PiCore: A Rely-guarantee Framework for Concurrent Reactive Systems

Yongwang Zhao
zhaoyongwang@gmail.com, zhaoyw@buaa.edu.cn

January 12, 2020

Contents

1 Abstract Syntax of PiCore Language

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

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

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

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

datatype $('l,'k,'s,'p)$ *esys* $=$
  EAnon $'p$
  | *EBasic* $('l,'s,'p)$ *event*
  | *EAtom* $('l,'s,'p)$ *event*
  | *ESeq* $('l,'k,'s,'p)$ *esys* $('l,'k,'s,'p)$ *esys* (- *NEXT* - [*81*,*81*] *80*)

  | *EChc* $('l,'k,'s,'p)$ *esys* $('l,'k,'s,'p)$ *esys* (- *OR* - [*81*,*81*] *80*)

  | *EJoin* $('l,'k,'s,'p)$ *esys* $('l,'k,'s,'p)$ *esys* (- $\bowtie$ - [*81*,*81*] *80*)

  | *EWhile* $'s\ set$ $('l,'k,'s,'p)$ *esys*

primrec *es-size* :: ‹$('l,'k,'s,'p)$ *esys* $\Rightarrow$ *nat*› **where**
‹*es-size* (*EAnon* -) = *1*› |
‹*es-size* (*EBasic* -) = *1*› |
‹*es-size* (*EAtom* -) = *1*› |

⟨*es-size* (*ESeq es1 es2*) = *Suc* (*es-size es1* + *es-size es2*)⟩ |
⟨*es-size* (*EChc es1 es2*) = *Suc* (*es-size es1* + *es-size es2*)⟩ |
⟨*es-size* (*EJoin es1 es2*) = *Suc* (*es-size es1* + *es-size es2*)⟩ |
⟨*es-size* (*EWhile* - *es*) = *Suc* (*es-size es*)⟩

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

end

2 Small-step Operational Semantics of PiCore Language

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

2.1 Datatypes for Semantics

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

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

abbreviation *mk-actk* :: (*'l,'s,'prog*) *act* ⇒ *'k* ⇒ (*'l,'k,'s,'prog*) *actk* (-♯- [*91*,*91*] *90*)
where *mk-actk a k* ≡ ⦇*Act*=*a*, *K*=*k*⦈

lemma *actk-destruct*:
⟨*a* = *Act a*♯*K a*⟩ **by** *simp*

type-synonym (*'l,'k,'s,'prog*) *ectx* = *'k* ⇀ (*'l,'s,'prog*) *event*

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

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

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

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

type-synonym ('l,'k,'s,'prog) *esconf* = ('l,'k,'s,'prog) *esys* × ('s × ('l,'k,'s,'prog) *ectx*)
type-synonym ('l,'k,'s,'prog) *pesconf* = (('l,'k,'s,'prog) *paresys*) × ('s × ('l,'k,'s,'prog) *ectx*)

locale *event* =
  fixes *ptran* :: 'Env ⇒ (('s,'prog) *pconf* × ('s,'prog) *pconf*) *set*
  fixes *fin-com* :: 'prog

  assumes *none-no-tran'*: ((*fin-com*, *s*),(*P*,*t*)) ∉ *ptran* Γ
  assumes *ptran-neq*: ((*P*, *s*),(*P*,*t*)) ∉ *ptran* Γ
begin

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

declare *ptran'-def*[*simp*]

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

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

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

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

2.2 Semantics of Event Systems

abbreviation ‹*fin* ≡ *EAnon fin-com*›

inductive *estran-p* :: 'Env ⇒ ('l,'k,'s,'prog) *esconf* ⇒ ('l,'k,'s,'prog) *actk* ⇒ ('l,'k,'s,'prog) *esconf* ⇒ *bool*
  (- ⊢ - −*es*[-]→ - [*81*,*81*] *80*)
  where
    EAnon: ⟦Γ ⊢ (*P*, *s*) −*c*→ (*Q*, *t*); *Q* ≠ *fin-com* ⟧ ⟹
        Γ ⊢ (*EAnon P*, *s*,*x*) −*es*[*Cmd*♯*k*]→ (*EAnon Q*, *t*,*x*)
  | *EAnon-fin*: ⟦ Γ ⊢ (*P*, *s*) −*c*→ (*Q*, *t*); *Q* = *fin-com*; *y* = *x*(*k* := *None*) ⟧ ⟹
        Γ ⊢ (*EAnon P*, *s*,*x*) −*es*[*Cmd*♯*k*]→ (*EAnon Q*, *t*, *y*)
  | *EBasic*: ⟦ *P* = *body e*; *s* ∈ *guard e*; *y* = *x*(*k*:=*Some e*) ⟧ ⟹
        Γ ⊢ (*EBasic e*, *s*,*x*) −*es*[(*EvtEnt e*)♯*k*]→ ((*EAnon P*), *s*,*y*)
  | *EAtom*: ⟦ *P* = *body e*; *s* ∈ *guard e*; Γ ⊢ (*P*,*s*)−*c*∗→(*fin-com*,*t*) ⟧ ⟹

$\Gamma \vdash (EAtom\ e,\ s,x)\ -es[(AtomEvt\ e)\sharp k]\rightarrow\ (fin,\ t,x)$
| *ESeq*: $[\![\Gamma \vdash (es1,\ s,x)\ -es[a]\rightarrow(es1',\ t,y);\ es1' \neq fin\ ]\!] \Longrightarrow$
$\Gamma \vdash (ESeq\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (ESeq\ es1'\ es2,\ t,y)$
| *ESeq-fin*: $[\![\Gamma \vdash (es1,\ s,x)\ -es[a]\rightarrow(fin,\ t,y)]\!] \Longrightarrow$
$\Gamma \vdash (ESeq\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (es2,\ t,y)$

| *EChc1*: $\Gamma \vdash (es1,s,x)\ -es[a]\rightarrow\ (es1',t,y) \Longrightarrow$
$\Gamma \vdash (EChc\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (es1',\ t,y)$
| *EChc2*: $\Gamma \vdash (es2,s,x)\ -es[a]\rightarrow\ (es2',t,y) \Longrightarrow$
$\Gamma \vdash (EChc\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (es2',\ t,y)$

| *EJoin1*: $\Gamma \vdash (es1,s,x)\ -es[a]\rightarrow\ (es1',t,y) \Longrightarrow$
$\Gamma \vdash (EJoin\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (EJoin\ es1'\ es2,\ t,y)$
| *EJoin2*: $\Gamma \vdash (es2,s,x)\ -es[a]\rightarrow\ (es2',t,y) \Longrightarrow$
$\Gamma \vdash (EJoin\ es1\ es2,\ s,x)\ -es[a]\rightarrow\ (EJoin\ es1\ es2',\ t,y)$
| *EJoin-fin*: $\langle\Gamma \vdash (EJoin\ fin\ fin,\ s,x)\ -es[Cmd\sharp k]\rightarrow\ (fin,s,x)\rangle$
| *EWhileT*: $s{\in}b \Longrightarrow P \neq fin \Longrightarrow \Gamma \vdash (EWhile\ b\ P,\ s,x)\ -es[Cmd\sharp k]\rightarrow\ (ESeq\ P\ (EWhile\ b\ P),\ s,x)$
| *EWhileF*: $s{\notin}b \Longrightarrow \Gamma \vdash (EWhile\ b\ P,\ s,x)\ -es[Cmd\sharp k]\rightarrow\ (fin,\ s,x)$

primrec *Choice-height* :: $('l,'k,'s,'p)$ *esys* $\Rightarrow$ *nat* **where**
Choice-height (*EAnon p*) = *0* |
Choice-height (*EBasic p*) = *0* |
Choice-height (*EAtom p*) = *0* |
Choice-height (*ESeq p q*) = *max* (*Choice-height p*) (*Choice-height q*) |
Choice-height (*EChc p q*) = *Suc* (*max* (*Choice-height p*) (*Choice-height q*)) |
Choice-height (*EJoin p q*) = *max* (*Choice-height p*) (*Choice-height q*) |
Choice-height (*EWhile - p*) = *Choice-height p*

primrec *Join-height* :: $('l,'k,'s,'p)$ *esys* $\Rightarrow$ *nat* **where**
Join-height (*EAnon p*) = *0* |
Join-height (*EBasic p*) = *0* |
Join-height (*EAtom p*) = *0* |
Join-height (*ESeq p q*) = *max* (*Join-height p*) (*Join-height q*) |
Join-height (*EChc p q*) = *max* (*Join-height p*) (*Join-height q*) |
Join-height (*EJoin p q*) = *Suc* (*max* (*Join-height p*) (*Join-height q*)) |
Join-height (*EWhile - p*) = *Join-height p*

lemma *chcneq-specneq*: *Choice-height es1* $\neq$ *Choice-height es2* $\Longrightarrow$ *es1* $\neq$ *es2*
by *auto*

lemma *allneq-specneq*: *All-height es1* $\neq$ *All-height es2* $\Longrightarrow$ *es1* $\neq$ *es2*
by *auto*

inductive-cases *estran-from-basic-cases*: $\langle\Gamma \vdash (EBasic\ e,\ s)\ -es[a]\rightarrow\ (es,\ t)\rangle$

lemma *chc-hei-convg*: $\Gamma \vdash (es1,s)\ -es[a]\rightarrow\ (es2,t) \Longrightarrow$ *Choice-height es1* $\geq$ *Choice-height es2*
apply(*induct es1 arbitrary*: *es2 a s t*; *rule estran-p.cases*, *auto*)

by *fastforce+*

lemma *join-hei-convg*: $\Gamma \vdash (es1,s) -es[a]\rightarrow (es2,t) \Longrightarrow$ *Join-height es1* $\geq$ *Join-height es2*
apply (*induct es1 arbitrary*: *es2 a s t*; *rule estran-p.cases*, *auto*)
by *fastforce+*

lemma $\neg(\exists$ *es2 t a*. $\Gamma \vdash (es1,s) -es[a]\rightarrow$ (*EChc es1 es2*,*t*)$)$
using *chc-hei-convg* **by** *fastforce*

lemma *seq-neq2*:
‹*P NEXT Q* $\neq$ *Q*›
proof
assume ‹*P NEXT Q* = *Q*›
then have ‹*es-size* (*P NEXT Q*) = *es-size Q*› **by** *simp*
then show *False* **by** *simp*
qed

lemma *join-neq1*: ‹*P* $\bowtie$ *Q* $\neq$ *P*› **by** (*induct P*) *auto*
lemma *join-neq2*: ‹*P* $\bowtie$ *Q* $\neq$ *Q*› **by** (*induct Q*) *auto*

lemma *spec-neq*: $\Gamma \vdash (es1,s,x)-es[a]\rightarrow(es2,t,y) \Longrightarrow es1 \neq es2$
proof(*induct es1 arbitrary*: *es2 s x t y a*)
case (*EAnon x*)
then show *?case* **apply**−
apply(*erule estran-p.cases*, *auto*) **using** *ptran-neq* **by** *simp+*
next
case (*EBasic x*)
then show *?case* **using** *estran-p.cases* **by** *fast*
next
case (*EAtom x*)
then show *?case* **using** *estran-p.cases* **by** *fast*
next
case (*ESeq es11 es12*)
then show *?case* **apply**−
apply(*erule estran-p.cases*, *auto*)
using *seq-neq2* **by** *blast+*
next
case (*EChc es11 es12*)
then show *?case* **apply**−
apply(*rule estran-p.cases*, *auto*)
proof−
assume ‹$\Gamma \vdash (es11,\ s,x) -es[a]\rightarrow$ (*es11 OR es12*, *t*,*y*)›
with *chc-hei-convg* **have** ‹*Choice-height* (*es11 OR es12*) $\leq$ *Choice-height es11*› **by** *blast*
then show *False* **by** *force*
next
assume ‹$\Gamma \vdash (es12,\ s,x) -es[a]\rightarrow$ (*es11 OR es12*, *t*,*y*)›
with *chc-hei-convg* **have** ‹*Choice-height* (*es11 OR es12*) $\leq$ *Choice-height es12*›

by *blast*
  then show *False* **by** *force*
 qed
next
 case (*EJoin es11 es12*)
 then show *?case* **apply**−
  apply(*rule estran-p.cases*, *auto*)
  using *join-neq2* **apply** *blast*
  apply *blast*.
next
 case *EWhile*
 then show *?case* **using** *estran-p.cases* **by** *fast*
qed

2.3 Semantics of Parallel Event Systems

inductive
 pestran-p :: $'Env \Rightarrow ('l,'k,'s,'prog)$ *pesconf* $\Rightarrow ('l,'k,'s,'prog)$ *actk*
  $\Rightarrow ('l,'k,'s,'prog)$ *pesconf* $\Rightarrow$ *bool* (- ⊢ - −*pes*[-]→ - [*70*,*70*] *60*)
 where
  ParES: $\Gamma \vdash (pes\ k,\ s,x)\ -es[a\sharp k]\rightarrow\ (es',\ t,y) \Longrightarrow \Gamma \vdash (pes,\ s,x)\ -pes[a\sharp k]\rightarrow$ $(pes(k{:=}es'),\ t,y)$

2.4 Lemmas

2.4.1 Programs

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* **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*
 done

2.4.2 Event systems

lemma *no-estran-to-self*: ‹$\neg\ \Gamma \vdash (es,\ s,x)\ -es[a]\rightarrow\ (es,\ t,y)$›
 using *spec-neq* **by** *blast*

lemma *no-estran-from-fin*:
 ‹$\neg\ \Gamma \vdash$ (*EAnon fin-com*, *s*) −*es*[*a*]→ *c*›
proof
 assume ‹$\Gamma \vdash$ (*EAnon fin-com*, *s*) −*es*[*a*]→ *c*›
 then show *False*
  apply(*rule estran-p.cases*, *auto*)
  using *none-no-tran* **by** *simp*+
qed

lemma *no-pestran-to-self*: ‹¬ Γ ⊢ (*Ps*, *S*) −*pes*[*a*]→ (*Ps*, *T*)›
proof(*rule ccontr*, *simp*)
  assume ‹Γ ⊢ (*Ps*, *S*) −*pes*[*a*]→ (*Ps*, *T*)›
  then show *False*
  proof(*cases*)
    case *ParES*
    then show *?thesis* **using** *no-estran-to-self*
      by (*metis fun-upd-same*)
  qed
qed

definition ‹*estran* Γ ≡ {(*c*,*c*′). ∃ *a*. *estran-p* Γ *c a c*′}›
definition ‹*pestran* Γ ≡ {(*c*,*c*′). ∃ *a k*. *pestran-p* Γ *c* (*a*♯*k*) *c*′}›

lemma *no-estran-to-self*′: ‹¬((*P*,*S*),(*P*,*T*))∈*estran* Γ›
  apply(*simp add*: *estran-def*)
  using *no-estran-to-self surjective-pairing*[*of S*] *surjective-pairing*[*of T*] **by** *metis*

lemma *no-estran-to-self*′′: ‹*fst c1* = *fst c2* ⟹ (*c1*,*c2*)∉*estran* Γ›
  apply(*subst surjective-pairing*[*of c1*])
  apply(*subst surjective-pairing*[*of c2*])
  using *no-estran-to-self*′ **by** *metis*

lemma *no-pestran-to-self*′: ‹¬((*P*,*s*),(*P*,*t*))∈*pestran* Γ›
  apply(*simp add*: *pestran-def*)
  using *no-pestran-to-self* **by** *blast*

end

end
theory *Computation* **imports** *Main* **begin**

definition *etran* :: ((′*p*×′*s*) × (′*p*×′*s*)) *set* **where**
  etran ≡ {(*c*,*c*′). *fst c* = *fst c*′}

declare *etran-def*[*simp*]

definition *etran-p* :: ‹(′*p*×′*s*) ⇒ (′*p*×′*s*) ⇒ *bool*› (- −*e*→ - [*81*,*81*] *80*)
  where ‹*etran-p c c*′ ≡ (*c*,*c*′) ∈ *etran*›

declare *etran-p-def*[*simp*]

inductive-set *cpts* :: ‹((′*p*×′*s*) × (′*p*×′*s*)) *set* ⇒ (′*p*×′*s*) *list set*›
  for *tran* :: ((′*p*×′*s*) × (′*p*×′*s*)) *set* **where**
    CptsOne[*intro*]: [(*P*,*s*)] ∈ *cpts tran* |
    CptsEnv[*intro*]: (*P*,*t*)#*cs* ∈ *cpts tran* ⟹ (*P*,*s*)#(*P*,*t*)#*cs* ∈ *cpts tran* |
    CptsComp: ⟦ ((*P*,*s*),(*Q*,*t*)) ∈ *tran*; (*Q*,*t*)#*cs* ∈ *cpts tran* ⟧ ⟹ (*P*,*s*)#(*Q*,*t*)#*cs*
∈ *cpts tran*

```
lemma cpts-snoc-env:
  assumes h: cpt ∈ cpts tran
  assumes tran: ‹last cpt −e→ c›
  shows ‹cpt@[c] ∈ cpts tran›
  using h tran
proof(induct)
  case (CptsOne P s)
  then have ‹fst c = P› by simp
  then show ?case
    apply(subst surjective-pairing[of c])
    apply(erule ssubst)
    apply simp
    apply(rule CptsEnv)
    apply(rule cpts.CptsOne)
    done
next
  case (CptsEnv P t cs s)
  then have ‹last ((P, t) # cs) −e→ c› by simp
  with CptsEnv(2) have ‹((P, t) # cs) @ [c] ∈ cpts tran› by blast
  then show ?case using cpts.CptsEnv by fastforce
next
  case (CptsComp P s Q t cs)
  then have ‹((Q, t) # cs) @ [c] ∈ cpts tran› by fastforce
  with CptsComp(1) show ?case using cpts.CptsComp by fastforce
qed

lemma cpts-snoc-comp:
  assumes h: cpt ∈ cpts tran
  assumes tran: ‹(last cpt, c) ∈ tran›
  shows ‹cpt@[c] ∈ cpts tran›
  using h tran
proof(induct)
  case (CptsOne P s)
  then show ?case apply simp
    apply(subst (asm) surjective-pairing[of c])
    apply(subst surjective-pairing[of c])
    apply(rule CptsComp)
     apply simp
    apply(rule cpts.CptsOne)
    done
next
  case (CptsEnv P t cs s)
  then have ‹((P, t) # cs) @ [c] ∈ cpts tran› by fastforce
  then show ?case using cpts.CptsEnv by fastforce
next
  case (CptsComp P s Q t cs)
  then have ‹((Q, t) # cs) @ [c] ∈ cpts tran› by fastforce
  with CptsComp(1) show ?case using cpts.CptsComp by fastforce
```

qed

lemma *cpts-nonnil*:
  assumes *h*: ‹*cpt* ∈ *cpts tran*›
  shows ‹*cpt* ≠ []›
  using *h* **by** (*induct*; *simp*)

lemma *cpts-def'*: ‹*cpt* ∈ *cpts tran* ⟷ *cpt* ≠ [] ∧ (∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*)›
proof
  assume *cpt*: ‹*cpt* ∈ *cpts tran*›
  show ‹*cpt* ≠ [] ∧ (∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*)›
  proof
    show ‹*cpt* ≠ []› **by** (*rule cpts-nonnil*[*OF cpt*])
  next
    show ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*›
    proof
      fix *i*
      show ‹*Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*›
      proof
        assume *i-lt*: ‹*Suc i* < *length cpt*›
        show ‹(*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*›
          using *cpt i-lt*
        proof(*induct arbitrary*:*i*)
          case (*CptsOne P s*)
          then show *?case* **by** *simp*
        next
          case (*CptsEnv P t cs s*)
          show *?case*
          proof(*cases i*)
            case *0*
            then show *?thesis* **apply**−
              apply(*rule disjI2*)
              apply(*erule ssubst*)
              apply *simp*
              done
          next
            case (*Suc i'*)
            then show *?thesis* **using** *CptsEnv*(*2*)[*of i'*] *CptsEnv*(*3*) **by** *force*
          qed
        next
          case (*CptsComp P s Q t cs*)
          show *?case*
          proof(*cases i*)
            case *0*
            then show *?thesis* **apply**−

apply(*rule disjI1*)
apply(*erule ssubst*)
apply *simp*
by (*rule CptsComp*(*1*))
next
case (*Suc i'*)
then show *?thesis* **using** *CptsComp*(*3*)[*of i'*] *CptsComp*(*4*) **by** *force*
qed
qed
qed
qed
qed
next
assume *h*: ‹*cpt* ≠ [] ∧ (∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*)›
from *h* **have** *cpt-nonnil*: ‹*cpt* ≠ []› **by** (*rule conjunct1*)
from *h* **have** *ct-et*: ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∈ *tran* ∨ *cpt*!*i* −*e*→ *cpt*!*Suc i*› **by** (*rule conjunct2*)
show ‹*cpt* ∈ *cpts tran*› **using** *cpt-nonnil ct-et*
proof(*induct cpt*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons c cs*)
have *IH*: ‹*cs* ≠ [] ⟹ ∀ *i*. *Suc i* < *length cs* ⟶ (*cs* ! *i*, *cs* ! *Suc i*) ∈ *tran* ∨ *cs* ! *i* −*e*→ *cs* ! *Suc i* ⟹ *cs* ∈ *cpts tran*›
by (*rule Cons*(*1*))
have *ct-et'*: ‹∀ *i*. *Suc i* < *length* (*c* # *cs*) ⟶ ((*c* # *cs*) ! *i*, (*c* # *cs*) ! *Suc i*) ∈ *tran* ∨ (*c* # *cs*) ! *i* −*e*→ (*c* # *cs*) ! *Suc i*›
by (*rule Cons*(*3*))
show *?case*
proof(*cases cs*)
case *Nil*
then show *?thesis* **apply**−
apply(*erule ssubst*)
apply(*subst surjective-pairing*[*of c*])
by (*rule CptsOne*)
next
case (*Cons c' cs'*)
then have ‹*cs* ≠ []› **by** *simp*
moreover have ‹∀ *i*. *Suc i* < *length cs* ⟶ (*cs* ! *i*, *cs* ! *Suc i*) ∈ *tran* ∨ *cs* ! *i* −*e*→ *cs* ! *Suc i*›
using *ct-et'* **by** *auto*
ultimately have *cs-cpts*: ‹*cs* ∈ *cpts tran*› **using** *IH* **by** *fast*
show *?thesis* **apply** (*rule ct-et'*[*THEN allE*, *of 0*])
apply(*simp add*: *Cons*)
proof−
assume ‹(*c*, *c'*) ∈ *tran* ∨ *fst c* = *fst c'*›
then show ‹*c* # *c'* # *cs'* ∈ *cpts tran*›

```
        proof
          assume h: ‹(c, c') ∈ tran›
          show ‹c # c' # cs' ∈ cpts tran›
            apply(subst surjective-pairing[of c])
            apply(subst surjective-pairing[of c'])
            apply(rule CptsComp)
             apply simp
             apply (rule h)
            using cs-cpts by (simp add: Cons)
        next
          assume h: ‹fst c = fst c'›
          show ‹c # c' # cs' ∈ cpts tran›
            apply(subst surjective-pairing[of c])
            apply(subst surjective-pairing[of c'])
            apply(subst h)
            apply(rule CptsEnv)
            apply simp
            using cs-cpts by (simp add: Cons)
        qed
      qed
    qed
  qed
qed

lemma cpts-tran:
  ‹cpt ∈ cpts tran ⟹
   ∀ i. Suc i < length cpt ⟶
   (cpt!i, cpt!Suc i) ∈ tran ∨ cpt!i −e→ cpt!Suc i›
  using cpts-def' by blast

definition cpts-from :: ‹(('p×'s) × ('p×'s)) set ⇒ ('p×'s) ⇒ ('p×'s) list set›
where
  cpts-from tran c0 ≡ {cpt. cpt ∈ cpts tran ∧ hd cpt = c0}

declare cpts-from-def[simp]

lemma cpts-from-def':
  cpt ∈ cpts-from tran c0 ⟷ cpt ∈ cpts tran ∧ hd cpt = c0 by simp

definition cpts-from-ctran-only :: ‹(('p×'s) × ('p×'s)) set ⇒ ('p×'s) ⇒ ('p×'s)
list set› where
  cpts-from-ctran-only tran c0 ≡ {cpt. cpt ∈ cpts-from tran c0 ∧ (∀ i. Suc i <
length cpt ⟶ (cpt!i, cpt!Suc i) ∈ tran)}

lemma cpts-tl':
  assumes h: ‹cpt ∈ cpts tran›
    and cpt: ‹cpt = c0#c1#cs›
  shows c1#cs ∈ cpts tran
  using h cpt apply− apply(erule cpts.cases, auto) done
```

lemma *cpts-tl*:
  ‹*cpt* ∈ *cpts tran* ⟹ *tl cpt* ≠ [] ⟹ *tl cpt* ∈ *cpts tran*›
  using *cpts-tl'* **by** (*metis cpts-nonnil list.exhaust-sel*)

lemma *cpts-from-tl*:
  assumes *h*: ‹*cpt* ∈ *cpts-from tran* (*P*,*s*)›
    and *cpt*: ‹*cpt* = (*P*,*s*)#(*P*,*t*)#*cs*›
  shows (*P*,*t*)#*cs* ∈ *cpts-from tran* (*P*,*t*)
proof−
  from *h* **have** *cpt* ∈ *cpts tran* **by** *simp*
  with *cpt* **show** *?thesis* **apply**− **apply**(*erule cpts.cases*, *auto*) **done**
qed

lemma *cpts-drop*:
  assumes *h*: *cpt* ∈ *cpts tran*
    and *i*: *i* < *length cpt*
  shows *drop i cpt* ∈ *cpts tran*
  using *i*
proof(*induct i*)
  case *0*
  then show *?case* **using** *h* **by** *simp*
next
  case (*Suc i'*)
  then show *?case*
  proof−
    assume *h1*: ‹*i'* < *length cpt* ⟹ *drop i' cpt* ∈ *cpts tran*›
    assume *h2*: ‹(*Suc i'*) < *length cpt*›
    with *h1* **have** ‹*drop i' cpt* ∈ *cpts tran*› **by** *fastforce*
    let *?cpt'* = ‹*drop i' cpt*›
    have ‹*drop* (*Suc i'*) *cpt* = *tl ?cpt'*›
      by (*simp add*: *drop-Suc drop-tl*)
    with *h2* **have** ‹*tl ?cpt'* ≠ []› **by** *auto*
    then show ‹*drop* (*Suc i'*) *cpt* ∈ *cpts tran*› **using** *cpts-tl*[*of ?cpt'*]
      by (*simp add*: ‹*drop* (*Suc i'*) *cpt* = *tl* (*drop i' cpt*)› ‹*drop i' cpt* ∈ *cpts tran*›
cpts-tl)
  qed
qed

lemma *cpts-take'*:
  assumes *h*: *cpt* ∈ *cpts tran*
  shows *take* (*Suc i*) *cpt* ∈ *cpts tran*
  using *h*
proof(*induct i*)
  case *0*
  have [(*fst* (*hd cpt*), *snd* (*hd cpt*))] ∈ *cpts tran* **using** *CptsOne* **by** *fast*
  then show *?case*
    using *0.prems cpts-def'* **by** *fastforce*
next

```
  case (Suc i)
  then have cpt': ‹take (Suc i) cpt ∈ cpts tran› by blast
  let ?cpt' = take (Suc i) cpt
  show ?case
  proof(cases ‹Suc i < length cpt›)
    case True
    with cpts-drop have drop-i: ‹drop i cpt ∈ cpts tran›
      using Suc-lessD h by blast
    have ‹?cpt' @ [cpt!Suc i] ∈ cpts tran› using drop-i
    proof(cases)
      case (CptsOne P s)
      then show ?thesis using h
      by (metis Cons-nth-drop-Suc Suc-lessD True append.right-neutral append-eq-append-conv
append-take-drop-id list.simps(3) nth-via-drop take-Suc-conv-app-nth)
    next
      case (CptsEnv P t cs s)
      then show ?thesis apply−
        apply(rule cpts-snoc-env)
        apply(rule cpt')
      proof−
        assume h1: ‹drop i cpt = (P, s) # (P, t) # cs›
        assume h2: ‹(P, t) # cs ∈ cpts tran›
        from h1 h2 have ‹last (take (Suc i) cpt) = (P, s)›
            by (metis Suc-lessD True hd-drop-conv-nth list.sel(1) snoc-eq-iff-butlast
take-Suc-conv-app-nth)
        moreover from h1 h2 have cpt!Suc i = (P,t)
          by (metis Cons-nth-drop-Suc Suc-lessD True list.sel(1) list.sel(3))
        ultimately show ‹last (take (Suc i) cpt) −e→ cpt ! Suc i› by force
      qed
    next
      case (CptsComp P s Q t cs)
      then show ?thesis apply−
        apply(rule cpts-snoc-comp)
         apply(rule cpt')
      proof−
        assume h1: ‹drop i cpt = (P, s) # (Q, t) # cs›
        assume h2: ‹(Q, t) # cs ∈ cpts tran›
        assume h3: ‹((P, s), (Q, t)) ∈ tran›
        from h1 h2 have ‹last (take (Suc i) cpt) = (P, s)›
            by (metis Suc-lessD True hd-drop-conv-nth list.sel(1) snoc-eq-iff-butlast
take-Suc-conv-app-nth)
        moreover from h1 h2 have cpt!Suc i = (Q,t)
          by (metis Cons-nth-drop-Suc Suc-lessD True list.sel(1) list.sel(3))
         ultimately show ‹(last (take (Suc i) cpt), cpt ! Suc i) ∈ tran› using h3
by simp
      qed
    qed
    with True show ?thesis
      by (simp add: take-Suc-conv-app-nth)
```

```
  next
    case False
    then show ?thesis using cpt' by simp
  qed
qed

lemma cpts-take:
  assumes h: cpt ∈ cpts tran
  assumes i: i ≠ 0
  shows take i cpt ∈ cpts tran
proof−
  from i obtain i' where ‹i = Suc i'› using not0-implies-Suc by blast
  with h cpts-take' show ?thesis by blast
qed

lemma cpts-from-take:
  assumes h: cpt ∈ cpts-from tran c
  assumes i: i ≠ 0
  shows take i cpt ∈ cpts-from tran c
  apply simp
proof
  from h have cpt ∈ cpts tran by simp
  with i cpts-take show ‹take i cpt ∈ cpts tran› by blast
next
  from h have hd cpt = c by simp
  with i show ‹hd (take i cpt) = c› by simp
qed

type-synonym 'a tran = ‹'a × 'a›

lemma cpts-prepend:
  ‹[c0,c1]∈cpts tran ⟹ c1#cs ∈ cpts tran ⟹ c0#c1#cs ∈ cpts tran›
  apply(erule cpts.cases, auto)
  apply(rule CptsComp, auto)
  done

lemma all-etran-same-prog:
  assumes all-etran: ‹∀ i. Suc i < length cpt ⟶ cpt!i −e→ cpt!Suc i›
    and fst-hd-cpt: ‹fst (hd cpt) = P›
    and ‹cpt≠[]›
  shows ‹∀ i<length cpt. fst (cpt!i) = P›
proof
  fix i
  show ‹i < length cpt ⟶ fst (cpt ! i) = P›
  proof(induct i)
    case 0
    then show ?case
      apply(rule impI)
      apply(subst hd-conv-nth[THEN sym])
```

apply(*rule* ‹*cpt*≠[]›)
apply(*rule fst-hd-cpt*)
done
next
case (*Suc i*)
have *1*: *Suc i* < *length cpt* ⟶ *cpt* ! *i* −*e*→ *cpt* ! *Suc i*
by (*rule all-etran*[*THEN spec*[**where** *x*=*i*]])
show *?case*
proof
assume *Suc-i-lt*: ‹*Suc i* < *length cpt*›
with *1* **have** ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i*› **by** *blast*
moreover from *Suc Suc-i-lt*[*THEN Suc-lessD*] **have** ‹*fst* (*cpt* ! *i*) = *P*› **by** *blast*
ultimately show ‹*fst* (*cpt* ! *Suc i*) = *P*› **by** *simp*
qed
qed
qed

lemma *cpts-append-comp*:
‹*cs1* ∈ *cpts tran* ⟹ *cs2* ∈ *cpts tran* ⟹ (*last cs1*, *hd cs2*) ∈ *tran* ⟹ *cs1*@*cs2* ∈ *cpts tran*›
proof−
assume *c1*: ‹*cs1*∈*cpts tran*›
assume *c2*: ‹*cs2*∈*cpts tran*›
assume *tran*: ‹(*last cs1*, *hd cs2*) ∈ *tran*›
show *?thesis* **using** *c1 tran*
proof(*induct*)
case (*CptsOne P s*)
then show *?case*
apply *simp*
apply(*cases cs2*)
using *cpts-nonnil c2* **apply** *fast*
apply *simp*
apply(*rename-tac c cs*)
apply(*subst surjective-pairing*[*of c*])
apply(*rule CptsComp*)
apply *simp*
using *c2* **by** *simp*
next
case (*CptsEnv P t cs s*)
then show *?case*
apply *simp*
apply(*rule cpts.CptsEnv*)
by *simp*
next
case (*CptsComp P s Q t cs*)
then show *?case*
apply *simp*
apply(*rule cpts.CptsComp*)

apply *blast*
by *blast*
qed
qed

lemma *cpts-append-env*:
assumes *c1*: ‹*cs1*∈*cpts tran*› **and** *c2*: ‹*cs2*∈*cpts tran*›
and *etran*: ‹*fst* (*last cs1*) = *fst* (*hd cs2*)›
shows ‹*cs1*@*cs2* ∈ *cpts tran*›
using *c1 etran*
proof(*induct*)
case (*CptsOne P s*)
then show *?case*
apply *simp*
apply(*subst hd-Cons-tl*[*OF cpts-nonnil*[*OF c2*], *symmetric*]) **back**
apply(*subst surjective-pairing*[*of* ‹*hd cs2*›]) **back**
apply(*rule CptsEnv*)
using *hd-Cons-tl*[*OF cpts-nonnil*[*OF c2*]] *c2* **by** *simp*
next
case (*CptsEnv P t cs s*)
then show *?case*
apply *simp*
apply(*rule cpts.CptsEnv*)
by *simp*
next
case (*CptsComp P s Q t cs*)
then show *?case*
apply *simp*
apply(*rule cpts.CptsComp*)
apply *blast*
by *blast*
qed

lemma *cpts-remove-last*:
assumes ‹*c*#*cs*@[*c'*] ∈ *cpts tran*›
shows ‹*c*#*cs* ∈ *cpts tran*›
proof−
from *assms cpts-def'* **have** *1*: ‹∀ *i*. *Suc i* < *length* (*c*#*cs*@[*c'*]) ⟶ ((*c*#*cs*@[*c'*]) ! *i*, (*c*#*cs*@[*c'*]) ! *Suc i*) ∈ *tran* ∨ (*c*#*cs*@[*c'*]) ! *i* −*e*→ (*c*#*cs*@[*c'*]) ! *Suc i*› **by** *blast*
have ‹∀ *i*. *Suc i* < *length* (*c*#*cs*) ⟶ ((*c*#*cs*) ! *i*, (*c*#*cs*) ! *Suc i*) ∈ *tran* ∨ (*c*#*cs*) ! *i* −*e*→ (*c*#*cs*) ! *Suc i*› (**is** ‹∀ *i*. *?P i*›)
proof
fix *i*
show ‹*?P i*›
proof
assume *Suc-i-lt*: ‹*Suc i* < *length* (*c* # *cs*)›
show ‹((*c* # *cs*) ! *i*, (*c* # *cs*) ! *Suc i*) ∈ *tran* ∨ (*c* # *cs*) ! *i* −*e*→ (*c* # *cs*) ! *Suc i*›

using *1*[*THEN spec*[**where** $x{=}i$]] *Suc-i-lt*
by (*metis* (*no-types*, *hide-lams*) *Suc-lessD Suc-less-eq Suc-mono append-Cons length-Cons length-append-singleton nth-Cons-Suc nth-butlast snoc-eq-iff-butlast*)
qed
qed
then show *?thesis* **using** *cpts-def'* **by** *blast*
qed

lemma *cpts-append*:
assumes *a1*: ‹$cs@[c] \in cpts\ tran$›
and *a2*: ‹$c\#cs' \in cpts\ tran$›
shows ‹$cs@c\#cs' \in cpts\ tran$›
proof−
from *a1 cpts-def'* **have** *a1'*: ‹$\forall i.\ Suc\ i < length\ (cs@[c]) \longrightarrow ((cs@[c])\ !\ i, (cs@[c])\ !\ Suc\ i) \in tran \lor (cs@[c])\ !\ i\ {-e}{\rightarrow}\ (cs@[c])\ !\ Suc\ i$› **by** *blast*
from *a2 cpts-def'* **have** *a2'*: ‹$\forall i.\ Suc\ i < length\ (c\#cs') \longrightarrow ((c\#cs')\ !\ i, (c\#cs')\ !\ Suc\ i) \in tran \lor (c\#cs')\ !\ i\ {-e}{\rightarrow}\ (c\#cs')\ !\ Suc\ i$› **by** *blast*
have ‹$\forall i.\ Suc\ i < length\ (cs@c\#cs') \longrightarrow ((cs@c\#cs')\ !\ i, (cs@c\#cs')\ !\ Suc\ i) \in tran \lor (cs@c\#cs')\ !\ i\ {-e}{\rightarrow}\ (cs@c\#cs')\ !\ Suc\ i$›
proof
fix *i*
show ‹$Suc\ i < length\ (cs@c\#cs') \longrightarrow ((cs@c\#cs')\ !\ i, (cs@c\#cs')\ !\ Suc\ i) \in tran \lor (cs@c\#cs')\ !\ i\ {-e}{\rightarrow}\ (cs@c\#cs')\ !\ Suc\ i$›
proof
assume *Suc-i-lt*: ‹$Suc\ i < length\ (cs@c\#cs')$›
show ‹$((cs@c\#cs')\ !\ i, (cs@c\#cs')\ !\ Suc\ i) \in tran \lor (cs@c\#cs')\ !\ i\ {-e}{\rightarrow}\ (cs@c\#cs')\ !\ Suc\ i$›
proof(*cases* ‹$Suc\ i < length\ (cs@[c])$›)
case *True*
with *a1'*[*THEN spec*[**where** $x{=}i$]] **show** *?thesis*
by (*metis Suc-less-eq length-append-singleton less-antisym nth-append nth-append-length*)
next
case *False*
with *a2'*[*THEN spec*[**where** $x{=}i - length\ cs$]] **show** *?thesis*

by (*smt Suc-diff-Suc Suc-i-lt Suc-lessD add-diff-cancel-left' diff-Suc-Suc diff-less-mono length-append length-append-singleton less-Suc-eq-le not-less-eq nth-append*)
qed
qed
qed
with *cpts-def'* **show** *?thesis* **by** *blast*
qed

end
theory *List-Lemmata* **imports** *Main* **begin**

lemma *last-take-Suc*:

