i.\ Suc\ i<length\ esl \longrightarrow (\exists t.\ \Gamma \vdash esl!i\ -es-t\rightarrow esl!(Suc\ i))$
$\longrightarrow (gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof −
{
fix *i*
assume *c0*: $Suc\ i<length\ esl$

and *c1*: $\exists\, t.\ \Gamma \vdash$ *esl*!*i* $-es-t\rightarrow$ *esl*!(*Suc i*)

have (*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *guar*
proof(*cases i* = *0*)
assume *d0*: *i* = *0*
from *p2* **have** $\Gamma \vdash$ (*BasicEvent e*, *s*, *x*) $-et-$(*EvtEnt* (*BasicEvent e*))$\sharp k\rightarrow$ (*ev*, *s1*, *x1*)
using *evtsysent-evtent* **by** *fastforce*
with *a2 b4* **have** (*s*, *s1*) $\in$ *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* $\neq$ *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** $\Gamma \vdash$ *esl* ! *i* $-es-t\rightarrow$ *esl* ! *Suc i* **by** *auto*
then have *getspc-es* (*esl*!*i*) $\neq$ *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*) $\neq$ *EvtSys es*
from *p0 p1 c0* **have** *getspc-es* (*esl*!*i*) = *EvtSys es* $\vee$
($\exists$ *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** $\exists$ *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* $\vee$
($\exists$ *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*) $\wedge$ *esl*!(*Suc i*) = (*esi'*,*si'*,*xi'*)
by (*metis prod.collapse*)
with *c1* **obtain** *t* **where** *e3*: $\Gamma \vdash$ (*esi*,*si*,*xi*) $-es-t\rightarrow$ (*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** $\exists\, t.\ \Gamma \vdash$ (*e*, *si*, *xi*) $-et-t\rightarrow$ (*AnonyEvent fin-com*, *si'*,*xi'*)

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

              with e2 show ?thesis by (simp add:gets-es-def)
            next
              assume f0: ∃ e. getspc-es (esl!Suc i) = EvtSeq e (EvtSys es)
               then obtain e′ where f1: getspc-es (esl!Suc i) = EvtSeq e′
(EvtSys es)
                by auto
              with e1 e2 e3 have ∃ t. Γ ⊢ (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 list.simps(9) nth-Cons-Suc
nth-map old.nat.exhaust snd-conv)
              moreover
              from p1 p4 a3 c0 d0 e2 f1 have f3:el!Suc i = (e′, si′,xi′)
               using getspc-es-def getspc-e-def rm-evtsys-def rm-evtsys1-def
                  gets-es-def getx-es-def EvtSeqrm
                   by (smt Suc-lessD fst-conv less-Suc-eq-0-disj list.simps(9)
nth-Cons-Suc nth-map snd-conv)
              ultimately have (si,si′)∈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* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*;
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*);
¬(∃*j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*);
el = (*BasicEvent e*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*);

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

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

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

proof −
assume *p0*: $\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *a0*: $length\ esl \geq 2 \wedge getspc\text{-}es\ (esl!0) = EvtSys\ es \wedge getspc\text{-}es\ (esl!1) \neq EvtSys\ es$
and *c41*: $\neg(\exists\, j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
and *c1*: $esl{\in}cpts\text{-}es\ \Gamma$

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

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

with *c1 c3 c4 c41* **have** *c7*: *esl*∈*assume-es* Γ (*Pre ei*, *Rely ei*) ⟶ *esl*∈*commit-es* Γ (*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*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es* Γ;*length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*;
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)⟧
⟹ ∀ *m*∈*es*. (∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
⟶ (*esl*∈*assume-es* Γ (*Pre m*,*Rely m*) ⟶ *esl*∈*commit-es* Γ (*Guar m*,*Post m*))
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *a0*: *length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*
and *c41*: ¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length esl* ∧ *getspc-es* (*esl*!*j*) = *EvtSys es* ∧ *getspc-es* (*esl*!*Suc j*) ≠ *EvtSys es*)
and *c1*: *esl*∈*cpts-es* Γ
then show *?thesis*
proof −
{
fix *m*
assume *c01*: *m*∈*es*
and *c02*: ∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*
from *a0 c1* **have** *c2*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
by (*simp add*:*fst-esys-snd-eseq-exist*)
then obtain *s* **and** *x* **and** *ev* **and** *s1* **and** *x1* **and** *xs* **where** *c3*:
esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*
with *c02* **have** ∃ *k*. Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt m*)♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *simp*
then obtain *k* **where** *c4*: Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt m*)♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) **by** *auto*
then have ∃ *e*. *m* = *BasicEvent e* **by** (*meson evtent-is-basicevt*)
then obtain *e* **where** *c40*: *m* = *BasicEvent e* **by** *auto*
let *?el* = (*m*, *s*, *x*) # *rm-evtsys* ((*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*)
from *c1 c3 c4 c40 c41* **have** *c5*: *?el* ∈ *cpts-ev* Γ **using** *rm-evtsys-in-cptse* **by** *metis*

from *c3 c4 c40* **have** *c61*: Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1* **by** *simp*
have *c8*: *?el*∈*assume-e* Γ (*Pre m*, *Rely m*) ⟶ *?el*∈*commit-e* Γ (*Guar m*,*Post m*)
proof
assume *d0*: *?el*∈*assume-e* Γ (*Pre m*, *Rely m*)
moreover

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

lemma *EventSys-sound-seg-aux0-exist*:
⟦*esl*∈*cpts-es* Γ;*length esl* ≥ *2*; *getspc-es* (*esl*!*0*) = *EvtSys es*; *getspc-es* (*esl*!*1*) ≠ *EvtSys es*⟧
⟹ ∃ *m*∈*es*. (∃ *k*. Γ ⊢ *esl*!*0*−*es*−(*EvtEnt m*)♯*k*→*esl*!*1*)
proof −
assume *p0*: *esl*∈*cpts-es* Γ
and *p1*: *length esl* ≥ *2*
and *p2*: *getspc-es* (*esl*!*0*) = *EvtSys es*
and *p3*: *getspc-es* (*esl*!*1*) ≠ *EvtSys es*
then have *a1*: ∃ *s x ev s1 x1 xs*. *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs*
by (*simp add*:*fst-esys-snd-eseq-exist*)
then obtain *s* **and** *x* **and** *ev* **and** *s1* **and** *x1* **and** *xs* **where** *a2*:
esl = (*EvtSys es*, *s*, *x*) # (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*) # *xs* **by** *auto*
with *p0 a1* **have** ∃ *e k*. Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
using *fst-esys-snd-eseq-exist-evtent2* **by** *fastforce*
then obtain *e* **and** *k* **where** *a3*:
Γ ⊢ (*EvtSys es*, *s*, *x*) −*es*−(*EvtEnt* (*BasicEvent e*))♯*k*→ (*EvtSeq ev* (*EvtSys es*), *s1*,*x1*)
by *auto*
from *a3* **have** ∃ *i*∈*es*. *i* = *BasicEvent e* **using** *evtsysent-evtent* **by** *metis*
then obtain *ei* **where** *c6*: *ei*∈ *es* ∧ *ei* = *BasicEvent e* **by** *auto*
then show *?thesis* **using** *One-nat-def a2 a3 nth-Cons-0 nth-Cons-Suc* **by** *force*

qed

lemma *EventSys-sound-seg-aux0-forall*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
esl∈*cpts-es* Γ;*length esl* ≥ *2* ∧ *getspc-es* (*esl*!*0*) = *EvtSys es* ∧ *getspc-es* (*esl*!*1*) ≠ *EvtSys es*;
getspc-es (*last esl*) = *EvtSys es*;

$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)]\!]$
$\Longrightarrow \forall ei \in es.\ (\exists k.\ \Gamma \vdash esl!0 -es-(EvtEnt\ ei)\sharp k \rightarrow esl!1)$
$\longrightarrow (esl \in assume\text{-}es\ \Gamma\ (Pre\ ei, Rely\ ei) \longrightarrow esl \in commit\text{-}es\ \Gamma\ (Guar\ ei, Post\ ei)$
$\wedge\ gets\text{-}es\ (last\ esl) \in Post\ ei)$

proof −
assume *p0*: $\forall ef \in es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *a0*: $length\ esl \geq 2 \wedge getspc\text{-}es\ (esl!0) = EvtSys\ es \wedge getspc\text{-}es\ (esl!1) \neq EvtSys\ es$
and *p6*: $getspc\text{-}es\ (last\ esl) = EvtSys\ es$
and *c41*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ esl \wedge getspc\text{-}es\ (esl!j) = EvtSys\ es \wedge getspc\text{-}es\ (esl!Suc\ j) \neq EvtSys\ es)$
and *c1*: $esl \in cpts\text{-}es\ \Gamma$
then show *?thesis*
proof−
{
fix *ei*
assume *c01*: $ei \in es$
and *c02*: $\exists k.\ \Gamma \vdash esl!0 -es-(EvtEnt\ ei)\sharp k \rightarrow esl!1$

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** $\exists k.\ \Gamma \vdash (EvtSys\ es,\ s,\ x) -es-(EvtEnt\ ei)\sharp k \rightarrow (EvtSeq\ ev\ (EvtSys\ es),\ s1, x1)$ **by** *simp*
then obtain *k* **where** *c4*: $\Gamma \vdash (EvtSys\ es,\ s,\ x) -es-(EvtEnt\ ei)\sharp k \rightarrow (EvtSeq\ ev\ (EvtSys\ es),\ s1, x1)$ **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 $?el = (ei,\ s,\ x)\ \#\ rm\text{-}evtsys\ ((EvtSeq\ ev\ (EvtSys\ es),\ s1, x1)\ \#\ xs)$
from *c1 c3 c4 c6 c41* **have** *c5*: $?el \in cpts\text{-}ev\ \Gamma$ **using** *rm-evtsys-in-cptse* **by** *metis*

from *c3 c4 c6* **have** *c61*: $\Gamma \vdash esl!0 -es-(EvtEnt\ ei)\sharp k \rightarrow esl!1$ **by** *simp*
have *c8*: $?el \in assume\text{-}e\ \Gamma\ (Pre\ ei,\ Rely\ ei) \longrightarrow ?el \in commit\text{-}e\ \Gamma\ (Guar\ ei, Post\ ei)$
proof
assume *d0*: $?el \in assume\text{-}e\ \Gamma\ (Pre\ ei,\ Rely\ ei)$
moreover
from *p0 c01 c6* **have** *d1*: $\Gamma \models ei\ sat_e\ [Pre\ ei,\ Rely\ ei,\ Guar\ ei,\ Post\ ei]$ **by** *auto*
moreover
from *c5 c6* **have** $?el \in cpts\text{-}of\text{-}ev\ \Gamma\ (BasicEvent\ e)\ s\ x$ **by** (*simp add:cpts-of-ev-def*)
ultimately show $?el \in commit\text{-}e\ \Gamma\ (Guar\ ei, Post\ ei)$ **using** *evt-validity-def*

c6
by *fastforce*
qed
with *c1 c3 c4 c41 c6* **have** *c7*: *esl*∈*assume-es* Γ (*Pre ei*, *Rely ei*) ⟶ *esl*∈*commit-es* Γ (*Guar ei*,*Post ei*)
using *rm-evtsys-assum-comm* **by** *metis*
moreover
have *esl*∈*assume-es* Γ (*Pre ei*, *Rely ei*) ⟶ *gets-es* (*last esl*) ∈ *Post ei*
proof
assume *d0*: *esl*∈*assume-es* Γ (*Pre ei*, *Rely ei*)
from *c1 c3 c4 c41 c5 c6* **have** *d2*: *e-sim-es esl ?el es e* **using** *fstent-nomident-e-sim-es2* **by** *metis*
with *c1 c3 c4 c41 c5 c6 d0* **have** *d3*: *?el*∈*assume-e* Γ (*Pre ei*, *Rely ei*)
using *e-sim-es-same-assume* **by** *metis*
with *c8* **have** *d1*: *?el*∈*commit-e* Γ (*Guar ei*,*Post ei*) **by** *auto*

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

with *d2* **have** ∀ *i*. *i* > *0* ∧ *i* < *length ?el* ⟶
(*getspc-es* (*esl*!*i*) = *EvtSys es* ∧ *getspc-e* (*?el*!*i*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*esl*!*i*) = *EvtSeq* (*getspc-e* (*?el*!*i*)) (*EvtSys es*))
by (*simp add*: *e-sim-es-def*)
with *a0 e1* **have** (*getspc-es* (*last esl*) = *EvtSys es* ∧ *getspc-e* (*last ?el*) = *AnonyEvent fin-com*)
∨ (*getspc-es* (*last esl*) = *EvtSeq* (*getspc-e* (*last ?el*)) (*EvtSys es*))
by (*metis* (*no-types*, *lifting*) *c3 diff-less last-conv-nth length-greater-0-conv length-tl*
list.sel(*3*) *list.simps*(*3*) *zero-less-one*)
with *p6* **show** *?thesis* **by** *simp*
qed
with *d1* **have** *gets-e* (*last ?el*) ∈ *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*, *lifting*) *c3 diff-less last-conv-nth length-greater-0-conv length-tl*
list.sel(*3*) *list.simps*(*3*) *zero-less-one*)
qed

ultimately show *gets-es* (*last esl*) ∈ *Post ei* **by** *simp*
qed

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

lemma *EventSys-sound-aux-i-forall*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef sat*$_e$ [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
∀ *ef*∈*es*. *pre* ⊆ *Pre ef*; ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*;
∀ *ef*∈*es*. *Guar ef* ⊆ *guar*; ∀ *ef*∈*es*. *Post ef* ⊆ *post*;
∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*;
esl∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es* Γ (*pre*,*rely*);

$elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])\rrbracket$
$\Longrightarrow \forall i.\ Suc\ i < length\ elst \longrightarrow$
$(\forall ei{\in}es.\ (\exists k.\ \Gamma \vdash (elst!i@[(elst!Suc\ i)!0])!0{-}es{-}(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei)$

proof −
assume $p0$: $\forall ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and $p1$: $\forall ef{\in}es.\ pre \subseteq Pre\ ef$
and $p2$: $\forall ef{\in}es.\ rely \subseteq Rely\ ef$
and $p3$: $\forall ef{\in}es.\ Guar\ ef \subseteq guar$
and $p4$: $\forall ef{\in}es.\ Post\ ef \subseteq post$
and $p5[rule\text{-}format]$: $\forall ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and $p8$: $esl{\in}cpts\text{-}es\ \Gamma$
and $p9$: $esl = (EvtSys\ es,\ s,\ x)\ \#\ (EvtSeq\ e\ (EvtSys\ es),\ s1,x1)\ \#\ xs$
and $p10$: $esl{\in}assume\text{-}es\ \Gamma\ (pre,rely)$
and $p11$: $elst = tl\ (parse\text{-}es\text{-}cpts\text{-}i2\ esl\ es\ [[]])$
from $p9\ p8\ p11$ **have** $a0[rule\text{-}format]$: $\forall i.\ i < length\ elst \longrightarrow length\ (elst!i) \geq 2\ \wedge$
$getspc\text{-}es\ (elst!i!0) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!1) \neq EvtSys\ es$
using *parse-es-cpts-i2-start-aux* **by** *metis*
from $p9\ p8\ p11$ **have** $a1$: $\forall i.\ i < length\ elst \longrightarrow$
$\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ (elst!i)\ \wedge$
$getspc\text{-}es\ (elst!i!j) = EvtSys\ es \wedge getspc\text{-}es\ (elst!i!Suc\ j) \neq EvtSys\ es)$
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from $p9\ p8\ p11$ **have** $a2$: $concat\ elst = esl$ **using** *parse-es-cpts-i2-concat3* **by** *metis*
show *?thesis*
proof −
{
fix i
assume $b0$: $Suc\ i < length\ elst$
then have $\forall ei{\in}es.\ (\exists k.\ \Gamma \vdash (elst!i@[(elst!Suc\ i)!0])!0{-}es{-}(EvtEnt\ ei)\sharp k{\rightarrow}(elst!i@[(elst!Suc\ i)!0])!1)$
$\longrightarrow elst!i@[(elst!Suc\ i)!0]{\in}commit\text{-}es\ \Gamma\ (Guar\ ei,Post\ ei)$
$\wedge\ gets\text{-}es\ ((elst!Suc\ i)!0) \in Post\ ei$
proof$(induct\ i)$
case 0
assume $c0$: $Suc\ 0 < length\ elst$
let $?els = elst\ !\ 0\ @\ [elst\ !\ Suc\ 0\ !\ 0]$
have $c1$: $?els \in cpts\text{-}es\ \Gamma$
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*: $\neg(\exists j.\ j > 0 \wedge$ *Suc j* < *length ?els* $\wedge$ *getspc-es* (*?els*!*j*) = *EvtSys es*
$\wedge$ *getspc-es* (*?els*!*Suc j*) $\neq$ *EvtSys es*)
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*: $2 \leq$ *length ?els* $\wedge$ *getspc-es* (*?els* ! *0*) = *EvtSys es* $\wedge$ *getspc-es* (*?els* ! *1*) $\neq$ *EvtSys es*
by (*metis* (*no-types*, *hide-lams*) *Suc-1 Suc-eq-plus1-left Suc-le-lessD*
Suc-lessD add.right-neutral c0 length-append-singleton not-less nth-append)
with *p0 c1 c2 c3* **have** *c5*: $\forall$ *ei*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ *?els*!*0*–*es*–(*EvtEnt ei*)♯*k*→*?els*!*1*)
$\longrightarrow$ (*?els*$\in$*assume-es* Γ (*Pre ei*,*Rely ei*) $\longrightarrow$ *?els*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)
$\wedge$ *gets-es* (*last ?els*) $\in$ *Post ei*)
using *EventSys-sound-seg-aux0-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *auto*

from *p10 a2* **have** *?els*$\in$*assume-es* Γ (*pre*,*rely*)
proof –
from *a0* **have** *d1*: $\forall$ *i*<*length elst*. *elst* ! *i* $\neq$ []
using *list.size*(*3*) *not-numeral-le-zero* **by** *force*
with *a2 c0* **have** $\exists$ *m n*. $m \leq$ *length esl* $\wedge$ $n \leq$ *length esl* $\wedge$ $m \leq n$ $\wedge$ *?els* = *take* $(n - m)$ (*drop m esl*)
using *concat-i-lm* **by** *blast*
moreover
from *p10* **have** $\forall$ *i*. *Suc i*<*length esl* $\longrightarrow$ $\Gamma \vdash$ *esl*!*i* –*ese*→ *esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have $\forall$ *i*. *Suc i*<*length ?els* $\longrightarrow$ $\Gamma \vdash$ *?els*!*i* –*ese*→ *?els*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?els*!*i*), *gets-es* (*?els*!*Suc i*)) $\in$ *rely*
using *rely-takedrop-rely* **by** *blast*

moreover
have *gets-es* (*?els*!*0*) ∈ *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* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp*
by *metis*
with *c5* **show** *?case* **by** *auto*
next
case (*Suc j*)
let *?elstjj* = *elst* ! *j* @ [*elst* ! *Suc j* ! *0*]
let *?els* = *elst* ! *Suc j* @ [*elst* ! *Suc* (*Suc j*) ! *0*]
assume *c01*: *Suc j* < *length elst*
⟹ ∀ *ei*∈*es*. (∃ *k*. Γ ⊢ *?elstjj* ! *0* −*es*−*EvtEnt ei*♯*k*→ *?elstjj*
! *1*) ⟶
?elstjj ∈ *commit-es* Γ (*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*. Γ ⊢ *?els* ! *0* −*es*−*EvtEnt ei*♯*k*→ *?els* ! *1*

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

then obtain *ei′* **where** *c03*: *ei′*∈*es* ∧ (∃ *k*. Γ ⊢ *?elstjj*!*0*−*es*−(*EvtEnt*
ei′)♯*k*→*?elstjj*!*1*)
by *auto*
with *c01 c02* **have** *c04*: *?elstjj* ∈ *commit-es* Γ (*Guar ei′*, *Post*
ei′)
∧ *gets-es* (*elst* ! *Suc j* ! *0*) ∈ *Post ei′*

by *auto*

have *c1*: *?els* ∈ *cpts-es* Γ
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* Γ (*Pre ei*,*Rely ei*) ⟶ *?els*∈*commit-es* Γ (*Guar ei*,*Post ei*)
∧ *gets-es* (*last ?els*) ∈ *Post ei*)
using *EventSys-sound-seg-aux0-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *blast*
from *p10 a2* **have** *?els*∈*assume-es* Γ (*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* ⟶ Γ ⊢ *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* ⟶ Γ ⊢ *?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* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp*[*of Pre ei Pre ei rely Rely ei*]
d0 order-refl **by** *auto*

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

lemma *EventSys-sound-aux-i*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
∀ *ef*∈*es*. *pre* ⊆ *Pre ef*; ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*;
∀ *ef*∈*es*. *Guar ef* ⊆ *guar*; ∀ *ef*∈*es*. *Post ef* ⊆ *post*;
∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*;
esl∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es* Γ (*pre*,*rely*);
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])⟧
⟹ ∀ *i*. *Suc i* < *length elst* ⟶
(∃ *m*∈*es*. *elst*!*i*@[(*elst*!*Suc i*)!*0*]∈*commit-es* Γ (*Guar m*,*Post m*)
∧ *gets-es* ((*elst*!*Suc i*)!*0*) ∈ *Post m*
∧ (∃ *k*. Γ ⊢ (*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*0*−*es*−(*EvtEnt m*)♯*k*→(*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*1*))
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]

and *p1*: ∀ *ef*∈*es*. *pre* ⊆ *Pre ef*
and *p2*: ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*
and *p3*: ∀ *ef*∈*es*. *Guar ef* ⊆ *guar*
and *p4*: ∀ *ef*∈*es*. *Post ef* ⊆ *post*
and *p5*: ∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*
and *p8*: *esl*∈*cpts-es* Γ
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*∈*assume-es* Γ (*pre*,*rely*)
and *p11*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
from *p9 p8 p11* **have** *a0*[*rule-format*]: ∀ *i*. *i* < *length elst* ⟶ *length* (*elst*!*i*) ≥ *2* ∧
getspc-es (*elst*!*i*!*0*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*1*) ≠ *EvtSys es*
using *parse-es-cpts-i2-start-aux* **by** *metis*
from *p9 p8 p11* **have** *a1*: ∀ *i*. *i* < *length elst* ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*)
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
show *?thesis*
proof −
{
fix *i*
assume *b0*: *Suc i* < *length elst*
with *a0*[*of i*] **have** ∃ *m*∈*es*. (∃ *k*. Γ ⊢ *elst*!*i*!*0*−*es*−(*EvtEnt m*)♯*k*→*elst*!*i*!*1*)
using *EventSys-sound-seg-aux0-exist*[*of elst*!*i*@[(*elst*!*Suc i*)!*0*] Γ *es*]
parse-es-cpts-i2-in-cptes-i[*of esl es s x e s1 x1 xs* Γ *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*. Γ ⊢ *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*. Γ ⊢ (*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* Γ (*Guar ei*, *Post ei*) ∧ *gets-es* (*elst* ! *Suc i* ! *0*) ∈ *Post ei*)
using *EventSys-sound-aux-i-forall*[*of es* Γ *Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs elst*]
by *fastforce*
from *b0 b1 b2*[*of i m*] **have** *elst*!*i*@[(*elst*!*Suc i*)!*0*]∈*commit-es* Γ (*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* Γ (*Guar m*,*Post*

$m)$
$\wedge$ *gets-es* ((*elst*!*Suc i*)!*0*) $\in$ *Post m*
$\wedge$ ($\exists$ *k*. $\Gamma \vdash$ (*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*0*$-$*es*$-$(*EvtEnt m*)$\sharp$*k*$\rightarrow$(*elst*!*i*@[(*elst*!*Suc i*)!*0*])!*1*)
by (*smt One-nat-def Suc-lessD a0 b0 lessI less-le-trans nth-append numeral-2-eq-2*)

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

lemma *EventSys-sound-aux-last-forall*:
$[\![\forall$ *ef*$\in$*es*. $\Gamma \models$ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
$\forall$ *ef*$\in$*es*. *pre* $\subseteq$ *Pre ef*; $\forall$ *ef*$\in$*es*. *rely* $\subseteq$ *Rely ef*;
$\forall$ *ef*$\in$*es*. *Guar ef* $\subseteq$ *guar*; $\forall$ *ef*$\in$*es*. *Post ef* $\subseteq$ *post*;
$\forall$ *ef1 ef2*. *ef1*$\in$*es* $\wedge$ *ef2*$\in$*es* $\longrightarrow$ *Post ef1* $\subseteq$ *Pre ef2*;
esl$\in$*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl$\in$*assume-es* Γ (*pre*,*rely*);
elst = *tl* (*parse-es-cpts-i2 esl es* [[]])$]\!]$
$\Longrightarrow$ $\forall$ *ei*$\in$*es*. ($\exists$ *k*. $\Gamma \vdash$ (*last elst*)!*0*$-$*es*$-$(*EvtEnt ei*)$\sharp$*k*$\rightarrow$(*last elst*)!*1*)
$\longrightarrow$ *last elst*$\in$*commit-es* Γ (*Guar ei*,*Post ei*)
proof $-$
assume *p0*: $\forall$ *ef*$\in$*es*. $\Gamma \models$ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *p1*: $\forall$ *ef*$\in$*es*. *pre* $\subseteq$ *Pre ef*
and *p2*: $\forall$ *ef*$\in$*es*. *rely* $\subseteq$ *Rely ef*
and *p3*: $\forall$ *ef*$\in$*es*. *Guar ef* $\subseteq$ *guar*
and *p4*: $\forall$ *ef*$\in$*es*. *Post ef* $\subseteq$ *post*
and *p5*: $\forall$ *ef1 ef2*. *ef1*$\in$*es* $\wedge$ *ef2*$\in$*es* $\longrightarrow$ *Post ef1* $\subseteq$ *Pre ef2*
and *p8*: *esl*$\in$*cpts-es* Γ
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*$\in$*assume-es* Γ (*pre*,*rely*)
and *p11*: *elst* = *tl* (*parse-es-cpts-i2 esl es* [[]])
from *p9 p8 p11* **have** *a0*[*rule-format*]: $\forall$ *i*. *i* < *length elst* $\longrightarrow$ *length* (*elst*!*i*) $\geq$ *2* $\wedge$
getspc-es (*elst*!*i*!*0*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*1*) $\neq$ *EvtSys es*
using *parse-es-cpts-i2-start-aux* **by** *metis*
from *p9 p8 p11* **have** *a1*: $\forall$ *i*. *i* < *length elst* $\longrightarrow$
$\neg$($\exists$ *j*. *j* > *0* $\wedge$ *Suc j* < *length* (*elst*!*i*) $\wedge$
getspc-es (*elst*!*i*!*j*) = *EvtSys es* $\wedge$ *getspc-es* (*elst*!*i*!*Suc j*) $\neq$ *EvtSys es*)
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
with *p9* **have** *a3*: *elst* $\neq$ [] **by** *auto*
show *?thesis*
proof $-$
{

fix *ei*
assume *a01*: $ei \in es$
and *a02*: $\exists k.\ \Gamma \vdash (last\ elst)!0 -es-(EvtEnt\ ei)\sharp k\rightarrow (last\ elst)!1$
have $last\ elst \in commit\text{-}es\ \Gamma\ (Guar\ ei, Post\ ei)$
proof(*cases length elst* = *1*)
assume *b0*: *length elst* = *1*
from *a2 b0* **have** *b1*: *last elst* = *esl*
by (*metis* (*no-types*, *lifting*) *One-nat-def a3 append-butlast-last-id append-self-conv2 concat.simps*(*1*) *concat.simps*(*2*) *diff-Suc-1 length-0-conv length-butlast self-append-conv*)

let *?els* = *elst* ! *0*
from *p8 a2 b0* **have** *c1*: $?els \in cpts\text{-}es\ \Gamma$ **using** *b1 a3 last-conv-nth* **by** *fastforce*

from *a1 b0* **have** *c3*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ ?els \wedge getspc\text{-}es\ (?els!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (?els!Suc\ j) \neq EvtSys\ es)$ **by** *simp*
from *a0 b0* **have** *c4*: $2 \leq length\ ?els \wedge getspc\text{-}es\ (?els\ !\ 0) = EvtSys\ es \wedge getspc\text{-}es\ (?els\ !\ 1) \neq EvtSys\ es$
by *simp*

with *p0 c1 c3* **have** *c5*: $\forall m \in es.\ (\exists k.\ \Gamma \vdash ?els!0 -es-(EvtEnt\ m)\sharp k\rightarrow ?els!1)$