$i < length\ l \Longrightarrow last\ (take\ (Suc\ i)\ l) = l!i$
by (*simp add*: *take-Suc-conv-app-nth*)

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

lemma *nth-tl*: $\llbracket\ ys!0{=}a;\ ys{\neq}[]\ \rrbracket \Longrightarrow ys{=}(a\#(tl\ ys))$
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 *drop-destruct*:
$\langle Suc\ n \le length\ xs \Longrightarrow drop\ n\ xs = hd\ (drop\ n\ xs)\ \#\ drop\ (Suc\ n)\ xs\rangle$
by (*metis drop-Suc drop-eq-Nil hd-Cons-tl not-less-eq-eq tl-drop*)

lemma *drop-last*:
$\langle xs \neq [] \Longrightarrow drop\ (length\ xs - 1)\ xs = [last\ xs]\rangle$
by (*metis append-butlast-last-id append-eq-conv-conj length-butlast*)

end

3 Computations of PiCore Language

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

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

locale *event-comp* = *event ptran fin-com*
for $ptran :: 'Env \Rightarrow (('s,'prog)\ pconf \times ('s,'prog)\ pconf)\ set$
and $fin\text{-}com :: 'prog$

begin

inductive-cases *estran-from-anon-cases*: $\langle\Gamma \vdash (EAnon\ p,\ S)\ {-es[a]\rightarrow}\ c\rangle$

lemma *cpts-from-anon*:
assumes *h*: $\langle cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (EAnon\ p0,\ s0,x0)\rangle$
shows $\langle\forall\, i.\ i < length\ cpt \longrightarrow (\exists\, p.\ fst(cpt!i) = EAnon\ p)\rangle$
proof

```
  from h have cpt-nonnil: cpt ≠ [] using cpts-nonnil by auto
  from h have h1: ‹cpt ∈ cpts (estran Γ)› by fastforce
  from h have h2: ‹hd cpt = (EAnon p0, s0,x0)› by auto
  fix i
  show ‹i < length cpt ⟶ (∃p. fst(cpt!i) = EAnon p)›
  proof
    assume i-lt: ‹i < length cpt›
    show ‹(∃p. fst(cpt!i) = EAnon p)›
      using i-lt
    proof(induct i)
      case 0
      from h have hd cpt = (EAnon p0, s0,x0) by simp
      then show ?case using hd-conv-nth cpt-nonnil by fastforce
    next
      case (Suc i')
      then obtain p where fst-cpt-i': fst(cpt!i') = (EAnon p) by fastforce
      have ‹(cpt!i', cpt!(Suc i')) ∈ estran Γ ∨ cpt!i' −e→ cpt!(Suc i')›
        using cpts-tran h1 Suc(2) by blast
      then show ?case
      proof
        assume ‹(cpt ! i', cpt ! Suc i') ∈ estran Γ›
        then show ?thesis
          apply(simp add: estran-def)
          apply(erule exE)
          apply(subst(asm) surjective-pairing[of ‹cpt!i'›])
          apply(subst(asm) fst-cpt-i')
          apply(erule estran-from-anon-cases)
          by simp+
      next
        assume ‹cpt ! i' −e→ cpt ! Suc i'›
        then show ?thesis
          apply simp
          using fst-cpt-i' by metis
      qed
    qed
  qed
qed

lemma cpts-from-anon':
  assumes h: ‹cpt ∈ cpts-from (estran Γ) (EAnon p0, s0)›
  shows ‹∀i. i < length cpt ⟶ (∃p s x. cpt!i = (EAnon p, s, x))›
  using cpts-from-anon by (metis h prod.collapse)

primrec (nonexhaustive) unlift-prog where
  ‹unlift-prog (EAnon p) = p›

definition ‹unlift-conf ≡ λ(p,s,-). (unlift-prog p, s)›
definition unlift-cpt :: ‹(('l, 'k, 's, 'prog) esconf) list ⇒ ('prog × 's) list› where
  ‹unlift-cpt ≡ map unlift-conf›
```

declare *unlift-conf-def* [*simp*] *unlift-cpt-def* [*simp*]

definition *lift-conf* :: ($'l$,$'k$,$'s$,$'prog$) *ectx* ⇒ ($'prog$×$'s$) ⇒ (($'l$,$'k$,$'s$,$'prog$) *esconf*)
where
‹*lift-conf x* ≡ λ(*p*,*s*). (*EAnon p*, *s*,*x*)›

declare *lift-conf-def* [*simp*]

lemma *lift-conf-def'*: ‹*lift-conf x* (*p*, *s*) = (*EAnon p*, *s*,*x*)› **by** *simp*

definition *lift-cpt* :: ($'l$,$'k$,$'s$,$'prog$) *ectx* ⇒ ($'prog$×$'s$) *list* ⇒ (($'l$,$'k$,$'s$,$'prog$) *esconf*) *list* **where**
‹*lift-cpt x* ≡ *map* (*lift-conf x*)›

declare *lift-cpt-def* [*simp*]

inductive-cases *estran-anon-to-anon-cases*: ‹Γ ⊢ (*EAnon p*, *s*,*x*) −*es*[*a*]→ (*EAnon q*, *t*,*y*)›

lemma *unlift-tran*: ‹((*EAnon p*, *s*,*x*), (*EAnon q*, *t*,*x*)) ∈ *estran* Γ ⟹ ((*p*,*s*),(*q*,*t*)) ∈ *ptran* Γ›
  apply(*simp add*: *case-prod-unfold estran-def*)
  apply(*erule exE*)
  apply(*erule estran-anon-to-anon-cases*)
  apply *simp+*
  done

lemma *unlift-tran'*: ‹(*lift-conf x c*, *lift-conf x c'*) ∈ *estran* Γ ⟹ (*c*, *c'*) ∈ *ptran* Γ›
  apply (*simp add*: *case-prod-unfold*)
  apply(*subst surjective-pairing*[*of c*])
  apply(*subst surjective-pairing*[*of c'*])
  using *unlift-tran* **by** *fastforce*

lemma *cpt-unlift-aux*:
‹((*EAnon p0*, *s0*,*x*), *Q*, *t*,*y*) ∈ *estran* Γ ⟹ ∃ *Q'*. *Q* = *EAnon Q'* ∧ ((*p0*,*s0*),(*Q'*,*t*)) ∈ *ptran* Γ›
  by (*simp add*: *estran-def*, *erule exE*, *erule estran-p.cases*, *auto*)

lemma *ctran-or-etran*:
‹*cpt* ∈ *cpts* (*estran* Γ) ⟹
  Suc i < *length cpt* ⟹
  (*cpt*!*i*, *cpt*!*Suc i*) ∈ *estran* Γ ∧ (¬ *cpt*!*i* −*e*→ *cpt*!*Suc i*) ∨
  (*cpt*!*i* −*e*→ *cpt*!*Suc i*) ∧ (*cpt*!*i*, *cpt*!*Suc i*) ∉ *estran* Γ›
proof−
  assume *cpt*: ‹*cpt* ∈ *cpts* (*estran* Γ)›
  assume *Suc-i-lt*: ‹*Suc i* < *length cpt*›
  from *cpts-drop*[*OF cpt Suc-i-lt*[*THEN Suc-lessD*]] **have**
    ‹*drop i cpt* ∈ *cpts* (*estran* Γ)› **by** *assumption*

```
  then show
    ‹(cpt!i, cpt!Suc i) ∈ estran Γ ∧ (¬ cpt!i −e→ cpt!Suc i) ∨
     (cpt!i −e→ cpt!Suc i) ∧ (cpt!i, cpt!Suc i) ∉ estran Γ›
  proof(cases)
    case (CptsOne P s)
    then have False
      by (metis (no-types, lifting) Cons-nth-drop-Suc Suc-i-lt Suc-lessD drop-eq-Nil
list.inject not-less)
    then show ?thesis by blast
  next
    case (CptsEnv P t cs s)
    from nth-via-drop[OF CptsEnv(1)] have ‹cpt!i = (P,s)› by assumption
    moreover from CptsEnv(1) have ‹cpt!Suc i = (P,t)›
      by (metis Suc-i-lt drop-Suc hd-drop-conv-nth list.sel(1) list.sel(3) tl-drop)
    ultimately show ?thesis
      by (simp add: no-estran-to-self')
  next
    case (CptsComp P s Q t cs)
    from nth-via-drop[OF CptsComp(1)] have ‹cpt!i = (P,s)› by assumption
    moreover from CptsComp(1) have ‹cpt!Suc i = (Q,t)›
      by (metis Suc-i-lt drop-Suc hd-drop-conv-nth list.sel(1) list.sel(3) tl-drop)
    ultimately show ?thesis
      apply simp
      apply(rule disjI1)
      apply(rule conjI)
       apply(rule CptsComp(2))
      using CptsComp(2) no-estran-to-self' by blast
  qed
qed

lemma ctran-or-etran-par:
  ‹cpt ∈ cpts (pestran Γ) ⟹
   Suc i < length cpt ⟹
   (cpt!i, cpt!Suc i) ∈ pestran Γ ∧ (¬ cpt!i −e→ cpt!Suc i) ∨
   (cpt!i −e→ cpt!Suc i) ∧ (cpt!i, cpt!Suc i) ∉ pestran Γ›
proof−
  assume cpt: ‹cpt ∈ cpts (pestran Γ)›
  assume Suc-i-lt: ‹Suc i < length cpt›
  from cpts-drop[OF cpt Suc-i-lt[THEN Suc-lessD]] have
    ‹drop i cpt ∈ cpts (pestran Γ)› by assumption
  then show
    ‹(cpt!i, cpt!Suc i) ∈ pestran Γ ∧ (¬ cpt!i −e→ cpt!Suc i) ∨
     (cpt!i −e→ cpt!Suc i) ∧ (cpt!i, cpt!Suc i) ∉ pestran Γ›
  proof(cases)
    case (CptsOne P s)
    then have False using Suc-i-lt
      by (metis Cons-nth-drop-Suc drop-Suc drop-tl list.sel(3) list.simps(3))
    then show ?thesis by blast
  next
```

```
    case (CptsEnv P t cs s)
    from nth-via-drop[OF CptsEnv(1)] have ‹cpt!i = (P,s)› by assumption
    moreover from CptsEnv(1) have ‹cpt!Suc i = (P,t)›
      by (metis Suc-i-lt drop-Suc hd-drop-conv-nth list.sel(1) list.sel(3) tl-drop)
    ultimately show ?thesis
      using no-pestran-to-self
      by (simp add: no-pestran-to-self')
  next
    case (CptsComp P s Q t cs)
    from nth-via-drop[OF CptsComp(1)] have ‹cpt!i = (P,s)› by assumption
    moreover from CptsComp(1) have ‹cpt!Suc i = (Q,t)›
      by (metis Suc-i-lt drop-Suc hd-drop-conv-nth list.sel(1) list.sel(3) tl-drop)
    ultimately show ?thesis
      apply simp
      apply(rule disjI1)
      apply(rule conjI)
       apply(rule CptsComp(2))
      using CptsComp(2) no-pestran-to-self' by blast
  qed
qed

abbreviation lift-seq Q P ≡ ESeq P Q
primrec lift-seq-esconf where lift-seq-esconf Q (P,s) = (lift-seq Q P, s)
abbreviation ‹lift-seq-cpt Q ≡ map (lift-seq-esconf Q)›
primrec lift-seq-esconf' where lift-seq-esconf' Q (P,s) = (if P = fin then (Q,s)
else (lift-seq Q P, s))
abbreviation ‹lift-seq-cpt' Q ≡ map (lift-seq-esconf' Q)›

lemma all-fin-after-fin:
  ‹(fin, s) # cs ∈ cpts (estran Γ) ⟹ ∀c∈set cs. fst c = fin›
proof−
  obtain cpt where cpt: cpt = (fin, s)#cs by simp
  assume ‹(fin, s) # cs ∈ cpts (estran Γ)›
  with cpt have ‹cpt ∈ cpts (estran Γ)› by simp
  then show ?thesis using cpt
    apply (induct arbitrary: s cs)
     apply simp
  proof−
    fix P s t sa
    fix cs csa :: ‹('a,'k,'s,'prog) escpt›
    assume h: ‹⋀s csa. (P, t) # cs = (fin, s) # csa ⟹ ∀c∈set csa. fst c = fin›
    assume eq: ‹(P, s) # (P, t) # cs = (fin, sa) # csa›
    then have P-fin: ‹P = fin› by simp
    with h have ‹∀c∈set cs. fst c = fin› by blast
    moreover from eq P-fin have csa = (fin, t)#cs by fast
    ultimately show ‹∀c∈set csa. fst c = fin› by simp
  next
    fix P Q :: ‹('a,'k,'s,'prog) esys›
    fix s t sa :: ‹'s ×('a,'k,'s,'prog) ectx›
```

fix *cs csa* :: ‹(′*a*,′*k*,′*s*,′*prog*) *escpt*›
assume *tran*: ‹((*P*, *s*), *Q*, *t*) ∈ *estran* Γ›
assume ‹(*P*, *s*) # (*Q*, *t*) # *cs* = (*fin*, *sa*) # *csa*›
then have *P-fin*: ‹*P* = *fin*› **by** *simp*
with *tran* **have** ‹((*fin*, *s*), (*Q*,*t*)) ∈ *estran* Γ› **by** *simp*
then have *False*
apply(*simp add*: *estran-def*)
using *no-estran-from-fin* **by** *fast*
then show ‹∀ *c*∈*set csa*. *fst c* = *fin*› **by** *blast*
qed
qed

lemma *lift-seq-cpt-partial*:
assumes ‹*cpt*∈*cpts* (*estran* Γ)›
and ‹*fst* (*last cpt*) ≠ *fin*›
shows ‹*lift-seq-cpt Q cpt* ∈ *cpts* (*estran* Γ)›
using *assms*
proof(*induct*)
case (*CptsOne P s*)
show *?case* **by** *auto*
next
case (*CptsEnv P t cs s*)
then show *?case* **by** *auto*
next
case (*CptsComp P S Q1 T cs*)
from *CptsComp*(*4*) **have** *1*: ‹*fst* (*last* ((*Q1*, *T*) # *cs*)) ≠ *fin*› **by** *simp*
from *CptsComp*(*3*)[*OF 1*] **have** *IH′*: ‹*map* (*lift-seq-esconf Q*) ((*Q1*, *T*) # *cs*) ∈ *cpts* (*estran* Γ)› **.**
have ‹*Q1*≠*fin*›
proof
assume ‹*Q1*=*fin*›
with *all-fin-after-fin CptsComp*(*2*) **have** ‹*fst* (*last* ((*Q1*, *T*) # *cs*)) = *fin*› **by** *fastforce*
with *1* **show** *False* **by** *blast*
qed
obtain *s x* **where** *S*: ‹*S*=(*s*,*x*)› **by** *fastforce*
obtain *t y* **where** *T*: ‹*T*=(*t*,*y*)› **by** *fastforce*
show *?case*
apply *simp*
apply(*rule cpts.CptsComp*)
apply(*insert CptsComp*(*1*))
apply(*simp add*: *estran-def*) **apply**(*erule exE*) **apply**(*rule exI*)
apply(*simp add*: *S T*)
apply(*erule ESeq*)
apply(*rule* ‹*Q1*≠*fin*›)
using *IH′*[*simplified*] **.**
qed

lemma *lift-seq-cpt*:

assumes ‹*cpt*∈*cpts* (*estran* Γ)›
and ‹Γ ⊢ *last cpt* −*es*[*a*]→ (*fin*,*t*,*y*)›
shows ‹*lift-seq-cpt Q cpt* @ [(*Q*,*t*,*y*)] ∈ *cpts* (*estran* Γ)›
using *assms*
proof(*induct*)
case (*CptsOne P S*)
obtain *s x* **where** *S*: ‹*S*=(*s*,*x*)› **by** *fastforce*
show *?case* **apply** *simp*
apply(*rule CptsComp*)
apply (*simp add*: *estran-def*)
apply(*rule exI*)
apply(*subst S*)
apply(*rule ESeq-fin*)
using *CptsOne S* **apply** *simp*
by (*rule cpts.CptsOne*)
next
case (*CptsEnv P T1 cs S*)
have ‹*map* (*lift-seq-esconf Q*) ((*P*, *T1*) # *cs*) @ [(*Q*, *t*,*y*)] ∈ *cpts* (*estran* Γ)›
apply(*rule CptsEnv*(*2*))
using *CptsEnv*(*3*) **by** *fastforce*
then show *?case* **apply** *simp* **by** (*erule cpts.CptsEnv*)
next
case (*CptsComp P S Q1 T1 cs*)
from *CptsComp*(*1*) **have** *ctran*: ‹∃ *a*. Γ ⊢ (*P*,*S*)−*es*[*a*]→(*Q1*,*T1*)›
by (*simp add*: *estran-def*)
have ‹*Q1*≠*fin*›
proof
assume ‹*Q1*=*fin*›
with *all-fin-after-fin CptsComp*(*2*) **have** ‹∀ *c*∈*set cs*. *fst c* = *fin*› **by** *fastforce*
with ‹*Q1*=*fin*› **have** ‹*fst* (*last* ((*P*, *S*) # (*Q1*, *T1*) # *cs*)) = *fin*› **by** *simp*
with *CptsComp*(*4*) **have** ‹Γ ⊢ (*fin*, *snd* (*last* ((*P*, *S*) # (*Q1*, *T1*) # *cs*)))
−*es*[*a*]→ (*fin*, *t*,*y*)› **using** *surjective-pairing* **by** *metis*
with *no-estran-from-fin* **show** *False* **by** *blast*
qed
obtain *s x* **where** *S*:‹*S*=(*s*,*x*)› **by** *fastforce*
obtain *t1 y1* **where** *T1*:‹*T1*=(*t1*,*y1*)› **by** *fastforce*
have ‹*map* (*lift-seq-esconf Q*) ((*Q1*, *T1*) # *cs*) @ [(*Q*, *t*,*y*)] ∈ *cpts* (*estran* Γ)›
using *CptsComp*(*3*,*4*) **by** *fastforce*
then show *?case* **apply** *simp* **apply**(*rule cpts.CptsComp*)
apply(*simp add*: *estran-def*) **apply**(*insert ctran*) **apply**(*erule exE*) **apply**(*rule exI*)
apply(*simp add*: *S T1*)
apply(*erule ESeq*)
apply(*rule* ‹*Q1*≠*fin*›)
by *assumption*
qed

lemma *all-etran-from-fin*:
assumes *cpt*: *cpt* ∈ *cpts* (*estran* Γ)

and *cpt-eq*: *cpt* = (*fin*, *t*) # *cs*
shows ‹∀ *i*. *Suc i* < *length cpt* ⟶ *cpt*!*i* −*e*→ *cpt*!*Suc i*›
using *cpt cpt-eq*
proof(*induct arbitrary*:*t cs*)
case (*CptsOne P s*)
then show *?case* **by** *simp*
next
case (*CptsEnv P t1 cs1 s*)
then have *et*: ‹∀ *i*. *Suc i* < *length* ((*P*, *t1*) # *cs1*) ⟶ ((*P*, *t1*) # *cs1*) ! *i* −*e*→ ((*P*, *t1*) # *cs1*) ! *Suc i*› **by** *fast*
show *?case*
proof
fix *i*
show ‹*Suc i* < *length* ((*P*, *s*) # (*P*, *t1*) # *cs1*) ⟶ ((*P*, *s*) # (*P*, *t1*) # *cs1*) ! *i* −*e*→ ((*P*, *s*) # (*P*, *t1*) # *cs1*) ! *Suc i*›
proof(*cases i*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc i′*)
then show *?thesis* **using** *et* **by** *auto*
qed
qed
next
case (*CptsComp P s Q t1 cs1*)
then have ‹((*EAnon fin-com*, *t*), *Q*, *t1*) ∈ *estran* Γ› **by** *fast*
then obtain *a* **where**
‹Γ ⊢ (*EAnon fin-com*, *t*) −*es*[*a*]→ (*Q*, *t1*)› **using** *estran-def* **by** *blast*
then have *False* **using** *no-estran-from-fin* **by** *blast*
then show *?case* **by** *blast*
qed

lemma *no-ctran-from-fin*:
assumes *cpt*: *cpt* ∈ *cpts* (*estran* Γ)
and *cpt-eq*: *cpt* = (*fin*, *t*) # *cs*
shows ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!*Suc i*) ∉ *estran* Γ›
proof
fix *i*
have *1*: ‹∀ *i*. *Suc i* < *length cpt* ⟶ *cpt*!*i* −*e*→ *cpt*!*Suc i*› **by** (*rule all-etran-from-fin*[*OF cpt cpt-eq*])
show ‹*Suc i* < *length cpt* ⟶ (*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ›
proof
assume ‹*Suc i* < *length cpt*›
with *1* **have** ‹*cpt*!*i* −*e*→ *cpt*!*Suc i*› **by** *blast*
then show ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ›
apply *simp*
using *no-estran-to-self″* **by** *blast*
qed
qed

inductive-set *cpts-es-mod* **for** Γ **where**
CptsModOne[intro]: [(P,s,x)] ∈ cpts-es-mod Γ |
CptsModEnv[intro]: (P,t,y)#cs ∈ cpts-es-mod Γ ⟹ (P,s,x)#(P,t,y)#cs ∈ cpts-es-mod Γ |
CptsModAnon: ⟦ Γ ⊢ (P, s) −c→ (Q, t); Q ≠ fin-com; (EAnon Q, t,x)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EAnon P, s,x)#(EAnon Q, t,x)#cs ∈ cpts-es-mod Γ |
CptsModAnon-fin: ⟦ Γ ⊢ (P, s) −c→ (Q, t); Q = fin-com; y = x(k:=None); (EAnon Q, t,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EAnon P, s,x)#(EAnon Q, t,y)#cs ∈ cpts-es-mod Γ |
CptsModBasic: ‹⟦ P = body e; s ∈ guard e; y=x(k:=Some e); (EAnon P, s,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EBasic e, s,x)#(EAnon P, s,y)#cs ∈ cpts-es-mod Γ› |
CptsModAtom: ‹⟦ P = body e; s ∈ guard e; Γ ⊢ (P,s)−c∗→(fin-com,t); (EAnon fin-com, t,x)#cs ∈ cpts-es-mod Γ ⟧
⟹ (EAtom e, s,x)#(EAnon fin-com, t,x)#cs ∈ cpts-es-mod Γ› |
CptsModSeq: ‹Γ ⊢ (P,s,x)−es[a]→(Q,t,y) ⟹ Q≠fin ⟹ (ESeq Q R, t,y)#cs ∈ cpts-es-mod Γ ⟹ (ESeq P R, s,x)#(ESeq Q R, t,y)#cs ∈ cpts-es-mod Γ› |
CptsModSeq-fin: ‹Γ ⊢ (P,s,x)−es[a]→(fin,t,y) ⟹ (Q,t,y)#cs ∈ cpts-es-mod Γ ⟹ (P NEXT Q, s,x)#(Q,t,y)#cs ∈ cpts-es-mod Γ› |
CptsModChc1: ‹⟦ Γ ⊢ (P,s,x) −es[a]→ (Q,t,y); (Q,t,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EChc P R, s,x)#(Q,t,y)#cs ∈ cpts-es-mod Γ› |
CptsModChc2: ‹⟦ Γ ⊢ (P,s,x) −es[a]→ (Q,t,y); (Q,t,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EChc R P, s,x)#(Q,t,y)#cs ∈ cpts-es-mod Γ› |

CptsModJoin1: ‹⟦ Γ ⊢ (P,s,x) −es[a]→ (Q,t,y); (EJoin Q R, t,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EJoin P R, s,x)#(EJoin Q R, t,y)#cs ∈ cpts-es-mod Γ› |
CptsModJoin2: ‹⟦ Γ ⊢ (P,s,x) −es[a]→ (Q,t,y); (EJoin R Q, t,y)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EJoin R P, s,x)#(EJoin R Q, t,y)#cs ∈ cpts-es-mod Γ› |
CptsModJoin-fin: ‹(fin,t,y)#cs ∈ cpts-es-mod Γ ⟹ (fin⋈fin,t,y)#(fin,t,y)#cs ∈ cpts-es-mod Γ› |
CptsModWhileTMore: ‹⟦ s∈b; (P,s,x)#cs ∈ cpts (estran Γ); Γ ⊢ (last ((P,s,x)#cs)) −es[a]→ (fin,t,y); (EWhile b P, t,y)#cs′ ∈ cpts-es-mod Γ ⟧
⟹ (EWhile b P, s,x) # lift-seq-cpt (EWhile b P) ((P,s,x)#cs) @ (EWhile b P, t,y) # cs′ ∈ cpts-es-mod Γ› |
CptsModWhileTOnePartial: ‹⟦ s∈b; (P,s,x)#cs ∈ cpts (estran Γ); fst (last ((P,s,x)#cs)) ≠ fin ⟧ ⟹ (EWhile b P, s,x) # lift-seq-cpt (EWhile b P) ((P,s,x)#cs) ∈ cpts-es-mod Γ› |
CptsModWhileTOneFull: ‹⟦ s∈b; (P,s,x)#cs ∈ cpts (estran Γ); Γ ⊢ (last ((P,s,x)#cs))−es[a]→(fin,t,y); (fin,t,y)#cs′ ∈ cpts-es-mod Γ ⟧ ⟹
(EWhile b P, s,x) # lift-seq-cpt (EWhile b P) ((P,s,x)#cs) @ map (λ(-,s,x). (EWhile b P, s,x)) ((fin,t,y)#cs′) ∈ cpts-es-mod Γ› |
CptsModWhileF: ‹⟦ s∉b; (fin, s,x)#cs ∈ cpts-es-mod Γ ⟧ ⟹ (EWhile b P, s,x)#(fin, s,x)#cs ∈ cpts-es-mod Γ›

definition ‹all-seq Q cs ≡ ∀ c∈set cs. ∃ P. fst c = P NEXT Q›

lemma *equiv-aux1*:
‹cs ∈ cpts (estran Γ) ⟹
hd cs = (P NEXT Q, s) ⟹

```
  P ≠ fin ⟹
  all-seq Q cs ⟹
  ∃cs0. cs = lift-seq-cpt Q ((P, s) # cs0) ∧ (P,s)#cs0 ∈ cpts (estran Γ) ∧ fst
(last ((P,s)#cs0)) ≠ fin›
proof−
 assume cpt: ‹cs ∈ cpts (estran Γ)›
 assume cs: ‹hd cs = (P NEXT Q, s)›
 assume ‹P≠fin›
 assume all-seq: ‹all-seq Q cs›
 show ?thesis
   using cpt cs ‹P≠fin› all-seq
 proof(induct arbitrary: P s)
   case (CptsOne P1 s1)
   then show ?case apply−
     apply(rule exI[where x=‹[]›])
     apply simp
     by (rule cpts.CptsOne)
 next
   case (CptsEnv P1 t cs s1)
   from CptsEnv(3) have 1: ‹hd ((P1, t) # cs) = (P NEXT Q, t)› by simp
   from ‹all-seq Q ((P1, s1) # (P1, t) # cs)› have 2: ‹all-seq Q ((P1, t) # cs)›
by (simp add: all-seq-def)
   from CptsEnv(3) have ‹s1=s› by simp
   from CptsEnv(2)[OF 1 CptsEnv(4) 2] obtain cs0 where
     ‹(P1, t) # cs = map (lift-seq-esconf Q) ((P, t) # cs0) ∧ (P, t) # cs0 ∈ cpts
(estran Γ) ∧ fst (last ((P, t) # cs0)) ≠ fin› by meson
   then show ?case apply− apply(rule exI[where x=‹(P,t)#cs0›])
     apply (simp add: ‹s1=s›)
     apply(rule cpts.CptsEnv)
     by blast
 next
   case (CptsComp P1 s1 Q1 t cs)
   from CptsComp(6) obtain P′ where Q1: ‹Q1 = P′ NEXT Q› by (auto simp
add: all-seq-def)
   then have 1: ‹hd ((Q1, t) # cs) = (P′ NEXT Q, t)› by simp
   from CptsComp(4) have P1: ‹P1=P NEXT Q› and ‹s1=s› by simp+
   from CptsComp(1) P1 Q1 have ‹P′≠fin›
     apply (simp add: estran-def)
     apply(erule exE)
     apply(erule estran-p.cases, auto)[]
     using Q1 seq-neq2 by blast
   from CptsComp(1) P1 Q1 have tran: ‹((P, s), P′, t) ∈ estran Γ›
     apply(simp add: estran-def) apply(erule exE) apply(erule estran-p.cases,
auto)[]
     apply(rule exI) apply (simp add: ‹s1=s›)
     using seq-neq2 by blast
  from CptsComp(6) have 2: ‹all-seq Q ((Q1, t) # cs)› by (simp add: all-seq-def)
   from CptsComp(3)[OF 1 ‹P′≠fin› 2] obtain cs0 where
     ‹(Q1, t) # cs = map (lift-seq-esconf Q) ((P′, t) # cs0) ∧ (P′, t) # cs0 ∈
```

cpts (*estran* Γ) ∧ *fst* (*last* ((*P'*, *t*) # *cs0*)) ≠ *fin*› **by** *meson*
then show *?case* **apply**− **apply**(*rule exI*[**where** *x*=‹(*P'*,*t*)#*cs0*›])
apply(*rule conjI*)
apply (*simp add*: ‹*s1*=*s*› *P1*)
apply(*rule conjI*)
apply(*rule cpts.CptsComp*)
apply(*rule tran*)
apply *blast*
by *simp*
qed
qed

lemma *split-seq-mod*:
assumes *cpt*: ‹*cpt* ∈ *cpts-es-mod* Γ›
and *hd-cpt*: ‹*hd cpt* = (*es1 NEXT es2*, *S0*)›
and *not-all-seq*: ‹¬ *all-seq es2 cpt*›
shows
∃ *i S'*. *cpt*!*i* = (*es2*, *S'*) ∧
i ≠ *0* ∧
i < *length cpt* ∧
(∃ *cpt'*. *take i cpt* = *lift-seq-cpt es2* ((*es1*,*S0*)#*cpt'*) ∧ ((*es1*,*S0*)#*cpt'*)∈*cpts* (*estran* Γ) ∧ (*last* ((*es1*,*S0*)#*cpt'*), (*fin*, *S'*))∈*estran* Γ) ∧
all-seq es2 (*take i cpt*) ∧
drop i cpt ∈ *cpts-es-mod* Γ
using *cpt hd-cpt not-all-seq*
proof(*induct arbitrary*: *es1 S0*)
case (*CptsModOne P S*)
then show *?case* **by** (*simp add*: *all-seq-def*)
next
case (*CptsModEnv P t y cs s x*)

from *CptsModEnv*(*3*) **have** *P-dest*: ‹*P* = *es1 NEXT es2*› **by** *simp*
from *P-dest* **have** *1*: ‹(*hd* ((*P*, *t*, *y*) # *cs*)) = (*es1 NEXT es2*, *t*, *y*)› **by** *simp*
from *CptsModEnv*(*4*) **have** *2*: ‹¬ *all-seq es2* ((*P*, *t*, *y*) # *cs*)› **by** (*simp add*: *all-seq-def*)
from *CptsModEnv*(*2*)[*OF 1 2*] **obtain** *i S'* **where**
‹((*P*, *t*, *y*) # *cs*) ! *i* = (*es2*, *S'*) ∧
i ≠ *0* ∧
i < *length* ((*P*, *t*, *y*) # *cs*) ∧
(∃ *cpt'*. *take i* ((*P*, *t*, *y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*es1*, *t*, *y*) # *cpt'*) ∧ (*es1*, *t*, *y*) # *cpt'* ∈ *cpts* (*estran* Γ) ∧ (*last* ((*es1*, *t*, *y*) # *cpt'*), *fin*, *S'*) ∈ *estran* Γ) ∧
all-seq es2 (*take i* ((*P*, *t*, *y*) # *cs*)) ∧ *drop i* ((*P*, *t*, *y*) # *cs*) ∈ *cpts-es-mod* Γ›
by *meson*
then have
p1: ‹((*P*, *t*, *y*) # *cs*) ! *i* = (*es2*, *S'*)› **and**
p2: ‹*i*≠*0*› **and**
p3: ‹*i* < *length* ((*P*, *t*, *y*) # *cs*)› **and**
p4: ‹∃ *cpt'*. *take i* ((*P*, *t*, *y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*es1*, *t*, *y*) #

cpt') ∧ ((*es1*, *t*, *y*) # *cpt'*) ∈ *cpts* (*estran* Γ) ∧ (*last* ((*es1*, *t*, *y*) # *cpt'*), *fin*, *S'*) ∈ *estran* Γ› **and**
p5: ‹*all-seq es2* (*take i* ((*P*, *t*, *y*) # *cs*))› **and**
p6: ‹*drop i* ((*P*, *t*, *y*) # *cs*) ∈ *cpts-es-mod* Γ› **by** *argo+*
from *p4* **obtain** *cpt'* **where**
p4-1: ‹*take i* ((*P*, *t*, *y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*es1*, *t*, *y*) # *cpt'*)› **and**
p4-2: ‹((*es1*, *t*, *y*) # *cpt'*) ∈ *cpts* (*estran* Γ)› **and**
p4-3: ‹(*last* ((*es1*, *t*, *y*) # *cpt'*), *fin*, *S'*) ∈ *estran* Γ› **by** *meson*
show *?case*
apply(*rule exI*[**where** *x*=*Suc i*])
apply(*rule exI*[**where** *x*=*S'*])
apply(*rule conjI*)
using *p1* **apply** *simp*
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **using** *p3* **apply** *simp*
apply(*rule conjI*)
apply(*rule exI*[**where** *x*=‹(*es1*,*t*,*y*)#*cpt'*›])
apply(*rule conjI*)
using *p4-1 P-dest* **apply** *simp*
using *CptsModEnv*(*3*) **apply** *simp*
apply(*rule conjI*)
apply(*rule CptsEnv*)
using *p4-2* **apply** *fastforce*
using *p4-3* **apply** *fastforce*
using *p5 P-dest* **apply**(*simp add*: *all-seq-def*)
using *p6* **apply** *simp*.
next
case (*CptsModAnon*)
then show *?case* **by** *simp*
next
case (*CptsModAnon-fin*)
then show *?case* **by** *simp*
next
case (*CptsModBasic*)
then show *?case* **by** *simp*
next
case (*CptsModAtom*)
then show *?case* **by** *simp*
next
case (*CptsModSeq P s x a Q t y R cs*)
from *CptsModSeq*(*5*) **have** ‹*R*=*es2*› **by** *simp*
then have *1*: ‹(*hd* ((*Q NEXT R*, *t*,*y*) # *cs*)) = (*Q NEXT es2*, *t*,*y*)› **by** *simp*
from *CptsModSeq*(*6*) ‹*R*=*es2*› **have** *2*: ‹¬ *all-seq es2* ((*Q NEXT R*, *t*,*y*) # *cs*)› **by** (*simp add*: *all-seq-def*)
from *CptsModSeq*(*4*)[*OF 1 2*] **obtain** *i S'* **where**
‹((*Q NEXT R*, *t*, *y*) # *cs*) ! *i* = (*es2*, *S'*) ∧
i ≠ *0* ∧
i < *length* ((*Q NEXT R*, *t*, *y*) # *cs*) ∧