$\longrightarrow (?els \in assume\text{-}es\ \Gamma\ (Pre\ m, Rely\ m) \longrightarrow ?els \in commit\text{-}es\ \Gamma\ (Guar\ m, Post\ m))$
using *EventSys-sound-aux1-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** $?els \in assume\text{-}es\ \Gamma\ (pre, rely)$
proof −

from *a2 b0* **have** $\exists m\ n.\ m \leq length\ esl \wedge last\ elst = (drop\ m\ esl)$
using *concat-last-lm* **using** *b1* **by** *auto*
moreover
from *p10* **have** $\forall i.\ Suc\ i < length\ esl \longrightarrow \Gamma \vdash esl!i\ -ese\rightarrow esl!(Suc\ i) \longrightarrow$
$(gets\text{-}es\ (esl!i), gets\text{-}es\ (esl!Suc\ i)) \in rely$ **by** (*simp add*:*assume-es-def*)
ultimately have $\forall i.\ Suc\ i < length\ ?els \longrightarrow \Gamma \vdash ?els!i\ -ese\rightarrow ?els!(Suc\ i) \longrightarrow$
$(gets\text{-}es\ (?els!i), gets\text{-}es\ (?els!Suc\ i)) \in rely$
using *a3 b0 b1 last-conv-nth* **by** *force*
moreover
have $gets\text{-}es\ (?els!0) \in pre$
proof −
from *a2* **have** $?els!0 = esl!0$
using *a3 b0 b1 last-conv-nth* **by** *fastforce*
moreover
from *p10* **have** $gets\text{-}es\ (esl!0) \in pre$ **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **by** *simp*

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

with *p1 p2 a01* **have** $?els \in assume\text{-}es\ \Gamma\ (Pre\ ei,\ Rely\ ei)$
using *assume-es-imp*[*of pre Pre ei rely Rely ei elst ! 0*] **by** *simp*
with *a01 a02 c5* **have** *c6*: $?els \in commit\text{-}es\ \Gamma\ (Guar\ ei, Post\ ei)$
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 $length\ elst \neq 1$
with *a3* **have** *b0*: $length\ elst > 1$ **by** (*simp add*: *Suc-lessI*)
let $?els = last\ elst$
from *p8 a2 b0* **have** *c1*: $?els \in cpts\text{-}es\ \Gamma$
proof −
from *a2 b0* **have** $\exists m .\ m \leq length\ esl \wedge ?els = drop\ m\ esl$
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*: $\neg(\exists j.\ j > 0 \wedge Suc\ j < length\ ?els \wedge getspc\text{-}es\ (?els!j) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (?els!Suc\ j) \neq EvtSys\ es)$
by (*metis One-nat-def Suc-lessD a3 diff-less last-conv-nth zero-less-one*)
from *a0 b0* **have** *c4*: $2 \leq length\ ?els \wedge getspc\text{-}es\ (?els\ !\ 0) = EvtSys\ es \wedge getspc\text{-}es\ (?els\ !\ 1) \neq EvtSys\ es$
by (*simp add*: *a3 last-conv-nth*)

with *p0 c1 c3* **have** *c5*: $\forall m \in es.\ (\exists k.\ \Gamma \vdash ?els!0 -es-(EvtEnt\ m)\sharp k \rightarrow ?els!1)$

$\longrightarrow (?els \in assume\text{-}es\ \Gamma\ (Pre\ m, Rely\ m) \longrightarrow ?els \in commit\text{-}es\ \Gamma\ (Guar\ m, Post\ m))$
using *EventSys-sound-aux1-forall*[*of es* Γ *Pre Rely Guar Post ?els*] **by** *fastforce*

from *p10 a2* **have** *c6*: $?els \in assume\text{-}es\ \Gamma\ (Pre\ ei, rely)$
proof −
from *a2 b0* **have** $\exists m .\ m \leq length\ esl \wedge ?els = drop\ m\ esl$
by (*simp add*: *concat-last-lm a3*)
moreover
from *p10* **have** $\forall i.\ Suc\ i < length\ esl \longrightarrow \Gamma \vdash esl!i -ese\rightarrow esl!(Suc\ i) \longrightarrow$

(gets-es (esl!i), gets-es (esl!Suc i)) ∈ *rely* **by** *(simp add:assume-es-def)*
ultimately have ∀ *i. Suc i<length ?els* ⟶ Γ ⊢ *?els!i −ese→ ?els!(Suc i)* ⟶
(gets-es (?els!i), gets-es (?els!Suc i)) ∈ *rely*
using *a3 b0 last-conv-nth* **by** *force*
moreover
have *gets-es (?els!0)* ∈ *Pre ei*
proof −
from *p0 p1 p2 p3 p4 p5 p8 p9 p10 p11*
have *c1[rule-format]*: ∀ *i. Suc i < length elst* ⟶
(∀ *ei*∈*es.* (∃ *k.* Γ ⊢ *(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* Γ *(Guar ei,Post ei)*
∧ *gets-es ((elst!Suc i)!0)* ∈ *Post ei)*
using *EventSys-sound-aux-i-forall[of es* Γ *Pre Rely Guar Post pre rely guar*
post esl s x e s1 x1 xs elst] **by** *blast*
let *?els1 = elst!(length elst − 2)@[(elst!(length elst − 1))!0]*
have *d1: ?els1* ∈ *cpts-es* Γ
proof −
from *a0* **have** *c11*: ∀ *i<length elst. elst ! i* ≠ []
using *list.size(3) not-numeral-le-zero* **by** *force*
with *a2 b0* **have** ∃ *m n. m* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n* ∧ *?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* ≤ *length esl* ∧ *n* ≤ *length esl* ∧ *m* ≤ *n*
∧ *?els1 = take (n − m) (drop m esl)* **by** *auto*
have *?els1* ≠ [] **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.* Γ ⊢ *?els1!0−es−(EvtEnt m)♯k→?els1!1)*
using *EventSys-sound-seg-aux0-exist[of ?els1* Γ *es]* **by** *simp*
then obtain *m* **where** *d2: m*∈*es* ∧ (∃ *k.* Γ ⊢ *?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* Γ (*Pre ei*, *Rely ei*)
using *assume-es-imp* **by** *blast*
with *c5* **have** *c6*: *?els*∈*commit-es* Γ (*Guar ei*,*Post ei*) **using** *a01 a02* **by** *blast*

with *c5* **show** *?thesis* **using** *a3 b0 last-conv-nth* **by** *blast*
qed
}
then show *?thesis* **by** *auto* **qed**
qed

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

getspc-es (*elst*!*i*!*0*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*1*) ≠ *EvtSys es*
using *parse-es-cpts-i2-start-aux* **by** *metis*
from *p9 p8 p11* **have** *a1*: ∀ *i*. *i* < *length elst* ⟶
¬(∃ *j*. *j* > *0* ∧ *Suc j* < *length* (*elst*!*i*) ∧
getspc-es (*elst*!*i*!*j*) = *EvtSys es* ∧ *getspc-es* (*elst*!*i*!*Suc j*) ≠ *EvtSys es*)
using *parse-es-cpts-i2-noent-mid* **by** *metis*
from *p9 p8 p11* **have** *a2*: *concat elst* = *esl* **using** *parse-es-cpts-i2-concat3* **by** *metis*
with *p9* **have** *a3*: *elst* ≠ [] **by** *auto*
from *p8 p9 p11 a0*[*of length elst* − *1*] **have** ∃ *m*∈*es*. (∃ *k*. Γ ⊢ *last elst*!*0*−*es*−(*EvtEnt m*)♯*k*→*last elst*!*1*)
using *EventSys-sound-seg-aux0-exist*[*of last elst* Γ *es*]
parse-es-cpts-i2-in-cptes-last[*of esl es s x e s1 x1 xs* Γ *elst*]
by (*metis a3 diff-less last-conv-nth length-greater-0-conv less-one*)
then obtain *m* **where** *b1*: *m*∈*es* ∧ (∃ *k*. Γ ⊢ *last elst*!*0*−*es*−(*EvtEnt m*)♯*k*→*last elst*!*1*) **by** *auto*
with *p0 p1 p2 p3 p4 p5 p8 p9 p10 p11*
have *last elst*∈*commit-es* Γ (*Guar m*,*Post m*)
using *EventSys-sound-aux-last-forall*[*of es* Γ *Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs elst*] **by** *blast*
with *b1* **show** *?thesis* **by** *auto*
qed

lemma *EventSys-sound-0*:
⟦∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*];
∀ *ef*∈*es*. *pre* ⊆ *Pre ef*; ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*;
∀ *ef*∈*es*. *Guar ef* ⊆ *guar*; ∀ *ef*∈*es*. *Post ef* ⊆ *post*;
∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*;
stable-e pre rely; ∀ *s*. (*s*, *s*)∈*guar*;
esl∈*cpts-es* Γ; *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*;
esl∈*assume-es* Γ (*pre*,*rely*)⟧
⟹ ∀ *i*. *Suc i*<*length esl* ⟶ (∃ *t*. Γ ⊢ *esl*!*i* −*es*−*t*→ *esl*!(*Suc i*)) ⟶
(*gets-es* (*esl*!*i*), *gets-es* (*esl*!*Suc i*)) ∈ *guar*
proof −
assume *p0*: ∀ *ef*∈*es*. Γ ⊨ *ef* sat_e [*Pre ef*, *Rely ef*, *Guar ef*, *Post ef*]
and *p1*: ∀ *ef*∈*es*. *pre* ⊆ *Pre ef*
and *p2*: ∀ *ef*∈*es*. *rely* ⊆ *Rely ef*
and *p3*: ∀ *ef*∈*es*. *Guar ef* ⊆ *guar*
and *p4*: ∀ *ef*∈*es*. *Post ef* ⊆ *post*
and *p5*: ∀ *ef1 ef2*. *ef1*∈*es* ∧ *ef2*∈*es* ⟶ *Post ef1* ⊆ *Pre ef2*
and *p6*: *stable-e pre rely*
and *p7*: ∀ *s*. (*s*, *s*)∈*guar*
and *p8*: *esl*∈*cpts-es* Γ
and *p9*: *esl* = (*EvtSys es*, *s*, *x*) # (*EvtSeq e* (*EvtSys es*), *s1*,*x1*) # *xs*
and *p10*: *esl*∈*assume-es* Γ (*pre*,*rely*)
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: ∀ 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 p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
  have ∀ i. Suc i < length ?elst ⟶
          (∃ m∈es. ?elst!i@[(?elst!Suc i)!0]∈commit-es Γ (Guar m,Post m)
                      ∧ gets-es ((?elst!Suc i)!0) ∈ Post m)
    using EventSys-sound-aux-i
      [of es Γ Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs ?elst] by
blast
  then have a2: ∀ i. Suc i < length ?elst ⟶
          (∃ m∈es. ?elst!i@[(?elst!Suc i)!0]∈commit-es Γ (Guar m,Post m)) by
auto

  from p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
  have a3: ∃ m∈es. last ?elst∈commit-es Γ (Guar m,Post m)
    using EventSys-sound-aux-last
      [of es Γ Pre Rely Guar Post pre rely guar post esl s x e s1 x1 xs ?elst] by
blast
  then obtain m where a4: m∈es ∧ last ?elst∈commit-es Γ (Guar m,Post m)
by auto
  show ?thesis
    proof –
    {
      fix i
      assume b0: Suc i < length esl
        and b1: ∃ t. Γ ⊢ esl ! i −es−t→ esl ! Suc i
      from p9 have b01: esl ≠ [] by simp
      moreover
      from a1 have b3: ∀ i<length ?elst. length (?elst!i) ≥ 2 by simp
      ultimately have ∃ k j. k < length ?elst ∧ j ≤ length (?elst!k) ∧
              drop i esl = (drop j (?elst!k)) @ concat (drop (Suc k) ?elst)
        using concat-equiv [of esl ?elst] a0 b0 by auto
     then obtain k and j where b2: k < length ?elst ∧ j ≤ length (?elst!k) ∧
            drop i esl = (drop j (?elst!k)) @ concat (drop (Suc k) ?elst) by auto
      have (gets-es (esl!i), gets-es (esl!Suc i)) ∈ guar
        proof(cases k = length ?elst − 1)
          assume c0: k = length ?elst − 1
          with b2 have c1: drop i esl = drop j (last ?elst)
            by (metis (no-types, lifting) Nitpick.size-list-simp(2) Suc-leI b01
                a0 concat.simps(1) drop-all last-conv-nth length-tl self-append-conv)
          with b0 b01 have c2: drop j (last ?elst) ≠ [] by auto
          with b2 c0 have c3: j < length (last ?elst) by auto
          with c1 have c4: esl ! i = (last ?elst) ! j
            by (metis Suc-lessD b0 hd-drop-conv-nth)
          from c1 c3 have c5: esl ! Suc i = (last ?elst) ! Suc j
            by (metis Cons-nth-drop-Suc Suc-lessD b0 list.sel(3) nth-via-drop)
```