($\exists$ *cpt'*. *take i* ((*Q NEXT R*, *t*, *y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*Q*, *t*, *y*) # *cpt'*) $\wedge$ (*Q*, *t*, *y*) # *cpt'* $\in$ *cpts* (*estran* Γ) $\wedge$ (*last* ((*Q*, *t*, *y*) # *cpt'*), *fin*, *S'*) $\in$ *estran* Γ) $\wedge$
all-seq es2 (*take i* ((*Q NEXT R*, *t*, *y*) # *cs*)) $\wedge$ *drop i* ((*Q NEXT R*, *t*, *y*) # *cs*) $\in$ *cpts-es-mod* Γ› **by** *meson*
then have
p1: ‹((*Q NEXT R*, *t*,*y*) # *cs*) ! *i* = (*es2*, *S'*)› **and**
p2: ‹*i*$\neq$*0*› **and**
p3: ‹*i* < *length* ((*Q NEXT R*, *t*,*y*) # *cs*)› **and**
p4: ‹$\exists$ *cpt'*. *take i* ((*Q NEXT R*, *t*,*y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*Q*, *t*,*y*) # *cpt'*) $\wedge$ ((*Q*, *t*,*y*) # *cpt'*) $\in$ *cpts* (*estran* Γ) $\wedge$ (*last* ((*Q*, *t*,*y*) # *cpt'*), *fin*, *S'*) $\in$ *estran* Γ› **and**
p5: ‹*all-seq es2* (*take i* ((*Q NEXT R*, *t*,*y*) # *cs*))› **and**
p6: ‹*drop i* ((*Q NEXT R*, *t*,*y*) # *cs*) $\in$ *cpts-es-mod* Γ› **by** *argo+*
from *p4* **obtain** *cpt'* **where**
p4-1: ‹*take i* ((*Q NEXT R*, *t*,*y*) # *cs*) = *map* (*lift-seq-esconf es2*) ((*Q*, *t*,*y*) # *cpt'*)› **and**
p4-2: ‹((*Q*, *t*,*y*) # *cpt'*) $\in$ *cpts* (*estran* Γ)› **and**
p4-3: ‹(*last* ((*Q*, *t*,*y*) # *cpt'*), *fin*, *S'*) $\in$ *estran* Γ› **by** *meson*
show *?case*
apply(*rule exI*[**where** *x*=*Suc i*])
apply(*rule exI*[**where** *x*=*S'*])
apply(*rule conjI*)
using *p1* **apply** *simp*
apply(*rule conjI*) **apply** *simp*
apply(*rule conjI*) **using** *p3* **apply** *simp*
apply(*rule conjI*)
apply(*rule exI*[**where** *x*=‹(*Q*,*t*,*y*)#*cpt'*›])
apply(*rule conjI*)
using *p4-1 CptsModSeq*(*5*) **apply** *simp*
apply(*rule conjI*)
apply(*rule CptsComp*)
using *CptsModSeq*(*1*,*5*) **apply** (*auto simp add*: *estran-def*)[]
using *p4-2* **apply** *simp*
using *p4-3* **apply** *simp*
using *p5* ‹*R*=*es2*› **apply**(*simp add*: *all-seq-def*)
using *p6* **by** *fastforce*
next
case (*CptsModSeq-fin P s x a t y Q cs*)
from *CptsModSeq-fin*(*4*) **have** ‹*P*=*es1*› ‹*Q*=*es2*› ‹(*s*,*x*)=*S0*› **by** *simp+*
show *?case*
apply(*rule exI*[**where** *x*=*1*])
apply(*rule exI*[**where** *x*=‹(*t*,*y*)›])
apply(*simp add*: *all-seq-def* ‹*P*=*es1*› ‹*Q*=*es2*› ‹(*s*,*x*)=*S0*›)
apply(*rule conjI*)
apply(*rule CptsOne*)
apply(*rule conjI*)
using *CptsModSeq-fin*(*1*) ‹*P*=*es1*› ‹(*s*,*x*)=*S0*› **apply** (*auto simp add*: *estran-def*)[]
using *CptsModSeq-fin*(*2*) ‹*Q*=*es2*› **by** *simp*

```
next
  case (CptsModChc1)
  then show ?case by simp
next
  case (CptsModChc2)
  then show ?case by simp
next
  case (CptsModJoin1)
  then show ?case by simp
next
  case (CptsModJoin2)
  then show ?case by simp
next
  case (CptsModJoin-fin)
  then show ?case by simp
next
  case (CptsModWhileTMore)
  then show ?case by simp
next
  case (CptsModWhileTOnePartial)
  then show ?case by simp
next
  case (CptsModWhileTOneFull)
  then show ?case by simp
next
  case (CptsModWhileF)
  then show ?case by simp
qed

lemma equiv-aux2:
  ‹∀ i<length cs. fst (cs!i) = P ⟹ (P,s)#cs ∈ cpts tran›
proof(induct cs arbitrary:s)
  case Nil
  show ?case by (rule CptsOne)
next
  case (Cons c cs)
  from Cons(2)[THEN spec[where x=0]] have ‹fst c = P› by simp
  show ?case apply(subst surjective-pairing[of c]) apply(subst ‹fst c = P›)
    apply(rule CptsEnv)
    apply(rule Cons(1))
    using Cons(2) by fastforce
qed

theorem cpts-es-mod-equiv:
  ‹cpts (estran Γ) = cpts-es-mod Γ›
proof
  show ‹cpts (estran Γ) ⊆ cpts-es-mod Γ›
  proof
    fix cpt
```

```
assume ‹cpt ∈ cpts (estran Γ)›
then show ‹cpt ∈ cpts-es-mod Γ›
proof(induct)
  case (CptsOne P S)
  obtain s x where ‹S=(s,x)› by fastforce
  from CptsOne this CptsModOne show ?case by fast
next
  case (CptsEnv P T cs S)
  obtain s x where S:‹S=(s,x)› by fastforce
  obtain t y where T:‹T=(t,y)› by fastforce
  show ?case using CptsModEnv estran-def S T CptsEnv by fast
next
  case (CptsComp P S Q T cs)
  from CptsComp(1) obtain a where h:
    ‹Γ ⊢ (P,S)−es[a]→(Q,T)› using estran-def by blast
  then show ?case
  proof(cases)
    case (EAnon)
    then show ?thesis apply clarify
      apply(erule CptsModAnon) apply blast
      using CptsComp EAnon by blast
  next
    case (EAnon-fin)
    then show ?thesis apply clarify
      apply(erule CptsModAnon-fin) apply blast+
      using CptsComp EAnon by blast
  next
    case (EBasic)
    then show ?thesis apply clarify
      apply(rule CptsModBasic, auto)
      using CptsComp EBasic by simp
  next
    case (EAtom)
    then show ?thesis apply clarify
      apply(rule CptsModAtom) using CptsComp by auto
  next
    case (ESeq)
    then show ?thesis apply clarify
      apply(rule CptsModSeq) using CptsComp by auto
  next
    case (ESeq-fin)
    then show ?thesis apply clarify
      apply(rule CptsModSeq-fin) using CptsComp by auto
  next
    case (EChc1)
    then show ?thesis apply clarify
      apply(rule CptsModChc1) using CptsComp by auto
  next
    case (EChc2)
```

then show *?thesis* **apply** *clarify*
apply(*rule CptsModChc2*) **using** *CptsComp* **by** *auto*
next
case (*EJoin1*)
then show *?thesis* **apply** *clarify*
apply(*rule CptsModJoin1*) **using** *CptsComp* **by** *auto*
next
case (*EJoin2*)
then show *?thesis* **apply** *clarify*
apply(*rule CptsModJoin2*) **using** *CptsComp* **by** *auto*
next
case *EJoin-fin*
then show *?thesis* **apply** *clarify*
apply(*rule CptsModJoin-fin*) **using** *CptsComp* **by** *auto*
next
case *EWhileF*
then show *?thesis* **apply** *clarify*
apply(*rule CptsModWhileF*) **using** *CptsComp* **by** *auto*
next
case (*EWhileT s b P1 x k*)
thm *CptsComp*

show *?thesis*
proof(*cases* ‹*all-seq* (*EWhile b P1*) ((*P1 NEXT EWhile b P1*, *T*) # *cs*)›)
case *True*
from *EWhileT*(*4*) **have** *1*: ‹*hd* ((*Q*, *T*) # *cs*) = (*P1 NEXT EWhile b P1*, *T*)› **by** *simp*
from *True EWhileT*(*4*) **have** *2*: ‹*all-seq* (*EWhile b P1*) ((*Q*, *T*) # *cs*)› **by** *simp*
from *equiv-aux1*[*OF CptsComp*(*2*) *1* ‹*P1*≠*fin*› *2*] **obtain** *cs0* **where**
3: ‹(*Q*, *T*) # *cs* = *map* (*lift-seq-esconf* (*EWhile b P1*)) ((*P1*, *T*) # *cs0*) ∧ (*P1*, *T*) # *cs0* ∈ *cpts* (*estran* Γ) ∧ *fst* (*last* ((*P1*, *T*) # *cs0*)) ≠ *fin*› **by** *meson*

then have *p3-1*: ‹(*Q*, *T*) # *cs* = *map* (*lift-seq-esconf* (*EWhile b P1*)) ((*P1*, *T*) # *cs0*)› **and**
p3-2: ‹(*P1*, *s*, *x*) # *cs0* ∈ *cpts* (*estran* Γ)› **and**
p3-3: ‹*fst* (*last* ((*P1*, *s*, *x*) # *cs0*)) ≠ *fin*› **using** ‹*T*=(*s*,*x*)› **by** *blast+*
from *CptsModWhileTOnePartial*[*OF* ‹*s*∈*b*› *p3-2 p3-3*]
have ‹(*EWhile b P1*, *s*,*x*) # *map* (*lift-seq-esconf* (*EWhile b P1*)) ((*P1*, *s*,*x*) # *cs0*) ∈ *cpts-es-mod* Γ› **.**
with *EWhileT 3* **show** *?thesis* **by** *simp*
next
case *False*
with *EWhileT*(*4*) **have** *not-all-seq*: ‹¬*all-seq* (*EWhile b P1*) ((*Q*,*T*)#*cs*)› **by** *simp*
from *EWhileT*(*4*) **have** ‹(*hd* ((*Q*, *T*) # *cs*)) = (*P1 NEXT EWhile b P1*, *T*)› **by** *simp*
from *split-seq-mod*[*OF CptsComp*(*3*) *this not-all-seq*] **obtain** *i S*′ **where** *split*:

⟨((Q, T) # cs) ! i = (EWhile b P1, S′) ∧
i ≠ 0 ∧
i < length ((Q, T) # cs) ∧
(∃ cpt′. take i ((Q, T) # cs) = map (lift-seq-esconf (EWhile b P1)) ((P1, T) # cpt′) ∧ (P1, T) # cpt′ ∈ cpts (estran Γ) ∧ (last ((P1, T) # cpt′), fin, S′) ∈ estran Γ) ∧
all-seq (EWhile b P1) (take i ((Q, T) # cs)) ∧ drop i ((Q, T) # cs) ∈ cpts-es-mod Γ⟩
by blast
then have 3: ⟨all-seq (EWhile b P1) (take i ((Q, T) # cs))⟩
and ⟨i≠0⟩
and i-lt: ⟨i < length ((Q, T) # cs)⟩
and part2-cpt: ⟨drop i ((Q, T) # cs) ∈ cpts-es-mod Γ⟩
and ex-cpt′: ⟨∃ cpt′. take i ((Q, T) # cs) = map (lift-seq-esconf (EWhile b P1)) ((P1, T) # cpt′) ∧ (P1, T) # cpt′ ∈ cpts (estran Γ) ∧ (last ((P1, T) # cpt′), fin, S′) ∈ estran Γ⟩ by blast+
from ex-cpt′ obtain cpt′ where cpt′1: ⟨take i ((Q, T) # cs) = map (lift-seq-esconf (EWhile b P1)) ((P1, T) # cpt′)⟩ and
cpt′2: ⟨((P1, s,x) # cpt′) ∈ cpts (estran Γ)⟩ and
cpt′3: ⟨(last ((P1, s,x) # cpt′), fin, S′) ∈ estran Γ⟩ using ⟨T=(s,x)⟩ by meson
from cpts-take[OF CptsComp(2)] ⟨i≠0⟩ have 1: ⟨take i ((Q, T) # cs) ∈ cpts (estran Γ)⟩ by fast
have 2: ⟨hd (take i ((Q, T) # cs)) = (P1 NEXT EWhile b P1, T)⟩ using ⟨i≠0⟩ EWhileT(4) by simp
obtain s′ x′ where S′: ⟨S′=(s′,x′)⟩ by fastforce
obtain cs′ where part2-eq: ⟨drop i ((Q, T) # cs) = (EWhile b P1, S′) # cs′⟩
proof
from split have ⟨((Q, T) # cs) ! i = (EWhile b P1, S′)⟩ by argo
with i-lt show ⟨drop i ((Q, T) # cs) = (EWhile b P1, S′) # drop (Suc i) ((Q,T)#cs)⟩
using Cons-nth-drop-Suc by metis
qed
with part2-cpt S′ have ⟨(EWhile b P1, s′,x′) # cs′ ∈ cpts-es-mod Γ⟩ by argo
from cpt′3 have ⟨∃ a. Γ ⊢ last ((P1, s,x) # cpt′) −es[a]→ (fin, S′)⟩ by (simp add: estran-def)
then obtain a where ⟨Γ ⊢ last ((P1, s,x) # cpt′) −es[a]→ (fin, s′, x′)⟩ using S′ by meson
from CptsModWhileTMore[OF ⟨s∈b⟩ cpt′2[simplified] this ⟨(EWhile b P1, s′,x′) # cs′ ∈ cpts-es-mod Γ⟩] have
⟨(EWhile b P1, s, x) # map (lift-seq-esconf (EWhile b P1)) ((P1, s, x) # cpt′) @ (EWhile b P1, s′, x′) # cs′ ∈ cpts-es-mod Γ⟩ .
moreover have ⟨(Q,T)#cs = map (lift-seq-esconf (EWhile b P1)) ((P1, T) # cpt′) @ (EWhile b P1, S′) # cs′⟩
using cpt′1 part2-eq i-lt by (metis append-take-drop-id)
ultimately show ?thesis using EWhileT S′ by argo
qed

qed
qed
qed
next
show ‹*cpts-es-mod* Γ ⊆ *cpts* (*estran* Γ)›
proof
fix *cpt*
assume ‹*cpt* ∈ *cpts-es-mod* Γ›
then show ‹*cpt* ∈ *cpts* (*estran* Γ)›
proof(*induct*)
case (*CptsModOne*)
then show *?case* **by** (*rule CptsOne*)
next
case (*CptsModEnv*)
then show *?case* **using** *CptsEnv* **by** *fast*
next
case (*CptsModAnon P s Q t x cs*)
from *CptsModAnon*(*1*) **have** ‹((*P*,*s*),(*Q*,*t*))∈*ptran* Γ› **by** *simp*
with *CptsModAnon* **show** *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*)
apply(*rule exI*)
apply(*rule EAnon*)
apply *simp*+
done
next
case (*CptsModAnon-fin P s Q t y x k cs*)
from *CptsModAnon-fin*(*1*) **have** ‹((*P*,*s*),(*Q*,*t*))∈*ptran* Γ› **by** *simp*
with *CptsModAnon-fin* **show** *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*)
apply(*rule exI*)
apply(*rule EAnon-fin*)
by *simp*+
next
case (*CptsModBasic*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EBasic*, *auto*) **done**
next
case (*CptsModAtom*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EAtom*, *auto*) **done**
next
case (*CptsModSeq*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule ESeq*, *auto*) **done**
next
case *CptsModSeq-fin*

then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule ESeq-fin*).
next
case (*CptsModChc1*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EChc1*, *auto*) **done**
next
case (*CptsModChc2*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EChc2*, *auto*) **done**
next
case (*CptsModJoin1*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EJoin1*, *auto*) **done**
next
case (*CptsModJoin2*)
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EJoin2*, *auto*) **done**
next
case *CptsModJoin-fin*
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EJoin-fin*).
next
case *CptsModWhileF*
then show *?case* **apply**− **apply**(*rule CptsComp*, *auto simp add*: *estran-def*, *rule exI*)
apply(*rule EWhileF*, *auto*) **done**
next
case (*CptsModWhileTMore s b P x cs a t y cs'*)
from *CptsModWhileTMore*(*2*,*3*) *all-fin-after-fin no-estran-from-fin* **have** ‹$P \neq fin$›
by (*metis last-in-set list.distinct*(*1*) *prod.collapse set-ConsD*)
have *1*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*) @ (*EWhile b P*, *t*,*y*) # *cs'* ∈ *cpts* (*estran* Γ)›
proof−
from *lift-seq-cpt*[*OF* ‹(*P*, *s*,*x*) # *cs* ∈ *cpts* (*estran* Γ)› *CptsModWhileT-More*(*3*)]
have ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*) @ [(*EWhile b P*, *t*,*y*)] ∈ *cpts* (*estran* Γ)› .
then have *cpt-part1*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*) ∈ *cpts* (*estran* Γ)›
apply *simp* **using** *cpts-remove-last* **by** *fast*
from *CptsModWhileTMore*(*3*)

have *tran*: ‹(*last* (*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*)), *hd* ((*EWhile b P*, *t*,*y*) # *cs'*)) ∈ *estran* Γ›
apply (*auto simp add*: *estran-def*)
apply(*rule exI*)
apply(*erule ESeq-fin*)
apply(*rule exI*)
apply(*subst last-map*)
apply *assumption*
apply(*simp add*: *lift-seq-esconf-def case-prod-unfold*)
apply(*subst surjective-pairing*[*of* ‹*snd* (*last cs*)›])
apply(*rule ESeq-fin*)
by *simp*
show *?thesis*
apply(*rule cpts-append-comp*)
apply(*rule cpt-part1*)
apply(*rule CptsModWhileTMore*(*5*))
apply(*rule tran*)
done
qed
show *?case*
apply *simp*
apply(*rule CptsComp*)
apply (*simp add*: *estran-def*)
apply(*rule exI*)
apply(*rule EWhileT*)
apply(*rule* ‹*s*∈*b*›)
apply(*rule* ‹*P*≠*fin*›)
using *1* **by** *fastforce*
next
case (*CptsModWhileTOnePartial s b P x cs*)
from *CptsModWhileTOnePartial*(*3*) *all-fin-after-fin* **have** ‹*P*≠*fin*›
by (*metis CptsModWhileTOnePartial.hyps*(*2*) *fst-conv last-in-set list.distinct*(*1*) *set-ConsD*)
from *lift-seq-cpt-partial*[*OF* ‹(*P*, *s*,*x*) # *cs* ∈ *cpts* (*estran* Γ)› ‹*fst* (*last* ((*P*, *s*,*x*) # *cs*)) ≠ *fin*›]
have *1*: ‹*lift-seq-cpt* (*EWhile b P*) ((*P*, *s*,*x*) # *cs*) ∈ *cpts* (*estran* Γ)› **.**
show *?case*
apply *simp*
apply(*rule CptsComp*)
apply (*simp add*: *estran-def*)
apply(*rule exI*)
apply(*rule EWhileT*)
apply(*rule* ‹*s*∈*b*›)
apply(*rule* ‹*P*≠*fin*›)
using *1* **by** *simp*
next
case (*CptsModWhileTOneFull s b P x cs a t y cs'*)
from *lift-seq-cpt*[*OF* ‹(*P*, *s*,*x*) # *cs* ∈ *cpts* (*estran* Γ)› ‹Γ ⊢ *last* ((*P*, *s*,*x*) # *cs*) −*es*[*a*]→ (*fin*, *t*,*y*)›]

have *1*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*) @ [(*EWhile b P*, *t*,*y*)] ∈ *cpts* (*estran* Γ)› **.**
let *?map* = ‹*map* (λ(-, *s*,*x*). (*EWhile b P*, *s*,*x*)) *cs'*›
have *p*: ‹∀ *i*<*length ?map*. *fst* (*?map*!*i*) = *EWhile b P*› **by** (*simp add*: *case-prod-unfold*)
have *2*: ‹(*EWhile b P*, *t*,*y*) # *map* (λ(-, *s*,*x*). (*EWhile b P*, *s*,*x*)) *cs'* ∈ *cpts* (*estran* Γ)›
using *equiv-aux2*[*OF p*] **.**
from *cpts-append*[*OF 1 2*] **have** *3*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*,*x*) # *cs*) @ (*EWhile b P*, *t*,*y*) # *map* (λ(-, *s*,*x*). (*EWhile b P*, *s*,*x*)) *cs'* ∈ *cpts* (*estran* Γ)› **.**
from *CptsModWhileTOneFull*(*2*,*3*) *all-fin-after-fin no-estran-from-fin* **have** ‹*P*≠*fin*›
by (*metis last-in-set list.distinct*(*1*) *prod.collapse set-ConsD*)
show *?case*
apply *simp*
apply(*rule CptsComp*)
apply(*simp add*: *estran-def*) **apply** (*rule exI*) **apply**(*rule EWhileT*) **apply**(*rule* ‹*s*∈*b*›)
apply(*rule* ‹*P*≠*fin*›)
using *3*[*simplified*] **.**
qed
qed
qed

lemma *ctran-imp-not-etran*:
‹(*c1*,*c2*)∈*estran* Γ ⟹ ¬ *c1* −*e*→ *c2*›
apply (*simp add*: *estran-def*)
apply(*erule exE*)
using *no-estran-to-self* **by** (*metis prod.collapse*)

fun *split* :: ‹('*l*,'*k*,'*s*,'*prog*) *escpt* ⇒ ('*l*,'*k*,'*s*,'*prog*) *escpt* × ('*l*,'*k*,'*s*,'*prog*) *escpt*›
where
‹*split* ((*P* ⋈ *Q*, *s*) # *rest*) = ((*P*,*s*) # *fst* (*split rest*), (*Q*,*s*) # *snd* (*split rest*))› |
‹*split* - = ([],[])›

inductive-cases *estran-all-cases*: ‹(*P* ⋈ *Q*, *s*) # (*R*, *t*) # *cs* ∈ *cpts-es-mod* Γ›

lemma *split-same-length*:
‹*length* (*fst* (*split cpt*)) = *length* (*snd* (*split cpt*))›
by (*induct cpt rule*: *split.induct*) *auto*

lemma *split-same-state1*:
‹*i* < *length* (*fst* (*split cpt*)) ⟹ *snd* (*fst* (*split cpt*) ! *i*) = *snd* (*cpt* ! *i*)›
apply (*induct cpt arbitrary*: *i rule*: *split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-same-state2*:
⟨i < *length* (*snd* (*split cpt*)) $\Longrightarrow$ *snd* (*snd* (*split cpt*) ! i) = *snd* (*cpt* ! i)⟩
apply (*induct cpt arbitrary*: i *rule*: *split.induct*, *auto*)
apply(*case-tac* i; *simp*)
done

lemma *split-length-le1*:
⟨*length* (*fst* (*split cpt*)) $\leq$ *length cpt*⟩
by (*induct cpt rule*: *split.induct*, *auto*)

lemma *split-length-le2*:
⟨*length* (*snd* (*split cpt*)) $\leq$ *length cpt*⟩
by (*induct cpt rule*: *split.induct*, *auto*)

lemma *all-neq1*[*simp*]: ⟨$P \bowtie Q \neq P$⟩
proof
assume ⟨$P \bowtie Q = P$⟩
then have ⟨*es-size* ($P \bowtie Q$) = *es-size* P⟩ **by** *simp*
then show *False* **by** *simp*
qed

lemma *all-neq2*[*simp*]: ⟨$P \bowtie Q \neq Q$⟩
proof
assume ⟨$P \bowtie Q = Q$⟩
then have ⟨*es-size* ($P \bowtie Q$) = *es-size* Q⟩ **by** *simp*
then show *False* **by** *simp*
qed

lemma *split-cpt-aux1*:
⟨(($P \bowtie Q$, *s0*), *fin*, t) $\in$ *estran* $\Gamma \Longrightarrow P$ = *fin* $\wedge$ Q = *fin*⟩
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
done

lemma *split-cpt-aux3*:
⟨(($P \bowtie Q$, s), (R, t)) $\in$ *estran* $\Gamma \Longrightarrow$
$R \neq$ *fin* $\Longrightarrow$
$\exists P'\ Q'.\ R = P' \bowtie Q' \wedge (P = P' \wedge ((Q,s),(Q',t)) \in$ *estran* $\Gamma \vee Q = Q' \wedge ((P,s),(P',t)) \in$ *estran* Γ)⟩
proof−
assume ⟨(($P \bowtie Q$, s), (R, t)) $\in$ *estran* Γ⟩
with *estran-def* **obtain** a **where** h: ⟨$\Gamma \vdash$ ($P \bowtie Q$, s) −*es*[a]→ (R, t)⟩ **by** *blast*
assume ⟨$R \neq$ *fin*⟩
with h **show** *?thesis* **apply**− **by** (*erule estran-p.cases*, *auto simp add*: *estran-def*)
qed

lemma *split-cpt*:

```
  assumes cpt-from:
    ‹cpt ∈ cpts-from (estran Γ) (P ⋈ Q, s0)›
  shows
    ‹fst (split cpt) ∈ cpts-from (estran Γ) (P, s0) ∧
     snd (split cpt) ∈ cpts-from (estran Γ) (Q, s0)›
proof−
  from cpt-from have cpt: ‹cpt ∈ cpts (estran Γ)› and hd-cpt: ‹hd cpt = (P ⋈ Q,
s0)› by auto
  show ?thesis using cpt hd-cpt
  proof(induct arbitrary: P Q s0)
    case (CptsOne)
    then show ?case
      apply(simp add: split-def)
      apply(rule conjI; rule cpts.CptsOne)
      done
  next
    case (CptsEnv)
    then show ?case
      apply(simp add: split-def)
      apply(rule conjI; rule cpts.CptsEnv, simp)
      done
  next
    case (CptsComp P1 S Q1 T cs)
    show ?case
    proof(cases ‹Q1 = fin›)
      case True
      with CptsComp show ?thesis
        apply(simp add: split-def)
        apply(drule split-cpt-aux1)
        apply clarify
        apply(rule conjI; rule CptsOne)
        done
    next
      case False
      with CptsComp show ?thesis
        apply(simp add: split-def)
        apply(rule conjI)
         apply(drule split-cpt-aux3, assumption)
         apply clarify
         apply simp
         apply(erule disjE)
        apply simp
          apply(rule CptsEnv) using surjective-pairing apply metis
        apply clarify
         apply (rule cpts.CptsComp, assumption)
         apply simp
        using surjective-pairing apply metis
        apply(drule split-cpt-aux3) apply assumption
        apply clarsimp
```

apply(*erule disjE*)
 apply *clarify*
 apply(*rule cpts.CptsComp*, *assumption*)
 using *surjective-pairing* **apply** *metis*
apply *clarify*
 apply(*rule CptsEnv*)
 using *surjective-pairing* **apply** *metis*
done
qed
qed
qed

lemma *estran-from-all-both-fin*:
 ‹$\Gamma \vdash (fin \bowtie fin,\ s)\ -es[a]\rightarrow\ (Q1,\ t) \Longrightarrow Q1 = fin$›
 apply(*erule estran-p.cases*, *auto*)
 using *no-estran-from-fin* **apply** *blast+*
 done

lemma *estran-from-all*:
 ‹$\Gamma \vdash (P \bowtie Q,\ s)\ -es[a]\rightarrow\ (Q1,\ t) \Longrightarrow \neg\ (P = fin \wedge Q = fin) \Longrightarrow \exists P'\ Q'.\ Q1 = P' \bowtie Q'$›
 by (*erule estran-p.cases*, *auto*)

lemma *all-fin-after-fin'*:
 ‹$(fin,\ s)\ \#\ cs \in cpts\ (estran\ \Gamma) \Longrightarrow i < Suc\ (length\ cs) \Longrightarrow fst\ (((fin,s)\#cs)!i) = fin$›
 apply(*cases i*) **apply** *simp*
 using *all-fin-after-fin* **by** *fastforce*

lemma *all-fin-after-fin''*:
 assumes *cpt*: ‹$cpt \in cpts\ (estran\ \Gamma)$›
 and *i-lt*: ‹$i < length\ cpt$›
 and *fin*: ‹$fst\ (cpt!i) = fin$›
 shows ‹$\forall j.\ j > i \longrightarrow j < length\ cpt \longrightarrow fst\ (cpt!j) = fin$›
proof(*auto*)

 have ‹$drop\ i\ cpt = cpt!i\ \#\ drop\ (Suc\ i)\ cpt$›
 by (*simp add*: *Cons-nth-drop-Suc i-lt*)
 then have ‹$drop\ i\ cpt = (fst\ (cpt!i),\ snd\ (cpt!i))\ \#\ drop\ (Suc\ i)\ cpt$›
 using *surjective-pairing* **by** *simp*
 with *fin* **have** *1*: ‹$drop\ i\ cpt = (fin,\ snd\ (cpt!i))\ \#\ drop\ (Suc\ i)\ cpt$› **by** *simp*

 from *cpts-drop*[*OF cpt i-lt*] **have** ‹$drop\ i\ cpt \in cpts\ (estran\ \Gamma)$› **.**
 with *1* **have** *2*: ‹$(fin,\ snd\ (cpt!i))\ \#\ drop\ (Suc\ i)\ cpt \in cpts\ (estran\ \Gamma)$› **by** *simp*

 fix *j*
 assume ‹$i < j$›
 assume ‹$j < length\ cpt$›

have ‹$j-i$ < *Suc* (*length* (*drop* (*Suc i*) *cpt*))›
by (*simp add*: *Suc-diff-Suc* ‹$i < j$› ‹$j <$ *length cpt*› *diff-less-mono i-lt less-imp-le*)

from *all-fin-after-fin*′[*OF 2 this*] *1* **have** ‹*fst* (*drop i cpt* ! ($j-i$)) = *fin*› **by** *simp*

then show ‹*fst* (*cpt*!*j*) = *fin*›
apply(*subst* (*asm*) *nth-drop*) **using** *i-lt* **apply** *linarith*
using ‹$i<j$› **by** *simp*
qed

lemma *estran-from-fin-AND-fin*:
‹((*fin* ⋈ *fin*, *s*), *Q1*, *t*) ∈ *estran* Γ ⟹ *Q1* = *fin*›
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
using *no-estran-from-fin* **by** *blast*+

lemma *split-etran-aux*:
‹*P1* = *P* ⋈ *Q* ⟹ ((*P1*,*s*),(*Q1*,*t*)) ∈ *estran* Γ ⟹ (*Q1*,*t*)#*cs* ∈ *cpts* (*estran* Γ) ⟹ *Suc i* < *length* ((*P1*, *s*) # (*Q1*, *t*) # *cs*) ⟹ *fst* (((*P1*, *s*) # (*Q1*, *t*) # *cs*) ! *Suc i*) ≠ *fin* ⟹ ∃ *P*′ *Q*′. *Q1* = *P*′ ⋈ *Q*′›
apply(*cases* ‹*P* = *fin* ∧ *Q* = *fin*›)
apply *simp*
apply(*drule estran-from-fin-AND-fin*)
apply *simp*
using *all-fin-after-fin*′ **apply** *blast*
apply(*simp add*: *estran-def*)
apply(*erule exE*)
using *estran-from-all* **by** *blast*

lemma *split-etran*:
assumes *cpt*: *cpt* ∈ *cpts* (*estran* Γ)
assumes *fst-hd-cpt*: ‹*fst* (*hd cpt*) = *P* ⋈ *Q*›
assumes *Suc-i-lt*: *Suc i* < *length cpt*
assumes *etran*: *cpt*!*i* −*e*→ *cpt*!*Suc i*
assumes *not-fin*: ‹*fst* (*cpt*!*Suc i*) ≠ *fin*›
shows
fst (*split cpt*) ! *i* −*e*→ *fst* (*split cpt*) ! *Suc i* ∧
snd (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i*
using *cpt fst-hd-cpt Suc-i-lt etran not-fin*
proof(*induct arbitrary*:*P Q i*)
case (*CptsOne P s*)
then show *?case* **by** *simp*
next
case (*CptsEnv P1 t cs s*)
show *?case*
proof(*cases i*)
case *0*
with *CptsEnv* **show** *?thesis* **by** *simp*

  next
   case $(Suc\ i')$
    from $CptsEnv(3)$ **have** 1:
      ‹$fst\ (hd\ ((P1,\ t)\ \#\ cs)) = P \bowtie Q$› **by** *simp*
    then have *P1-conv*: ‹$P1 = P \bowtie Q$› **by** *simp*
    from *Suc* ‹$Suc\ i < length\ ((P1,\ s)\ \#\ (P1,\ t)\ \#\ cs)$› **have** 2: ‹$Suc\ i' < length\ ((P1,t)\#cs)$› **by** *simp*
    from *Suc* ‹$((P1,\ s)\ \#\ (P1,\ t)\ \#\ cs)\ !\ i\ {-e\rightarrow}\ ((P1,\ s)\ \#\ (P1,\ t)\ \#\ cs)\ !\ Suc\ i$› **have** 3:
      ‹$((P1,\ t)\ \#\ cs)\ !\ i'\ {-e\rightarrow}\ ((P1,\ t)\ \#\ cs)\ !\ Suc\ i'$› **by** *simp*
    from $CptsEnv(6)$ *Suc* **have** 4: ‹$fst\ (((P1,\ t)\ \#\ cs)\ !\ Suc\ i') \neq fin$› **by** *simp*
    have
      ‹$fst\ (split\ ((P1,\ t)\ \#\ cs))\ !\ i'\ {-e\rightarrow}\ fst\ (split\ ((P1,\ t)\ \#\ cs))\ !\ Suc\ i' \wedge$
       $snd\ (split\ ((P1,\ t)\ \#\ cs))\ !\ i'\ {-e\rightarrow}\ snd\ (split\ ((P1,\ t)\ \#\ cs))\ !\ Suc\ i'$›
      by (*rule* $CptsEnv(2)[OF\ 1\ 2\ 3\ 4]$)
    with *Suc P1-conv* **show** *?thesis* **by** *simp*
  qed
next
  case $(CptsComp\ P1\ s\ Q1\ t\ cs)$
  show *?case*
  proof(*cases i*)
    case 0
    with *CptsComp* **show** *?thesis* **using** *no-estran-to-self'* **by** *auto*
  next
    case $(Suc\ i')$
    from $CptsComp(4)$ **have** 1: ‹$P1 = P \bowtie Q$› **by** *simp*
     have ‹$\exists P'\ Q'.\ Q1 = P' \bowtie Q'$› **using** *split-etran-aux*$[OF\ 1\ CptsComp(1)\ CptsComp(2)]$ $CptsComp(5,7)$ **by** *force*
    then obtain $P'\ Q'$ **where** 2: ‹$Q1 = P' \bowtie Q'$› **by** *blast*
    from 2 **have** 3: ‹$fst\ (hd\ ((Q1,\ t)\ \#\ cs)) = P' \bowtie Q'$› **by** *simp*
    from $CptsComp(5)$ *Suc* **have** 4: ‹$Suc\ i' < length\ ((Q1,t)\#cs)$› **by** *simp*
     from $CptsComp(6)$ *Suc* **have** 5: ‹$((Q1,\ t)\ \#\ cs)\ !\ i'\ {-e\rightarrow}\ ((Q1,\ t)\ \#\ cs)\ !\ Suc\ i'$› **by** *simp*
    from $CptsComp(7)$ *Suc* **have** 6: ‹$fst\ (((Q1,\ t)\ \#\ cs)\ !\ Suc\ i') \neq fin$› **by** *simp*
    have
      ‹$fst\ (split\ ((Q1,\ t)\ \#\ cs))\ !\ i'\ {-e\rightarrow}\ fst\ (split\ ((Q1,\ t)\ \#\ cs))\ !\ Suc\ i' \wedge$
       $snd\ (split\ ((Q1,\ t)\ \#\ cs))\ !\ i'\ {-e\rightarrow}\ snd\ (split\ ((Q1,\ t)\ \#\ cs))\ !\ Suc\ i'$›
      by (*rule* $CptsComp(3)[OF\ 3\ 4\ 5\ 6]$)
    with *Suc* 1 **show** *?thesis* **by** *simp*
  qed
qed

lemma *all-join-aux*:
  ‹$(c1,\ c2) \in estran\ \Gamma \Longrightarrow$
   $fst\ c1 = P \bowtie Q \Longrightarrow$
   $fst\ c2 \neq fin \Longrightarrow$
   $\exists P'\ Q'.\ fst\ c2 = P' \bowtie Q'$›
  apply(*simp add*: *estran-def*, *erule exE*)
  apply(*erule estran-p.cases*, *auto*)

```
  done

lemma all-join:
  ‹cpt ∈ cpts (estran Γ) ⟹
   fst (hd cpt) = P ⋈ Q ⟹
   n < length cpt ⟹
   fst (cpt!n) ≠ fin ⟹
   ∀ i ≤ n. ∃ P′ Q′. fst (cpt!i) = P′ ⋈ Q′›
proof−
  assume cpt: ‹cpt ∈ cpts (estran Γ)›
  with cpts-nonnil have ‹cpt≠[]› by blast
  from cpt cpts-def′ have ct-or-et:
    ‹∀ i. Suc i < length cpt ⟶ (cpt!i, cpt!Suc i) ∈ estran Γ ∨ cpt!i −e→ cpt!Suc
i› by blast
  assume fst-hd-cpt: ‹fst (hd cpt) = P ⋈ Q›
  assume n-lt: ‹n < length cpt›
  assume not-fin: ‹fst (cpt!n) ≠ fin›
  show ‹∀ i ≤ n. ∃ P′ Q′. fst (cpt!i) = P′ ⋈ Q′›
  proof
    fix i
    show ‹i ≤ n ⟶ (∃ P′ Q′. fst (cpt!i) = P′ ⋈ Q′)›
    proof(induct i)
      case 0
      then show ?case
        apply(rule impI)
        apply(rule exI)+
        apply(subst hd-conv-nth[THEN sym])
        apply(rule ‹cpt≠[]›)
        apply(rule fst-hd-cpt)
        done
    next
      case (Suc i)
      show ?case
      proof
        assume Suc-i-le: ‹Suc i ≤ n›
        then have ‹i ≤ n› by simp
        with Suc obtain P′ Q′ where fst-cpt-i: ‹fst (cpt ! i) = P′ ⋈ Q′› by blast
        from Suc-i-le n-lt have Suc-i-lt: ‹Suc i < length cpt› by linarith
       have ‹Suc i < length cpt ⟶ (cpt ! i, cpt ! Suc i) ∈ estran Γ ∨ cpt ! i −e→
cpt ! Suc i›
          by (rule ct-or-et[THEN spec[where x=i]])
        with Suc-i-lt have ct-or-et′:
          ‹(cpt ! i, cpt ! Suc i) ∈ estran Γ ∨ cpt ! i −e→ cpt ! Suc i› by blast
        then show ‹∃ P′ Q′. fst (cpt ! Suc i) = P′ ⋈ Q′›
        proof
          assume ctran: ‹(cpt ! i, cpt ! Suc i) ∈ estran Γ›
          show ‹∃ P′ Q′. fst (cpt ! Suc i) = P′ ⋈ Q′›
          proof(cases ‹fst (cpt!Suc i) = fin›)
            case True
```

have *1*: ‹(*fin*, *snd* (*cpt*!*Suc i*)) # *drop* (*Suc* (*Suc i*)) *cpt* ∈ *cpts* (*estran* Γ)›
proof−
have *cpt-Suc-i*: ‹*cpt*!*Suc i* = (*fin*, *snd* (*cpt*!*Suc i*))›
apply(*subst True*[*THEN sym*]) **by** *simp*
moreover have ‹*drop* (*Suc i*) *cpt* ∈ *cpts* (*estran* Γ)› **by** (*rule cpts-drop*[*OF cpt Suc-i-lt*])
ultimately show *?thesis*
by (*simp add*: *Cons-nth-drop-Suc Suc-i-lt*)
qed
let *?cpt*′ = ‹*drop* (*Suc* (*Suc i*)) *cpt*›
have ‹∀ *c*∈*set ?cpt*′. *fst c* = *fin*› **by** (*rule all-fin-after-fin*[*OF 1*])
then have ‹∀ *j*<*length ?cpt*′. *fst* (*?cpt*′!*j*) = *fin*› **using** *nth-mem* **by** *blast*
then have *all-fin*: ‹∀ *j*. *Suc* (*Suc i*) + *j* < *length cpt* ⟶ *fst* (*cpt*!(*Suc* (*Suc i*) + *j*)) = *fin*› **by** *auto*
have ‹*fst* (*cpt*!*n*) = *fin*›
proof(*cases* ‹*Suc i* = *n*›)
case *True*
then show *?thesis* **using** ‹*fst* (*cpt* ! *Suc i*) = *fin*› **by** *simp*
next
case *False*
with ‹*Suc i* ≤ *n*› **have** ‹*Suc* (*Suc i*) ≤ *n*› **by** *linarith*
then show *?thesis* **using** *all-fin n-lt le-Suc-ex* **by** *blast*
qed
with *not-fin* **have** *False* **by** *blast*
then show *?thesis* **by** *blast*
next
case *False*
from *Suc* ‹*i*≤*n*› **obtain** *P*′ *Q*′ **where** *1*: ‹*fst* (*cpt* ! *i*) = *P*′ ⋈ *Q*′› **by** *blast*
show *?thesis* **by** (*rule all-join-aux*[*OF ctran 1 False*])
qed
next
assume *etran*: ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i*›
then show ‹∃ *P*′ *Q*′. *fst* (*cpt* ! *Suc i*) = *P*′ ⋈ *Q*′›
apply *simp*
using *fst-cpt-i* **by** *metis*
qed
qed
qed
qed
qed

lemma *all-join-aux*′:
‹*fst* (*cpt* ! *m*) = *fin* ⟹ *length* (*fst* (*split cpt*)) ≤ *m* ∧ *length* (*snd* (*split cpt*)) ≤ *m*›
apply(*induct cpt arbitrary*:*m rule*:*split.induct*; *simp*)
apply(*case-tac m*; *simp*)
done

lemma *all-join1*:
⟨$\forall i < length\ (fst\ (split\ cpt)).\ \exists P'\ Q'.\ fst\ (cpt!i) = P' \bowtie Q'$⟩
apply(*induct cpt rule:split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *all-join2*:
⟨$\forall i < length\ (snd\ (split\ cpt)).\ \exists P'\ Q'.\ fst\ (cpt!i) = P' \bowtie Q'$⟩
apply(*induct cpt rule:split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-length*:
⟨$cpt \in cpts\ (estran\ \Gamma) \Longrightarrow$
$fst\ (hd\ cpt) = P \bowtie Q \Longrightarrow$
$Suc\ m < length\ cpt \Longrightarrow$
$fst\ (cpt\ !\ m) \neq fin \Longrightarrow$
$fst\ (cpt\ !\ Suc\ m) = fin \Longrightarrow$
$length\ (fst\ (split\ cpt)) = Suc\ m \wedge length\ (snd\ (split\ cpt)) = Suc\ m$⟩
proof(*induct cpt arbitrary*: *P Q m rule*: *split.induct*; *simp*)
fix *P Q s Pa Qa m*
fix *rest*
assume *IH*:
⟨$\bigwedge P\ Q\ m.$
$rest \in cpts\ (estran\ \Gamma) \Longrightarrow$
$fst\ (hd\ rest) = P \bowtie Q \Longrightarrow$
$Suc\ m < length\ rest \Longrightarrow fst\ (rest\ !\ m) \neq fin \Longrightarrow fst\ (rest\ !\ Suc\ m) = fin \Longrightarrow$
$length\ (fst\ (split\ rest)) = Suc\ m \wedge length\ (snd\ (split\ rest)) = Suc\ m$⟩
assume *a1*: ⟨$(Pa \bowtie Qa,\ s)\ \#\ rest \in cpts\ (estran\ \Gamma)$⟩
assume *a2*: ⟨$m < length\ rest$⟩
then have ⟨$rest \neq []$⟩ **by** *fastforce*
from *cpts-tl*[*OF a1*] *this* **have** *1*: ⟨$rest \in cpts\ (estran\ \Gamma)$⟩ **by** *simp*
assume *a3*: ⟨$fst\ (((Pa \bowtie Qa,\ s)\ \#\ rest)\ !\ m) \neq fin$⟩
from *all-join*[*OF a1*] *a2 a3* **have** *2*: ⟨$\forall i \le m.\ \exists P'\ Q'.\ fst\ (((Pa \bowtie Qa,\ s)\ \#\ rest)\ !\ i) = P' \bowtie Q'$⟩
by (*metis fstI length-Cons less-SucI list.sel*(*1*))
assume *a4*: ⟨$fst\ (rest\ !\ m) = fin$⟩
show ⟨$length\ (fst\ (split\ rest)) = m \wedge length\ (snd\ (split\ rest)) = m$⟩
proof(*cases* ⟨$m=0$⟩)
case *True*
with *a4* **have** ⟨$fst\ (rest\ !\ 0) = fin$⟩ **by** *simp*
with *hd-conv-nth*[*OF* ⟨$rest \neq []$⟩] **have** ⟨$fst\ (hd\ rest) = fin$⟩ **by** *simp*
then obtain *t* **where** ⟨$hd\ rest = (fin,t)$⟩ **using** *surjective-pairing* **by** *metis*
then have ⟨$rest = (fin,t)\ \#\ tl\ rest$⟩ **using** *hd-Cons-tl*[*OF* ⟨$rest \neq []$⟩] **by** *simp*
then have ⟨$split\ rest = ([],[])$⟩ **apply**− **apply**(*erule ssubst*) **by** *simp*
then show *?thesis* **using** *True* **by** *simp*
next
case *False*

then have ‹$m \geq 1$› **by** *fastforce*
from *2*[*rule-format*, *of 1*, *OF this*] **obtain** P' Q' **where** ‹*fst* (((*Pa* ⋈ *Qa*, *s*) # *rest*) ! *1*) = P' ⋈ Q'› **by** *blast*
with *hd-conv-nth*[*OF* ‹*rest*≠[]›] **have** *fst-hd-rest*: ‹*fst* (*hd rest*) = P' ⋈ Q'› **by** *simp*
from *not0-implies-Suc*[*OF False*] **obtain** m' **where** m': ‹m = *Suc* m'› **by** *blast*
from *a2* m' **have** *Suc-m'-lt*: ‹*Suc* m' < *length rest*› **by** *simp*
from *a3* m' **have** *not-fin*: ‹*fst* (*rest* ! m') ≠ *fin*› **by** *simp*
from *a4* m' **have** *fin*: ‹*fst* (*rest* ! *Suc* m') = *fin*› **by** *simp*
from *IH*[*OF 1 fst-hd-rest Suc-m'-lt not-fin fin*] m' **show** *?thesis* **by** *simp*
qed
qed

lemma *split-prog1*:
‹i < *length* (*fst* (*split cpt*)) ⟹ *fst* (*cpt*!i) = P ⋈ Q ⟹ *fst* (*fst* (*split cpt*) ! i) = P›
apply(*induct cpt arbitrary*:*i rule*:*split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-prog2*:
‹i < *length* (*snd* (*split cpt*)) ⟹ *fst* (*cpt*!i) = P ⋈ Q ⟹ *fst* (*snd* (*split cpt*) ! i) = Q›
apply(*induct cpt arbitrary*:*i rule*:*split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-ctran-aux*:
‹((*P* ⋈ *Q*, *s*), P' ⋈ Q', *t*) ∈ *estran* Γ ⟹
((*P*, *s*), P', *t*) ∈ *estran* Γ ∧ *Q* = Q' ∨ ((*Q*, *s*), Q', *t*) ∈ *estran* Γ ∧ *P* = P'›
apply(*simp add*: *estran-def*, *erule exE*)
apply(*erule estran-p.cases*, *auto*)
done

lemma *split-ctran*:
assumes *cpt*: *cpt* ∈ *cpts* (*estran* Γ)
assumes *fst-hd-cpt*: ‹*fst* (*hd cpt*) = *P* ⋈ *Q*›
assumes *not-fin* : ‹*fst* (*cpt*!*Suc i*) ≠ *fin*›
assumes *Suc-i-lt*: *Suc i* < *length cpt*
assumes *ctran*: (*cpt*!*i*, *cpt*!*Suc i*) ∈ *estran* Γ
shows
‹(*fst* (*split cpt*) ! *i*, *fst* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *snd* (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i* ∨
(*snd* (*split cpt*) ! *i*, *snd* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *fst* (*split cpt*) ! *i* −*e*→ *fst* (*split cpt*) ! *Suc i*›
proof−
have *all-All'*: ‹∀ *j*≤*Suc i*. ∃ P' Q'. *fst* (*cpt* ! *j*) = P' ⋈ Q'› **by** (*rule all-join*[*OF cpt fst-hd-cpt Suc-i-lt not-fin*])
show *?thesis*

using *cpt fst-hd-cpt Suc-i-lt ctran all-All'*
proof(*induct arbitrary*:*P Q i*)
case (*CptsOne P s*)
then show *?case* **by** *simp*
next
case (*CptsEnv P1 t cs s*)
from *CptsEnv*(*3*) **have** *1*: ‹*fst* (*hd* ((*P1*, *t*) # *cs*)) = *P* ⋈ *Q*› **by** *simp*
show *?case*
proof(*cases i*)
case *0*
with *CptsEnv* **show** *?thesis*
apply (*simp add*: *split-def*)
using *no-estran-to-self'* **by** *blast*
next
case (*Suc i'*)
with *CptsEnv* **have**
‹(*fst* (*split* ((*P1*, *t*) # *cs*)) ! *i'*, *fst* (*split* ((*P1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ ∧ *snd* (*split* ((*P1*, *t*) # *cs*)) ! *i'* −*e*→ *snd* (*split* ((*P1*, *t*) # *cs*)) ! *Suc i'* ∨
(*snd* (*split* ((*P1*, *t*) # *cs*)) ! *i'*, *snd* (*split* ((*P1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ ∧ *fst* (*split* ((*P1*, *t*) # *cs*)) ! *i'* −*e*→ *fst* (*split* ((*P1*, *t*) # *cs*)) ! *Suc i'*›
by *fastforce*
then show *?thesis* **using** *Suc 1* **by** *simp*
qed
next
case (*CptsComp P1 s Q1 t cs*)
from *CptsComp*(*7*)[*THEN spec*[**where** *x*=*1*]] **obtain** *P' Q'* **where** *Q1*: ‹*Q1* = *P'* ⋈ *Q'*› **by** *auto*
show *?case*
proof(*cases i*)
case *0*
with *Q1 CptsComp* **show** *?thesis*
apply(*simp add*: *split-def*)
using *split-ctran-aux* **by** *fast*
next
case (*Suc i'*)
from *Q1* **have** *1*: ‹*fst* (*hd* ((*Q1*, *t*) # *cs*)) = *P'* ⋈ *Q'*› **by** *simp*
from *CptsComp*(*5*) *Suc* **have** *2*: ‹*Suc i'* < *length* ((*Q1*, *t*) # *cs*)› **by** *simp*
from *CptsComp*(*6*) *Suc* **have** *3*: ‹(((*Q1*, *t*) # *cs*) ! *i'*, ((*Q1*, *t*) # *cs*) ! *Suc i'*) ∈ *estran* Γ› **by** *simp*
from *CptsComp*(*7*) *Suc* **have** *4*: ‹∀ *j*≤*Suc i'*. ∃ *P' Q'*. *fst* (((*Q1*, *t*) # *cs*) ! *j*) = *P'* ⋈ *Q'*› **by** *auto*
have
‹(*fst* (*split* ((*Q1*, *t*) # *cs*)) ! *i'*, *fst* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ ∧ *snd* (*split* ((*Q1*, *t*) # *cs*)) ! *i'* −*e*→ *snd* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'* ∨
(*snd* (*split* ((*Q1*, *t*) # *cs*)) ! *i'*, *snd* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ ∧ *fst* (*split* ((*Q1*, *t*) # *cs*)) ! *i'* −*e*→ *fst* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*›
by (*rule CptsComp*(*3*)[*OF 1 2 3 4*])
with *Suc CptsComp*(*4*) **show** *?thesis* **by** *simp*
qed

qed
qed

lemma *etran-imp-not-ctran*:
⟨$c1\ -e\rightarrow\ c2 \Longrightarrow \neg((c1,c2) \in estran\ \Gamma)$⟩
using *no-estran-to-self''* **by** *fastforce*

lemma *split-etran1-aux*:
⟨$((P' \bowtie Q,\ s),\ P' \bowtie Q',\ t) \in estran\ \Gamma \Longrightarrow P = P' \Longrightarrow ((Q,\ s),\ Q',\ t) \in estran\ \Gamma$⟩
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
using *no-estran-to-self* **by** *blast*

lemma *split-etran1*:
assumes *cpt*: ⟨$cpt \in cpts\ (estran\ \Gamma)$⟩
and *fst-hd-cpt*: ⟨$fst\ (hd\ cpt) = P \bowtie Q$⟩
and *Suc-i-lt*: ⟨$Suc\ i < length\ cpt$⟩
and *not-fin*: ⟨$fst\ (cpt\ !\ Suc\ i) \neq fin$⟩
and *etran*: ⟨$fst\ (split\ cpt)\ !\ i\ -e\rightarrow fst\ (split\ cpt)\ !\ Suc\ i$⟩
shows
⟨$cpt\ !\ i\ -e\rightarrow cpt\ !\ Suc\ i\ \vee$
$(snd\ (split\ cpt)\ !\ i,\ snd\ (split\ cpt)\ !\ Suc\ i) \in estran\ \Gamma$⟩
proof−
have *all-All'*: ⟨$\forall j \le Suc\ i.\ \exists P'\ Q'.\ fst\ (cpt\ !\ j) = P' \bowtie Q'$⟩
by (*rule all-join*[*OF cpt fst-hd-cpt Suc-i-lt not-fin*])
show *?thesis*
using *cpt fst-hd-cpt Suc-i-lt not-fin etran all-All'*
proof(*induct arbitrary*:*P Q i*)
case (*CptsOne P s*)
then show *?case* **by** *simp*
next
case (*CptsEnv P1 t cs s*)
show *?case*
proof(*cases i*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc i'*)
from *CptsEnv*(*3*) **have** *1*: ⟨$fst\ (hd\ ((P1,\ t)\ \#\ cs)) = P \bowtie Q$⟩ **by** *simp*
then have *P1*: ⟨$P1 = P \bowtie Q$⟩ **by** *simp*
from *CptsEnv*(*4*) *Suc* **have** *2*: ⟨$Suc\ i' < length\ ((P1,\ t)\ \#\ cs)$⟩ **by** *simp*
from *CptsEnv*(*5*) *Suc* **have** *3*: ⟨$fst\ (((P1,\ t)\ \#\ cs)\ !\ Suc\ i') \neq fin$⟩ **by** *simp*
from *CptsEnv*(*6*) *Suc P1*
have *4*: ⟨$fst\ (split\ ((P1,\ t)\ \#\ cs))\ !\ i'\ -e\rightarrow fst\ (split\ ((P1,\ t)\ \#\ cs))\ !\ Suc\ i'$⟩ **by** *simp*
from *CptsEnv*(*7*) *Suc* **have** *5*: ⟨$\forall j \le Suc\ i'.\ \exists P'\ Q'.\ fst\ (((P1,\ t)\ \#\ cs)\ !\ j) = P' \bowtie Q'$⟩ **by** *auto*

from *CptsEnv*(*2*)[*OF 1 2 3 4 5*]
have ‹((*P1*, *t*) # *cs*) ! *i'* −*e*→ ((*P1*, *t*) # *cs*) ! *Suc i'* ∨ (*snd* (*split* ((*P1*, *t*) # *cs*)) ! *i'*, *snd* (*split* ((*P1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ› **.**
then show *?thesis* **using** *Suc P1* **by** *simp*
qed
next
case (*CptsComp P1 s Q1 t cs*)
from *CptsComp*(*4*) **have** *P1*: ‹*P1* = *P* ⋈ *Q*› **by** *simp*
from *CptsComp*(*8*)[*THEN spec*[**where** *x=1*]] **obtain** *P' Q'* **where** *Q1*: ‹*Q1* = *P'* ⋈ *Q'*› **by** *auto*
show *?case*
proof(*cases i*)
case *0*
with *P1 Q1 CptsComp*(*1*) *CptsComp*(*7*) **show** *?thesis*
apply (*simp add*: *split-def*)
apply(*rule disjI2*)
apply(*erule split-etran1-aux*, *assumption*)
done
next
case (*Suc i'*)
have *1*: ‹*fst* (*hd* ((*Q1*, *t*) # *cs*)) = *P'* ⋈ *Q'*› **using** *Q1* **by** *simp*
from *CptsComp*(*5*) *Suc* **have** *2*: ‹*Suc i'* < *length* ((*Q1*, *t*) # *cs*)› **by** *simp*
from *CptsComp*(*6*) *Suc* **have** *3*: ‹*fst* (((*Q1*, *t*) # *cs*) ! *Suc i'*) ≠ *fin*› **by** *simp*
from *CptsComp*(*7*) *Suc P1* **have** *4*: ‹*fst* (*split* ((*Q1*, *t*) # *cs*)) ! *i'* −*e*→ *fst* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*› **by** *simp*
from *CptsComp*(*8*) *Suc* **have** *5*: ‹∀ *j*≤*Suc i'*. ∃ *P' Q'*. *fst* (((*Q1*, *t*) # *cs*) ! *j*) = *P'* ⋈ *Q'*› **by** *auto*
from *CptsComp*(*3*)[*OF 1 2 3 4 5*]
have ‹((*Q1*, *t*) # *cs*) ! *i'* −*e*→ ((*Q1*, *t*) # *cs*) ! *Suc i'* ∨ (*snd* (*split* ((*Q1*, *t*) # *cs*)) ! *i'*, *snd* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ› **.**
then show *?thesis* **using** *Suc P1* **by** *simp*
qed
qed
qed

lemma *split-etran2-aux*:
‹((*P* ⋈ *Q'*, *s*), *P'* ⋈ *Q'*, *t*) ∈ *estran* Γ ⟹ *Q* = *Q'* ⟹ ((*P*, *s*), *P'*, *t*) ∈ *estran* Γ›
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
using *no-estran-to-self* **by** *blast*

lemma *split-etran2*:
assumes *cpt*: ‹*cpt* ∈ *cpts* (*estran* Γ)›
and *fst-hd-cpt*: ‹*fst* (*hd cpt*) = *P* ⋈ *Q*›
and *Suc-i-lt*: ‹*Suc i* < *length cpt*›
and *not-fin*: ‹*fst* (*cpt* ! *Suc i*) ≠ *fin*›
and *etran*: ‹*snd* (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i*›

shows
‹cpt ! i −e→ cpt ! Suc i ∨
(fst (split cpt) ! i, fst (split cpt) ! Suc i) ∈ estran Γ›
proof−
have *all-All′*: ‹∀ j≤Suc i. ∃ P′ Q′. fst (cpt ! j) = P′ ⋈ Q′›
by (*rule all-join*[*OF cpt fst-hd-cpt Suc-i-lt not-fin*])
show *?thesis*
using *cpt fst-hd-cpt Suc-i-lt not-fin etran all-All′*
proof(*induct arbitrary*:*P Q i*)
case (*CptsOne P s*)
then show *?case* **by** *simp*
next
case (*CptsEnv P1 t cs s*)
show *?case*
proof(*cases i*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc i′*)
from *CptsEnv*(*3*) **have** *1*: ‹fst (hd ((P1, t) # cs)) = P ⋈ Q› **by** *simp*
then have *P1*: ‹P1 = P ⋈ Q› **by** *simp*
from *CptsEnv*(*4*) *Suc* **have** *2*: ‹Suc i′ < length ((P1, t) # cs)› **by** *simp*
from *CptsEnv*(*5*) *Suc* **have** *3*: ‹fst (((P1, t) # cs) ! Suc i′) ≠ fin› **by** *simp*
from *CptsEnv*(*6*) *Suc P1* **have** *4*: ‹snd (split ((P1, t) # cs)) ! i′ −e→ snd (split ((P1, t) # cs)) ! Suc i′› **by** *simp*
from *CptsEnv*(*7*) *Suc* **have** *5*: ‹∀ j≤Suc i′. ∃ P′ Q′. fst (((P1, t) # cs) ! j) = P′ ⋈ Q′› **by** *auto*
have ‹((P1, t) # cs) ! i′ −e→ ((P1, t) # cs) ! Suc i′ ∨ (fst (split ((P1, t) # cs)) ! i′, fst (split ((P1, t) # cs)) ! Suc i′) ∈ estran Γ›
by (*rule CptsEnv*(*2*)[*OF 1 2 3 4 5*])
then show *?thesis* **using** *Suc P1* **by** *simp*
qed
next
case (*CptsComp P1 s Q1 t cs*)
from *CptsComp*(*4*) **have** *P1*: ‹P1 = P ⋈ Q› **by** *simp*
from *CptsComp*(*8*)[*THEN spec*[**where** *x=1*]] **obtain** *P′ Q′* **where** *Q1*: ‹Q1 = P′ ⋈ Q′› **by** *auto*
show *?case*
proof(*cases i*)
case *0*
with *P1 Q1 CptsComp*(*1*) *CptsComp*(*7*) **show** *?thesis*
apply (*simp add*: *split-def*)
apply(*rule disjI2*)
apply(*erule split-etran2-aux*, *assumption*)
done
next
case (*Suc i′*)
have *1*: ‹fst (hd ((Q1, t) # cs)) = P′ ⋈ Q′› **using** *Q1* **by** *simp*
from *CptsComp*(*5*) *Suc* **have** *2*: ‹Suc i′ < length ((Q1, t) # cs)› **by** *simp*

from *CptsComp*(*6*) *Suc* **have** *3*: ‹*fst* (((*Q1*, *t*) # *cs*) ! *Suc i'*) ≠ *fin*› **by** *simp*
from *CptsComp*(*7*) *Suc P1* **have** *4*: ‹*snd* (*split* ((*Q1*, *t*) # *cs*)) ! *i'* −*e*→ *snd* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*› **by** *simp*
from *CptsComp*(*8*) *Suc* **have** *5*: ‹∀ *j*≤*Suc i'*. ∃ *P' Q'*. *fst* (((*Q1*, *t*) # *cs*) ! *j*) = *P'* ⋈ *Q'*› **by** *auto*
have ‹((*Q1*, *t*) # *cs*) ! *i'* −*e*→ ((*Q1*, *t*) # *cs*) ! *Suc i'* ∨ (*fst* (*split* ((*Q1*, *t*) # *cs*)) ! *i'*, *fst* (*split* ((*Q1*, *t*) # *cs*)) ! *Suc i'*) ∈ *estran* Γ›
by (*rule CptsComp*(*3*)[*OF 1 2 3 4 5*])
then show *?thesis* **using** *Suc P1* **by** *simp*
qed
qed
qed

lemma *split-ctran1-aux*:
‹*i* < *length* (*fst* (*split cpt*)) ⟹
fst (*cpt*!*i*) ≠ *fin*›
apply(*induct cpt arbitrary*: *i rule*: *split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-ctran1*:
‹*cpt* ∈ *cpts* (*estran* Γ) ⟹
fst (*hd cpt*) = *P* ⋈ *Q* ⟹
Suc i < *length* (*fst* (*split cpt*)) ⟹
(*fst* (*split cpt*) ! *i*, *fst* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ⟹
(*cpt*!*i*, *cpt*!*Suc i*) ∈ *estran* Γ›
proof(*rule ccontr*)
assume *cpt*: ‹*cpt* ∈ *cpts* (*estran* Γ)›
assume *fst-hd-cpt*: ‹*fst* (*hd cpt*) = *P* ⋈ *Q*›
assume *Suc-i-lt1*: ‹*Suc i* < *length* (*fst* (*split cpt*))›
with *split-length-le1*[*of cpt*]
have *Suc-i-lt*: ‹*Suc i* < *length cpt*› **by** *fastforce*
assume *ctran1*: ‹(*fst* (*split cpt*) ! *i*, *fst* (*split cpt*) ! *Suc i*) ∈ *estran* Γ›
assume ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ›
with *ctran-or-etran*[*OF cpt Suc-i-lt*] **have** *etran*: ‹*cpt*!*i* −*e*→ *cpt*!*Suc i*› **by** *blast*
from *split-ctran1-aux*[*OF Suc-i-lt1*] **have** ‹*fst* (*cpt* ! *Suc i*) ≠ *fin*› **.**
from *split-etran*[*OF cpt fst-hd-cpt Suc-i-lt etran this*, *THEN conjunct1*] **have** ‹*fst* (*split cpt*) ! *i* −*e*→ *fst* (*split cpt*) ! *Suc i*› **.**
with *ctran1 no-estran-to-self''* **show** *False* **by** *fastforce*
qed

lemma *split-ctran2-aux*:
‹*i* < *length* (*snd* (*split cpt*)) ⟹
fst (*cpt*!*i*) ≠ *fin*›
apply(*induct cpt arbitrary*: *i rule*: *split.induct*, *auto*)
apply(*case-tac i*; *simp*)
done

lemma *split-ctran2*:

⟨cpt ∈ cpts (estran Γ) ⟹
fst (hd cpt) = P ⋈ Q ⟹
Suc i < length (snd (split cpt)) ⟹
(snd (split cpt) ! i, snd (split cpt) ! Suc i) ∈ estran Γ ⟹
(cpt!i, cpt!Suc i) ∈ estran Γ⟩
proof(*rule ccontr*)
assume *cpt*: ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩
assume *fst-hd-cpt*: ⟨*fst* (*hd cpt*) = *P* ⋈ *Q*⟩
assume *Suc-i-lt2*: ⟨*Suc i* < *length* (*snd* (*split cpt*))⟩
with *split-length-le2*[*of cpt*]
have *Suc-i-lt*: ⟨*Suc i* < *length cpt*⟩ **by** *fastforce*
assume *ctran2*: ⟨(*snd* (*split cpt*) ! *i*, *snd* (*split cpt*) ! *Suc i*) ∈ *estran* Γ⟩
assume ⟨(*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ⟩
with *ctran-or-etran*[*OF cpt Suc-i-lt*] **have** *etran*: ⟨*cpt*!*i* −*e*→ *cpt*!*Suc i*⟩ **by** *blast*
from *split-ctran2-aux*[*OF Suc-i-lt2*] **have** ⟨*fst* (*cpt* ! *Suc i*) ≠ *fin*⟩ **.**
from *split-etran*[*OF cpt fst-hd-cpt Suc-i-lt etran this*, *THEN conjunct2*] **have** ⟨*snd* (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i*⟩ **.**
with *ctran2 no-estran-to-self''* **show** *False* **by** *fastforce*
qed

lemma *no-fin-before-non-fin*:
assumes *cpt*: ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩
and *m-lt*: ⟨*m* < *length cpt*⟩
and *m-not-fin*: *fst* (*cpt*!*m*) ≠ *fin*
and ⟨*i*≤*m*⟩
shows ⟨*fst* (*cpt*!*i*) ≠ *fin*⟩
proof(*rule ccontr*, *simp*)
assume *i-fin*: ⟨*fst* (*cpt*!*i*) = *fin*⟩
from *m-lt* ⟨*i*≤*m*⟩ **have** *i-lt*: ⟨*i* < *length cpt*⟩ **by** *simp*
from *cpts-drop*[*OF cpt this*] **have** ⟨*drop i cpt* ∈ *cpts* (*estran* Γ)⟩ **by** *assumption*
have *1*: ⟨*drop i cpt* = (*fin*, *snd* (*cpt*!*i*)) # *drop* (*Suc i*) *cpt*⟩ **using** *i-fin i-lt*
by (*metis Cons-nth-drop-Suc surjective-pairing*)
from *cpts-drop*[*OF cpt i-lt*] **have** ⟨*drop i cpt* ∈ *cpts* (*estran* Γ)⟩ **by** *assumption*
with *1* **have** ⟨(*fin*, *snd* (*cpt*!*i*)) # *drop* (*Suc i*) *cpt* ∈ *cpts* (*estran* Γ)⟩ **by** *simp*
from *all-fin-after-fin*[*OF this*] **have** ⟨∀ *c*∈*set* (*drop* (*Suc i*) *cpt*). *fst c* = *fin*⟩ **by** *assumption*
then have ⟨∀ *j*<*length* (*drop* (*Suc i*) *cpt*). *fst* (*drop* (*Suc i*) *cpt* ! *j*) = *fin*⟩ **using** *nth-mem* **by** *blast*
then have *2*: ⟨∀ *j*. *Suc i* + *j* < *length cpt* ⟶ *fst* (*cpt* ! (*Suc i* + *j*)) = *fin*⟩ **by** *simp*
find-theorems *nth drop*
show *False*
proof(*cases* ⟨*i*=*m*⟩)
case *True*
then show *False* **using** *m-not-fin i-fin* **by** *simp*
next
case *False*
with ⟨*i*≤*m*⟩ **have** ⟨*i*<*m*⟩ **by** *simp*
with *2 m-not-fin* **show** *False*

using *Suc-leI le-Suc-ex m-lt* **by** *blast*
qed
qed

lemma *no-estran-from-fin′*:
‹$(c1,\ c2) \in estran\ \Gamma \Longrightarrow fst\ c1 \neq fin$›
apply(*simp add*: *estran-def*)
apply(*subst* (*asm*) *surjective-pairing*[*of c1*])
using *no-estran-from-fin* **by** *metis*

3.1 Compositionality of the Semantics

3.1.1 Definition of the conjoin operator

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

definition *same-state* :: $('l,'k,'s,'prog)$ *pesconf list* $\Rightarrow$ $('k \Rightarrow ('l,'k,'s,'prog)$ *esconf list*$)$ $\Rightarrow$ *bool* **where**
$same\text{-}state\ c\ cs \equiv \forall k\ j.\ j < length\ c \longrightarrow snd\ (c!j) = snd\ (cs\ k\ !\ j)$

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

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

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

3.1.2 Properties of the conjoin operator

lemma *conjoin-ctran*:
assumes *conjoin*: ‹$pc \propto cs$›
assumes *Suc-i-lt*: ‹$Suc\ i < length\ pc$›
assumes *ctran*: ‹$\Gamma \vdash pc!i\ -pes[a\sharp k]\rightarrow\ pc!Suc\ i$›
shows
‹$(\Gamma \vdash cs\ k\ !\ i\ -es[a\sharp k]\rightarrow\ cs\ k\ !\ Suc\ i) \wedge$
$(\forall k'.\ k' \neq k \longrightarrow (cs\ k'\ !\ i\ -e\rightarrow\ cs\ k'\ !\ Suc\ i))$›

proof −
 from *conjoin* **have** ‹*compat-tran pc cs*› **using** *conjoin-def* **by** *blast*
 then have
 h: ‹$\forall j$. *Suc j* < *length pc* $\longrightarrow$
 ($\exists\, t\ k\ \Gamma$. $\Gamma \vdash$ *pc* ! *j* −*pes*[*t*♯*k*]→ *pc* ! *Suc j*) $\wedge$
 ($\forall\, k\ t\ \Gamma$. ($\Gamma \vdash$ *pc* ! *j* −*pes*[*t*♯*k*]→ *pc* ! *Suc j*) $\longrightarrow$ ($\Gamma \vdash$ *cs k* ! *j* −*es*[*t*♯*k*]→ *cs k* ! *Suc j*) $\wedge$ ($\forall\, k'$. $k' \neq k \longrightarrow$ *fst* (*cs k'* ! *j*) = *fst* (*cs k'* ! *Suc j*))) $\vee$
 fst (*pc* ! *j*) = *fst* (*pc* ! *Suc j*) $\wedge$ ($\forall\, k$. *fst* (*cs k* ! *j*) = *fst* (*cs k* ! *Suc j*))› **by** (*simp add*: *compat-tran-def*)
 from *ctran* **have** ‹*fst* (*pc* ! *i*) $\neq$ *fst* (*pc* ! *Suc i*)› **using** *no-pestran-to-self* **by** (*metis prod.collapse*)
 with *h*[*rule-format*, *OF Suc-i-lt*] **have**
 ‹$\forall\, k\ t\ \Gamma$. ($\Gamma \vdash$ *pc* ! *i* −*pes*[*t*♯*k*]→ *pc* ! *Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k* ! *i* −*es*[*t*♯*k*]→ *cs k* ! *Suc i*) $\wedge$ ($\forall\, k'$. $k' \neq k \longrightarrow$ *fst* (*cs k'* ! *i*) = *fst* (*cs k'* ! *Suc i*))›
 by *argo*
 from *this*[*rule-format*, *OF ctran*] **show** *?thesis* **by** *fastforce*
qed

lemma *conjoin-etran*:
 assumes *conjoin*: ‹*pc* $\propto$ *cs*›
 assumes *Suc-i-lt*: ‹*Suc i* < *length pc*›
 assumes *etran*: ‹*pc*!*i* −*e*→ *pc*!*Suc i*›
 shows ‹$\forall\, k$. *cs k* ! *i* −*e*→ *cs k* ! *Suc i*›
proof −
 from *conjoin* **have** ‹*compat-tran pc cs*› **using** *conjoin-def* **by** *blast*
 then have
 ‹$\forall j$. *Suc j* < *length pc* $\longrightarrow$
 ($\exists\, t\ k\ \Gamma$. $\Gamma \vdash$ *pc* ! *j* −*pes*[*t*♯*k*]→ *pc* ! *Suc j*) $\wedge$
 ($\forall\, k\ t\ \Gamma$. ($\Gamma \vdash$ *pc* ! *j* −*pes*[*t*♯*k*]→ *pc* ! *Suc j*) $\longrightarrow$ ($\Gamma \vdash$ *cs k* ! *j* −*es*[*t*♯*k*]→ *cs k* ! *Suc j*) $\wedge$ ($\forall\, k'$. $k' \neq k \longrightarrow$ *fst* (*cs k'* ! *j*) = *fst* (*cs k'* ! *Suc j*))) $\vee$
 fst (*pc* ! *j*) = *fst* (*pc* ! *Suc j*) $\wedge$ ($\forall\, k$. *fst* (*cs k* ! *j*) = *fst* (*cs k* ! *Suc j*))› **by** (*simp add*: *compat-tran-def*)
 from *this*[*rule-format*, *OF Suc-i-lt*] **have** *h*:
‹($\exists\, t\ k\ \Gamma$. $\Gamma \vdash$ *pc* ! *i* −*pes*[*t*♯*k*]→ *pc* ! *Suc i*) $\wedge$
 ($\forall\, k\ t\ \Gamma$. ($\Gamma \vdash$ *pc* ! *i* −*pes*[*t*♯*k*]→ *pc* ! *Suc i*) $\longrightarrow$ ($\Gamma \vdash$ *cs k* ! *i* −*es*[*t*♯*k*]→ *cs k* ! *Suc i*) $\wedge$ ($\forall\, k'$. $k' \neq k \longrightarrow$ *fst* (*cs k'* ! *i*) = *fst* (*cs k'* ! *Suc i*))) $\vee$
 fst (*pc* ! *i*) = *fst* (*pc* ! *Suc i*) $\wedge$ ($\forall\, k$. *fst* (*cs k* ! *i*) = *fst* (*cs k* ! *Suc i*))› **by** *blast*
 from *etran* **have** ‹$\neg$($\exists\, t\ k\ \Gamma$. $\Gamma \vdash$ *pc* ! *i* −*pes*[*t*♯*k*]→ *pc* ! *Suc i*)› **using** *no-pestran-to-self*
 by (*metis* (*mono-tags*, *lifting*) *etran-def etran-p-def mem-Collect-eq prod.simps*(*2*) *surjective-pairing*)
 with *h* **have** ‹$\forall\, k$. *fst* (*cs k* ! *i*) = *fst* (*cs k* ! *Suc i*)› **by** *blast*
 then show *?thesis* **by** *simp*
qed

lemma *conjoin-cpt*:
 assumes *pc*: ‹*pc* $\in$ *cpts* (*pestran* Γ)›
 assumes *conjoin*: ‹*pc* $\propto$ *cs*›
 shows ‹*cs k* $\in$ *cpts* (*estran* Γ)›
proof −

from *pc cpts-def′*[*of pc* ‹*pestran* Γ›] **have**
‹*pc* ≠ []› **and** *1*: ‹(∀ *i*. *Suc i* < *length pc* ⟶ (*pc* ! *i*, *pc* ! *Suc i*) ∈ *pestran* Γ ∨ *pc* ! *i* −*e*→ *pc* ! *Suc i*)›
by *auto*
from ‹*pc*≠[]› **have** ‹*length pc* ≠ *0*› **by** *simp*
then have ‹*length* (*cs k*) ≠ *0*› **using** *conjoin* **by** (*simp add*: *conjoin-def same-length-def*)
then have ‹*cs k* ≠ []› **by** *simp*
moreover have ‹∀ *i*. *Suc i* < *length* (*cs k*) ⟶ (*cs k* ! *i*) −*e*→ (*cs k* ! *Suc i*) ∨ (*cs k* ! *i*, *cs k* ! *Suc i*) ∈ *estran* Γ›
proof(*rule allI*, *rule impI*)
fix *i*
assume ‹*Suc i* < *length* (*cs k*)›
then have *Suc-i-lt*: ‹*Suc i* < *length pc*› **using** *conjoin conjoin-def same-length-def* **by** *metis*
from *1*[*rule-format*, *OF this*]
have *ctran-or-etran-par*: ‹(*pc* ! *i*, *pc* ! *Suc i*) ∈ *pestran* Γ ∨ *pc* ! *i* −*e*→ *pc* ! *Suc i*› **by** *assumption*
then show ‹*cs k* ! *i* −*e*→ *cs k* ! *Suc i* ∨ (*cs k* ! *i*, *cs k* ! *Suc i*) ∈ *estran* Γ›
proof
assume ‹(*pc* ! *i*, *pc* ! *Suc i*) ∈ *pestran* Γ›
then have ‹∃ *a k*. Γ ⊢ *pc*!*i* −*pes*[*a*♯*k*]→ *pc*!*Suc i*› **by** (*simp add*: *pestran-def*)
then obtain *a k′* **where** ‹Γ ⊢ *pc*!*i* −*pes*[*a*♯*k′*]→ *pc*!*Suc i*› **by** *blast*
from *conjoin-ctran*[*OF conjoin Suc-i-lt this*]
have *2*: ‹(Γ ⊢ *cs k′* ! *i* −*es*[*a*♯*k′*]→ *cs k′* ! *Suc i*) ∧ (∀ *k′a*. *k′a* ≠ *k′* ⟶ *cs k′a* ! *i* −*e*→ *cs k′a* ! *Suc i*)›
by *assumption*
show *?thesis*
proof(*cases* ‹*k′*=*k*›)
case *True*
then show *?thesis*
using *2* **apply** (*simp add*: *estran-def*)
apply(*rule disjI2*)
by *auto*
next
case *False*
then show *?thesis* **using** *2* **by** *simp*
qed
next
assume ‹*pc* ! *i* −*e*→ *pc* ! *Suc i*›
from *conjoin-etran*[*OF conjoin Suc-i-lt this*] **show** *?thesis*
apply−
apply (*rule disjI1*)
by *blast*
qed
qed
ultimately show ‹*cs k* ∈ *cpts* (*estran* Γ)› **using** *cpts-def′* **by** *blast*
qed

lemma *conjoin-cpt′*:

assumes *pc*: ‹*pc* ∈ *cpts-from* (*pestran* Γ) (*Ps*, *s0*)›
assumes *conjoin*: ‹*pc* ∝ *cs*›
shows ‹*cs k* ∈ *cpts-from* (*estran* Γ) (*Ps k*, *s0*)›
proof−
from *pc* **have** *pc-cpt*: ‹*pc* ∈ *cpts* (*pestran* Γ)› **and** *hd-pc*: ‹*hd pc* = (*Ps*, *s0*)› **by** *auto*
from *pc-cpt cpts-nonnil* **have** ‹*pc*≠[]› **by** *blast*
have *ck-cpt*: ‹*cs k* ∈ *cpts* (*estran* Γ)› **using** *conjoin-cpt*[*OF pc-cpt conjoin*] **by** *assumption*
moreover have ‹*hd* (*cs k*) = (*Ps k*, *s0*)›
proof−
from *ck-cpt cpts-nonnil* **have** ‹*cs k* ≠ []› **by** *blast*
from *conjoin conjoin-def* **have** ‹*same-spec pc cs*› **and** ‹*same-state pc cs*› **by** *blast*+
then show *?thesis* **using** *hd-pc* ‹*pc*≠[]› ‹*cs k* ≠ []›
apply(*simp add*: *same-spec-def same-state-def hd-conv-nth*)
apply(*erule allE*[**where** *x*=*k*])
apply(*erule allE*[**where** *x*=*0*])
apply *simp*
by (*simp add*: *prod-eqI*)
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *conjoin-same-length*:
‹*pc* ∝ *cs* ⟹ *length pc* = *length* (*cs k*)›
by (*simp add*: *conjoin-def same-length-def*)

lemma *conjoin-same-spec*:
‹*pc* ∝ *cs* ⟹ ∀ *k i*. *i* < *length pc* ⟶ *fst* (*pc*!*i*) *k* = *fst* (*cs k* ! *i*)›
by (*simp add*: *conjoin-def same-spec-def*)

lemma *conjoin-same-state*:
‹*pc* ∝ *cs* ⟹ ∀ *k i*. *i* < *length pc* ⟶ *snd* (*pc*!*i*) = *snd* (*cs k*!*i*)›
by (*simp add*: *conjoin-def same-state-def*)

lemma *conjoin-all-etran*:
assumes *conjoin*: ‹*pc* ∝ *cs*›
and *Suc-i-lt*: ‹*Suc i* < *length pc*›
and *all-etran*: ‹∀ *k*. *cs k* ! *i* −*e*→ *cs k* ! *Suc i*›
shows ‹*pc*!*i* −*e*→ *pc*!*Suc i*›
proof−
from *conjoin-same-spec*[*OF conjoin*]
have *same-spec*: ‹∀ *k i*. *i* < *length pc* ⟶ *fst* (*pc* ! *i*) *k* = *fst* (*cs k* ! *i*)› **by** *assumption*
from *same-spec*[*rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*]]
have *eq1*: ‹∀ *k*. *fst* (*pc* ! *i*) *k* = *fst* (*cs k* ! *i*)› **by** *blast*
from *same-spec*[*rule-format*, *OF Suc-i-lt*]
have *eq2*: ‹∀ *k*. *fst* (*pc* ! *Suc i*) *k* = *fst* (*cs k* ! *Suc i*)› **by** *blast*

```
  have ‹∀ k. fst (pc!i) k = fst (pc!Suc i) k›
  proof
    fix k
    from eq1[THEN spec[where x=k]] have 1: ‹fst (pc ! i) k = fst (cs k ! i)› by
assumption
    from eq2[THEN spec[where x=k]] have 2: ‹fst (pc!Suc i) k = fst (cs k ! Suc
i)› by assumption
    from 1 2 all-etran[THEN spec[where x=k]]
    show ‹fst (pc!i) k = fst (pc!Suc i) k› by simp
  qed
  then have ‹fst (pc!i) = fst (pc!Suc i)› by blast
  then show ?thesis by simp
qed

lemma conjoin-etran-k:
  assumes pc: ‹pc ∈ cpts (pestran Γ)›
    and conjoin: ‹pc ∝ cs›
    and Suc-i-lt: ‹Suc i < length pc›
    and etran: ‹cs k!i −e→ cs k!Suc i›
  shows ‹(pc!i −e→ pc!Suc i) ∨ (∃ k'. k'≠k ∧ (cs k'!i, cs k'!Suc i) ∈ estran Γ)›
proof(rule ccontr, clarsimp)
  assume neq: ‹fst (pc ! i) ≠ fst (pc ! Suc i)›
  assume 1: ‹∀ k'. k' = k ∨ (cs k' ! i, cs k' ! Suc i) ∉ estran Γ›
  have ‹∀ k'. cs k' ! i −e→ cs k' ! Suc i›
  proof
    fix k'
    show ‹cs k' ! i −e→ cs k' ! Suc i›
    proof(cases ‹k=k'›)
      case True
      then show ?thesis using etran by blast
    next
      case False
      with 1 have not-ctran: ‹(cs k' ! i, cs k' ! Suc i) ∉ estran Γ› by fast
     from conjoin-same-length[OF conjoin] Suc-i-lt have Suc-i-lt': ‹Suc i < length
(cs k')› by simp
     from conjoin-cpt[OF pc conjoin] have ‹cs k' ∈ cpts (estran Γ)› by assumption
      from ctran-or-etran[OF this Suc-i-lt'] not-ctran
      show ?thesis by blast
    qed
  qed
  from conjoin-all-etran[OF conjoin Suc-i-lt this]
  have ‹fst (pc!i) = fst (pc!Suc i)› by simp
  with neq show False by blast
qed

end

end
theory Validity imports Computation begin
```

definition *assume* :: $'s$ *set* ⇒ $('s \times 's)$ *set* ⇒ $('p \times 's)$ *list set* **where**
assume pre rely ≡ {*cpt. snd (hd cpt)* ∈ *pre* ∧ (∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i* −*e*→ *cpt*!(*Suc i*)) ⟶ (*snd* (*cpt*!*i*), *snd* (*cpt*!*Suc i*)) ∈ *rely*)}

definition *commit* :: $(('p \times 's) \times ('p \times 's))$ *set* ⇒ $'p$ *set* ⇒ $('s \times 's)$ *set* ⇒ $'s$ *set* ⇒ $('p \times 's)$ *list set* **where**
commit tran fin guar post ≡
{*cpt*. (∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i*, *cpt*!(*Suc i*)) ∈ *tran* ⟶ (*snd* (*cpt*!*i*), *snd* (*cpt*!(*Suc i*))) ∈ *guar*) ∧
(*fst* (*last cpt*) ∈ *fin* ⟶ *snd* (*last cpt*) ∈ *post*)}

definition *validity* :: $(('p \times 's) \times ('p \times 's))$ *set* ⇒ $'p$ *set* ⇒ $'p$ ⇒ $'s$ *set* ⇒ $('s \times 's)$ *set* ⇒ $('s \times 's)$ *set* ⇒ $'s$ *set* ⇒ *bool* **where**
validity tran fin P pre rely guar post ≡ ∀ *s0*. *cpts-from tran* (*P*,*s0*) ∩ *assume pre rely* ⊆ *commit tran fin guar post*

declare *validity-def*[*simp*]

lemma *commit-Cons-env*:
‹∀ *P s t*. ((*P*,*s*),(*P*,*t*)) ∉ *tran* ⟹
(*P*,*t*)#*cpt* ∈ *commit tran fin guar post* ⟹
(*P*,*s*)#(*P*,*t*)#*cpt* ∈ *commit tran fin guar post*›
apply (*simp add*: *commit-def*)
apply *clarify*
apply(*case-tac i*, *auto*)
done

lemma *commit-Cons-comp*:
‹(*Q*,*t*)#*cpt* ∈ *commit tran fin guar post* ⟹
((*P*,*s*),(*Q*,*t*)) ∈ *tran* ⟹
(*s*,*t*)∈*guar* ⟹
(*P*,*s*)#(*Q*,*t*)#*cpt* ∈ *commit tran fin guar post*›
apply (*simp add*: *commit-def*)
apply *clarify*
apply(*case-tac i*, *auto*)
done

lemma *cpts-from-assume-take*:
assumes *h*: *cpt* ∈ *cpts-from tran c* ∩ *assume pre rely*
assumes *i*: *i* ≠ *0*
shows *take i cpt* ∈ *cpts-from tran c* ∩ *assume pre rely*
proof
from *h* **have** ‹*cpt* ∈ *cpts-from tran c*› **by** *blast*
with *i cpts-from-take* **show** ‹*take i cpt* ∈ *cpts-from tran c*› **by** *blast*
next
from *h* **have** ‹*cpt* ∈ *assume pre rely*› **by** *blast*
with *i* **show** ‹*take i cpt* ∈ *assume pre rely*› **by** (*simp add*: *assume-def*)
qed

lemma *assume-snoc*:
  assumes *assume*: ‹*cpt* $\in$ *assume pre rely*›
    and *nonnil*: ‹*cpt* $\neq$ []›
    and *tran*: ‹$\neg$(*last cpt* $-e\rightarrow$ *c*)›
  shows ‹*cpt*@[*c*] $\in$ *assume pre rely*›
  using *assume nonnil* **apply** (*simp add*: *assume-def*)
proof
  fix *i*
  show ‹*i* < *length cpt* $\longrightarrow$
        fst ((*cpt* @ [*c*]) ! *i*) = *fst* ((*cpt* @ [*c*]) ! *Suc i*) $\longrightarrow$ (*snd* ((*cpt* @ [*c*]) ! *i*), *snd* ((*cpt* @ [*c*]) ! *Suc i*)) $\in$ *rely*›
  proof(*cases* ‹*Suc i* < *length cpt*›)
    case *True*
    then show *?thesis* **using** *assume nonnil*
      apply (*simp add*: *assume-def*)
      apply *clarify*
      apply(*erule allE*[**where** *x*=*i*])
      by (*simp add*: *nth-append*)
  next
    case *False*
    then show *?thesis*
      apply *clarsimp*
      apply(*subgoal-tac Suc i* = *length cpt*)
       apply *simp*
     apply (*smt Suc-lessD append-eq-conv-conj etran-def etran-p-def hd-drop-conv-nth last-snoc length-append-singleton lessI mem-Collect-eq prod.simps*(*2*) *take-hd-drop tran*)
      apply *simp*
      done
  qed
qed

lemma *commit-tl*:
  ‹(*P*,*s*)#(*Q*,*t*)#*cs* $\in$ *commit tran fin guar post* $\Longrightarrow$
   (*Q*,*t*)#*cs* $\in$ *commit tran fin guar post*›
  apply(*unfold commit-def*)
  apply(*unfold mem-Collect-eq*)
  apply *clarify*
  apply(*rule conjI*)
   apply *fastforce*
  by *simp*

lemma *assume-appendD*:
  ‹(*P*,*s*)#*cs*@*cs'* $\in$ *assume pre rely* $\Longrightarrow$ (*P*,*s*)#*cs* $\in$ *assume pre rely*›
  apply(*auto simp add*: *assume-def*)
  apply(*erule-tac x*=*i* **in** *allE*)
  apply *auto*
  apply (*metis append-Cons length-Cons lessI less-trans nth-append*)

by (*metis Suc-diff-1 Suc-lessD linorder-neqE-nat nth-Cons' nth-append zero-order*(*3*))

lemma *assume-appendD2*:
‹*cs*@*cs'* ∈ *assume pre rely* ⟹ ∀ *i*. *Suc i* < *length cs'* ⟶ *cs'*!*i* −*e*→ *cs'*!*Suc i* ⟶ (*snd*(*cs'*!*i*), *snd*(*cs'*!*Suc i*))∈*rely*›
apply(*auto simp add*: *assume-def*)
apply(*erule-tac x*=‹*length cs*+*i*› **in** *allE*)
apply *simp*
by (*metis add-Suc-right nth-append-length-plus*)

lemma *commit-append*:
assumes *cmt1*: ‹*cs* ∈ *commit tran fin guar mid*›
and *guar*: ‹(*snd* (*last cs*), *snd c'*) ∈ *guar*›
and *cmt2*: ‹*c'*#*cs'* ∈ *commit tran fin guar post*›
shows ‹*cs*@*c'*#*cs'* ∈ *commit tran fin guar post*›
apply(*auto simp add*: *commit-def*)
using *cmt1* **apply**(*simp add*: *commit-def*)
using *guar* **apply** (*metis Suc-lessI append-Nil2 append-eq-conv-conj hd-drop-conv-nth nth-append nth-append-length snoc-eq-iff-butlast take-hd-drop*)
using *cmt2* **apply**(*simp add*: *commit-def*)
apply(*case-tac* ‹*Suc i* < *length cs*›)
using *cmt1* **apply**(*simp add*: *commit-def*) **apply** (*simp add*: *nth-append*)
apply(*case-tac* ‹*Suc i* = *length cs*›)
using *guar* **apply** (*metis Cons-nth-drop-Suc drop-eq-Nil id-take-nth-drop last.simps last-appendR le-refl lessI less-irrefl-nat less-le-trans nth-append nth-append-length*)
using *cmt2* **apply**(*simp add*: *commit-def*) **apply** (*simp add*: *nth-append*)
using *cmt2* **apply**(*simp add*: *commit-def*) **.**

lemma *assume-append*:
assumes *asm1*: ‹*cs* ∈ *assume pre rely*›
and *asm2*: ‹∀ *i*. *Suc i* < *length* (*c'*#*cs'*) ⟶ (*c'*#*cs'*)!*i* −*e*→ (*c'*#*cs'*)!*Suc i* ⟶ (*snd*((*c'*#*cs'*)!*i*), *snd*((*c'*#*cs'*)!*Suc i*)) ∈ *rely*›
and *rely*: ‹*last cs* −*e*→ *c'* ⟶ (*snd* (*last cs*), *snd c'*) ∈ *rely*›
and ‹*cs*≠[]›
shows ‹*cs*@*c'*#*cs'* ∈ *assume pre rely*›
using *asm1* ‹*cs*≠[]›
apply(*auto simp add*: *assume-def*)
apply(*case-tac* ‹*Suc i* < *length cs*›)
apply(*erule-tac x*=*i* **in** *allE*)
apply (*metis Suc-lessD append-eq-conv-conj nth-take*)
apply(*case-tac* ‹*Suc i* = *length cs*›)
apply *simp*
using *rely* **apply**(*simp add*: *last-conv-nth*) **apply** (*metis diff-Suc-Suc diff-zero lessI nth-append*)
subgoal for *i*
using *asm2*[*THEN spec*[**where** *x*=‹*i*−*length cs*›]] **by** (*simp add*: *nth-append*)
done

end

4 Rely-guarantee Validity of PiCore Computations

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

4.1 Definitions Correctness Formulas

record $('p,'s)$ *rgformula* $=$
 Com :: $'p$
 Pre :: $'s$ *set*
 Rely :: $('s \times 's)$ *set*
 Guar :: $('s \times 's)$ *set*
 Post :: $'s$ *set*

locale *event-validity* $=$ *event-comp ptran fin-com*
for *ptran* :: $'Env \Rightarrow (('prog \times 's) \times 'prog \times 's)$ *set*
and *fin-com* :: $'prog$
$+$
fixes *prog-validity* :: $'Env \Rightarrow 'prog \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
 $(\text{-} \models \text{-}\ sat_p\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,60,0,0,0,0]\ 45)$

assumes *prog-validity-def*: $\Gamma \models P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow validity\ (ptran\ \Gamma)\ \{fin\text{-}com\}\ P\ pre\ rely\ guar\ post$

begin

definition *lift-state-set* :: ‹$'s\ set \Rightarrow ('s \times 'a)\ set$› **where**
 ‹$lift\text{-}state\text{-}set\ P \equiv \{(s,x).s{\in}P\}$›

definition *lift-state-pair-set* :: ‹$('s \times 's)\ set \Rightarrow (('s \times 'a) \times ('s \times 'a))set$› **where**
 ‹$lift\text{-}state\text{-}pair\text{-}set\ P \equiv \{((s,x),(t,y)).\ (s,t){\in}P\}$›

definition *es-validity* :: $'Env \Rightarrow ('l,'k,'s,'prog)\ esys \Rightarrow 's\ set \Rightarrow ('s \times 's)\ set \Rightarrow ('s \times 's)\ set \Rightarrow 's\ set \Rightarrow bool$
 $(\text{-} \models \text{-}\ sat_e\ [\text{-}, \text{-}, \text{-}, \text{-}]\ [60,0,0,0,0,0]\ 45)$ **where**
 $\Gamma \models es\ sat_e\ [pre,\ rely,\ guar,\ post] \equiv validity\ (estran\ \Gamma)\ \{fin\}\ es\ (lift\text{-}state\text{-}set\ pre)\ (lift\text{-}state\text{-}pair\text{-}set\ rely)\ (lift\text{-}state\text{-}pair\text{-}set\ guar)\ (lift\text{-}state\text{-}set\ post)$

declare *es-validity-def* [*simp*]

abbreviation ‹$par\text{-}fin \equiv \{Ps.\ \forall k.\ Ps\ k = fin\}$›

abbreviation ‹$par\text{-}com\ prgf \equiv \lambda k.\ Com\ (prgf\ k)$›

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_e [-, -, -, -] [*60*,*0*,*0*,*0*,*0*,*0*] *45*) **where**
‹$\Gamma \models Ps\ SAT_e$ [*pre*, *rely*, *guar*, *post*] $\equiv$ *validity* (*pestran* Γ) *par-fin Ps* (*lift-state-set pre*) (*lift-state-pair-set rely*) (*lift-state-pair-set guar*) (*lift-state-set post*)›

declare *pes-validity-def* [*simp*]

lemma *commit-Cons-env-p*:
‹(*P*,*t*)#*cpt* $\in$ *commit* (*ptran* Γ) {*fin-com*} *guar post* $\Longrightarrow$ (*P*,*s*)#(*P*,*t*)#*cpt* $\in$ *commit* (*ptran* Γ) {*fin-com*} *guar post*›
using *commit-Cons-env ptran-neq* **by** *metis*

lemma *commit-Cons-env-es*:
‹(*P*,*t*)#*cpt* $\in$ *commit* (*estran* Γ) {*EAnon fin-com*} *guar post* $\Longrightarrow$ (*P*,*s*)#(*P*,*t*)#*cpt* $\in$ *commit* (*estran* Γ) {*EAnon fin-com*} *guar post*›
using *commit-Cons-env no-estran-to-self'* **by** *metis*

lemma *cpt-from-ptran-star*:
assumes *h*: ‹$\Gamma \vdash$ (*P*, *s0*) $-c*\rightarrow$ (*fin-com*, *t*)›
shows ‹$\exists$ *cpt*. *cpt* $\in$ *cpts-from* (*ptran* Γ) (*P*, *s0*) $\cap$ *assume* {*s0*} {} $\wedge$ *last cpt* = (*fin-com*, *t*)›
proof−
from *h* **have** ‹((*P*,*s0*),(*fin-com*,*t*)) $\in$(*ptran* Γ) ^*› **by** (*simp add*: *ptrans-def*)
then show *?thesis*
proof(*induct*)
case *base*
show *?case*
proof
show ‹[(*P*,*s0*)] $\in$ *cpts-from* (*ptran* Γ) (*P*, *s0*) $\cap$ *assume* {*s0*} {} $\wedge$ *last* [(*P*,*s0*)] = (*P*, *s0*)›
apply (*simp add*: *assume-def*)
apply(*rule CptsOne*)
done
qed
next
case (*step c c'*)
from *step*(*3*) **obtain** *cpt* **where** *cpt*: ‹*cpt* $\in$ *cpts-from* (*ptran* Γ) (*P*, *s0*) $\cap$ *assume* {*s0*} {} $\wedge$ *last cpt* = *c*› **by** *blast*
with *step* **have** *tran*: ‹(*last cpt*, *c'*) $\in$ *ptran* Γ› **by** *simp*
then have *prog-neq*: ‹*fst* (*last cpt*) $\neq$ *fst c'*› **using** *ptran-neq*
by (*metis prod*.*exhaust-sel*)
from *cpt* **have** *cpt1*:‹*cpt* $\in$ *cpts* (*ptran* Γ)› **by** *simp*
then have *cpt-nonnil*: ‹*cpt* $\neq$ []› **using** *cpts-nonnil* **by** *blast*
show *?case*
proof
show ‹(*cpt*@[*c'*]) $\in$ *cpts-from* (*ptran* Γ) (*P*, *s0*) $\cap$ *assume* {*s0*} {} $\wedge$ *last* (*cpt*@[*c'*]) = *c'*›
proof

```
      show ‹cpt @ [c'] ∈ cpts-from (ptran Γ) (P, s0) ∩ assume {s0} {}›
      proof
        from cpt1 tran cpts-snoc-comp have ‹cpt@[c'] ∈ cpts (ptran Γ)› by blast
        moreover from cpt have ‹hd (cpt@[c']) = (P, s0)›
          using cpt-nonnil by fastforce
        ultimately show ‹cpt @ [c'] ∈ cpts-from (ptran Γ) (P, s0)› by fastforce
      next
        from cpt have assume: ‹cpt ∈ assume {s0} {}› by blast
        then have ‹snd (hd cpt) ∈ {s0}› using assume-def by blast
        then have 1: ‹snd (hd (cpt@[c'])) ∈ {s0}› using cpt-nonnil
          by (simp add: nth-append)
          from assume have assume2: ‹∀ i. Suc i < length cpt ⟶ (cpt!i −e→
cpt!(Suc i)) ⟶ (snd (cpt!i), snd (cpt!Suc i)) ∈ {}›
          by (simp add: assume-def)
      have 2: ‹∀ i. Suc i < length (cpt@[c']) ⟶ ((cpt@[c'])!i −e→ (cpt@[c'])!(Suc
i)) ⟶ (snd ((cpt@[c'])!i), snd ((cpt@[c'])!Suc i)) ∈ {}›
        proof
          fix i
          show ‹Suc i < length (cpt @ [c']) ⟶
        (cpt @ [c']) ! i −e→ (cpt @ [c']) ! Suc i ⟶ (snd ((cpt @ [c']) ! i), snd
((cpt @ [c']) ! Suc i)) ∈ {}›
          proof
            assume Suc-i: ‹Suc i < length (cpt @ [c'])›
            show ‹(cpt @ [c']) ! i −e→ (cpt @ [c']) ! Suc i ⟶ (snd ((cpt @ [c'])
! i), snd ((cpt @ [c']) ! Suc i)) ∈ {}›
            proof(cases ‹Suc i < length cpt›)
              case True
              then show ?thesis using assume2
                by (simp add: Suc-lessD nth-append)
            next
              case False
              with Suc-i have ‹Suc i = length cpt› by fastforce
              then have i: i = length cpt − 1 by fastforce
              find-theorems last length ?x − 1
              show ?thesis
              proof
                have eq1: ‹(cpt @ [c']) ! i = last cpt› using i cpt-nonnil
                  by (simp add: last-conv-nth nth-append)
                have eq2: ‹(cpt @ [c']) ! Suc i = c'› using Suc-i
                  by (simp add: ‹Suc i = length cpt›)
                assume ‹(cpt @ [c']) ! i −e→ (cpt @ [c']) ! Suc i›
                with eq1 eq2 have ‹last cpt −e→ c'› by simp
                with prog-neq have False by simp
               then show ‹(snd ((cpt @ [c']) ! i), snd ((cpt @ [c']) ! Suc i)) ∈ {}›
by blast
              qed
            qed
          qed
        qed
```

from *1 2 assume-def* **show** ‹*cpt @ [c']* ∈ *assume {s0} {}*› **by** *blast*
qed
next
show ‹*last (cpt @ [c']) = c'*› **by** *simp*
qed
qed
qed
qed

end

end

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

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

5.1 Proof System for Programs

definition *stable* :: *'a set* ⇒ *('a × 'a) set* ⇒ *bool* **where**
stable P R ≡ ∀ *s s'*. *s* ∈ *P* ⟶ *(s, s')* ∈ *R* ⟶ *s'* ∈ *P*