```
          from a4 have ∀ i. Suc i<length (last ?elst) ⟶ (∃ t. Γ ⊢ (last ?elst)!i
−es−t→ (last ?elst)!(Suc i))
              ⟶ (gets-es ((last ?elst)!i), gets-es ((last ?elst)!Suc i)) ∈ Guar m
           by (simp add: commit-es-def)
          with b1 c3 c4 c5 have (gets-es (esl ! i), gets-es (esl ! Suc i)) ∈ Guar m
          by (metis Cons-nth-drop-Suc b0 c1 length-drop list.sel(3) zero-less-diff)

          with p3 a4 show ?thesis by auto
        next
          assume c00: k ≠ length ?elst − 1
          with b2 have c0: k < length ?elst − 1 by auto
          show ?thesis
            proof(cases j = length (?elst!k))
              assume d0: j = length (?elst!k)
              with b2 have d1: drop i esl = concat (drop (Suc k) ?elst) by auto
              from b3 c0 have d2: length (?elst ! (Suc k)) ≥ 2 by auto
             from c0 have concat (drop (Suc k) ?elst) = ?elst ! (Suc k) @ concat
(drop (Suc (Suc k)) ?elst)
                  by (metis (no-types, hide-lams) Cons-nth-drop-Suc List.nth-tl
concat.simps(2) drop-Suc length-tl)
              with d1 have d3: drop i esl = ?elst ! (Suc k) @ concat (drop (Suc
(Suc k)) ?elst) by simp
              with b0 c0 d2 have d4: esl ! i = ?elst ! (Suc k) ! 0
                  by (metis (no-types, hide-lams) Cons-nth-drop-Suc One-nat-def
Suc-1
                  less-or-eq-imp-le not-less not-less-eq-eq nth-Cons-0 nth-append)

              from b0 c0 d2 d3 have d5: esl ! Suc i = ?elst ! (Suc k) ! 1
                by (metis (no-types, hide-lams) Cons-nth-drop-Suc One-nat-def
                Suc-1 Suc-le-lessD Suc-lessD nth-Cons-0 nth-Cons-Suc nth-append)

              from c0 have Suc k < length ?elst by auto
              show ?thesis
                proof(cases Suc k = length ?elst − 1)
                  assume e0: Suc k = length ?elst − 1
                  with d4 have e1: esl ! i = (last ?elst) ! 0
                    by (metis a0 b01 concat.simps(1) last-conv-nth)
                  from e0 d4 have e2: esl ! Suc i = (last ?elst) ! 1
                    by (metis a0 b01 concat.simps(1) d5 last-conv-nth)
                    from a4 have ∀ i. Suc i<length (last ?elst) ⟶ (∃ t. Γ ⊢ (last
?elst)!i −es−t→ (last ?elst)!(Suc i))
                        ⟶ (gets-es ((last ?elst)!i), gets-es ((last ?elst)!Suc i)) ∈
Guar m
                    by (simp add: commit-es-def)
                 with b1 e1 e2 have (gets-es (esl ! i), gets-es (esl ! Suc i)) ∈ Guar
m
                   by (metis One-nat-def Suc-1 Suc-le-lessD a0 b01 concat.simps(1)
d2 e0 last-conv-nth)
                  with p3 a4 show ?thesis by auto
```

next
assume *Suc k* ≠ *length ?elst* − *1*
with *c0* **have** *e0*: *Suc k* < *length ?elst* − *1* **by** *auto*
let *?els'* = *?elst*!(*Suc k*)@[(*?elst*!*Suc* (*Suc k*))!*0*]
from *e0* **have** *Suc* (*Suc k*) < *length ?elst* **by** *auto*
with *a2* **have** ∃ *m*∈*es*. *?els'*∈*commit-es* Γ (*Guar m*,*Post m*)
by *blast*
then obtain *m* **where** *e1*: *m*∈*es* ∧ *?els'*∈*commit-es* Γ (*Guar m*,*Post m*)
by *auto*
then have *e2*: ∀ *i*. *Suc i*<*length ?els'* ⟶ (∃ *t*. Γ ⊢ *?els'*!*i* −*es*−*t*→ *?els'*!(*Suc i*))
⟶ (*gets-es* (*?els'*!*i*), *gets-es* (*?els'*!*Suc i*)) ∈ *Guar m*
by (*simp add*: *commit-es-def*)
from *d4* **have** *e3*: *esl* ! *i* = *?els'* ! *0*
by (*metis* (*no-types*, *lifting*) *Suc-le-eq d2 dual-order.strict-trans lessI nth-append numeral-2-eq-2*)
from *d5* **have** *e4*: *esl* ! *Suc i* = *?els'* ! *1*
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-le-lessD d2 nth-append*)
from *b1 e3 e4* **have** *e5*: ∃ *t*. Γ ⊢ *?els'*!*0* −*es*−*t*→ *?els'*!*1* **by** *simp*
have *length ?els'* > *1* **using** *d2* **by** *auto*
with *e2 e5* **have** (*gets-es* (*?els'*!*0*), *gets-es* (*?els'*!*1*)) ∈ *Guar m* **by** *simp*
with *e3 e4* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) ∈ *Guar m* **by** *simp*
with *p3 e1* **show** *?thesis* **by** *auto*
qed
next
assume *d00*: *j* ≠ *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** ∃ *m*∈*es*. *?els′*∈*commit-es* Γ (*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e7*: *m*∈*es* ∧
?els′∈*commit-es* Γ (*Guar m*,*Post m*)
by *auto*
then have *e8*: ∀ *i*. *Suc i*<*length ?els′* ⟶ (∃ *t*. Γ ⊢ *?els′*!*i* −*es*−*t*→ *?els′*!(*Suc i*))
⟶ (*gets-es* (*?els′*!*i*), *gets-es* (*?els′*!*Suc i*)) ∈ *Guar m*
by (*simp add*: *commit-es-def*)

from *b1 e5 e6* **have** *e9*: ∃ *t*. Γ ⊢ *?els′*!*j* −*es*−*t*→ *?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*)) ∈ *Guar m* **by** *simp*
with *e5 e6* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) ∈ *Guar m* **by** *simp*
with *p3 e7* **show** *?thesis* **by** *auto*

next
assume *e0*: *j* ≠ *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** ∃ *m*∈*es*. *?els′*∈*commit-es* Γ (*Guar m*,*Post m*)
by *simp*
then obtain *m* **where** *e2*: *m*∈*es* ∧ *?els′*∈*commit-es* Γ (*Guar m*,*Post m*)
by *auto*
then have *e3*: ∀ *i*. *Suc i*<*length ?els′* ⟶ (∃ *t*. Γ ⊢ *?els′*!*i* −*es*−*t*→ *?els′*!(*Suc i*))
⟶ (*gets-es* (*?els′*!*i*), *gets-es* (*?els′*!*Suc i*)) ∈ *Guar m*

by (*simp add*: *commit-es-def*)
from *d1 e00* **have** *e4*: *esl* ! *i* = *?els'* ! *j*
by (*simp add*: *d0 nth-append*)
from *e1 e00* **have** *e5*: *esl* ! *Suc i* = *?els'* ! *Suc j*
by (*simp add*: *Suc-lessI nth-append*)
from *b1 e5 e4* **have** *e6*: $\exists\, t.\ \Gamma \vdash$ *?els'!j* $-es-t\rightarrow$ *?els'!Suc j* **by** *simp*
have *Suc j* < *length ?els'* **using** *e00* **by** *auto*
with *e3 e4 e6* **have** (*gets-es* (*?els'!j*), *gets-es* (*?els'!Suc j*)) $\in$ *Guar m* **by** *simp*
with *e4 e5* **have** (*gets-es* (*esl* ! *i*), *gets-es* (*esl* ! *Suc i*)) $\in$ *Guar m* **by** *simp*
with *p3 e2* **show** *?thesis* **by** *auto*
qed
qed
qed
}
then show *?thesis* **by** *auto*
qed

qed

lemma *EventSys-sound* :
$[\![\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef];$
$\forall\, ef{\in}es.\ pre \subseteq Pre\ ef;\ \ \forall\, ef{\in}es.\ rely \subseteq Rely\ ef;$
$\forall\, ef{\in}es.\ Guar\ ef \subseteq guar;\ \forall\, ef{\in}es.\ Post\ ef \subseteq post;$
$\forall\, ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2;$
$stable\text{-}e\ pre\ rely;\ \forall\, s.\ (s,\ s){\in}guar\ ]\!]$
$\Longrightarrow \Gamma \models EvtSys\ es\ sat_s\ [pre,\ rely,\ guar,\ post]$
proof −
assume *p0*: $\forall\, ef{\in}es.\ \Gamma \models ef\ sat_e\ [Pre\ ef,\ Rely\ ef,\ Guar\ ef,\ Post\ ef]$
and *p1*: $\forall\, ef{\in}es.\ pre \subseteq Pre\ ef$
and *p2*: $\forall\, ef{\in}es.\ rely \subseteq Rely\ ef$
and *p3*: $\forall\, ef{\in}es.\ Guar\ ef \subseteq guar$
and *p4*: $\forall\, ef{\in}es.\ Post\ ef \subseteq post$
and *p5*: $\forall\, ef1\ ef2.\ ef1{\in}es \wedge ef2{\in}es \longrightarrow Post\ ef1 \subseteq Pre\ ef2$
and *p6*: *stable-e pre rely*
and *p7*: $\forall\, s.\ (s,\ s){\in}guar$
then have $\forall\, s\ x.$ (*cpts-of-es* Γ (*EvtSys es*) *s x*) $\cap$ *assume-es* Γ (*pre*, *rely*) $\subseteq$ *commit-es* Γ (*guar*, *post*)
proof−
{
fix *s x*
have $\forall$ *esl*. *esl*$\in$(*cpts-of-es* Γ (*EvtSys es*) *s x*) $\cap$ *assume-es* Γ (*pre*, *rely*) $\longrightarrow$ *esl*$\in$ *commit-es* Γ (*guar*, *post*)
proof −
{
fix *esl*
assume *a0*: *esl*$\in$(*cpts-of-es* Γ (*EvtSys es*) *s x*) $\cap$ *assume-es* Γ (*pre*, *rely*)

then have $a1$: $esl \in (cpts\text{-}of\text{-}es\ \Gamma\ (EvtSys\ es)\ s\ x)$ **by** *simp*
then have $a1\text{-}1$: $esl!0 = (EvtSys\ es,\ s,\ x)$ **by** (*simp add*:*cpts-of-es-def*)
from $a1$ **have** $a1\text{-}2$: $esl \in cpts\text{-}es\ \Gamma$ **by** (*simp add*:*cpts-of-es-def*)
from $a0$ **have** $a2$: $esl \in assume\text{-}es\ \Gamma\ (pre,\ rely)$ **by** *simp*
then have $\forall i.\ Suc\ i < length\ esl \longrightarrow (\exists t.\ \Gamma \vdash esl!i\ {-es-t\rightarrow}\ esl!(Suc\ i)) \longrightarrow$
$(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof –
{
fix i
assume $b0$: $Suc\ i < length\ esl$
and $b1$: $\exists t.\ \Gamma \vdash esl!i\ {-es-t\rightarrow}\ esl!(Suc\ i)$
then obtain t **where** $b2$: $\Gamma \vdash esl!i\ {-es-t\rightarrow}\ esl!(Suc\ i)$ **by** *auto*
from $a1\text{-}2\ b0\ b1$ **have** $(gets\text{-}es\ (esl!i),\ gets\text{-}es\ (esl!Suc\ i)) \in guar$
proof(*cases* $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$)
assume $c0$: $\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$

with $b0$ **have** $getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$ **by** *simp*
moreover from $b0\ c0$ **have** $getspc\text{-}es\ (esl\ !\ (Suc\ i)) = EvtSys\ es$ **by** *simp*
ultimately have $\neg(\exists t.\ \Gamma \vdash esl!i\ {-es-t\rightarrow}\ esl!(Suc\ i))$
using *evtsys-not-eq-in-tran2 getspc-es-def* **by** (*metis surjective-pairing*)

with $b1$ **show** *?thesis* **by** *simp*
next
assume $c0$: $\neg\ (\forall i.\ Suc\ i \le length\ esl \longrightarrow getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es)$
then obtain m **where** $c1$: $Suc\ m \le length\ esl \wedge getspc\text{-}es\ (esl\ !\ m) \ne EvtSys\ es$
by *auto*
from $a1\text{-}1$ **have** $c2$: $getspc\text{-}es\ (esl!0) = EvtSys\ es$ **by** (*simp add*:*getspc-es-def*)
from $c1$ **have** $\exists i.\ i \le m \wedge getspc\text{-}es\ (esl\ !\ i) \ne EvtSys\ es$ **by** *auto*
with $a1\text{-}2\ a1\text{-}1\ c1\ c2$ **have** $\exists i.\ (i < m \wedge getspc\text{-}es\ (esl\ !\ i) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (esl\ !\ Suc\ i) \ne EvtSys\ es)$
$\wedge\ (\forall j.\ j < i \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es)$
using *evtsys-fst-ent* **by** *blast*
then obtain n **where** $c3$: $(n < m \wedge getspc\text{-}es\ (esl\ !\ n) = EvtSys\ es$
$\wedge\ getspc\text{-}es\ (esl\ !\ Suc\ n) \ne EvtSys\ es)$
$\wedge\ (\forall j.\ j < n \longrightarrow getspc\text{-}es\ (esl\ !\ j) = EvtSys\ es)$ **by** *auto*
with $b1$ **have** $c4$: $i \ge n$
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*. Γ ⊢ *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* Γ **using** *cpts-es-dropi*
by (*metis* (*no-types*, *lifting*) *b0 c4 drop-0 dual-order.strict-trans le-0-eq le-SucE le-imp-less-Suc zero-induct*)
from *c5 c6 c7* **have** ∃ *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* Γ (*pre*,*rely*)
proof(*cases n* = *0*)
assume *d0*: *n* = *0*
then have *?esl* = *esl* **by** *simp*
with *a2* **show** *?thesis* **by** *simp*
next
assume *d0*: *n* ≠ *0*
let *?eslh* = *take* (*n* + *1*) *esl*
from *a2* **have** *d1*: ∀ *i*. *Suc i*<*length ?esl* ⟶ Γ ⊢ *?esl*!*i* −*ese*→ *?esl*!(*Suc i*)
⟶ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *rely* **by** (*simp add*:*assume-es-def*)
have *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** $\forall$ *i*. *Suc i<length ?eslh* $\longrightarrow$ $\Gamma \vdash$ *?eslh*!*i* $-ese\rightarrow$ *?eslh*!(*Suc i*)
$\longrightarrow$ (*gets-es* (*?eslh*!*i*), *gets-es* (*?eslh*!*Suc i*)) $\in$ *rely* **by** (*simp add*:*assume-es-def*)
ultimately have *?eslh* $\in$ *assume-es* Γ (*pre*, *rely*) **by** (*simp add*:*assume-es-def*)
moreover
from *c3* **have** $\forall$ *i<length ?eslh*. *getspc-es* (*?eslh*!*i*) = *EvtSys es*
by (*metis Suc-eq-plus1 length-take less-antisym min-less-iff-conj nth-take*)
ultimately have $\forall$ *i<length ?eslh*. *gets-es* (*?eslh*!*i*) $\in$ *pre*
using *p6 pre-trans* **by** *blast*
with *d0* **have** *gets-es* (*?eslh* ! *n*) $\in$ *pre*
using *b0 c4* **by** *auto*
then show *?thesis* **by** (*simp add*: *c8 nth-via-drop*)
qed
with *d1* **show** *?thesis* **by** (*simp add*:*assume-es-def*)
qed

from *p0 p1 p2 p3 p4 p5 p6 p7 c7 c8 c10*
have *c11*: $\forall$ *i*. *Suc i<length ?esl* $\longrightarrow$ ($\exists$ *t*. $\Gamma \vdash$ *?esl*!*i* $-es-t\rightarrow$ *?esl*!(*Suc i*)) $\longrightarrow$
(*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *guar*
using *EventSys-sound-0*
[*of es* Γ *Pre Rely Guar Post pre rely guar post ?esl s x e s1 x1 xs*] **by** *simp*

from *b0 c4* **have** *c12*: *esl* ! *i* = *?esl* ! (*i* − *n*) **by** *auto*
moreover
from *b0 c4* **have** *c13*: *esl* ! *Suc i* = *?esl* ! *Suc* (*i* − *n*) **by** *auto*
moreover
from *b0 c4* **have** *Suc* (*i* − *n*) < *length ?esl* **by** *auto*
moreover
from *b1 c12 c13* **have** $\exists$ *t*. $\Gamma \vdash$ *?esl* ! (*i* − *n*) $-es-t\rightarrow$ *?esl* ! *Suc* (*i* − *n*) **by** *simp*
ultimately
have (*gets-es* (*?esl* ! (*i* − *n*)), *gets-es* (*?esl* ! *Suc* (*i* − *n*))) $\in$ *guar*
using *c11* **by** *simp*

with *c12 c13* **show** *?thesis* **by** *simp*

qed

}
then show *?thesis* **by** *auto*
qed
then have *esl*$\in$*commit-es* Γ (*guar*, *post*) **by** (*simp add*:*commit-es-def*)

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

then show $\Gamma \models$ *EvtSys es sat*$_s$ [*pre*, *rely*, *guar*, *post*] **by** (*simp add*:*es-validity-def*)
qed

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

lemma *EventSys-sound'*:
assumes *p0*: $\forall$ *ef*$\in$*esf*. $\Gamma \vdash E_e$ *ef sat*$_e$ [*Pre*$_e$ *ef*, *Rely*$_e$ *ef*, *Guar*$_e$ *ef*, *Post*$_e$ *ef*]
and *p1*: $\forall$ *ef*$\in$*esf*. *pre* $\subseteq$ *Pre*$_e$ *ef*
and *p2*: $\forall$ *ef*$\in$*esf*. *rely* $\subseteq$ *Rely*$_e$ *ef*
and *p3*: $\forall$ *ef*$\in$*esf*. *Guar*$_e$ *ef* $\subseteq$ *guar*
and *p4*: $\forall$ *ef*$\in$*esf*. *Post*$_e$ *ef* $\subseteq$ *post*
and *p5*: $\forall$ *ef1 ef2*. *ef1*$\in$*esf* $\wedge$ *ef2*$\in$*esf* $\longrightarrow$ *Post*$_e$ *ef1* $\subseteq$ *Pre*$_e$ *ef2*
and *p6*: *stable-e pre rely*

and $p7$: $\forall s.\ (s,\ s) \in guar$
shows $\Gamma \models$ *evtsys-spec* (*rgf-EvtSys esf*) sat_s [*pre*, *rely*, *guar*, *post*]
proof −
let *?es* = *Domain esf*
let *?RG* = λe. *SOME rg*. (*e*,*rg*)$\in$*esf*
have *a1*: $\forall e \in$*?es*. $\exists ef \in$*esf*. *?RG e* = *snd ef* **by** (*metis Domain.cases snd-conv someI*)

let *?Pre* = *pre-rgf* $\circ$ *?RG*