5.2 Rely-guarantee Condition

locale *event-hoare* = *event-validity ptran fin-com prog-validity*
for *ptran* :: *'Env* ⇒ *(('prog × 's) × 'prog × 's) set*
and *fin-com* :: *'prog*
and *prog-validity* :: *'Env* ⇒ *'prog* ⇒ *'s set* ⇒ *('s × 's) set* ⇒ *('s × 's) set* ⇒ *'s set* ⇒ *bool*
(- ⊨ - sat_p [-, -, -, -] [60,60,0,0,0,0] 45)
\+
fixes *rghoare-p* :: *'Env* ⇒ *['prog, 's set, ('s × 's) set, ('s × 's) set, 's set]* ⇒ *bool*
(- ⊢ - sat_p [-, -, -, -] [60,60,0,0,0,0] 45)
assumes *rgsound-p*: Γ ⊢ *P* sat_p *[pre, rely, guar, post]* ⟹ Γ ⊨ *P* sat_p *[pre, rely, guar, post]*
begin

lemma *stable-lift*:
‹*stable P R* ⟹ *stable (lift-state-set P) (lift-state-pair-set R)*›
by (*simp add*: *lift-state-set-def lift-state-pair-set-def stable-def*)

5.3 Proof System for Events

lemma *estran-anon-inv*:
assumes ‹*((EAnon p,s,x), (EAnon q,t,y))*∈*estran* Γ›
shows ‹*((p,s), (q,t))*∈*ptran* Γ›

```
  using assms apply-
  apply(simp add: estran-def)
  apply(erule exE)
  apply(erule estran-p.cases, auto)
  done

lemma unlift-cpt:
  assumes ‹cpt ∈ cpts-from (estran Γ) (EAnon p0, s0, x0)›
  shows ‹unlift-cpt cpt ∈ cpts-from (ptran Γ) (p0, s0)›
  using assms
proof(auto)
  assume a1: ‹cpt ∈ cpts (estran Γ)›
  assume a2: ‹hd cpt = (EAnon p0, s0, x0)›
  show ‹map (λ(p, s, -). (unlift-prog p, s)) cpt ∈ cpts (ptran Γ)›
    using a1 a2
  proof(induct arbitrary:p0 s0 x0)
    case (CptsOne P s)
    then show ?case by auto
  next
    case (CptsEnv P T cs S)
    obtain t y where T: ‹T=(t,y)› by fastforce
    from CptsEnv(3) T have ‹hd ((P,T)#cs) = (EAnon p0, t, y)› by simp
     from CptsEnv(2)[OF this] have ‹map (λa. case a of (p, s, -) ⇒ (unlift-prog
p, s)) ((P, T) # cs) ∈ cpts (ptran Γ)› .
    then show ?case by (auto simp add: case-prod-unfold)
  next
    case (CptsComp P S Q T cs)
    from CptsComp(4) have P: ‹P = EAnon p0› by simp
    obtain q where ptran: ‹((p0,fst S),(q,fst T))∈ptran Γ› and Q: ‹Q = EAnon
q›
    proof-
      assume a: ‹⋀q. ((p0, fst S), q, fst T) ∈ ptran Γ ⟹ Q = EAnon q ⟹
thesis›
      show thesis
        using CptsComp(1) apply(simp add: P estran-def)
        apply(erule exE)
        apply(erule estran-p.cases, auto)
        apply(rule a) apply simp+
        by (simp add: a)
    qed
    obtain t y where T: ‹T=(t,y)› by fastforce
    have ‹hd ((Q, T) # cs) = (EAnon q, t, y)› by (simp add: Q T)
     from CptsComp(3)[OF this] have *: ‹map (λa. case a of (p, s, uu-) ⇒
(unlift-prog p, s)) ((Q, T) # cs) ∈ cpts (ptran Γ)› .
    show ?case
      apply(simp add: case-prod-unfold)
      apply(rule cpts.CptsComp)
      using ptran Q apply(simp add: P)
      using * by (simp add: case-prod-unfold)
```

qed
next
assume *a1*: ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩
assume *a2*: ⟨*hd cpt* = (*EAnon p0*, *s0*, *x0*)⟩
show ⟨*hd* (*map* (λ(*p*, *s*, -). (*unlift-prog p*, *s*)) *cpt*) = (*p0*, *s0*)⟩
by (*simp add*: *hd-map*[*OF cpts-nonnil*[*OF a1*]] *case-prod-unfold a2*)
qed

theorem *Anon-sound*:
assumes *h*: ⟨Γ ⊢ *p* sat_p [*pre*, *rely*, *guar*, *post*]⟩
shows ⟨Γ ⊨ *EAnon p* sat_e [*pre*, *rely*, *guar*, *post*]⟩
proof−
from *h* **have** Γ ⊨ *p* sat_p [*pre*, *rely*, *guar*, *post*] **using** *rgsound-p* **by** *blast*
then have ⟨*validity* (*ptran* Γ) {*fin-com*} *p pre rely guar post*⟩ **using** *prog-validity-def* **by** *simp*
then have *p-valid*[*rule-format*]: ⟨∀ *S0*. *cpts-from* (*ptran* Γ) (*p*,*S0*) ∩ *assume pre rely* ⊆ *commit* (*ptran* Γ) {*fin-com*} *guar post*⟩ **using** *validity-def* **by** *fast*

let *?pre* = ⟨*lift-state-set pre*⟩
let *?rely* = ⟨*lift-state-pair-set rely*⟩
let *?guar* = ⟨*lift-state-pair-set guar*⟩
let *?post* = ⟨*lift-state-set post*⟩
have ⟨∀ *S0*. *cpts-from* (*estran* Γ) (*EAnon p*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*EAnon fin-com*} *?guar ?post*⟩
proof
fix *S0*
show ⟨*cpts-from* (*estran* Γ) (*EAnon p*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*EAnon fin-com*} *?guar ?post*⟩
proof
fix *cpt*
assume *h1*: ⟨*cpt* ∈ *cpts-from* (*estran* Γ) (*EAnon p*, *S0*) ∩ *assume ?pre ?rely*⟩
from *h1* **have** *cpt*: ⟨*cpt* ∈ *cpts-from* (*estran* Γ) (*EAnon p*, *S0*)⟩ **by** *blast*
then have ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩ **by** *simp*
from *h1* **have** *cpt-assume*: ⟨*cpt* ∈ *assume ?pre ?rely*⟩ **by** *blast*
have *cpt-unlift*: ⟨*unlift-cpt cpt* ∈ *cpts-from* (*ptran* Γ) (*p*, *fst S0*) ∩ *assume pre rely*⟩
proof
show ⟨*unlift-cpt cpt* ∈ *cpts-from* (*ptran* Γ) (*p*, *fst S0*)⟩
using *unlift-cpt cpt surjective-pairing* **by** *metis*
next
from *cpt-assume* **have** ⟨*snd* (*hd* (*map* (λ(*p*, *s*, -). (*unlift-prog p*, *s*)) *cpt*)) ∈ *pre*⟩
by (*auto simp add*: *assume-def hd-map*[*OF cpts-nonnil*[*OF* ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩]] *case-prod-unfold lift-state-set-def*)
then show ⟨*unlift-cpt cpt* ∈ *assume pre rely*⟩
using *h1*
apply(*auto simp add*: *assume-def case-prod-unfold*)
apply(*erule-tac x*=*i* **in** *allE*)
apply(*simp add*: *lift-state-pair-set-def case-prod-unfold*)

by (*metis* (*mono-tags*, *lifting*) *Suc-lessD cpt cpts-from-anon′ fst-conv unlift-prog.simps*)
qed
with *p-valid* **have** *unlift-commit*: ‹*unlift-cpt cpt* ∈ *commit* (*ptran* Γ) {*fin-com*} *guar post*› **by** *blast*
show *cpt* ∈ *commit* (*estran* Γ) {*EAnon fin-com*} *?guar ?post*
proof(*auto simp add*: *commit-def*)
fix *i*
assume *a1*: ‹*Suc i* < *length cpt*›
assume *estran*: ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ›
from *cpts-from-anon′*[*OF cpt*, *rule-format*, *OF a1*[*THEN Suc-lessD*]]
obtain *p1 s1 x1* **where** *1*: ‹*cpt*!*i* = (*EAnon p1*,*s1*,*x1*)› **by** *blast*
from *cpts-from-anon′*[*OF cpt*, *rule-format*, *OF a1*]
obtain *p2 s2 x2* **where** *2*: ‹*cpt*!*Suc i* = (*EAnon p2*,*s2*,*x2*)› **by** *blast*
from *estran* **have** ‹((*p1*,*s1*), (*p2*,*s2*)) ∈ *ptran* Γ›
using *1 2 estran-anon-inv* **by** *fastforce*
then have ‹(*unlift-conf* (*cpt*!*i*), *unlift-conf* (*cpt*!*Suc i*)) ∈ *ptran* Γ›
by (*simp add*: *1 2*)
then have ‹(*fst* (*snd* (*cpt*!*i*)), *fst* (*snd* (*cpt*!*Suc i*))) ∈ *guar*› **using** *unlift-commit*
apply(*simp add*: *commit-def case-prod-unfold*)
apply *clarify*
apply(*erule allE*[**where** *x*=*i*])
using *a1* **by** *blast*
then show ‹(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *lift-state-pair-set guar*›
by (*simp add*: *lift-state-pair-set-def case-prod-unfold*)
next
assume *a1*: ‹*fst* (*last cpt*) = *fin*›
from *cpt cpts-nonnil* **have** ‹*cpt*≠[]› **by** *auto*
have ‹*fst* (*last* (*map* (λ*p*. (*unlift-prog* (*fst p*), *fst* (*snd p*))) *cpt*)) = *fin-com*›
by (*simp add*: *last-map*[*OF* ‹*cpt*≠[]›] *a1*)
then have ‹*snd* (*last* (*map* (λ*p*. (*unlift-prog* (*fst p*), *fst* (*snd p*))) *cpt*)) ∈ *post*› **using** *unlift-commit*
by (*simp add*: *commit-def case-prod-unfold*)
then show ‹*snd* (*last cpt*) ∈ *lift-state-set post*›
by (*simp add*: *last-map*[*OF* ‹*cpt*≠[]›] *lift-state-set-def case-prod-unfold*)
qed
qed
qed
then have ‹*validity* (*estran* Γ) {*EAnon fin-com*} (*EAnon p*) *?pre ?rely ?guar ?post*›
by (*subst validity-def*, *assumption*)
then show *?thesis*
by (*subst es-validity-def*, *assumption*)
qed

type-synonym *′a tran* = ‹*′a* × *′a*›

inductive-cases *estran-from-basic*: ‹Γ ⊢ (*EBasic ev*, *s*) −*es*[*a*]→ (*es*, *t*)›

lemma *assume-tl-comp*:
⟨$(P,\ s)\ \#\ (P,\ t)\ \#\ cs \in$ *assume pre rely* $\Longrightarrow$
stable pre rely $\Longrightarrow$
$(P,\ t)\ \#\ cs \in$ *assume pre rely*⟩
apply (*simp add*: *assume-def*)
apply *clarify*
apply(*rule conjI*)
apply(*erule-tac x=0* **in** *allE*)
apply(*simp add*: *stable-def*)
apply *auto*
done

lemma *assume-tl-env*:
assumes ⟨$(P,s)\#(Q,s)\#cs \in$ *assume pre rely*⟩
shows ⟨$(Q,s)\#cs \in$ *assume pre rely*⟩
using *assms*
apply(*clarsimp simp add*: *assume-def*)
apply(*erule-tac x*=⟨*Suc i*⟩ **in** *allE*)
by *auto*

lemma *Basic-sound*:
assumes *h*: ⟨$\Gamma \vdash$ *body* (*ev*::(′*l*,′*s*,′*prog*)*event*) sat_p [*pre* ∩ *guard ev*, *rely*, *guar*, *post*]⟩
and *stable*: ⟨*stable pre rely*⟩
and *guar-refl*: ⟨$\forall s.\ (s,\ s) \in$ *guar*⟩
shows ⟨$\Gamma \models$ *EBasic ev* sat_e [*pre*, *rely*, *guar*, *post*]⟩
proof−
let *?pre* = ⟨*lift-state-set pre*⟩
let *?rely* = ⟨*lift-state-pair-set rely*⟩
let *?guar* = ⟨*lift-state-pair-set guar*⟩
let *?post* = ⟨*lift-state-set post*⟩

from *stable* **have** *stable′*: ⟨*stable ?pre ?rely*⟩
by (*simp add*: *lift-state-set-def lift-state-pair-set-def stable-def*)

from *h Anon-sound* **have**
⟨$\Gamma \models$ *EAnon* (*body ev*) sat_e [*pre* ∩ *guard ev*, *rely*, *guar*, *post*]⟩ **by** *blast*
then have *es-valid*:
⟨∀ *S0*. *cpts-from* (*estran* Γ) (*EAnon* (*body ev*), *S0*) ∩ *assume* (*lift-state-set* (*pre* ∩ *guard ev*)) *?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*⟩
using *es-validity-def* **by** (*simp*)

have ⟨∀ *S0*. *cpts-from* (*estran* Γ) (*EBasic ev*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*⟩
proof
fix *S0*
show ⟨*cpts-from* (*estran* Γ) (*EBasic ev*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*⟩

proof
fix *cpt*
assume *cpt*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (EBasic\ ev,\ S0) \cap assume\ ?pre\ ?rely$›
then have *cpt-nonnil*: ‹$cpt \neq []$› **using** *cpts-nonnil* **by** *auto*
then have *cpt-Cons*: $cpt = hd\ cpt\ \#\ tl\ cpt$ **using** *hd-Cons-tl* **by** *simp*
let $?c0 = hd\ cpt$
from *cpt* **have** *fst-c0*: $fst\ (hd\ cpt) = EBasic\ ev$ **by** *auto*
from *cpt* **have** *cpt1*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (EBasic\ ev,\ S0)$› **by** *blast*
then have *cpt1-1*: ‹$cpt \in cpts\ (estran\ \Gamma)$› **using** *cpts-from-def* **by** *blast*
from *cpt* **have** *cpt-assume*: ‹$cpt \in assume\ ?pre\ ?rely$› **by** *blast*

show ‹$cpt \in commit\ (estran\ \Gamma)\ \{fin\}\ ?guar\ ?post$›
using *cpt1-1 cpt*
proof(*induct arbitrary*:*S0*)
case (*CptsOne P S*)
then have ‹$(P,S) = (EBasic\ ev,\ S0)$› **by** *simp*
then show *?case* **by** (*simp add*: *commit-def*)
next
case (*CptsEnv P T cs S*)
from *CptsEnv*(*3*) **have** *P-s*:
‹$(P,S) = (EBasic\ ev,\ S0)$› **by** *simp*
from *CptsEnv*(*3*) **have**
‹$(P,\ S)\ \#\ (P,\ T)\ \#\ cs \in assume\ ?pre\ ?rely$› **by** *blast*
with *assume-tl-comp stable′* **have** *assume′*:
‹$(P,T)\#cs \in assume\ ?pre\ ?rely$› **by** *fast*
have ‹$(P,\ T)\ \#\ cs \in cpts\text{-}from\ (estran\ \Gamma)\ (EBasic\ ev,\ T)$› **using** *CptsEnv*(*1*) *P-s* **by** *simp*
with *assume′* **have** ‹$(P,\ T)\ \#\ cs \in cpts\text{-}from\ (estran\ \Gamma)\ (EBasic\ ev,\ T) \cap assume\ ?pre\ ?rely$› **by** *blast*
with *CptsEnv*(*2*) **have** ‹$(P,\ T)\ \#\ cs \in commit\ (estran\ \Gamma)\ \{fin\}\ ?guar\ ?post$› **by** *blast*
then show *?case* **using** *commit-Cons-env-es* **by** *blast*
next
case (*CptsComp P S Q T cs*)
obtain *s0 x0* **where** *S0*: ‹$S0=(s0,x0)$› **by** *fastforce*
obtain *s x* **where** *S*: ‹$S=(s,x)$› **by** *fastforce*
obtain *t y* **where** *T*: ‹$T=(t,y)$› **by** *fastforce*
from *CptsComp*(*4*) **have** *P-s*:
‹$(P,S) = (EBasic\ ev,\ S0)$› **by** *simp*
from *CptsComp*(*4*) **have**
‹$(P,\ S)\ \#\ (Q,\ T)\ \#\ cs \in assume\ ?pre\ ?rely$› **by** *blast*
then have *pre*:
‹$snd\ (hd\ ((P,S)\#(Q,T)\#cs)) \in ?pre$›
and *rely*:
‹$\forall i.\ Suc\ i < length\ ((P,S)\#(Q,T)\#cs) \longrightarrow$
$(((P,S)\#(Q,T)\#cs)!i\ {-e\rightarrow}\ ((P,S)\#(Q,T)\#cs)!(Suc\ i)) \longrightarrow$
$(snd\ (((P,S)\#(Q,T)\#cs)!i),\ snd\ (((P,S)\#(Q,T)\#cs)!Suc\ i)) \in ?rely$›
using *assume-def* **by** *blast+*

from *pre* **have** ‹S ∈ ?pre› **by** *simp*
then have ‹s∈pre› **by** (*simp add*: *lift-state-set-def S*)
from *CptsComp*(*1*) **have** ‹∃ a k. Γ ⊢ (P,S) −es[a♯k]→ (Q,T)›
apply(*simp add*: *estran-def*)
apply(*erule exE*) **apply**(*rule-tac x*=‹Act a› **in** *exI*) **apply**(*rule-tac x*=‹K a› **in** *exI*)
apply(*subst*(*asm*) *actk-destruct*) **by** *assumption*
then obtain *a k* **where** ‹Γ ⊢ (P,S) −es[a♯k]→ (Q,T)› **by** *blast*
with *P-s* **have** *tran*: ‹Γ ⊢ (EBasic ev, S0) −es[a♯k]→ (Q,T)› **by** *simp*
then have *a*: ‹a = EvtEnt ev› **apply**− **apply**(*erule estran-from-basic*) **apply** *simp* **done**
from *tran* **have** *guard*: ‹s0 ∈ guard ev› **apply**− **apply**(*erule estran-from-basic*) **apply** (*simp add*: *S0*) **done**
from *tran* **have** *s0*=*t* **apply**− **apply**(*erule estran-from-basic*) **using** *a guard* **apply** (*simp add*: *T S0*) **done**
with *P-s S S0* **have** *s*=*t* **by** *simp*
with *guar-refl* **have** *guar*: ‹(s, t) ∈ guar› **by** *simp*

have ‹(Q,T)#cs ∈ cpts-from (estran Γ) (EAnon (body ev), T)›
proof−
have (*Q*,*T*)#*cs* ∈ *cpts* (*estran* Γ) **by** (*rule CptsComp*(*2*))
moreover have *Q* = *EAnon* (*body ev*) **using** *estran-from-basic* **using** *tran* **by** *blast*
ultimately show *?thesis* **by** *auto*
qed
moreover have ‹(Q,T)#cs ∈ assume (lift-state-set (pre ∩ guard ev)) ?rely›
proof−
have ‹fst (snd (hd ((Q,T)#cs))) ∈ (pre ∩ guard ev)›
proof
show ‹fst (snd (hd ((Q, T) # cs))) ∈ pre› **using** ‹s=t› ‹s∈pre› *T* **by** *simp*
next
show ‹fst (snd (hd ((Q, T) # cs))) ∈ guard ev› **using** ‹s0=t› *guard T* **by** *fastforce*
qed
then have ‹snd (hd ((Q,T)#cs)) ∈ lift-state-set (pre ∩ guard ev)› **using** *lift-state-set-def* **by** *fastforce*
moreover have
‹∀ i. Suc i < length ((Q,T)#cs) ⟶ (((Q,T)#cs)!i −e→ ((Q,T)#cs)!(Suc i)) ⟶ (snd (((Q,T)#cs)!i), snd (((Q,T)#cs)!Suc i)) ∈ ?rely›
using *rely* **by** *auto*
ultimately show *?thesis* **using** *assume-def* **by** *blast*
qed
ultimately have ‹(Q,T)#cs ∈ cpts-from (estran Γ) (EAnon (body ev), T) ∩ assume (lift-state-set (pre ∩ guard ev)) ?rely› **by** *blast*
then have ‹(Q,T)#cs ∈ commit (estran Γ) {fin} ?guar ?post› **using** *es-valid* **by** *blast*
then show *?case* **using** *commit-Cons-comp CptsComp*(*1*) *guar S T*

lift-state-set-def lift-state-pair-set-def **by** *fast*
      qed
    qed
  qed
  then show *?thesis* **by** *simp*
qed

inductive-cases *estran-from-atom*: ⟨Γ ⊢ (*EAtom ev*, *s*) −*es*[*a*]→ (*Q*, *t*)⟩

lemma *estran-from-atom′*:
  assumes *h*: ⟨Γ ⊢ (*EAtom ev*, *s*,*x*) −*es*[*a*♯*k*]→ (*Q*, *t*,*y*)⟩
  shows ⟨*a* = *AtomEvt ev* ∧ *s* ∈ *guard ev* ∧ Γ ⊢ (*body ev*, *s*) −*c*∗→ (*fin-com*, *t*)
∧ *Q* = *EAnon fin-com*⟩
  using *h estran-from-atom* **by** *blast*

lemma *last-sat-post*:
  assumes *t*: ⟨*t* ∈ *post*⟩
    and *cpt*: *cpt* = (*Q*,*t*)#*cs*
    and *etran*: ⟨∀ *i*. *Suc i* < *length cpt* ⟶ *cpt*!*i* −*e*→ *cpt*!*Suc i*⟩
    and *stable*: ⟨*stable post rely*⟩
    and *rely*: ⟨∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt*!*i* −*e*→ *cpt*!*Suc i*) ⟶ (*snd* (*cpt*!*i*),
snd (*cpt*!*Suc i*)) ∈ *rely*⟩
  shows ⟨*snd* (*last cpt*) ∈ *post*⟩
proof−
  from *etran rely* **have** *rely′*:
    ⟨∀ *i*. *Suc i* < *length cpt* ⟶ (*snd* (*cpt*!*i*), *snd* (*cpt*!*Suc i*)) ∈ *rely*⟩ **by** *auto*
  show *?thesis* **using** *cpt rely′*
  proof(*induct cs arbitrary*:*cpt rule*:*rev-induct*)
    case *Nil*
    then show *?case* **using** *t* **by** *simp*
  next
    case (*snoc x xs*)
    have
      ⟨∀ *i*. *Suc i* < *length* ((*Q*,*t*)#*xs*) ⟶ (*snd* (((*Q*,*t*)#*xs*) ! *i*), *snd* (((*Q*,*t*)#*xs*) !
Suc i)) ∈ *rely*⟩
    proof
      fix *i*
      show ⟨*Suc i* < *length* ((*Q*,*t*)#*xs*) ⟶ (*snd* (((*Q*,*t*)#*xs*) ! *i*), *snd* (((*Q*,*t*)#*xs*)
! *Suc i*)) ∈ *rely*⟩
      proof
        assume *Suc-i-lt*: ⟨*Suc i* < *length* ((*Q*,*t*)#*xs*)⟩
        then have *eq1*:
          ((*Q*,*t*)#*xs*)!*i* = *cpt*!*i* **using** *snoc*(*2*)
          by (*metis Suc-lessD butlast.simps*(*2*) *nth-butlast snoc-eq-iff-butlast*)
        from *Suc-i-lt snoc*(*2*) **have** *eq2*:
          ((*Q*,*t*)#*xs*)!*Suc i* = *cpt*!*Suc i*
          by (*simp add*: *nth-append*)
        have ⟨(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *rely*⟩
          using *Suc-i-lt snoc.prems*(*1*) *snoc.prems*(*2*) **by** *auto*

then show ‹(snd (((Q,t)#xs) ! i), snd (((Q,t)#xs) ! $Suc\ i$)) ∈ $rely$› **using** *eq1 eq2* **by** *simp*
qed
qed
then have *last-post*: ‹snd ($last$ ((Q, t) # xs)) ∈ $post$›
using *snoc.hyps* **by** *blast*
have ‹(snd ($last$ ((Q,t)#xs)), $snd\ x$) ∈ $rely$› **using** *snoc*(2,3)
by (*metis List.nth-tl append-butlast-last-id append-is-Nil-conv butlast.simps*(2) *butlast-snoc length-Cons length-append-singleton lessI list.distinct*(1) *list.sel*(3) *nth-append-length nth-butlast*)
with *last-post stable*
have $snd\ x \in post$ **by** (*simp add*: *stable-def*)
then show *?case* **using** *snoc*(2) **by** *simp*
qed
qed

lemma *Atom-sound*:
assumes *h*: ‹∀ V. Γ ⊢ *body* (ev::('l,'s,'$prog$)$event$) sat_p [pre ∩ $guard\ ev$ ∩ {V}, Id, $UNIV$, {s. (V,s)∈$guar$} ∩ $post$]›
and *stable-pre*: ‹*stable pre rely*›
and *stable-post*: ‹*stable post rely*›
shows ‹Γ ⊨ *EAtom ev* sat_e [pre, $rely$, $guar$, $post$]›
proof−
let *?pre* = ‹*lift-state-set pre*›
let *?rely* = ‹*lift-state-pair-set rely*›
let *?guar* = ‹*lift-state-pair-set guar*›
let *?post* = ‹*lift-state-set post*›

from *stable-pre* **have** *stable-pre'*: ‹*stable ?pre ?rely*›
by (*simp add*: *lift-state-set-def lift-state-pair-set-def stable-def*)
from *stable-post* **have** *stable-post'*: ‹*stable ?post ?rely*›
by (*simp add*: *lift-state-set-def lift-state-pair-set-def stable-def*)

from *h rgsound-p* **have**
‹∀ V. Γ ⊨ (*body ev*) sat_p [pre ∩ $guard\ ev$ ∩ {V}, Id, $UNIV$, {s. (V,s)∈$guar$} ∩ $post$]› **by** *blast*
then have *body-valid*:
‹∀ V $s0$. *cpts-from* (*ptran* Γ) ((*body ev*), $s0$) ∩ *assume* (pre ∩ $guard\ ev$ ∩ {V}) Id ⊆ *commit* (*ptran* Γ) {*fin-com*} $UNIV$ ({s. (V,s)∈$guar$} ∩ $post$)›
using *prog-validity-def* **by** (*meson validity-def*)

have ‹∀ $s0$. *cpts-from* (*estran* Γ) (*EAtom ev*, $s0$) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *S0*
show ‹*cpts-from* (*estran* Γ) (*EAtom ev*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *cpt*

assume *cpt*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*EAtom ev*, *S0*) ∩ *assume ?pre ?rely*›
then have *cpt1*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*EAtom ev*, *S0*)› **by** *blast*
then have *cpt1-1*: ‹*cpt* ∈ *cpts* (*estran* Γ)› **by** *simp*
from *cpt1* **have** *hd cpt* = (*EAtom ev*, *S0*) **by** *fastforce*
show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
using *cpt1-1 cpt*
proof(*induct arbitrary*:*S0*)
case (*CptsOne P S*)
then show *?case* **by** (*simp add*: *commit-def*)
next
case (*CptsEnv P T cs S*)
have ‹(*P*, *T*) # *cs* ∈ *cpts-from* (*estran* Γ) (*EAtom ev*, *T*) ∩ *assume ?pre ?rely*›
proof
from *CptsEnv*(*3*) **have** ‹(*P*, *S*) # (*P*, *T*) # *cs* ∈ *cpts-from* (*estran* Γ) (*EAtom ev*, *S0*)› **by** *blast*
then show ‹(*P*, *T*) # *cs* ∈ *cpts-from* (*estran* Γ) (*EAtom ev*, *T*)›
using *CptsEnv.hyps*(*1*) **by** *auto*
next
from *CptsEnv*(*3*) **have** ‹(*P*, *S*) # (*P*, *T*) # *cs* ∈ *assume ?pre ?rely*› **by** *blast*
with *assume-tl-comp stable-pre′* **show** ‹(*P*, *T*) # *cs* ∈ *assume ?pre ?rely*› **by** *fast*
qed
then have ‹(*P*, *T*) # *cs* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*› **using** *CptsEnv*(*2*) **by** *blast*
then show *?case* **using** *commit-Cons-env-es* **by** *blast*
next
case (*CptsComp P S Q T cs*)
obtain *s0 x0* **where** *S0*: ‹*S0*=(*s0*,*x0*)› **by** *fastforce*
obtain *s x* **where** *S*: ‹*S*=(*s*,*x*)› **by** *fastforce*
obtain *t y* **where** *T*: ‹*T*=(*t*,*y*)› **by** *fastforce*
from *CptsComp*(*1*) **have** ‹∃ *a k*. Γ ⊢ (*P*,*S*) −*es*[*a*♯*k*]→ (*Q*,*T*)›
apply− **apply**(*simp add*: *estran-def*) **apply**(*erule exE*) **apply**(*rule-tac x*=‹*Act a*› **in** *exI*) **apply**(*rule-tac x*=‹*K a*› **in** *exI*)
apply(*subst* (*asm*) *actk-destruct*) **by** *assumption*
then obtain *a k* **where** Γ ⊢ (*P*,*S*) −*es*[*a*♯*k*]→ (*Q*,*T*) **by** *blast*
moreover from *CptsComp*(*4*) **have** *P-s*: (*P*,*S*) = (*EAtom ev*, *S0*) **by** *force*
ultimately have *tran*: ‹Γ ⊢ (*EAtom ev*, *S0*) −*es*[*a*♯*k*]→ (*Q*,*T*)› **by** *simp*
then have *tran-inv*:
a = *AtomEvt ev* ∧ *s0* ∈ *guard ev* ∧ Γ ⊢ (*body ev*, *s0*) −*c*∗→ (*fin-com*, *t*) ∧ *Q* = *EAnon fin-com*
using *estran-from-atom′ S0 T* **by** *fastforce*
from *tran-inv* **have** *Q*: ‹*Q* = *EAnon fin-com*› **by** *blast*

from *CptsComp*(*4*) **have** *assume*: ‹(*P*, *S*) # (*Q*, *T*) # *cs* ∈ *assume ?pre ?rely*› **by** *blast*
from *assume* **have** *assume1*: ‹*snd* (*hd* ((*P*,*S*)#(*Q*,*T*)#*cs*)) ∈ *?pre*› **using**

assume-def **by** *blast*
then have ‹$S \in$ *?pre*› **by** *simp*
then have ‹*s*∈*pre*› **by** (*simp add*: *lift-state-set-def S*)
then have ‹*s0* ∈ *pre*› **using** *P-s S0 S* **by** *simp*
have ‹*s0* ∈ *guard ev*› **using** *tran-inv* **by** *blast*
have ‹*S0* ∈ {*S0*}› **by** *simp*

from *assume* **have** *assume2*:
‹∀ *i*. *Suc i* < *length* ((*P*,*S*)#(*Q*,*T*)#*cs*) ⟶ (((*P*,*S*)#(*Q*,*T*)#*cs*)!*i* −*e*→ ((*P*,*S*)#(*Q*,*T*)#*cs*)!(*Suc i*)) ⟶ (*snd* (((*P*,*S*)#(*Q*,*T*)#*cs*)!*i*), *snd* (((*P*,*S*)#(*Q*,*T*)#*cs*)!*Suc i*)) ∈ *?rely*›
using *assume-def* **by** *blast*
then have *assume2-tl*:
‹∀ *i*. *Suc i* < *length* ((*Q*,*T*)#*cs*) ⟶ ((((*Q*,*T*)#*cs*)!*i* −*e*→ ((*Q*,*T*)#*cs*)!(*Suc i*)) ⟶ (*snd* ((((*Q*,*T*)#*cs*)!*i*), *snd* ((((*Q*,*T*)#*cs*)!*Suc i*)) ∈ *?rely*›
by *fastforce*
from *tran-inv* **have** ‹Γ ⊢ (*body ev*, *s0*) −*c*∗→ (*fin-com*, *t*)› **by** *blast*
with *cpt-from-ptran-star* **obtain** *pcpt* **where** *pcpt*:
‹*pcpt* ∈ *cpts-from* (*ptran* Γ) (*body ev*, *s0*) ∩ *assume* {*s0*} {} ∧ *last pcpt* = (*fin-com*, *t*)› **by** *blast*

from *pcpt* **have**
‹*pcpt* ∈ *assume* {*s0*} {}› **by** *blast*
with ‹*s0*∈*pre*› ‹*s0*∈*guard ev*› **have** ‹*pcpt* ∈ *assume* (*pre* ∩ *guard ev* ∩ {*s0*}) *Id*›
by (*simp add*: *assume-def*)
with *pcpt body-valid* **have** *pcpt-commit*:
‹*pcpt* ∈ *commit* (*ptran* Γ) {*fin-com*} *UNIV* ({*s*. (*s0*, *s*) ∈ *guar*} ∩ *post*)›
by *blast*
then have ‹*t* ∈ ({*s*. (*s0*, *s*) ∈ *guar*} ∩ *post*)›
by (*simp add*: *pcpt commit-def*)
with *P-s S0 S T* **have** ‹(*s*,*t*)∈*guar*› **by** *simp*
from *pcpt-commit* **have**
‹*fst* (*last pcpt*) = *fin-com* ⟶ *snd* (*last pcpt*) ∈ ({*s*. (*s0*, *s*) ∈ *guar*} ∩ *post*)›
by (*simp add*: *commit-def*)
with *pcpt* **have** *t*:
‹*t* ∈ ({*s*. (*s0*, *s*) ∈ *guar*} ∩ *post*)› **by** *force*

have *rest-etran*:
‹∀ *i*. *Suc i* < *length* ((*Q*,*T*)#*cs*) ⟶ ((*Q*,*T*)#*cs*)!*i* −*e*→ ((*Q*,*T*)#*cs*)!*Suc i*› **using** *all-etran-from-fin*
using *CptsComp.hyps*(*2*) *Q* **by** *blast*
from *rest-etran assume2-tl* **have** *rely*:
‹∀ *i*. *Suc i* < *length* ((*Q*,*T*)#*cs*) ⟶ (*snd* (((*Q*, *T*) # *cs*) ! *i*), *snd* (((*Q*, *T*) # *cs*) ! *Suc i*)) ∈ *?rely*›
by *blast*
have *commit1*:
‹∀ *i*. *Suc i* < *length* ((*P*,*S*)#(*Q*,*T*)#*cs*) ⟶ (((*P*,*S*)#(*Q*,*T*)#*cs*)!*i*,

$((P,S)\#(Q,T)\#cs)!(Suc\ i)) \in (estran\ \Gamma) \longrightarrow (snd\ (((P,S)\#(Q,T)\#cs)!i),\ snd\ (((P,S)\#(Q,T)\#cs)!(Suc\ i))) \in ?guar\rangle$

proof

fix i

show $\langle Suc\ i < length\ ((P,S)\#(Q,T)\#cs) \longrightarrow (((P,S)\#(Q,T)\#cs)!i,\ ((P,S)\#(Q,T)\#cs)!(Suc\ i)) \in (estran\ \Gamma) \longrightarrow (snd\ (((P,S)\#(Q,T)\#cs)!i),\ snd\ (((P,S)\#(Q,T)\#cs)!(Suc\ i))) \in ?guar\rangle$

proof

assume $\langle Suc\ i < length\ ((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\rangle$

show $\langle(((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\ !\ i,\ ((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\ !\ Suc\ i) \in (estran\ \Gamma) \longrightarrow$
$(snd\ (((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\ !\ i),\ snd\ (((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\ !\ Suc\ i)) \in ?guar\rangle$

proof(*cases i*)

case *0*

then show *?thesis* **apply** *simp* **using** $\langle(s,t)\in guar\rangle$ *lift-state-pair-set-def S T* **by** *blast*

next

case $(Suc\ i')$

then show *?thesis* **apply** *simp* **apply**(*subst Q*)

using *no-ctran-from-fin*

using *CptsComp.hyps(2) Q* $\langle Suc\ i < length\ ((P,\ S)\ \#\ (Q,\ T)\ \#\ cs)\rangle$

by (*metis Suc-less-eq length-Cons nth-Cons-Suc*)

qed

qed

qed

have *commit2-aux*:

$\langle fst\ (last\ ((Q,T)\#cs)) = fin \longrightarrow snd\ (last\ ((Q,T)\#cs)) \in ?post\rangle$

proof

assume $\langle fst\ (last\ ((Q,\ T)\ \#\ cs)) = fin\rangle$

from *t* **have** *1*: $\langle T\in ?post\rangle$ **using** *T* **by** (*simp add*: *lift-state-set-def*)

from *last-sat-post*[*OF 1 refl rest-etran stable-post'*] *rely*

show $\langle snd\ (last\ ((Q,\ T)\ \#\ cs)) \in ?post\rangle$ **by** *blast*

qed

then have *commit2*:

$\langle fst\ (last\ ((P,S)\#(Q,T)\#cs)) = fin \longrightarrow snd\ (last\ ((P,S)\#(Q,T)\#cs)) \in ?post\rangle$ **by** *simp*

show *?case* **using** *commit1 commit2*

by (*simp add*: *commit-def*)

qed

qed

qed

then show *?thesis*

by (*simp*)

qed

theorem *conseq-sound*:

assumes *h*: $\langle\Gamma \models es\ sat_e\ [pre',\ rely',\ guar',\ post']\rangle$

and *pre*: $pre \subseteq pre'$

and *rely*: *rely* ⊆ *rely'*
and *guar*: *guar'* ⊆ *guar*
and *post*: *post'* ⊆ *post*
shows ‹Γ ⊨ *es sat*$_e$ [*pre*, *rely*, *guar*, *post*]›
proof−
let *?pre* = ‹*lift-state-set pre*›
let *?rely* = ‹*lift-state-pair-set rely*›
let *?guar* = ‹*lift-state-pair-set guar*›
let *?post* = ‹*lift-state-set post*›
let *?pre'* = ‹*lift-state-set pre'*›
let *?rely'* = ‹*lift-state-pair-set rely'*›
let *?guar'* = ‹*lift-state-pair-set guar'*›
let *?post'* = ‹*lift-state-set post'*›

from *h* **have**
valid: ‹∀ *S0*. *cpts-from* (*estran* Γ) (*es*, *S0*) ∩ *assume ?pre' ?rely'* ⊆ *commit* (*estran* Γ) {*fin*} *?guar' ?post'*›
by *auto*
have ‹∀ *S0*. *cpts-from* (*estran* Γ) (*es*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *S0*
show ‹*cpts-from* (*estran* Γ) (*es*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *cpt*
assume *cpt*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*es*, *S0*) ∩ *assume ?pre ?rely*›
then have *cpt1*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*es*, *S0*)› **by** *blast*
from *cpt* **have** *assume*: ‹*cpt* ∈ *assume ?pre ?rely*› **by** *blast*
then have *assume'*: ‹*cpt* ∈ *assume ?pre' ?rely'*›
apply(*simp add*: *assume-def lift-state-set-def lift-state-pair-set-def case-prod-unfold*)
using *pre rely* **by** *auto*
from *cpt1 assume'* **have** ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*es*, *S0*) ∩ *assume ?pre' ?rely'*› **by** *blast*
with *valid* **have** *commit*: *cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar' ?post'* **by** *blast*
then show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
apply(*simp add*: *commit-def lift-state-set-def lift-state-pair-set-def case-prod-unfold*)
using *guar post* **by** *auto*
qed
qed
then have ‹*validity* (*estran* Γ) {*fin*} *es ?pre ?rely ?guar ?post*› **using** *validity-def* **by** *metis*
then show *?thesis* **using** *es-validity-def* **by** *simp*
qed

primrec (*nonexhaustive*) *unlift-seq* **where**
‹*unlift-seq* (*ESeq P Q*) = *P*›

primrec *unlift-seq-esconf* **where**
‹*unlift-seq-esconf* (*P*,*s*) = (*unlift-seq P*, *s*)›

abbreviation ‹*unlift-seq-cpt* ≡ *map unlift-seq-esconf*›

lemma *split-seq*:
assumes *cpt*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*)›
and *not-all-seq*: ‹¬ *all-seq es2 cpt*›
shows
∃ *i S'*. *cpt*!*Suc i* = (*es2*, *S'*) ∧
Suc i < *length cpt* ∧
all-seq es2 (*take* (*Suc i*) *cpt*) ∧
unlift-seq-cpt (*take* (*Suc i*) *cpt*) @ [(*fin*,*S'*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∧
(*cpt*!*i*, *cpt*!*Suc i*)∈*estran* Γ ∧
(*unlift-seq-esconf* (*cpt*!*i*), (*fin*,*S'*))∈*estran* Γ
proof−
from *cpt* **have** *hd-cpt*: ‹*hd cpt* = (*ESeq es1 es2*, *S0*)› **by** *simp*
from *cpt* **have** ‹*cpt* ∈ *cpts* (*estran* Γ)› **by** *simp*
then have ‹*cpt* ∈ *cpts-es-mod* Γ› **using** *cpts-es-mod-equiv* **by** *blast*
then show *?thesis* **using** *hd-cpt not-all-seq*
proof(*induct arbitrary*:*S0 es1*)
case (*CptsModOne*)
then show *?case*
by (*simp add*: *all-seq-def*)
next
case (*CptsModEnv P t y cs s x*)
from *CptsModEnv*(*3*) **have** *1*: ‹*hd* ((*P*,*t*,*y*)#*cs*) = (*es1 NEXT es2*, *t*,*y*)› **by** *simp*
from *CptsModEnv*(*4*) **have** *2*: ‹¬ *all-seq es2* ((*P*,*t*,*y*)#*cs*)› **by** (*simp add*: *all-seq-def*)
from *CptsModEnv*(*2*)[*OF 1 2*] **obtain** *i S'* **where**
‹((*P*, *t*, *y*) # *cs*) ! *Suc i* = (*es2*, *S'*) ∧
Suc i < *length* ((*P*, *t*, *y*) # *cs*) ∧
all-seq es2 (*take* (*Suc i*) ((*P*, *t*, *y*) # *cs*)) ∧
map unlift-seq-esconf (*take* (*Suc i*) ((*P*, *t*, *y*) # *cs*)) @ [(*fin*, *S'*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *t*, *y*) ∧ (((*P*, *t*, *y*) # *cs*) ! *i*, ((*P*, *t*, *y*) # *cs*) ! *Suc i*) ∈ *estran* Γ ∧ (*unlift-seq-esconf* (((*P*, *t*, *y*) # *cs*) ! *i*), *fin*, *S'*) ∈ *estran* Γ›
by *blast*
then show *?case* **apply**−
apply(*rule exI*[**where** *x*=*Suc i*])
apply (*simp add*: *all-seq-def*)
apply(*rule conjI*)
apply(*rule CptsEnv*)
apply *fastforce*
apply(*rule conjI*)
using *CptsModEnv*(*3*) **apply** *simp*
by *argo*
next

case (*CptsModAnon*)
then show *?case* **by** *simp*
next
case (*CptsModAnon-fin*)
then show *?case* **by** *simp*
next
case (*CptsModBasic*)
then show *?case* **by** *simp*
next
case (*CptsModAtom*)
then show *?case* **by** *simp*
next
case (*CptsModSeq P s x a Q t y R cs*)
from *CptsModSeq*(*5*) **have** ‹(*s*,*x*) = *S0*› **and** ‹*R*=*es2*› **and** ‹*P*=*es1*› **by** *simp*+
from *CptsModSeq*(*5*) **have** *1*: ‹*hd* ((*Q NEXT R*, *t*,*y*) # *cs*) = (*Q NEXT es2*, *t*,*y*)› **by** *simp*
from *CptsModSeq*(*6*) **have** *2*: ‹¬ *all-seq es2* ((*Q NEXT R*, *t*,*y*) # *cs*)› **by** (*simp add*: *all-seq-def*)
from *CptsModSeq*(*4*)[*OF 1 2*] **obtain** *i S*′ **where**
‹((*Q NEXT R*, *t*, *y*) # *cs*) ! *Suc i* = (*es2*, *S*′) ∧
Suc i < *length* ((*Q NEXT R*, *t*, *y*) # *cs*) ∧
all-seq es2 (*take* (*Suc i*) ((*Q NEXT R*, *t*, *y*) # *cs*)) ∧
map unlift-seq-esconf (*take* (*Suc i*) ((*Q NEXT R*, *t*, *y*) # *cs*)) @ [(*fin*, *S*′)] ∈ *cpts-from* (*estran* Γ) (*Q*, *t*, *y*) ∧
(((*Q NEXT R*, *t*, *y*) # *cs*) ! *i*, ((*Q NEXT R*, *t*, *y*) # *cs*) ! *Suc i*) ∈ *estran* Γ ∧
(*unlift-seq-esconf* (((*Q NEXT R*, *t*, *y*) # *cs*) ! *i*), *fin*, *S*′) ∈ *estran* Γ›
by *blast*
then show *?case* **apply**−
apply(*rule exI*[**where** *x*=*Suc i*])
apply(*simp add*: *all-seq-def*)
apply(*rule conjI*)
apply(*rule CptsComp*)
apply(*simp add*: *estran-def*; *rule exI*)
apply(*rule CptsModSeq*(*1*))
apply *fast*
apply(*rule conjI*)
apply(*rule* ‹*P*=*es1*›)
apply(*rule conjI*)
apply(*rule* ‹(*s*,*x*) = *S0*›)
by *argo*
next
case (*CptsModSeq-fin Q s x a t y cs cs*′)
then show *?case*
apply−
apply(*rule exI*[**where** *x*=*0*])
apply (*simp add*: *all-seq-def*)
apply(*rule conjI*)
apply(*rule CptsComp*)

```
      apply(simp add: estran-def; rule exI; assumption)
     apply(rule CptsOne)
    apply(rule conjI)
     apply(simp add: estran-def; rule exI)
    using ESeq-fin apply blast
    apply(simp add: estran-def)
    apply(rule exI)
    by assumption
  next
    case (CptsModChc1)
    then show ?case by simp
  next
    case (CptsModChc2)
    then show ?case by simp
  next
    case (CptsModJoin1)
    then show ?case by simp
  next
    case (CptsModJoin2)
    then show ?case by simp
  next
    case (CptsModJoin-fin)
    then show ?case by simp
  next
    case (CptsModWhileTOnePartial)
    then show ?case by simp
  next
    case (CptsModWhileTOneFull)
    then show ?case by simp
  next
    case (CptsModWhileTMore)
    then show ?case by simp
  next
    case (CptsModWhileF)
    then show ?case by simp
  qed
qed

lemma all-seq-unlift:
  assumes all-seq: all-seq Q cpt
    and h: ‹cpt ∈ cpts-from (estran Γ) (ESeq P Q, S0) ∩ assume pre rely›
  shows ‹unlift-seq-cpt cpt ∈ cpts-from (estran Γ) (P, S0) ∩ assume pre rely›
proof
  from h have h1:
    ‹cpt ∈ cpts-from (estran Γ) (ESeq P Q, S0)› by blast
  then have cpt: ‹cpt ∈ cpts (estran Γ)› by simp
  with cpts-es-mod-equiv have cpt-mod: cpt ∈ cpts-es-mod Γ by auto
  from h1 have hd-cpt: ‹hd cpt = (ESeq P Q, S0)› by simp
  show ‹map unlift-seq-esconf cpt ∈ cpts-from (estran Γ) (P, S0)› using cpt-mod
```

```
hd-cpt all-seq
  proof(induct arbitrary:P S0)
    case (CptsModOne P s)
    then show ?case apply simp apply(rule CptsOne) done
  next
    case (CptsModEnv P1 t y cs s x)
     from CptsModEnv(3) have ‹hd ((P1, t,y) # cs) = (P  NEXT  Q, t,y)› by
simp
    moreover from CptsModEnv(4) have ‹all-seq Q ((P1, t,y) # cs)›
      apply− apply(unfold all-seq-def) apply auto done
    ultimately have ‹map unlift-seq-esconf ((P1, t,y) # cs) ∈ cpts-from (estran
Γ) (P, t,y)›
      using CptsModEnv(2) by blast
    moreover have (s,x)=S0 using CptsModEnv(3) by simp
    ultimately show ?case apply clarsimp apply(erule CptsEnv) done
  next
    case (CptsModAnon)
    then show ?case by simp
  next
    case (CptsModAnon-fin)
    then show ?case by simp
  next
    case (CptsModBasic)
    then show ?case by simp
  next
    case (CptsModAtom)
    then show ?case by simp
  next
    case (CptsModSeq P1 s x a Q1 t y R cs)
    from CptsModSeq(5) have ‹hd ((Q1  NEXT  R, t,y) # cs) = (Q1  NEXT  Q,
t,y)› by simp
    moreover from CptsModSeq(6) have ‹all-seq Q ((Q1  NEXT  R, t,y) # cs)›
      apply(unfold all-seq-def) by auto
   ultimately have ‹map unlift-seq-esconf ((Q1  NEXT  R, t,y) # cs) ∈ cpts-from
(estran Γ) (Q1, t,y)›
      using CptsModSeq(4) by blast
    moreover from CptsModSeq(5) have (s,x)=S0 and P1=P by simp-all
    ultimately show ?case apply (simp add: estran-def)
      apply(rule CptsComp) using CptsModSeq(1) by auto
  next
    case (CptsModSeq-fin)
    from CptsModSeq-fin(5) have False
      apply(auto simp add: all-seq-def)
      using seq-neq2 by metis
    then show ?case by blast
  next
    case (CptsModChc1)
    then show ?case by simp
  next
```

case (*CptsModChc2*)
then show *?case* **by** *simp*
next
case (*CptsModJoin1*)
then show *?case* **by** *simp*
next
case (*CptsModJoin2*)
then show *?case* **by** *simp*
next
case (*CptsModJoin-fin*)
then show *?case* **by** *simp*
next
case *CptsModWhileTOnePartial*
then show *?case* **by** *simp*
next
case *CptsModWhileTOneFull*
then show *?case* **by** *simp*
next
case *CptsModWhileTMore*
then show *?case* **by** *simp*
next
case *CptsModWhileF*
then show *?case* **by** *simp*
qed
next
from *h* **have** *h2*: *cpt* ∈ *assume pre rely* **by** *blast*
then have *a1*: ‹*snd* (*hd cpt*) ∈ *pre*› **by** (*simp add*: *assume-def*)
from *h2* **have** *a2*:
‹∀ *i*. *Suc i* < *length cpt* ⟶
fst ((*cpt* ! *i*)) = *fst* ((*cpt* ! *Suc i*)) ⟶
(*snd* ((*cpt* ! *i*)), *snd* ((*cpt* ! *Suc i*))) ∈ *rely*› **by** (*simp add*: *assume-def*)
from *h* **have** ‹*cpt* ∈ *cpts* (*estran* Γ)› **by** *fastforce*
with *cpts-nonnil* **have** *cpt-nonnil*: *cpt* ≠ [] **by** *blast*
show ‹*map unlift-seq-esconf cpt* ∈ *assume pre rely*›
apply (*simp add*: *assume-def*)
proof
show ‹*snd* (*hd* (*map unlift-seq-esconf cpt*)) ∈ *pre*› **using** *a1 cpt-nonnil*
by (*metis eq-snd-iff hd-map unlift-seq-esconf.simps*)
next
show ‹∀ *i*. *Suc i* < *length cpt* ⟶
fst (*unlift-seq-esconf* (*cpt* ! *i*)) = *fst* (*unlift-seq-esconf* (*cpt* ! *Suc i*)) ⟶
(*snd* (*unlift-seq-esconf* (*cpt* ! *i*)), *snd* (*unlift-seq-esconf* (*cpt* ! *Suc i*))) ∈
rely›
using *a2* **by** (*metis Suc-lessD all-seq all-seq-def fst-conv nth-mem prod.collapse*
snd-conv unlift-seq.simps unlift-seq-esconf.simps)
qed
qed

lemma *cpts-from-assume-snoc-fin*:

assumes *cpt*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*P*, *S0*) ∩ *assume pre rely*›
and *tran*: ‹(*last cpt*, (*fin*, *S1*)) ∈ (*estran* Γ)›
shows ‹*cpt* @ [(*fin*, *S1*)] ∈ *cpts-from* (*estran* Γ) (*P*, *S0*) ∩ *assume pre rely*›
proof
from *cpt* **have** *cpt-from*:
‹*cpt* ∈ *cpts-from* (*estran* Γ) (*P*,*S0*)› **by** *blast*
with *cpts-snoc-comp tran cpts-from-def* **show** ‹*cpt* @ [(*fin*, *S1*)] ∈ *cpts-from* (*estran* Γ) (*P*, *S0*)›
using *cpts-nonnil* **by** *fastforce*
next
from *cpt* **have** *cpt-assume*:
‹*cpt* ∈ *assume pre rely*› **by** *blast*
from *cpt* **have** *cpt-nonnil*:
‹*cpt* ≠ []› **using** *cpts-nonnil* **by** *fastforce*
from *tran ctran-imp-not-etran* **have** *not-etran*:
‹¬ *last cpt* −*e*→ (*fin*, *S1*)› **by** *fast*
show ‹*cpt* @ [(*fin*, *S1*)] ∈ *assume pre rely*›
using *assume-snoc cpt-assume cpt-nonnil not-etran* **by** *blast*
qed

lemma *unlift-seq-estran*:
assumes *all-seq*: ‹*all-seq Q cpt*›
and *cpt*: ‹*cpt* ∈ *cpts* (*estran* Γ)›
and *i*: ‹*Suc i* < *length cpt*›
and *tran*: ‹(*cpt*!*i*, *cpt*!*Suc i*) ∈ (*estran* Γ)›
shows ‹(*unlift-seq-cpt cpt* ! *i*, *unlift-seq-cpt cpt* ! *Suc i*) ∈ (*estran* Γ)›
proof−
let *?part* = ‹*drop i cpt*›
from *i* **have** *i'*: ‹*i* < *length cpt*› **by** *simp*
from *cpts-drop cpt i'* **have** ‹*?part* ∈ *cpts* (*estran* Γ)› **by** *blast*
with *cpts-es-mod-equiv* **have** *part-cpt*: ‹*?part* ∈ *cpts-es-mod* Γ› **by** *blast*
show *?thesis* **using** *part-cpt*
proof(*cases*)
case (*CptsModOne P s*)
then show *?thesis* **using** *i*
by (*metis Cons-nth-drop-Suc i' list.discI list.sel*(*3*))
next
case (*CptsModEnv P t y cs s x*)
with *tran* **have** ‹((*P*,*s*,*x*),(*P*,*t*,*y*)) ∈ (*estran* Γ)›
using *Cons-nth-drop-Suc i' nth-via-drop* **by** *fastforce*
then have *False* **apply** (*simp add*: *estran-def*)
using *no-estran-to-self* **by** *fast*
then show *?thesis* **by** *blast*
next
case (*CptsModAnon*)
from *CptsModAnon*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i' nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModAnon-fin*)

from *CptsModAnon-fin*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModBasic*)
from *CptsModBasic*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModAtom*)
from *CptsModAtom*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModSeq P1 s x a Q1 t y R cs*)
then have *eq1*:
‹*map unlift-seq-esconf cpt* ! *i* = (*P1*,*s*,*x*)›
by (*simp add*: *i'* *nth-via-drop*)
from *CptsModSeq* **have** *eq2*:
‹*map unlift-seq-esconf cpt* ! *Suc i* = (*Q1*,*t*,*y*)›
by (*metis Cons-nth-drop-Suc i i'* *list.sel*(*1*) *list.sel*(*3*) *nth-map unlift-seq.simps unlift-seq-esconf.simps*)
from *CptsModSeq*(*2*) *eq1 eq2* **show** *?thesis*
apply(*unfold estran-def*) **by** *auto*
next
case (*CptsModSeq-fin*)
from *CptsModSeq-fin*(*1*) *all-seq all-seq-def* **obtain** *P2* **where** ‹*Q* = *P2 NEXT Q*›
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc esys.inject*(*4*) *fst-conv i i'* *list.inject nth-mem*)
then show *?thesis* **using** *seq-neq2* **by** *metis*
next
case (*CptsModChc1*)
from *CptsModChc1*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModChc2*)
from *CptsModChc2*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModJoin1*)
from *CptsModJoin1*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case (*CptsModJoin2*)
from *CptsModJoin2*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next
case *CptsModJoin-fin*
from *CptsModJoin-fin*(*1*) *all-seq all-seq-def* **show** *?thesis*
using *i'* *nth-mem nth-via-drop* **by** *fastforce*
next

```
    case CptsModWhileTOnePartial
    with all-seq all-seq-def show ?thesis
      using i' nth-mem nth-via-drop by fastforce
  next
    case CptsModWhileTOneFull
    with all-seq all-seq-def show ?thesis
      using i' nth-mem nth-via-drop by fastforce
  next
    case CptsModWhileTMore
    with all-seq all-seq-def show ?thesis
      using i' nth-mem nth-via-drop by fastforce
  next
    case CptsModWhileF
    with all-seq all-seq-def show ?thesis
      using i' nth-mem nth-via-drop by fastforce
  qed
qed

lemma fin-imp-not-all-seq:
  assumes ‹fst (last cpt) = fin›
    and ‹cpt ≠ []›
  shows ‹¬ all-seq Q cpt›
  apply(unfold all-seq-def)
proof
  assume ‹∀ c∈set cpt. ∃ P. fst c = P NEXT Q›
  then obtain P where ‹fst (last cpt) = P NEXT Q›
    using assms(2) last-in-set by blast
  with assms(1) show False by simp
qed

lemma all-seq-guar:
  assumes all-seq: ‹all-seq es2 cpt›
     and h1': ‹∀ s0. cpts-from (estran Γ) (es1, s0) ∩ assume pre rely ⊆ commit
(estran Γ) {fin} guar post›
    and cpt: ‹cpt ∈ cpts-from (estran Γ) (ESeq es1 es2, s0) ∩ assume pre rely›
  shows ‹∀ i. Suc i < length cpt ⟶ (cpt ! i, cpt ! Suc i) ∈ (estran Γ) ⟶ (snd
(cpt ! i), snd (cpt ! Suc i)) ∈ guar›
proof−
  let ?cpt' = ‹unlift-seq-cpt cpt›
  from all-seq-unlift[of es2 cpt Γ es1 s0 pre rely] all-seq cpt have cpt':
    ‹?cpt' ∈ cpts-from (estran Γ) (es1, s0) ∩ assume pre rely› by blast
  with h1' have ‹?cpt' ∈ commit (estran Γ) {fin} guar post› by blast
  then have guar:
     ‹∀ i. Suc i < length ?cpt' ⟶ (?cpt'!i, ?cpt'!Suc i) ∈ (estran Γ) ⟶ (snd
(?cpt'!i), snd (?cpt'!Suc i)) ∈ guar›
    by (simp add: commit-def)
  show ?thesis
  proof
    fix i
```

from *guar* **have** *guar-i*: ‹*Suc i* < *length ?cpt′* ⟶ (*?cpt′*!*i*, *?cpt′*!*Suc i*) ∈ (*estran* Γ) ⟶ (*snd* (*?cpt′*!*i*), *snd* (*?cpt′*!*Suc i*)) ∈ *guar*› **by** *blast*
show ‹*Suc i* < *length cpt* ⟶ (*cpt* ! *i*, *cpt* ! *Suc i*) ∈ (*estran* Γ) ⟶ (*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *guar*› **apply** *clarify*
proof−
assume *i*: ‹*Suc i* < *length cpt*›
assume *tran*: ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ (*estran* Γ)›
from *cpt* **have** ‹*cpt* ∈ *cpts* (*estran* Γ)› **by** *force*
with *unlift-seq-estran*[*of es2 cpt* Γ *i*] *all-seq i tran* **have** *tran′*:
‹(*?cpt′*!*i*, *?cpt′*!*Suc i*) ∈ (*estran* Γ)› **by** *blast*
with *guar-i i* **show** ‹(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *guar*›
by (*metis* (*no-types*, *lifting*) *Suc-lessD length-map nth-map prod.collapse sndI unlift-seq-esconf.simps*)
qed
qed
qed

lemma *part1-cpt-assume*:
assumes *split*:
‹*cpt*!*Suc i* = (*es2*, *S*) ∧
Suc i < *length cpt* ∧
all-seq es2 (*take* (*Suc i*) *cpt*) ∧
unlift-seq-cpt (*take* (*Suc i*) *cpt*) @ [(*fin*,*S*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∧
(*unlift-seq-esconf* (*cpt*!*i*), (*fin*,*S*))∈*estran* Γ›
and *h1′*:
‹∀ *S0*. *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume pre rely* ⊆ *commit* (*estran* Γ) {*fin*} *guar mid*›
and *cpt*:
‹*cpt* ∈ *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*) ∩ *assume pre rely*›
shows ‹*unlift-seq-cpt* (*take* (*Suc i*) *cpt*)@[(*fin*,*S*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume pre rely*›
proof−
let *?part1* = ‹*take* (*Suc i*) *cpt*›
let *?part2* = ‹*drop* (*Suc i*) *cpt*›
let *?part1′* = ‹*unlift-seq-cpt ?part1*›
let *?part1″* = ‹*?part1′*@[(*fin*,*S*)]›

show ‹*?part1″* ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume pre rely*›
proof
show ‹*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*)›
using *split* **by** *blast*
next
from *cpt cpts-nonnil* **have** ‹*cpt*≠[]› **by** *auto*
then have ‹*take* (*Suc i*) *cpt* ≠ []› **by** *simp*
have *1*: ‹*snd* (*hd* (*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*))) ∈ *pre*›
apply(*simp add*: *hd-map*[*OF* ‹*take*(*Suc i*)*cpt*≠[]›])
using *cpt* **by** (*auto simp add*: *assume-def*)
show ‹*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)] ∈ *assume pre rely*›

apply(*auto simp add*: *assume-def*)
using *1* ‹*cpt*≠[]› **apply** *fastforce*
subgoal for *j*
proof(*cases j*=*i*)
case *True*
assume *contra*: ‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *j*) = *fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *Suc j*)›
from *split* **have** ‹*Suc i* < *length cpt*› **by** *argo*
have *1*: ‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *i*) ≠ *fin*›
proof−
from *split* **have** *tran*: ‹(*unlift-seq-esconf* (*cpt*!*i*), (*fin*,*S*))∈*estran* Γ› **by** *argo*
have ∗: ‹*i* < *length* (*take*(*Suc i*)*cpt*)›
by (*simp add*: ‹*Suc i* < *length cpt*›[*THEN Suc-lessD*])
have ‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *i*) ≠ *fin*›
apply(*simp add*: *nth-map*[*OF* ∗])
using *no-estran-from-fin'*[*OF tran*] **.**
then show *?thesis* **by** (*simp add*: ‹*Suc i* < *length cpt*›[*THEN Suc-lessD*] *nth-append*)
qed
have *2*: ‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *Suc i*) = *fin*›
using ‹*cpt*≠[]› ‹*Suc i* < *length cpt*›
by (*metis* (*no-types*, *lifting*) *Suc-leI Suc-lessD length-map length-take min.absorb2 nth-append-length prod.collapse prod.inject*)
from *contra* **have** *False* **using** *True 1 2* **by** *argo*
then show *?thesis* **by** *blast*
next
case *False*
assume *a2*: ‹*j*<*Suc i*›
with *False* **have** ‹*j*<*i*› **by** *simp*
from *split* **have** ‹*Suc i* < *length cpt*› **by** *argo*
from *split* **have** *all-seq*: ‹*all-seq es2* (*take* (*Suc i*) *cpt*)› **by** *argo*
have ∗: ‹*Suc j* < *length* (*take* (*Suc i*) *cpt*)›
using ‹*Suc i* < *length cpt*› ‹*j*<*i*› **by** *auto*
assume *a3*:
‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *j*) =
fst ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *Suc j*)›
then have
‹*fst* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *j*) =
fst ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *Suc j*)›
using ‹*j*<*i*› ‹*Suc i* < *length cpt*›
by (*smt Suc-lessD Suc-mono length-map length-take less-trans-Suc min-less-iff-conj nth-append*)
then have ‹*fst* (*unlift-seq-esconf* (*take* (*Suc i*) *cpt* ! *j*)) = *fst* (*unlift-seq-esconf* (*take* (*Suc i*) *cpt* ! *Suc j*))›
by (*simp add*: *nth-map*[*OF* ∗] *nth-map*[*OF* ∗[*THEN Suc-lessD*]])
then have ‹*fst* (*cpt*!*j*) = *fst* (*cpt*!*Suc j*)›

```
proof−
assume a: ‹fst (unlift-seq-esconf (take (Suc i) cpt ! j)) = fst (unlift-seq-esconf
(take (Suc i) cpt ! Suc j))›
    have 1: ‹take (Suc i) cpt ! j = cpt ! j›
      by (simp add: a2)
    have 2: ‹take (Suc i) cpt ! Suc j = cpt ! Suc j›
      by (simp add: ‹j<i›)
    obtain P1 S1 where 3: ‹cpt!j = (P1 NEXT es2, S1)›
      using all-seq apply(simp add: all-seq-def)
      by (metis ∗ 1 Suc-lessD nth-mem prod.collapse)
    obtain P2 S2 where 4: ‹cpt!Suc j = (P2 NEXT es2, S2)›
      using all-seq apply(simp add: all-seq-def)
      by (metis ∗ 2 nth-mem prod.collapse)
    from a have ‹fst (unlift-seq-esconf (cpt ! j)) = fst (unlift-seq-esconf (cpt
! Suc j))›
      by (simp add: 1 2)
    then show ?thesis by (simp add: 3 4)
  qed
  from cpt have ‹cpt ∈ assume pre rely› by blast
  then have ‹fst (cpt!j) = fst (cpt!Suc j) ⟹ (snd (cpt!j), snd (cpt!Suc
j))∈rely›
    apply(auto simp add: assume-def)
    apply(erule allE[where x=j])
    using ‹Suc i < length cpt› ‹j<i› by fastforce
  from this[OF ‹fst (cpt!j) = fst (cpt!Suc j)›]
  have ‹(snd ((map unlift-seq-esconf (take (Suc i) cpt)) ! j), snd ((map
unlift-seq-esconf (take (Suc i) cpt)) ! Suc j)) ∈ rely›
    apply(simp add: nth-map[OF ∗] nth-map[OF ∗[THEN Suc-lessD]])
    using ‹j<i› all-seq
   by (metis (no-types, lifting) Suc-mono a2 nth-take prod.collapse prod.inject
unlift-seq-esconf.simps)
  then show ?thesis
    by (metis (no-types, lifting) ∗ Suc-lessD length-map nth-append)
 qed
 done
qed
qed

lemma part2-assume:
  assumes split:
    ‹cpt!Suc i = (es2, S) ∧
     Suc i < length cpt ∧
     all-seq es2 (take (Suc i) cpt) ∧
     unlift-seq-cpt (take (Suc i) cpt) @ [(fin,S)] ∈ cpts-from (estran Γ) (es1, S0) ∧
     (unlift-seq-esconf (cpt!i), (fin,S))∈estran Γ›
    and h1':
    ‹∀S0. cpts-from (estran Γ) (es1, S0) ∩ assume pre rely ⊆ commit (estran Γ)
{fin} guar mid›
    and cpt:
```