let *?Rely* = *rely-rgf* $\circ$ *?RG*
let *?Guar* = *guar-rgf* $\circ$ *?RG*
let *?Post* = *post-rgf* $\circ$ *?RG*
from *p0* **have** *a2*: $\forall i \in esf.\ \Gamma \models E_e\ i\ sat_e\ [Pre_e\ i,\ Rely_e\ i,\ Guar_e\ i,\ Post_e\ i]$
by (*simp add*: *rgsound-e*)
have $\forall ef \in$*?es*. $\Gamma \models$ *ef* sat_e [*?Pre ef*, *?Rely ef*, *?Guar ef*, *?Post ef*]
by (*metis* (*mono-tags*, *lifting*) *Domain.cases* E_e*-def* $Guar_e$*-def* $Post_e$*-def* Pre_e*-def* $Rely_e$*-def a2 comp-apply fst-conv snd-conv someI-ex*)
moreover
have $\forall ef \in$*?es*. *pre* $\subseteq$ *?Pre ef* **by** (*metis* Pre_e*-def a1 comp-def p1*)
moreover
have $\forall ef \in$*?es*. *rely* $\subseteq$ *?Rely ef* **by** (*metis* $Rely_e$*-def a1 comp-apply p2*)
moreover
have $\forall ef \in$*?es*. *?Guar ef* $\subseteq$ *guar* **by** (*metis* $Guar_e$*-def a1 comp-apply p3*)
moreover
have $\forall ef \in$*?es*. *?Post ef* $\subseteq$ *post* **by** (*metis* $Post_e$*-def a1 comp-apply p4*)
moreover
have $\forall$ *ef1 ef2*. *ef1* $\in$ *?es* $\wedge$ *ef2* $\in$ *?es* $\longrightarrow$ *?Post ef1* $\subseteq$ *?Pre ef2*
by (*metis* (*mono-tags*, *lifting*) $Post_e$*-def* Pre_e*-def a1 comp-def p5*)
ultimately have $\Gamma \models$ *EvtSys* (*Domain esf*) sat_s [*pre*, *rely*, *guar*, *post*]
using *p6 p7 EventSys-sound* [*of ?es* Γ *?Pre ?Rely ?Guar ?Post pre rely guar post*] **by** *simp*
then show $\Gamma \models$ *evtsys-spec* (*rgf-EvtSys esf*) sat_s [*pre*, *rely*, *guar*, *post*] **by** *simp*
qed

theorem *rgsound-es*: $\Gamma \vdash$ (*esf*::('*l*,'*k*,'*s*,'*prog*) *rgformula-ess*) sat_s [*pre*, *rely*, *guar*, *post*]
$\Longrightarrow \Gamma \models$ *evtsys-spec esf* sat_s [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-es.induct*)
apply *auto*[*1*]
using *EventSeq-sound rgsound-e* **apply** *smt*
using *EventSys-sound'* **apply** *blast*
using *esys-seq-sound* **apply** *blast*
done

7.5 Soundness of Parallel Event Systems

lemma *conjoin-comm-imp-rely-n*[*rule-format*]:
$[\![\forall k.\ pre \subseteq Pre\ k;\ \forall k.\ rely \subseteq Rely\ k;$

$\forall k\ j.\ j{\neq}k \longrightarrow Guar\ j \subseteq Rely\ k$;
$\forall k.\ cs\ k \in commit\text{-}es\ \Gamma\ (Guar\ k,\ Post\ k)$;
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$; $c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely)$; $\Gamma\ c \propto cs\rrbracket \Longrightarrow$
$\forall n\ k.\ n \leq length\ (cs\ k) \wedge n > 0 \longrightarrow take\ n\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
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*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p0*: $\forall k.\ cs\ k \in commit\text{-}es\ \Gamma\ (Guar\ k,\ Post\ k)$
from *p6* **have** *p8*: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)
from *p4 p6* **have** *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$ **using** *conjoin-imp-cptses-k* **by** *auto*
then have *p9*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma \wedge cs\ k\ !0 = (pes\ k,s,x)$ **by** (*simp add*:*cpts-of-es-def*)
from *p6* **have** *p10*: $\forall k\ j.\ j < length\ c \longrightarrow gets\ (c!j) = gets\text{-}es\ ((cs\ k)!j)$ **by** (*simp add*:*conjoin-def same-state-def*)
{
fix *n*
have $\forall k.\ n \leq length\ (cs\ k) \wedge n > 0 \longrightarrow take\ n\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof(*induct n*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc m*)
assume *b0*: $\forall k.\ m \leq length\ (cs\ k) \wedge 0 < m \longrightarrow take\ m\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
{
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 $take\ (Suc\ m)\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof(*cases* $m = 0$)
assume *d0*: $m = 0$
have $gets\text{-}es\ (take\ (Suc\ m)\ (cs\ k)!0) \in Pre\ k$
proof −
from *p6 c2 c3* **have** $gets\ (c!0) = gets\text{-}es\ ((cs\ k)!0)$
by (*simp add*:*conjoin-def same-state-def*)

moreover
from *p5* **have** *gets* $(c!0) \in$ *pre* **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
from *d0* **have** *d1*: *length* (*take* (*Suc m*) (*cs k*)) = *1*
using *One-nat-def c2 gr0-implies-Suc length-take min-0R min-Suc-Suc*
by *fastforce*
moreover
from *d1* **have** $\forall$ *i*. *Suc i* < *length* (*take* (*Suc m*) (*cs k*))
$\longrightarrow \Gamma \vdash$ (*take* (*Suc m*) (*cs k*)) ! *i* $-ese\rightarrow$ (*take* (*Suc m*) (*cs k*))
! *Suc i*
$\longrightarrow$ (*gets-es* ((*take* (*Suc m*) (*cs k*)) ! *i*), *gets-es* ((*take* (*Suc m*)
(*cs k*)) ! *Suc i*)) $\in$ *rely*
by *auto*
moreover
have *assume-es* Γ (*Pre k*, *Rely k*) = {*c*. *gets-es* (*c* ! *0*) $\in$ *Pre k* $\wedge$
($\forall$ *i*. *Suc i* < *length c* $\longrightarrow \Gamma \vdash$ *c* ! *i* $-ese\rightarrow$ *c* ! *Suc i*
$\longrightarrow$ (*gets-es* (*c* ! *i*), *gets-es* (*c* ! *Suc i*)) $\in$ *Rely k*)} **by** (*simp*
add:*assume-es-def*)
ultimately show *?thesis* **using** *Suc-neq-Zero less-one mem-Collect-eq*
by *auto*
next
assume $m \neq 0$
then have *dd0*: $m > 0$ **by** *simp*
with *b0 c0* **have** *dd1*: *take m* (*cs k*) $\in$ *assume-es* Γ (*Pre k*, *Rely k*)
by *simp*

have *gets-es* (*?esl* ! *0*) $\in$ *Pre k*
proof −
from *p6 c2 c3* **have** *gets* $(c!0)$ = *gets-es* $((cs\ k)!0)$
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** *gets* $(c!0) \in$ *pre* **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
have $\forall$ *i*. *Suc i*<*length ?esl* $\longrightarrow$
$\Gamma \vdash$ *?esl*!*i* $-ese\rightarrow$ *?esl*!(*Suc i*) $\longrightarrow$
(*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *Rely k*
proof −
{
fix *i*
assume *d0*: *Suc i*<*length ?esl*
and *d1*: $\Gamma \vdash$ *?esl*!*i* $-ese\rightarrow$ *?esl*!*Suc i*
then have *d2*: *?esl*!*i* = (*cs k*)!*i* $\wedge$ *?esl*!*Suc i* = (*cs k*)! *Suc i*
by *auto*
from *p6 c3 d0* **have** *d4*: ($\exists$ *t k*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\wedge$
($\forall$ *k t*. ($\Gamma \vdash$ *c*!*i* $-pes-(t\sharp k)\rightarrow$ *c*!*Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k*!*i*

$-es-(t\sharp k)\rightarrow$ *cs k*! *Suc i*) $\wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow\ cs\ k'!\ Suc\ i))))$
$\vee$
$(\Gamma \vdash (c!i)\ -pese\rightarrow (c!Suc\ i) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow$
$((cs\ k)!\ Suc\ i))))$
by (*simp add*:*conjoin-def compat-tran-def*)
from *d1* **have** *d5*: $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i)$
by (*simp add*: *d2*)
from *d4* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) $\in$ *Rely k*
proof
assume *e0*: $\exists\, t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i$
$-es-(t\sharp k)\rightarrow cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow cs\ k'!\ Suc\ i)))$
then obtain *ct* **and** k' **where** *e1*: $(\Gamma \vdash (c!i)\ -pes-(ct\sharp k')\rightarrow$
$(c!Suc\ i)) \wedge$
$(\Gamma \vdash ((cs\ k')!i)\ -es-(ct\sharp k')\rightarrow ((cs\ k')!\ Suc\ i))$ **by** *auto*
with *p6 p8 d0 d5* **have** *e2*: $k \neq k'$
using *conjoin-def*[*of* Γ *c cs*] *same-spec-def*[*of c cs*]
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i)$ **by**
auto
with *d0* **have** $\Gamma \vdash (?esl!i)\ -ese\rightarrow (?esl!\ Suc\ i)$ **by** *auto*
then show *?thesis*
proof(*cases* $i < m - 1$)
assume *f0*: $i < m - 1$
with *d2* **have** *f1*:*take* (*Suc m*) (*cs k*) ! *i* = *take m* (*cs k*) ! *i*
by (*simp add*: *diff-less-Suc less-trans-Suc*)

from *f0* **have** *f2*: *take* (*Suc m*) (*cs k*) ! *Suc i* = *take m* (*cs*
k) ! *Suc i*
by (*simp add*: *d2 gr-implies-not0 nat-le-linear*)
from *dd1* **have** $\forall\, i.$ *Suc i*<*length* (*take m* (*cs k*)) $\longrightarrow$
$\Gamma \vdash$ (*take m* (*cs k*))!*i* $-ese\rightarrow$ (*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′*) ∈ *assume-es* Γ (*Pre k′*, *Rely k′*)
by (*metis Suc-leD p8*)
moreover
from *e1 f0* **have** ¬(Γ ⊢ *cs k′* ! (*m*−*1*) −*ese*→ *cs k′* !*m*)
using *Suc-pred′ dd0 es-tran-not-etran1* **by** *fastforce*
ultimately have *f1*: *take* (*Suc m*) (*cs k′*) ∈ *assume-es* Γ (*Pre k′*, *Rely k′*)
using *assume-es-one-more*[*of cs k′* Γ *m Pre k′ Rely k′*] *p8 p9 c0 dd0*
by (*simp add*: *Suc-le-eq*)
from *p7* **have** *cs k′* ∈ *cpts-of-es* Γ (*pes k′*) *s x* **by** *simp*
with *p8 c0 dd0* **have** *f2*: *?esl2*∈*cpts-of-es* Γ (*pes k′*) *s x*
using *cpts-es-take*[*of cs k′* Γ *m*] *cpts-of-es-def*[*of* Γ *pes k′ s x*]
by (*simp add*: *Suc-le-lessD*)
from *p0 p8 c0* **have** *?esl2*∈*commit-es* Γ (*Guar k′*, *Post k′*)
using *commit-es-take-n*[*of Suc m cs k′* Γ *Guar k′ Post k′*] **by** *auto*
then have ∀ *i*. *Suc i*<*length ?esl2* ⟶
(∃ *t*. Γ ⊢ *?esl2*!*i* −*es*−*t*→ *?esl2*!(*Suc i*)) ⟶
(*gets-es* (*?esl2*!*i*), *gets-es* (*?esl2*!*Suc i*)) ∈ *Guar k′*
by (*simp add*:*commit-es-def*)

with *p8 e1 f0 c0 dd0* **have** (*gets-es* (*?esl2* ! (*m*−*1*)), *gets-es* (*?esl2* ! *m*))∈*Guar k′*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred diff-less-Suc length-take lessI min.absorb2 nth-take*)
with *p3 p10 c0 f0 e2* **show** *?thesis*
by (*smt Suc-diff-1 Suc-leD c3 dd0 le-less-linear not-less-eq-eq nth-take subsetCE*)
qed
next
assume *e0*: ((Γ ⊢ (*c*!*i*) −*pese*→ (*c*!*Suc i*)) ∧ (∀ *k*. (Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*))))
from *p5* **have** ∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*pese*→ *c*!(*Suc i*) ⟶
(*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely*
by (*simp add*:*assume-pes-def*)
moreover
from *p8 c0 d0* **have** *e1*:*Suc i* < *length c* **by** *simp*
ultimately have (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely* **using** *e0* **by** *simp*
with *p2* **have** (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *Rely k* **by** *auto*
with *p8 p10 c0 d0* **show** *?thesis*
using *Suc-lessD e1 d2* **by** *auto*
qed
}
then show *?thesis* **by** *auto*

qed
ultimately show *?thesis* **by** (*simp add*:*assume-es-def*)
qed
}
then show *?case* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *conjoin-comm-imp-rely*:
$[\![\forall k.\ pre \subseteq Pre\ k;\ \forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow Guar\ j \subseteq Rely\ k;$
$\forall k.\ cs\ k \in commit\text{-}es\ \Gamma\ (Guar\ k,\ Post\ k);$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);\ \Gamma\ c \propto cs]\!] \Longrightarrow$
$\forall k.\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof −
assume *a1*: $\forall k.\ pre \subseteq Pre\ k$
assume *a2*: $\forall k.\ rely \subseteq Rely\ k$
assume *a3*: $\forall k\ j.\ j \neq k \longrightarrow Guar\ j \subseteq Rely\ k$
assume *a4*: $\forall k.\ cs\ k \in commit\text{-}es\ \Gamma\ (Guar\ k,\ Post\ k)$
assume *a5*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
assume *a6*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
assume *a7*: $\Gamma\ c \propto cs$
have *f8*: $c \neq []$
using *a5 cpts-of-pes-def* **by** *force*
from *a7* **have** *p8*: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)
{
fix *k*
have $(cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
using *a1 a2 a3 a4 a5 a6 a7 p8 f8*
conjoin-comm-imp-rely-n[*of pre Pre rely Rely Guar cs* Γ *Post c pes s x length* (*cs k*) *k*] **by** *force*
}
then show *?thesis* **by** *simp*
qed

lemma *cpts-es-sat-rely*[*rule-format*]:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);$
$\Gamma\ c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x]\!] \Longrightarrow$
$\forall n\ k.\ n \leq length\ (cs\ k) \wedge n > 0 \longrightarrow take\ n\ (cs\ k){\in}assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$

and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k$
and *p4*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
from *p6* **have** *p8*: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)
from *p7* **have** *p9*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma$ **using** *cpts-of-es-def mem-Collect-eq* **by** *fastforce*
from *p6* **have** *p10*: $\forall k\ j.\ j < length\ c \longrightarrow gets\ (c!j) = gets\text{-}es\ ((cs\ k)!j)$ **by** (*simp add*:*conjoin-def same-state-def*)
{
fix *n*
have $\forall k.\ n \leq length\ (cs\ k) \wedge n > 0 \longrightarrow take\ n\ (cs\ k){\in}assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof(*induct n*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc m*)
assume *b0*: $\forall k.\ m \leq length\ (cs\ k) \wedge 0 < m \longrightarrow take\ m\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$

{
fix *k*
assume *c0*: $Suc\ m \leq length\ (cs\ k) \wedge 0 < Suc\ m$
from *p7* **have** *c2*: $length\ (cs\ k) > 0$
by (*metis* (*no-types*, *lifting*) *cpts-es-not-empty cpts-of-es-def gr0I length-0-conv mem-Collect-eq*)
from *p6* **have** *c3*: $length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)

let $?esl = take\ (Suc\ m)\ (cs\ k)$
have $?esl \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof(*cases* $m = 0$)
assume *d0*: $m = 0$
have $gets\text{-}es\ (take\ (Suc\ m)\ (cs\ k)!0) \in Pre\ k$
proof −
from *p6 c2 c3* **have** $gets\ (c!0) = gets\text{-}es\ ((cs\ k)!0)$
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** $gets\ (c!0) \in pre$ **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
from *d0* **have** *d1*: $length\ (take\ (Suc\ m)\ (cs\ k)) = 1$
using *One-nat-def c2 gr0-implies-Suc length-take min-0R min-Suc-Suc* **by** *fastforce*

moreover
from *d1* **have** $\forall i.\ Suc\ i < length\ (take\ (Suc\ m)\ (cs\ k))$
$\longrightarrow \Gamma \vdash (take\ (Suc\ m)\ (cs\ k))\ !\ i\ -ese\rightarrow (take\ (Suc\ m)\ (cs\ k))\ !\ Suc\ i$
$\longrightarrow (gets\text{-}es\ ((take\ (Suc\ m)\ (cs\ k))\ !\ i),\ gets\text{-}es\ ((take\ (Suc\ m)\ (cs\ k))\ !\ Suc\ i)) \in rely$
by *auto*
moreover
have $assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k) = \{c.\ gets\text{-}es\ (c\ !\ 0) \in Pre\ k\ \wedge$
$(\forall i.\ Suc\ i < length\ c \longrightarrow \Gamma \vdash c\ !\ i\ -ese\rightarrow c\ !\ Suc\ i$
$\longrightarrow (gets\text{-}es\ (c\ !\ i),\ gets\text{-}es\ (c\ !\ Suc\ i)) \in Rely\ k)\}$ **by** (*simp add*:*assume-es-def*)
ultimately show *?thesis* **using** *Suc-neq-Zero less-one mem-Collect-eq* **by** *auto*
next
assume $m \neq 0$
then have *dd0*: $m > 0$ **by** *simp*
with *b0 c0* **have** *dd1*: $take\ m\ (cs\ k) \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$ **by** *simp*

have $gets\text{-}es\ (?esl\ !\ 0) \in Pre\ k$
proof $-$
from *p6 c2 c3* **have** $gets\ (c!0) = gets\text{-}es\ ((cs\ k)!0)$
by (*simp add*:*conjoin-def same-state-def*)
moreover
from *p5* **have** $gets\ (c!0) \in pre$ **by** (*simp add*:*assume-pes-def*)
ultimately show *?thesis* **using** *p1 p8* **by** *auto*
qed
moreover
have $\forall i.\ Suc\ i<length\ ?esl \longrightarrow$
$\Gamma \vdash ?esl!i\ -ese\rightarrow ?esl!(Suc\ i) \longrightarrow$
$(gets\text{-}es\ (?esl!i),\ gets\text{-}es\ (?esl!Suc\ i)) \in Rely\ k$
proof $-$
{
fix *i*
assume *d0*: $Suc\ i<length\ ?esl$
and *d1*: $\Gamma \vdash ?esl!i\ -ese\rightarrow ?esl!Suc\ i$
then have *d2*: $?esl!i = (cs\ k)!i \wedge ?esl!Suc\ i = (cs\ k)!\ Suc\ i$
by *auto*
from *p6 c3 d0* **have** *d4*: $(\exists t\ k.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ -pes-(t\sharp k)\rightarrow c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i\ -es-(t\sharp k)\rightarrow cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ -ese\rightarrow cs\ k'!\ Suc\ i))))$
$\vee$
$((\Gamma \vdash (c!i)\ -pese\rightarrow (c!Suc\ i)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i))))$
by (*simp add*:*conjoin-def compat-tran-def*)
from *d1* **have** *d5*: $\Gamma \vdash ((cs\ k)!i)\ -ese\rightarrow ((cs\ k)!\ Suc\ i)$
by (*simp add*: *d2*)

from *d4* **have** (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *Rely k*
proof
assume *e0*: ∃ *t k*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ∧
(∀ *k t*. (Γ ⊢ *c*!*i* −*pes*−(*t*♯*k*)→ *c*!*Suc i*) ⟶ (Γ ⊢ *cs k*!*i* −*es*−(*t*♯*k*)→ *cs k*! *Suc i*) ∧
(∀ *k*′. *k*′ ≠ *k* ⟶ (Γ ⊢ *cs k*′!*i* −*ese*→ *cs k*′! *Suc i*)))
then obtain *ct* **and** *k*′ **where** *e1*: (Γ ⊢ (*c*!*i*) −*pes*−(*ct*♯*k*′)→ (*c*!*Suc i*)) ∧
(Γ ⊢ ((*cs k*′)!*i*) −*es*−(*ct*♯*k*′)→ ((*cs k*′)! *Suc i*)) **by** *auto*
with *p6 p8 d0 d5* **have** *e2*: *k* ≠ *k*′
using *conjoin-def*[*of* Γ *c cs*] *same-spec-def*[*of c cs*]
es-tran-not-etran1 **by** *blast*

with *e0 e1* **have** *e3*: Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*) **by** *auto*
with *d0* **have** Γ ⊢ (*?esl*!*i*) −*ese*→ (*?esl*! *Suc i*) **by** *auto*
then show *?thesis*
proof(*cases i* < *m* − *1*)
assume *f0*: *i* < *m* − *1*
with *d2* **have** *f1*:*take* (*Suc m*) (*cs k*) ! *i* = *take m* (*cs k*) ! *i*
by (*simp add*: *diff-less-Suc less-trans-Suc*)

from *f0* **have** *f2*: *take* (*Suc m*) (*cs k*) ! *Suc i* = *take m* (*cs k*) ! *Suc i*
by (*simp add*: *d2 gr-implies-not0 nat-le-linear*)
from *dd1* **have** ∀ *i*. *Suc i*<*length* (*take m* (*cs k*)) ⟶
Γ ⊢ (*take m* (*cs k*))!*i* −*ese*→ (*take m* (*cs k*))!(*Suc i*) ⟶
(*gets-es* ((*take m* (*cs k*))!*i*), *gets-es* ((*take m* (*cs k*))!*Suc i*)) ∈ *Rely k*
by (*simp add*:*assume-es-def*)
with *dd0 f0* **have** (*gets-es* (*take m* (*cs k*) ! *i*), *gets-es* (*take m* (*cs k*) ! *Suc i*)) ∈ *Rely k*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-mono Suc-pred d0 d1 f1 f2 length-take min-less-iff-conj*)
with *f1 f2* **show** *?thesis* **by** *simp*
next
assume ¬(*i* < *m* − *1*)
with *d0* **have** *f0*: *i* = *m* − *1*
by (*simp add*: *c0 dd0 less-antisym min.absorb2*)
let *?esl2* = *take* (*Suc m*) (*cs k*′)

from *b0 c0 dd0* **have** *take m* (*cs k*′) ∈ *assume-es* Γ (*Pre k*′, *Rely k*′)
by (*metis Suc-leD p8*)
moreover
from *e1 f0* **have** ¬(Γ ⊢ *cs k*′ ! (*m*−*1*) −*ese*→ *cs k*′ !*m*)
using *Suc-pred*′ *dd0 es-tran-not-etran1* **by** *fastforce*
ultimately have *f1*: *take* (*Suc m*) (*cs k*′) ∈ *assume-es* Γ (*Pre k*′, *Rely k*′)

using *assume-es-one-more*[*of cs k′ Γ m Pre k′ Rely k′*] *p8 p9 c0 dd0*
by (*simp add*: *Suc-le-eq*)
from *p7* **have** *cs k′* ∈ *cpts-of-es* Γ (*pes k′*) *s x* **by** *simp*
with *p8 c0 dd0* **have** *f2*: *?esl2*∈*cpts-of-es* Γ (*pes k′*) *s x*
using *cpts-es-take*[*of cs k′ Γ m*] *cpts-of-es-def*[*of Γ pes k′ s x*]
by (*simp add*: *Suc-le-lessD*)
from *p0* **have** *f3*: Γ $\models$ *pes k′* sat_s [*Pre k′*, *Rely k′*, *Guar k′*, *Post k′*] **by** *simp*
with *f1 f2* **have** *?esl2*∈*commit-es* Γ (*Guar k′*, *Post k′*)
using *es-validity-def*[*of Γ pes k′ Pre k′ Rely k′ Guar k′ Post k′*]
by *auto*
then have ∀ *i*. *Suc i*<*length ?esl2* ⟶
(∃ *t*. Γ ⊢ *?esl2*!*i* −*es*−*t*→ *?esl2*!(*Suc i*)) ⟶
(*gets-es* (*?esl2*!*i*), *gets-es* (*?esl2*!*Suc i*)) ∈ *Guar k′*
by (*simp add*:*commit-es-def*)

with *p8 e1 f0 c0 dd0* **have** (*gets-es* (*?esl2* ! (*m*−*1*)), *gets-es* (*?esl2* ! *m*))∈*Guar k′*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred diff-less-Suc length-take lessI min.absorb2 nth-take*)
with *p3 p10 c0 f0 e2* **show** *?thesis*
by (*smt Suc-diff-1 Suc-leD c3 dd0 le-less-linear not-less-eq-eq nth-take subsetCE*)
qed
next
assume *e0*: ((Γ ⊢ (*c*!*i*) −*pese*→ (*c*!*Suc i*)) ∧ (∀ *k*. (Γ ⊢ ((*cs k*)!*i*) −*ese*→ ((*cs k*)! *Suc i*))))
from *p5* **have** ∀ *i*. *Suc i*<*length c* ⟶
Γ ⊢ *c*!*i* −*pese*→ *c*!(*Suc i*) ⟶
(*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely*
by (*simp add*:*assume-pes-def*)
moreover
from *p8 c0 d0* **have** *e1*:*Suc i* < *length c* **by** *simp*
ultimately have (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *rely* **using** *e0* **by** *simp*
with *p2* **have** (*gets* (*c*!*i*), *gets* (*c*!*Suc i*)) ∈ *Rely k* **by** *auto*
with *p8 p10 c0 d0* **show** *?thesis*
using *Suc-lessD e1 d2* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

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

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

lemma *es-tran-sat-guar-aux*:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);$
$\Gamma\ c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x\ ]\!]$
$\Longrightarrow \forall k\ i\ m.\ m \leq length\ c \longrightarrow Suc\ i < length\ (take\ m\ (cs\ k)) \longrightarrow (\exists t.(\Gamma \vdash (take\ m\ (cs\ k))!i{-}es{-}t{\rightarrow}((take\ m\ (cs\ k))!Suc\ i)))$
$\longrightarrow (gets\text{-}es\ ((take\ m\ (cs\ k))!i), gets\text{-}es\ ((take\ m\ (cs\ k))!Suc\ i)) \in Guar\ k$

proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow Guar\ j \subseteq Rely\ k$
and *p4*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
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*: $\exists t.(\Gamma \vdash (take\ m\ (cs\ k))!i{-}es{-}t{\rightarrow}((take\ m\ (cs\ k))!Suc\ i))$
have $(gets\text{-}es\ ((take\ m\ (cs\ k))!i), gets\text{-}es\ ((take\ m\ (cs\ k))!Suc\ i)) \in Guar\ k$
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** $cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$ **by** *simp*
then have $cs\ k!0{=}(pes\ k,s,x) \wedge cs\ k \in cpts\text{-}es\ \Gamma$ **by** (*simp add*:*cpts-of-es-def*)
with *b0* **have** $?esl!0{=}(pes\ k,s,x) \wedge\ ?esl \in cpts\text{-}es\ \Gamma$
by (*metis Suc-pred a0 cpts-es-take leD not-less-eq nth-take p8*)
then have *r1*: $?esl \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$ **by** (*simp add*:*cpts-of-es-def*)
from *p0 p1 p2 p3 p4 p5 p6 p7*
have $\forall n.\ n \leq length\ (cs\ k) \wedge n > 0 \longrightarrow take\ n\ (cs\ k){\in}assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$

using *cpts-es-sat-rely*[*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*]
by *auto*
with *p8 a0 b0* **have** *r2*: *?esl*∈*assume-es* Γ (*Pre k*, *Rely k*) **by** *auto*

from *p0* **have** (*cpts-of-es* Γ (*pes k*) *s x*) ∩ *assume-es* Γ (*Pre k*, *Rely k*) ⊆ *commit-es* Γ (*Guar k*, *Post k*)
by (*simp add*:*es-validity-def*)
with *r1 r2* **have** *?esl* ∈ *commit-es* Γ (*Guar k*, *Post k*)
using *IntI subsetCE* **by** *auto*
then have ∀ *i*. *Suc i*<*length ?esl* ⟶
(∃ *t*. Γ ⊢ *?esl*!*i* −*es*−*t*→ *?esl*!(*Suc i*)) ⟶ (*gets-es* (*?esl*!*i*), *gets-es* (*?esl*!*Suc i*)) ∈ *Guar k*
by (*simp add*:*commit-es-def*)
with *a1 a2* **show** *?thesis* **by** *auto*
qed
}
then show *?thesis* **by** *auto*
qed

lemma *es-tran-sat-guar*:
⟦∀ *k*. Γ ⊨ (*pes k*) sat_s [*Pre k*, *Rely k*, *Guar k*, *Post k*];
∀ *k*. *pre* ⊆ *Pre k*;
∀ *k*. *rely* ⊆ *Rely k*;
∀ *k j*. *j*≠*k* ⟶ *Guar j* ⊆ *Rely k*;
c ∈ *cpts-of-pes* Γ *pes s x*; *c*∈*assume-pes* Γ (*pre*, *rely*);
Γ *c* ∝ *cs*; ∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes k*) *s x* ⟧
⟹∀ *k i*. *Suc i* < *length* (*cs k*) ⟶ (∃ *t*.(Γ ⊢ (*cs k*)!*i*−*es*−*t*→(*cs k*)!*Suc i*))
⟶ (*gets-es* ((*cs k*)!*i*),*gets-es* ((*cs k*)!*Suc i*)) ∈ *Guar k*
proof −
assume *p0*: ∀ *k*. Γ ⊨ (*pes k*) sat_s [*Pre k*, *Rely k*, *Guar k*, *Post k*]
and *p1*: ∀ *k*. *pre* ⊆ *Pre k*
and *p2*: ∀ *k*. *rely* ⊆ *Rely k*
and *p3*: ∀ *k j*. *j*≠*k* ⟶ *Guar j* ⊆ *Rely k*
and *p4*: *c* ∈ *cpts-of-pes* Γ *pes s x*
and *p5*: *c* ∈ *assume-pes* Γ (*pre*, *rely*)
and *p6*: Γ *c* ∝ *cs*
and *p7*: ∀ *k*. *cs k* ∈ *cpts-of-es* Γ (*pes k*) *s x*
then have ∀ *k i m*. *m* ≤ *length c* ⟶ *Suc i* < *length* (*take m* (*cs k*)) ⟶ (∃ *t*.(Γ ⊢ (*take m* (*cs k*))!*i*−*es*−*t*→((*take m* (*cs k*))!*Suc i*)))
⟶ (*gets-es* ((*take m* (*cs k*))!*i*),*gets-es* ((*take m* (*cs k*))!*Suc i*)) ∈ *Guar k*
using *es-tran-sat-guar-aux* [*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*] **by** *simp*
moreover
from *p6* **have** ∀ *k*. *length c* = *length* (*cs k*) **by** (*simp add*:*conjoin-def same-length-def*)
ultimately show *?thesis* **by** *auto*
qed

lemma *conjoin-es-sat-assume*:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely);$
$\Gamma\ c \propto cs;\ \forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x\ ]\!]$
$\Longrightarrow \forall k.\ cs\ k \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*[*rule-format*]: $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k$
and *p4*: $c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x$
and *p5*: $c \in assume\text{-}pes\ \Gamma\ (pre,\ rely)$
and *p6*: $\Gamma\ c \propto cs$
and *p7*: $\forall k.\ cs\ k \in cpts\text{-}of\text{-}es\ \Gamma\ (pes\ k)\ s\ x$
from *p6* **have** *p11*[*rule-format*]: $\forall k.\ length\ (cs\ k) = length\ c$ **by** (*simp add*:*conjoin-def same-length-def*)
from *p7* **have** *p12*: $\forall k.\ cs\ k \in cpts\text{-}es\ \Gamma$ **using** *cpts-of-es-def mem-Collect-eq* **by** *fastforce*
with *p11* **have** $c \neq Nil$ **using** *cpts-es-not-empty length-0-conv* **by** *auto*
then have *p13*: $length\ c > 0$ **by** *auto*
{
fix *k*
have $cs\ k \in assume\text{-}es\ \Gamma\ (Pre\ k,\ Rely\ k)$
using *p0 p1 p2 p3 p4 p5 p6 p7 p13 p11*
cpts-es-sat-rely[*of* Γ *pes Pre Rely Guar Post pre rely c s x cs length* (*cs k*) *k*] **by** *force*
}
then show *?thesis* **by** *auto*
qed

lemma *pes-tran-sat-guar*:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$\forall k.\ Guar\ k \subseteq guar;$
$c \in cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x;\ c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely)]\!]$
$\Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}\ c!(Suc\ i))$
$\longrightarrow (gets\ (c!i), gets\ (c!Suc\ i)) \in guar$
proof −
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k$
and *p4*: $\forall k.\ Guar\ k \subseteq guar$

and $p5$: $c \in$ *cpts-of-pes* Γ *pes s x*
and $p6$: $c \in$*assume-pes* Γ (*pre*, *rely*)
{
fix i
assume $a0$: *Suc i* $<$ *length c*
and $a1$: $\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}\ c!(Suc\ i)$
from $p5$ **have** $\exists cs.\ (\forall k.\ (cs\ k) \in$ *cpts-of-es* $\Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$
by (*meson cpt-imp-exist-conjoin-cs*)
then obtain *cs* **where** $a2$: $(\forall k.\ (cs\ k) \in$ *cpts-of-es* $\Gamma\ (pes\ k)\ s\ x) \wedge \Gamma\ c \propto cs$ **by** *auto*
then have *compat-tran* Γ *c cs* **by** (*simp add*:*conjoin-def*)
with $a0$ **have** $a3$: $(\exists t\ k.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i))))$
$\vee$
$((\Gamma \vdash (c!i)\ {-pese\rightarrow}\ (c!Suc\ i)) \wedge (\forall k.\ (\Gamma \vdash ((cs\ k)!i)\ {-ese\rightarrow}\ ((cs\ k)!\ Suc\ i))))$
by (*simp add*:*compat-tran-def*)
from $a1$ **have** $\neg(\Gamma \vdash (c!i)\ {-pese\rightarrow}\ (c!Suc\ i))$
using *pes-tran-not-etran1* **by** *blast*
with $a3$ **have** $\exists t\ k.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i)))$
by *simp*
then obtain t **and** k **where** $a4$: $(\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \wedge$
$(\forall k\ t.\ (\Gamma \vdash c!i\ {-pes-(t\sharp k)\rightarrow}\ c!Suc\ i) \longrightarrow (\Gamma \vdash cs\ k!i\ {-es-(t\sharp k)\rightarrow}\ cs\ k!\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (\Gamma \vdash cs\ k'!i\ {-ese\rightarrow}\ cs\ k'!\ Suc\ i)))$
by *auto*
from $p0\ p1\ p2\ p3\ p4\ p5\ p6\ a2$ **have**
$\forall k\ i.\ Suc\ i < length\ (cs\ k) \longrightarrow (\exists t.(\Gamma \vdash (cs\ k)!i{-es-t\rightarrow}(cs\ k)!Suc\ i))$
$\longrightarrow$ (*gets-es* $((cs\ k)!i)$,*gets-es* $((cs\ k)!Suc\ i)) \in$ *Guar k*
using *es-tran-sat-guar* [*of* Γ *pes Pre Rely Guar Post pre rely c s x cs*] **by** *simp*
then have $a5$: *Suc i* $<$ *length* $(cs\ k) \longrightarrow (\exists t.(\Gamma \vdash (cs\ k)!i{-es-t\rightarrow}(cs\ k)!Suc\ i))$
$\longrightarrow$ (*gets-es* $((cs\ k)!i)$,*gets-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-es* $((cs\ k)!i)$,*gets-es* $((cs\ k)!Suc\ i)) \in$ *Guar k* **by** *auto*
from $a0\ a2$ **have** $a8$: *gets-es* $((cs\ k)!i) =$ *gets* $(c!i)$ **by** (*simp add*:*conjoin-def same-state-def*)
from $a0\ a2$ **have** $a9$: *gets-es* $((cs\ k)!Suc\ i) =$ *gets* $(c!Suc\ i)$ **by** (*simp add*:*conjoin-def same-state-def*)
with $a7\ a8$ **have** (*gets* $(c!i)$,*gets* $(c!Suc\ i)) \in$ *Guar k* **by** *auto*
with $p4$ **have** (*gets* $(c!i)$,*gets* $(c!Suc\ i)) \in$ *guar* **by** *auto*

}
thus *?thesis* **by** *auto*
qed

lemma *parallel-sound*:
$[\![\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k];$
$\forall k.\ pre \subseteq Pre\ k;$
$\forall k.\ rely \subseteq Rely\ k;$
$\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k;$
$\forall k.\ Guar\ k \subseteq guar;$
$\forall k.\ Post\ k \subseteq post]\!]$
$\Longrightarrow \Gamma \models pes\ SAT\ [pre,\ rely,\ guar,\ post]$
proof –
assume *p0*: $\forall k.\ \Gamma \models (pes\ k)\ sat_s\ [Pre\ k,\ Rely\ k,\ Guar\ k,\ Post\ k]$
and *p1*: $\forall k.\ pre \subseteq Pre\ k$
and *p2*: $\forall k.\ rely \subseteq Rely\ k$
and *p3*: $\forall k\ j.\ j{\neq}k \longrightarrow\ Guar\ j \subseteq Rely\ k$
and *p4*: $\forall k.\ Guar\ k \subseteq guar$
and *p5*: $\forall k.\ Post\ k \subseteq post$
have $\forall s\ x.\ (cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely) \subseteq commit\text{-}pes\ \Gamma$
$(guar,\ post)$
proof –
{
fix *c s x*
assume *a0*: $c{\in}(cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \cap assume\text{-}pes\ \Gamma\ (pre,\ rely)$
then have *a1*: $c{\in}(cpts\text{-}of\text{-}pes\ \Gamma\ pes\ s\ x) \wedge c{\in}assume\text{-}pes\ \Gamma\ (pre,\ rely)$ **by**
simp
with *p0 p1 p2 p3 p4* **have** $\forall i.\ Suc\ i < length\ c \longrightarrow (\exists t.\ \Gamma \vdash c!i\ {-pes-t\rightarrow}$
$c!(Suc\ i))$
$\longrightarrow (gets\ (c!i), gets\ (c!Suc\ i)) \in guar$
using *pes-tran-sat-guar* [*of* Γ *pes Pre Rely Guar Post pre rely guar c s x*]
by *simp*
then have $c{\in}commit\text{-}pes\ \Gamma\ (guar,\ post)$
by (*simp add*: *commit-pes-def*)
}
then show *?thesis* **by** *auto*
qed

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

lemma *parallel-seq-sound*:
$[\![pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\Gamma \models pes\ SAT\ [pre',\ rely',\ guar',\ post']]\!]$
$\Longrightarrow \Gamma \models pes\ SAT\ [pre,\ rely,\ guar,\ post]$
proof –
assume *p0*: $pre \subseteq pre'$
and *p1*: $rely \subseteq rely'$
and *p2*: $guar' \subseteq guar$

and *p3*: *post'* ⊆ *post*
and *p4*: Γ ⊨ *pes SAT* [*pre'*, *rely'*, *guar'*, *post'*]
from *p4* **have** *p5*: ∀ *s x*. (*cpts-of-pes* Γ *pes s x*) ∩ *assume-pes* Γ (*pre'*, *rely'*) ⊆ *commit-pes* Γ (*guar'*, *post'*)
by (*simp add*: *pes-validity-def*)
have ∀ *s x*. (*cpts-of-pes* Γ *pes s x*) ∩ *assume-pes* Γ (*pre*, *rely*) ⊆ *commit-pes* Γ (*guar*, *post*)
proof −
{
fix *c s x*
assume *a0*: *c*∈(*cpts-of-pes* Γ *pes s x*) ∩ *assume-pes* Γ (*pre*, *rely*)
then have *c*∈(*cpts-of-pes* Γ *pes s x*) ∧ *c*∈*assume-pes* Γ (*pre*, *rely*) **by** *simp*
with *p0 p1* **have** *c*∈(*cpts-of-pes* Γ *pes s x*) ∧ *c*∈*assume-pes* Γ (*pre'*, *rely'*)
using *assume-pes-imp*[*of pre pre' rely rely' c*] **by** *simp*
with *p5* **have** *c*∈*commit-pes* Γ (*guar'*, *post'*) **by** *auto*
with *p2 p3* **have** *c*∈*commit-pes* Γ (*guar*, *post*)
using *commit-pes-imp*[*of guar' guar post' post c*] **by** *simp*
}
then show *?thesis* **by** *auto*
qed
then show *?thesis* **by** (*simp add*:*pes-validity-def*)
qed

lemma *parallel-sound'*:
assumes *p0*: ∀ *k*. Γ ⊢ *fst* ((*pes*::'*k* ⇒ ('*l*,'*k*,'*s*,'*prog*) *rgformula-es*) *k*) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
and *p1*: ∀ *k*. *pre* ⊆ Pre_{es} (*pes k*)
and *p2*: ∀ *k*. *rely* ⊆ $Rely_{es}$ (*pes k*)
and *p3*: ∀ *k j*. *j* ≠ *k* ⟶ $Guar_{es}$ (*pes j*) ⊆ $Rely_{es}$ (*pes k*)
and *p4*: ∀ *k*. $Guar_{es}$ (*pes k*) ⊆ *guar*
and *p5*: ∀ *k*. $Post_{es}$ (*pes k*) ⊆ *post*
shows Γ ⊨ *paresys-spec pes SAT* [*pre*, *rely*, *guar*, *post*]
proof −
from *p0* **have** ∀ *k*. Γ ⊨ *evtsys-spec* (*fst* (*pes k*)) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
proof −
{
fix *k*
from *p0* **have** Γ ⊢ *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 Γ ⊨ *evtsys-spec* (*fst* (*pes k*)) sat_s [Pre_{es} (*pes k*), $Rely_{es}$ (*pes k*), $Guar_{es}$ (*pes k*), $Post_{es}$ (*pes k*)]
using *rgsound-es* [*of* Γ *fst* (*pes k*) Pre_{es} (*pes k*) $Rely_{es}$ (*pes k*) $Guar_{es}$ (*pes k*) $Post_{es}$ (*pes k*)]
by *simp*
}
then show *?thesis* **by** *auto*
qed

with *p1 p2 p3 p4 p5* **show** $\Gamma \models$ *paresys-spec pes SAT* [*pre*, *rely*, *guar*, *post*]
using *parallel-sound* [*of* Γ *paresys-spec pes* $Pre_{es}\circ pes$ $Rely_{es}\circ pes$ $Guar_{es}\circ pes$ $Post_{es}\circ pes$
pre rely guar post] **by** (*simp add*:*paresys-spec-def*)
qed

theorem *rgsound-pes*: $\Gamma \vdash$ *rgf-par SAT* [*pre*, *rely*, *guar*, *post*] $\Longrightarrow \Gamma \models$ *paresys-spec rgf-par SAT* [*pre*, *rely*, *guar*, *post*]
apply(*erule rghoare-pes.induct*)

using *parallel-sound'* **apply** *blast*
using *parallel-seq-sound* **apply** *blast*
done

end

end

8 Rely-guarantee-based Safety Reasoning

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

type-synonym *'s invariant* = *'s* $\Rightarrow$ *bool*

context *event-hoare*
begin

definition *invariant-presv-pares*::*'Env* $\Rightarrow$ *'s invariant* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *paresys* $\Rightarrow$ *'s set* $\Rightarrow$ (*'s* $\times$ *'s*) *set* $\Rightarrow$ *bool*
where *invariant-presv-pares* Γ *invar pares init R* $\equiv$
$\forall$ *s0 x0 pesl*. *s0*$\in$*init* $\wedge$ *pesl* $\in$ (*cpts-of-pes* Γ *pares s0 x0* $\cap$ *assume-pes* Γ (*init*, *R*))
$\longrightarrow$ ($\forall$ *i*<*length pesl*. *invar* (*gets* (*pesl*!*i*)))

definition *invariant-presv-pares2*::*'Env* $\Rightarrow$ *'s invariant* $\Rightarrow$ (*'l*,*'k*,*'s*,*'prog*) *paresys* $\Rightarrow$ *'s set* $\Rightarrow$ (*'s* $\times$ *'s*) *set* $\Rightarrow$ *bool*
where *invariant-presv-pares2* Γ *invar pares init R* $\equiv$
$\forall$ *s0 x0 pesl*. *pesl* $\in$ (*cpts-of-pes* Γ *pares s0 x0* $\cap$ *assume-pes* Γ (*init*, *R*))
$\longrightarrow$ ($\forall$ *i*<*length pesl*. *invar* (*gets* (*pesl*!*i*)))

lemma *invariant-presv-pares* Γ *invar pares init R* = *invariant-presv-pares2* Γ *invar pares init R*
apply(*rule iffI*)
apply(*simp add*:*invariant-presv-pares-def invariant-presv-pares2-def cpts-of-pes-def assume-pes-def gets-def*)
apply *clarsimp*
apply(*simp add*:*invariant-presv-pares-def invariant-presv-pares2-def cpts-of-pes-def*

assume-pes-def gets-def)
done

theorem *invariant-theorem*:
assumes *parsys-sat-rg*: $\Gamma \vdash$ *pesf SAT* [*init*, *R*, *G*, *pst*]
and *stb-rely*: *stable-e* (*Collect invar*) *R*
and *stb-guar*: *stable-e* (*Collect invar*) *G*
and *init-in-invar*: *init* $\subseteq$ (*Collect invar*)
shows *invariant-presv-pares* Γ *invar* (*paresys-spec pesf*) *init R*
proof −
from *parsys-sat-rg* **have** $\Gamma \models$ *paresys-spec pesf SAT* [*init*, *R*, *G*, *pst*] **using** *rgsound-pes* **by** *fast*
hence *cpts-pes*: $\forall$ *s x*. (*cpts-of-pes* Γ (*paresys-spec pesf*) *s x*) $\cap$ *assume-pes* Γ (*init*, *R*) $\subseteq$ *commit-pes* Γ (*G*, *pst*)
by (*simp add*:*pes-validity-def*)
show *?thesis*
proof −
{
fix *s0 x0 pesl*
assume *a0*: *s0*$\in$*init*
and *a1*: *pesl*$\in$*cpts-of-pes* Γ (*paresys-spec pesf*) *s0 x0* $\cap$ *assume-pes* Γ (*init*, *R*)
from *a1* **have** *a3*: *pesl*!*0* = (*paresys-spec pesf*, *s0*, *x0*) $\wedge$ *pesl*$\in$*cpts-pes* Γ **by** (*simp add*:*cpts-of-pes-def*)
from *a1 cpts-pes* **have** *pesl-in-comm*: *pesl* $\in$ *commit-pes* Γ (*G*, *pst*) **by** *auto*
{
fix *i*
assume *b0*: *i*<*length pesl*
then have *gets* (*pesl*!*i*) $\in$ (*Collect invar*)
proof(*induct i*)
case *0*
with *a3* **have** *gets* (*pesl*!*0*) = *s0* **by** (*simp add*:*gets-def*)
with *a0 init-in-invar* **show** *?case* **by** *auto*
next
case (*Suc ni*)
assume *c0*: *ni* < *length pesl* $\Longrightarrow$ *gets* (*pesl* ! *ni*) $\in$ (*Collect invar*)
and *c1*: *Suc ni* < *length pesl*
then have *c2*: *gets* (*pesl* ! *ni*) $\in$ (*Collect invar*) **by** *auto*
from *c1* **have** *c3*: *ni* < *length pesl* **by** *simp*
with *c0* **have** *c4*: *gets* (*pesl* ! *ni*) $\in$ (*Collect invar*) **by** *simp*
from *a3 c1* **have** $\Gamma \vdash$ *pesl* ! *ni* −*pese*→ *pesl* ! *Suc ni* $\vee$ ($\exists$ *et*. $\Gamma \vdash$ *pesl* ! *ni* −*pes*−*et*→ *pesl* ! *Suc ni*)
using *incpts-pes-impl-evnorcomptran* **by** *blast*
then show *?case*
proof
assume *d0*: $\Gamma \vdash$ *pesl* ! *ni* −*pese*→ *pesl* ! *Suc ni*
then show *?thesis* **using** *c3 c4 a1 c1 stb-rely* **by**(*simp add*:*assume-pes-def stable-e-def*)
next

assume $\exists\, et.\ \Gamma \vdash pesl\ !\ ni\ -pes-et\rightarrow\ pesl\ !\ Suc\ ni$
then obtain *et* **where** *d0*: $\Gamma \vdash pesl\ !\ ni\ -pes-et\rightarrow\ pesl\ !\ Suc\ ni$ **by** *auto*
then show *?thesis* **using** *c3 c4 c1 pesl-in-comm stb-guar* **apply**(*simp add*:*commit-pes-def stable-e-def*)
by *blast*
qed
qed
}
}
then show *?thesis* **using** *invariant-presv-pares-def* **by** *blast*
qed
qed

end

end