⟨*cpt* ∈ *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*) ∩ *assume pre rely*⟩
shows ⟨*drop* (*Suc i*) *cpt* ∈ *assume mid rely*⟩
apply(*unfold assume-def*)
apply(*subst mem-Collect-eq*)
proof
let *?part1* = ⟨*take* (*Suc i*) *cpt*⟩
let *?part2* = ⟨*drop* (*Suc i*) *cpt*⟩
let *?part1'* = ⟨*unlift-seq-cpt ?part1*⟩
let *?part1''* = ⟨*?part1'*@[(*fin*,*S*)]⟩

have ⟨*?part1''* ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume pre rely*⟩
using *part1-cpt-assume*[*OF split h1' cpt*] **.**
with *h1'* **have** ⟨*?part1''* ∈ *commit* (*estran* Γ) {*fin*} *guar mid*⟩ **by** *blast*
then have ⟨*S*∈*mid*⟩
by (*auto simp add*: *commit-def*)
then show ⟨*snd* (*hd ?part2*) ∈ *mid*⟩
by (*simp add*: *split hd-drop-conv-nth*)
next
let *?part2* = ⟨*drop* (*Suc i*) *cpt*⟩
from *cpt* **have** ⟨*cpt* ∈ *assume pre rely*⟩ **by** *blast*
then have ⟨∀ *j*. *Suc j* < *length cpt* ⟶ *cpt*!*j* −*e*→ *cpt*!*Suc j* ⟶ (*snd* (*cpt*!*j*), *snd* (*cpt*!*Suc j*)) ∈ *rely*⟩ **by** (*simp add*: *assume-def*)
then show ⟨∀ *j*. *Suc j* < *length ?part2* ⟶ *?part2*!*j* −*e*→ *?part2*!*Suc j* ⟶ (*snd* (*?part2*!*j*), *snd* (*?part2*!*Suc j*)) ∈ *rely*⟩ **by** *simp*
qed

theorem *Seq-sound*:
assumes *h1*:
⟨Γ ⊨ *es1* sat_e [*pre*, *rely*, *guar*, *mid*]⟩
assumes *h2*:
⟨Γ ⊨ *es2* sat_e [*mid*, *rely*, *guar*, *post*]⟩
shows
⟨Γ ⊨ *ESeq es1 es2* sat_e [*pre*, *rely*, *guar*, *post*]⟩
proof−
let *?pre* = ⟨*lift-state-set pre*⟩
let *?rely* = ⟨*lift-state-pair-set rely*⟩
let *?guar* = ⟨*lift-state-pair-set guar*⟩
let *?post* = ⟨*lift-state-set post*⟩
let *?mid* = ⟨*lift-state-set mid*⟩

from *h1* **have** *h1'*:
⟨∀ *S0*. *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?mid*⟩
by (*simp*)
from *h2* **have** *h2'*:
⟨∀ *S0*. *cpts-from* (*estran* Γ) (*es2*, *S0*) ∩ *assume ?mid ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*⟩
by (*simp*)

have ‹∀ *S0*. *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *S0*
show ‹*cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *cpt*
assume *cpt*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*) ∩ *assume ?pre ?rely*›
from *cpt* **have** *cpt1*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*ESeq es1 es2*, *S0*)› **by** *blast*
then have *cpt-cpts*: ‹*cpt* ∈ *cpts* (*estran* Γ)› **by** *simp*
then have ‹*cpt* ≠ []› **using** *cpts-nonnil* **by** *auto*
from *cpt* **have** *hd-cpt*: ‹*hd cpt* = (*ESeq es1 es2*, *S0*)› **by** *simp*
from *cpt* **have** *cpt-assume*: ‹*cpt* ∈ *assume ?pre ?rely*› **by** *blast*
show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
apply (*simp add*: *commit-def*)
proof
show ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ ⟶ (*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *?guar*›
proof(*cases* ‹*all-seq es2 cpt*›)
case *True*
with *all-seq-guar h1′ cpt* **show** *?thesis* **by** *blast*
next
case *False*
with *split-seq*[*OF cpt1*] **obtain** *i S* **where** *split*:
‹*cpt* ! *Suc i* = (*es2*, *S*) ∧
Suc i < *length cpt* ∧
all-seq es2 (*take* (*Suc i*) *cpt*) ∧ *map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)] ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∧ (*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ ∧ (*unlift-seq-esconf* (*cpt* ! *i*), *fin*, *S*) ∈ *estran* Γ› **by** *blast*
let *?part1* = ‹*take* (*Suc i*) *cpt*›
let *?part1′* = ‹*unlift-seq-cpt ?part1*›
let *?part1″* = ‹*?part1′* @ [(*fin*,*S*)]›
let *?part2* = ‹*drop* (*Suc i*) *cpt*›
from *split* **have**
Suc-i-lt: ‹*Suc i* < *length cpt*› **and**
all-seq-part1: ‹*all-seq es2 ?part1*› **by** *argo*+
have *part1-cpt*:
‹*?part1* ∈ *cpts-from* (*estran* Γ) (*es1 NEXT es2*, *S0*) ∩ *assume ?pre ?rely*›
using *cpts-from-assume-take*[*OF cpt*, *of* ‹*Suc i*›] **by** *simp*
have *guar-part1*:
∀ *j*. *Suc j* < *length ?part1* ⟶ (*?part1*!*j*, *?part1*!*Suc j*)∈(*estran* Γ) ⟶ (*snd* (*?part1*!*j*), *snd* (*?part1*!*Suc j*))∈*?guar*
using *all-seq-guar all-seq-part1 h1′ part1-cpt* **by** *blast*
have *guar-part2*:
‹∀ *j*. *Suc j* < *length ?part2* ⟶ (*?part2*!*j*, *?part2*!*Suc j*)∈(*estran* Γ) ⟶ (*snd* (*?part2*!*j*), *snd* (*?part2*!*Suc j*))∈*?guar*›

proof−
from *part2-assume*[*OF* - *h1′ cpt*] *split* **have** ‹*?part2* ∈ *assume ?mid ?rely*› **by** *blast*
moreover from *cpts-drop cpt cpts-from-def split* **have** *?part2* ∈ *cpts* (*estran* Γ) **by** *blast*
moreover from *split* **have** ‹*hd ?part2* = (*es2*, *S*)› **by** (*simp add*: *hd-conv-nth*)
ultimately have ‹*?part2* ∈ *cpts-from* (*estran* Γ) (*es2*,*S*) ∩ *assume ?mid ?rely*› **by** *fastforce*
with *h2′* **have** ‹*?part2* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*› **by** *blast*
then show *?thesis* **by** (*simp add*: *commit-def*)
qed
have *guar-tran*:
‹(*snd* (*last ?part1*), *snd* (*hd ?part2*))∈*?guar*›
proof−
have ‹(*snd* (*?part1″*!*i*), *snd* (*?part1″*!*Suc i*))∈*?guar*›
proof−
have *part1″-cpt-asm*: ‹*?part1″* ∈ *cpts-from* (*estran* Γ) (*es1*, *S0*) ∩ *assume ?pre ?rely*›
using *part1-cpt-assume*[*of cpt i es2 S* Γ *es1 S0*, *OF* - *h1′ cpt*] *split* **by** *blast*
from *split* **have** *tran*: ‹(*unlift-seq-esconf* (*cpt* ! *i*), *fin*, *S*) ∈ *estran* Γ› **by** *argo*
have ‹(*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *i* = (*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *i*›
using ‹*Suc i* < *length cpt*› **by** (*simp add*: *nth-append*)
moreover have ‹(*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *i* = *unlift-seq-esconf* (*cpt* ! *i*)›
proof−
have ∗: ‹*i* < *length* (*take* (*Suc i*) *cpt*)› **using** ‹*Suc i* < *length cpt*› **by** *simp*
show *?thesis* **by** (*simp add*: *nth-map*[*OF* ∗])
qed
ultimately have *1*: ‹(*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *i* = (*unlift-seq-esconf* (*cpt*!*i*))› **by** *simp*
have *2*: ‹(*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *Suc i* = (*fin*, *S*)›
using ‹*Suc i* < *length cpt*›
by (*metis* (*no-types*, *lifting*) *length-map length-take min.absorb2 nat-less-le nth-append-length*)
from *tran* **have** *tran′*: ‹((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *i*, (*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *Suc i*) ∈ *estran* Γ›
by (*simp add*: *1 2*)
from *h1′ part1″-cpt-asm* **have** ‹*?part1″* ∈ *commit* (*estran* Γ) {*fin*} (*lift-state-pair-set guar*) (*lift-state-set mid*)›
by *blast*
then show *?thesis*
apply(*auto simp add*: *commit-def*)

apply(*erule allE*[**where** x=i])
using ‹*Suc i* < *length cpt*› *tran'* **by** *linarith*
qed
moreover have ‹*snd* (*?part1''*!*i*) = *snd* (*last ?part1*)›
proof−
have *1*: ‹*snd* (*last* (*take* (*Suc i*) *cpt*)) = *snd* (*cpt*!*i*)› **using** *Suc-i-lt*
by (*simp add*: *last-take-Suc*)
have *2*: ‹*snd* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*) @ [(*fin*, *S*)]) ! *i*) = *snd* ((*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) ! *i*)›
using *Suc-i-lt*
by (*simp add*: *nth-append*)
have *3*: ‹*i* < *length* (*take* (*Suc i*) *cpt*)› **using** *Suc-i-lt* **by** *simp*
show *?thesis*
apply (*simp add*: *1 2 nth-map*[*OF 3*])
apply(*subst surjective-pairing*[*of* ‹*cpt*!*i*›])
apply(*subst unlift-seq-esconf.simps*)
by *simp*
qed
moreover have ‹*snd* (*?part1''*!*Suc i*) = *snd* (*hd ?part2*)›
proof−
have ‹*snd* (*?part1''*!*Suc i*) = *S*›
proof−
have ‹*length* (*map unlift-seq-esconf* (*take* (*Suc i*) *cpt*)) = *Suc i*› **using** *Suc-i-lt* **by** *simp*
then show *?thesis* **by** (*simp add*: *nth-via-drop*)
qed
moreover have ‹*snd* (*hd ?part2*) = *S*› **using** *split* **by** (*simp add*: *hd-conv-nth*)
ultimately show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
show *?thesis*
proof
fix j
show ‹*Suc j* < *length cpt* ⟶ (*cpt* ! *j*, *cpt* ! *Suc j*) ∈ *estran* Γ ⟶ (*snd* (*cpt* ! *j*), *snd* (*cpt* ! *Suc j*)) ∈ *?guar*›
proof(*cases* ‹*j*<*i*›)
case *True*
then show *?thesis* **using** *guar-part1* **by** *simp*
next
case *False*
then show *?thesis*
proof(*cases* ‹*j*=*i*›)
case *True*
then show *?thesis* **using** *guar-tran*
by (*metis Suc-lessD hd-drop-conv-nth last-take-Suc*)
next
case *False*

```
            with ‹¬j<i› have ‹j>i› by simp
            then obtain d where ‹Suc i + d = j›
              using Suc-leI le-Suc-ex by blast
            then show ?thesis using guar-part2[THEN spec, of d] by simp
          qed
        qed
      qed
    qed
  next
    show ‹fst (last cpt) = fin ⟶ snd (last cpt) ∈ ?post›
    proof
      assume fin: ‹fst (last cpt) = fin›
      then have
        ‹¬ all-seq es2 cpt›
        using fin-imp-not-all-seq ‹cpt≠[]› by blast

      with split-seq[OF cpt1] obtain i S where split:
        ‹cpt ! Suc i = (es2, S) ∧
      Suc i < length cpt ∧
       all-seq es2 (take (Suc i) cpt) ∧ map unlift-seq-esconf (take (Suc i) cpt)
@ [(fin, S)] ∈ cpts-from (estran Γ) (es1, S0) ∧ (cpt ! i, cpt ! Suc i) ∈ estran Γ ∧
(unlift-seq-esconf (cpt ! i), fin, S) ∈ estran Γ› by blast
      then have
        cpt-Suc-i: ‹cpt!(Suc i) = (es2, S)› and
        Suc-i-lt: ‹Suc i < length cpt› and
        all-seq: ‹all-seq es2 (take (Suc i) cpt)› by argo+
      let ?part2 = ‹drop (Suc i) cpt›
      from cpt-Suc-i have hd-part2:
        ‹hd ?part2 = (es2, S)›
        by (simp add: Suc-i-lt hd-drop-conv-nth)

      have ‹?part2 ∈ cpts (estran Γ)› using cpts-drop Suc-i-lt cpt1 by fastforce
      with cpt-Suc-i have ‹?part2 ∈ cpts-from (estran Γ) (es2, S)›
        using hd-drop-conv-nth Suc-i-lt by fastforce
      moreover have ‹?part2 ∈ assume ?mid ?rely›
        using part2-assume split h1' cpt by blast
      ultimately have ‹?part2 ∈ commit (estran Γ) {fin} ?guar ?post› using
h2' by blast
      then have fst (last ?part2) ∈ {fin} ⟶ snd (last ?part2) ∈ ?post
        by (simp add: commit-def)
     moreover from fin have fst (last ?part2) = fin using Suc-i-lt by fastforce
      ultimately have ‹snd (last ?part2) ∈ ?post› by blast
      then show ‹snd (last cpt) ∈ ?post› using Suc-i-lt by force
    qed
  qed
 qed
qed
then show ?thesis using es-validity-def validity-def
  by metis
```

qed

lemma *assume-choice1*:
‹$(P\ OR\ R,\ S)\ \#\ (Q,\ T)\ \#\ cs \in assume\ pre\ rely \Longrightarrow$
$\Gamma \vdash (P,S)\ -es[a]\rightarrow\ (Q,T) \Longrightarrow$
$(P,S)\#(Q,T)\#cs \in assume\ pre\ rely$›
apply(*simp add*: *assume-def*)
apply *clarify*
apply(*case-tac i*)
prefer *2*
apply *fastforce*
apply *simp*
using *no-estran-to-self surjective-pairing* **by** *metis*

lemma *assume-choice2*:
‹$(P\ OR\ R,\ S)\ \#\ (Q,\ T)\ \#\ cs \in assume\ pre\ rely \Longrightarrow$
$\Gamma \vdash (R,S)\ -es[a]\rightarrow\ (Q,T) \Longrightarrow$
$(R,S)\#(Q,T)\#cs \in assume\ pre\ rely$›
apply(*simp add*: *assume-def*)
apply *clarify*
apply(*case-tac i*)
prefer *2*
apply *fastforce*
apply *simp*
using *no-estran-to-self surjective-pairing* **by** *metis*

lemma *exists-least*:
‹$P\ (n::nat) \Longrightarrow \exists m.\ P\ m \wedge (\forall i{<}m.\ \neg\ P\ i)$›
using *exists-least-iff* **by** *auto*

lemma *choice-sound-aux1*:
‹$cpt' = map\ (\lambda(\text{-},\ s).\ (P,\ s))\ (take\ (Suc\ m)\ cpt)\ @\ drop\ (Suc\ m)\ cpt \Longrightarrow$
$Suc\ m < length\ cpt \Longrightarrow$
$\forall j{<}Suc\ m.\ fst\ (cpt'\ !\ j) = P$›
proof
fix *j*
assume *cpt'*: ‹$cpt' = map\ (\lambda(\text{-},\ s).\ (P,\ s))\ (take\ (Suc\ m)\ cpt)\ @\ drop\ (Suc\ m)\ cpt$›
assume *Suc-m-lt*: ‹$Suc\ m < length\ cpt$›
show ‹$j{<}Suc\ m \longrightarrow fst(cpt'!j) = P$›
proof
assume ‹$j{<}Suc\ m$›
with *cpt'* **have** ‹$cpt'!j = map\ (\lambda(\text{-},\ s).\ (P,\ s))\ (take\ (Suc\ m)\ cpt)\ !\ j$›
by (*metis* (*mono-tags*, *lifting*) *Suc-m-lt length-map length-take less-trans min-less-iff-conj nth-append*)
then have ‹$fst\ (cpt'!j) = fst\ (map\ (\lambda(\text{-},\ s).\ (P,\ s))\ (take\ (Suc\ m)\ cpt)\ !\ j)$› **by** *simp*
moreover have ‹$fst\ (map\ (\lambda(\text{-},\ s).\ (P,\ s))\ (take\ (Suc\ m)\ cpt)\ !\ j) = P$› **using** ‹$j{<}Suc\ m$›

by (*simp add*: *Suc-leI Suc-lessD Suc-m-lt case-prod-unfold min.absorb2*)
ultimately show ‹$fst(cpt'!j) = P$› **by** *simp*
qed
qed

theorem *Choice-sound*:
assumes *h1*:
‹$\Gamma \models P\ sat_e$ [*pre*, *rely*, *guar*, *post*]›
assumes *h2*:
‹$\Gamma \models Q\ sat_e$ [*pre*, *rely*, *guar*, *post*]›
shows
‹$\Gamma \models$ *EChc P Q* sat_e [*pre*, *rely*, *guar*, *post*]›
proof−
let *?pre* = ‹*lift-state-set pre*›
let *?rely* = ‹*lift-state-pair-set rely*›
let *?guar* = ‹*lift-state-pair-set guar*›
let *?post* = ‹*lift-state-set post*›

from *h1* **have** *h1'*:
‹∀ *S0*. *cpts-from* (*estran* Γ) (*P*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
by (*simp*)
from *h2* **have** *h2'*:
‹∀ *S0*. *cpts-from* (*estran* Γ) (*Q*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
by (*simp*)
have ‹∀ *S0*. *cpts-from* (*estran* Γ) (*EChc P Q*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *S0*
show ‹*cpts-from* (*estran* Γ) (*EChc P Q*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof
fix *cpt*
assume *cpt-from-assume*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*EChc P Q*, *S0*) ∩ *assume ?pre ?rely*›
then have *cpt*: ‹*cpt* ∈ *cpts* (*estran* Γ)›
and *hd-cpt*: ‹*hd cpt* = (*P OR Q*, *S0*)›
and *fst-hd-cpt*: *fst* (*hd cpt*) = *P OR Q*
and *cpt-assume*: ‹*cpt* ∈ *assume ?pre ?rely*› **by** *auto*
from *cpt cpts-nonnil* **have** ‹*cpt*≠[]› **by** *auto*
show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
proof(*cases* ‹∀ *i*. *Suc i* < *length cpt* ⟶ *cpt*!*i* −*e*→ *cpt*!*Suc i*›)
case *True*
then show *?thesis*
apply(*simp add*: *commit-def*)
proof
assume ‹∀ *i*. *Suc i* < *length cpt* ⟶ *fst* (*cpt* ! *i*) = *fst* (*cpt* ! *Suc i*)›
then show

```
      ‹∀ i. Suc i < length cpt ⟶ (cpt ! i, cpt ! Suc i) ∈ estran Γ ⟶
          (snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?guar›
      using no-estran-to-self'' by blast
  next
    assume ‹∀ i. Suc i < length cpt ⟶ fst (cpt ! i) = fst (cpt ! Suc i)›
    show ‹fst (last cpt) = fin ⟶ snd (last cpt) ∈ ?post›
    proof−
      have ‹∀ i<length cpt. fst (cpt ! i) = P OR Q›
        by (rule all-etran-same-prog[OF True fst-hd-cpt ‹cpt≠[]›])
       then have ‹fst (last cpt) = P OR Q› using last-conv-nth ‹cpt≠[]› by force
      then show ?thesis by simp
    qed
  qed
next
  case False
  then obtain i where 1: ‹Suc i < length cpt ∧ ¬ cpt ! i −e→ cpt ! Suc i› (is ?P i) by blast
  with exists-least[of ?P, OF 1] obtain m where 2: ‹?P m ∧ (∀ i<m. ¬?P i)› by blast
   from 2 have Suc-m-lt: ‹Suc m < length cpt› and all-etran: ‹∀ i<m. cpt!i −e→ cpt!Suc i› by simp-all
  from 2 have ‹¬ cpt!m −e→ cpt!Suc m› by blast
 then have ctran: ‹(cpt!m, cpt!Suc m) ∈ (estran Γ)› using ctran-or-etran[OF cpt Suc-m-lt] by simp
  have fst-cpt-m: ‹fst (cpt!m) = P OR Q›
  proof−
    let ?cpt = ‹take (Suc m) cpt›
   from Suc-m-lt all-etran have 1: ‹∀ i. Suc i < length ?cpt ⟶ ?cpt!i −e→ ?cpt!Suc i› by simp
    from fst-hd-cpt have 2: ‹fst (hd ?cpt) = P OR Q› by simp
    from ‹cpt≠[]› have ‹?cpt ≠ []› by simp
     have ‹∀ i<length (take (Suc m) cpt). fst (take (Suc m) cpt ! i) = P OR Q›
      by (rule all-etran-same-prog[OF 1 2 ‹?cpt≠[]›])
    then show ?thesis
      by (simp add: Suc-lessD Suc-m-lt)
  qed
  with ctran show ?thesis
    apply(subst (asm) estran-def)
    apply(subst (asm) mem-Collect-eq)
    apply(subst (asm) case-prod-unfold)
    apply(erule exE)
    apply(erule estran-p.cases, auto)
  proof−
    fix s a P' t
    assume cpt-m: ‹cpt!m = (P OR Q, s)›
    assume cpt-Suc-m: ‹cpt!Suc m = (P', t)›
    assume ctran-from-P: ‹Γ ⊢ (P, s) −es[a]→ (P', t)›
```

obtain *cpt'* **where** *cpt'*: ‹*cpt'* = *map* (λ(-,*s*). (*P*, *s*)) (*take* (*Suc m*) *cpt*) @ *drop* (*Suc m*) *cpt*› **by** *simp*
then have *cpt'-m*: ‹*cpt'*!*m* = (*P*, *s*)› **using** *Suc-m-lt*
by (*simp add*: *Suc-lessD cpt-m nth-append*)
have *len-eq*: ‹*length cpt'* = *length cpt*› **using** *cpt'* **by** *simp*
have *same-state*: ‹∀ *i*<*length cpt*. *snd* (*cpt'*!*i*) = *snd* (*cpt*!*i*)› **using** *cpt' Suc-m-lt*
by (*metis* (*mono-tags*, *lifting*) *append-take-drop-id length-map nth-append nth-map prod.collapse prod.simps*(*2*) *snd-conv*)
have ‹*cpt'* ∈ *cpts-from* (*estran* Γ) (*P*,*S0*) ∩ *assume ?pre ?rely*›
proof
show ‹*cpt'* ∈ *cpts-from* (*estran* Γ) (*P*,*S0*)›
apply(*subst cpts-from-def'*)
proof
show ‹*cpt'* ∈ *cpts* (*estran* Γ)›
apply(*subst cpts-def'*)
proof
show ‹*cpt'*≠[]› **using** *cpt'* ‹*cpt*≠[]› **by** *simp*
next
show ‹∀ *i*. *Suc i* < *length cpt'* ⟶ (*cpt'* ! *i*, *cpt'* ! *Suc i*) ∈ *estran* Γ ∨ *cpt'* ! *i* −*e*→ *cpt'* ! *Suc i*›
proof
fix *i*
show ‹*Suc i* < *length cpt'* ⟶ (*cpt'* ! *i*, *cpt'* ! *Suc i*) ∈ *estran* Γ ∨ *cpt'* ! *i* −*e*→ *cpt'* ! *Suc i*›
proof
assume *Suc-i-lt*: ‹*Suc i* < *length cpt'*›
show ‹(*cpt'* ! *i*, *cpt'* ! *Suc i*) ∈ *estran* Γ ∨ *cpt'* ! *i* −*e*→ *cpt'* ! *Suc i*›
proof(*cases* ‹*i*<*m*›)
case *True*
have ‹∀ *j* < *Suc m*. *fst*(*cpt'*!*j*) = *P*› **by** (*rule choice-sound-aux1*[*OF cpt' Suc-m-lt*])
then have *all-etran'*: ‹∀ *j*<*m*. *cpt'*!*j* −*e*→ *cpt'*!*Suc j*› **by** *simp*
have ‹*cpt'*!*i* −*e*→ *cpt'*!*Suc i*› **by** (*rule all-etran'*[*THEN spec*[**where** *x*=*i*], *rule-format*, *OF True*])
then show *?thesis* **by** *blast*
next
case *False*
have *eq-Suc-i*: ‹*cpt'*!*Suc i* = *cpt*!*Suc i*› **using** *cpt' False Suc-m-lt*
by (*metis* (*no-types*, *lifting*) *Suc-less-SucD append-take-drop-id length-map length-take min-less-iff-conj nth-append*)
show *?thesis*
proof(*cases* ‹*i*=*m*›)
case *True*
then show *?thesis*
apply *simp*
apply(*rule disjI1*)
using *cpt'-m eq-Suc-i cpt-Suc-m* **apply** (*simp add*: *estran-def*)

using *ctran-from-P* **by** *blast*
next
case *False*
with ‹¬ i < m› **have** ‹m<i› **by** *simp*
then have *eq-i*: ‹$cpt'!i = cpt!i$› **using** *cpt' Suc-m-lt*
by (*metis* (*no-types*, *lifting*) ‹¬ i < m› *append-take-drop-id length-map length-take less-SucE min-less-iff-conj nth-append*)
from *cpt* **have** ‹∀ i. *Suc* i < *length cpt* ⟶ (*cpt*!i, *cpt*!*Suc* i)∈*estran* Γ ∨ (*cpt*!i −*e*→ *cpt*!*Suc* i)› **using** *cpts-def'* **by** *metis*
then show *?thesis* **using** *eq-i eq-Suc-i Suc-i-lt len-eq* **by** *simp*
qed
qed
qed
qed
qed
next
show ‹*hd cpt'* = (P, *S0*)› **using** *cpt' hd-cpt*
by (*simp add*: ‹*cpt* ≠ []› *hd-map*)
qed
next
show ‹*cpt'* ∈ *assume ?pre ?rely*›
apply(*simp add*: *assume-def*)
proof
from *cpt'* **have** ‹*snd* (*hd cpt'*) = *snd* (*hd cpt*)›
by (*simp add*: ‹*cpt* ≠ []› *hd-cpt hd-map*)
then show ‹*snd* (*hd cpt'*) ∈ *?pre*›
using *cpt-assume* **by** (*simp add*: *assume-def*)
next
show ‹∀ i. *Suc* i < *length cpt'* ⟶ *fst* (*cpt'* ! i) = *fst* (*cpt'* ! *Suc* i) ⟶ (*snd* (*cpt'* ! i), *snd* (*cpt'* ! *Suc* i)) ∈ *?rely*›
proof
fix i
show ‹*Suc* i < *length cpt'* ⟶ *fst* (*cpt'* ! i) = *fst* (*cpt'* ! *Suc* i) ⟶ (*snd* (*cpt'* ! i), *snd* (*cpt'* ! *Suc* i)) ∈ *?rely*›
proof
assume ‹*Suc* i < *length cpt'*›
with *len-eq* **have** ‹*Suc* i < *length cpt*› **by** *simp*
show ‹*fst* (*cpt'* ! i) = *fst* (*cpt'* ! *Suc* i) ⟶ (*snd* (*cpt'* ! i), *snd* (*cpt'* ! *Suc* i)) ∈ *?rely*›
proof(*cases* ‹i<m›)
case *True*
from *same-state* ‹*Suc* i < *length cpt'*› *len-eq* **have**
‹*snd* (*cpt'*!i) = *snd* (*cpt*!i)› **and** ‹*snd* (*cpt'*!*Suc* i) = *snd* (*cpt*!*Suc* i)› **by** *simp-all*
then show *?thesis*
using *cpt-assume* ‹*Suc* i < *length cpt*› *all-etran True* **by** (*auto simp add*: *assume-def*)
next
case *False*

have *eq-Suc-i*: ‹*cpt'*!*Suc i* = *cpt*!*Suc i*› **using** *cpt' False Suc-m-lt*
by (*metis* (*no-types*, *lifting*) *Suc-less-SucD append-take-drop-id length-map length-take min-less-iff-conj nth-append*)
show *?thesis*
proof(*cases* ‹*i*=*m*›)
case *True*
have ‹*fst* (*cpt'*!*i*) ≠ *fst* (*cpt'*!*Suc i*)› **using** *True eq-Suc-i cpt'-m cpt-Suc-m ctran-from-P no-estran-to-self surjective-pairing* **by** *metis*
then show *?thesis* **by** *blast*
next
case *False*
with ‹¬ *i* < *m*› **have** ‹*m*<*i*› **by** *simp*
then have *eq-i*: ‹*cpt'*!*i* = *cpt*!*i*› **using** *cpt' Suc-m-lt*
by (*metis* (*no-types*, *lifting*) ‹¬ *i* < *m*› *append-take-drop-id length-map length-take less-SucE min-less-iff-conj nth-append*)
from *eq-i eq-Suc-i cpt-assume* ‹*Suc i* < *length cpt*›
show *?thesis* **by** (*auto simp add*: *assume-def*)
qed
qed
qed
qed
qed
qed
with *h1'* **have** *cpt'-commit*: ‹*cpt'* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*› **by** *blast*
show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
apply(*simp add*: *commit-def*)
proof
show ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ ⟶ (*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *?guar*›
(**is** ‹∀ *i*. *?P i*›)
proof
fix *i*
show ‹*?P i*›
proof(*cases i*<*m*)
case *True*
then show *?thesis*
apply *clarify*
apply(*insert all-etran*[*THEN spec*[**where** *x*=*i*]])
apply *auto*
using *no-estran-to-self''* **apply** *blast*
done
next
case *False*
have *eq-Suc-i*: ‹*cpt'*!*Suc i* = *cpt*!*Suc i*› **using** *cpt' False Suc-m-lt*
by (*metis* (*no-types*, *lifting*) *Suc-less-SucD append-take-drop-id length-map length-take min-less-iff-conj nth-append*)
show *?thesis*
proof(*cases i*=*m*)

case *True*
with *eq-Suc-i* **have** *eq-Suc-m*: ⟨$cpt'!Suc\ m = cpt!Suc\ m$⟩ **by** *simp*
have *snd-cpt-m-eq*: ⟨$snd\ (cpt!m) = s$⟩ **using** *cpt-m* **by** *simp*
from *True* **show** *?thesis* **using** *cpt′-commit*
apply(*simp add*: *commit-def*)
apply *clarify*
apply(*erule allE*[**where** $x=i$])
apply (*simp add*: *cpt′-m eq-Suc-m cpt-Suc-m estran-def snd-cpt-m-eq len-eq*)
using *ctran-from-P* **by** *blast*
next
case *False*
with ⟨$\neg\ i < m$⟩ **have** ⟨$m<i$⟩ **by** *simp*
then have *eq-i*: ⟨$cpt'!i = cpt!i$⟩ **using** *cpt′ Suc-m-lt*
by (*metis* (*no-types*, *lifting*) ⟨$\neg\ i < m$⟩ *append-take-drop-id length-map length-take less-SucE min-less-iff-conj nth-append*)
from *False* **show** *?thesis* **using** *cpt′-commit*
apply(*simp add*: *commit-def*)
apply *clarify*
apply(*erule allE*[**where** $x=i$])
apply(*simp add*: *eq-i eq-Suc-i len-eq*)
done
qed
qed
qed
next
have *eq-last*: ⟨*last cpt* = *last cpt′*⟩ **using** *cpt′ Suc-m-lt* **by** *simp*
show ⟨*fst* (*last cpt*) = *fin* ⟶ *snd* (*last cpt*) ∈ *?post*⟩
using *cpt′-commit*
by (*simp add*: *commit-def eq-last*)
qed
next
fix $s\ a\ Q'\ t$
assume *cpt-m*: ⟨$cpt!m = (P\ OR\ Q,\ s)$⟩
assume *cpt-Suc-m*: ⟨$cpt!Suc\ m = (Q',\ t)$⟩
assume *ctran-from-Q*: ⟨$\Gamma \vdash (Q,\ s)\ -es[a]\rightarrow\ (Q',\ t)$⟩
obtain *cpt′* **where** *cpt′*: ⟨$cpt' = map\ (\lambda(\text{-},s).\ (Q,\ s))\ (take\ (Suc\ m)\ cpt)$ @ $drop\ (Suc\ m)\ cpt$⟩ **by** *simp*
then have *cpt′-m*: ⟨$cpt'!m = (Q,\ s)$⟩ **using** *Suc-m-lt*
by (*simp add*: *Suc-lessD cpt-m nth-append*)
have *len-eq*: ⟨*length cpt′* = *length cpt*⟩ **using** *cpt′* **by** *simp*
have *same-state*: ⟨∀ *i*<*length cpt*. *snd* (*cpt′*!*i*) = *snd* (*cpt*!*i*)⟩ **using** *cpt′ Suc-m-lt*
by (*metis* (*mono-tags*, *lifting*) *append-take-drop-id length-map nth-append nth-map prod.collapse prod.simps*(*2*) *snd-conv*)
have ⟨$cpt' \in cpts\text{-}from\ (estran\ \Gamma)\ (Q,S0) \cap assume\ ?pre\ ?rely$⟩
proof
show ⟨$cpt' \in cpts\text{-}from\ (estran\ \Gamma)\ (Q,S0)$⟩
apply(*subst cpts-from-def′*)

proof
show ‹$cpt' \in cpts$ ($estran\ \Gamma$)›
apply(*subst cpts-def'*)
proof
show ‹$cpt' \neq []$› **using** *cpt'* ‹$cpt \neq []$› **by** *simp*
next
show ‹$\forall i.\ Suc\ i < length\ cpt' \longrightarrow (cpt'\ !\ i,\ cpt'\ !\ Suc\ i) \in estran\ \Gamma \vee cpt'\ !\ i\ -e\rightarrow\ cpt'\ !\ Suc\ i$›
proof
fix *i*
show ‹$Suc\ i < length\ cpt' \longrightarrow (cpt'\ !\ i,\ cpt'\ !\ Suc\ i) \in estran\ \Gamma \vee cpt'\ !\ i\ -e\rightarrow\ cpt'\ !\ Suc\ i$›
proof
assume *Suc-i-lt*: ‹$Suc\ i < length\ cpt'$›
show ‹$(cpt'\ !\ i,\ cpt'\ !\ Suc\ i) \in estran\ \Gamma \vee cpt'\ !\ i\ -e\rightarrow\ cpt'\ !\ Suc\ i$›
proof(*cases* ‹$i<m$›)
case *True*
have ‹$\forall j < Suc\ m.\ fst(cpt'!j) = Q$› **by** (*rule choice-sound-aux1*[*OF cpt' Suc-m-lt*])
then have *all-etran'*: ‹$\forall j<m.\ cpt'!j\ -e\rightarrow\ cpt'!Suc\ j$› **by** *simp*
have ‹$cpt'!i\ -e\rightarrow\ cpt'!Suc\ i$› **by** (*rule all-etran'*[*THEN spec*[**where** $x=i$], *rule-format*, *OF True*])
then show *?thesis* **by** *blast*
next
case *False*
have *eq-Suc-i*: ‹$cpt'!Suc\ i = cpt!Suc\ i$› **using** *cpt' False Suc-m-lt*
by (*metis* (*no-types*, *lifting*) *Suc-less-SucD append-take-drop-id length-map length-take min-less-iff-conj nth-append*)
show *?thesis*
proof(*cases* ‹$i=m$›)
case *True*
then show *?thesis*
apply *simp*
apply(*rule disjI1*)
using *cpt'-m eq-Suc-i cpt-Suc-m* **apply** (*simp add*: *estran-def*)
using *ctran-from-Q* **by** *blast*
next
case *False*
with ‹$\neg\ i < m$› **have** ‹$m<i$› **by** *simp*
then have *eq-i*: ‹$cpt'!i = cpt!i$› **using** *cpt' Suc-m-lt*
by (*metis* (*no-types*, *lifting*) ‹$\neg\ i < m$› *append-take-drop-id length-map length-take less-SucE min-less-iff-conj nth-append*)
from *cpt* **have** ‹$\forall i.\ Suc\ i < length\ cpt \longrightarrow (cpt!i,\ cpt!Suc\ i) \in estran\ \Gamma \vee (cpt!i\ -e\rightarrow\ cpt!Suc\ i)$› **using** *cpts-def'* **by** *metis*
then show *?thesis* **using** *eq-i eq-Suc-i Suc-i-lt len-eq* **by** *simp*
qed
qed
qed

qed
qed
next
show ‹*hd cpt′* = (*Q*, *S0*)› **using** *cpt′ hd-cpt*
by (*simp add*: ‹*cpt* ≠ []› *hd-map*)
qed
next
show ‹*cpt′* ∈ *assume ?pre ?rely*›
apply(*simp add*: *assume-def*)
proof
from *cpt′* **have** ‹*snd* (*hd cpt′*) = *snd* (*hd cpt*)›
by (*simp add*: ‹*cpt* ≠ []› *hd-cpt hd-map*)
then show ‹*snd* (*hd cpt′*) ∈ *?pre*›
using *cpt-assume* **by** (*simp add*: *assume-def*)
next
show ‹∀ *i*. *Suc i* < *length cpt′* ⟶ *fst* (*cpt′* ! *i*) = *fst* (*cpt′* ! *Suc i*) ⟶ (*snd* (*cpt′* ! *i*), *snd* (*cpt′* ! *Suc i*)) ∈ *?rely*›
proof
fix *i*
show ‹*Suc i* < *length cpt′* ⟶ *fst* (*cpt′* ! *i*) = *fst* (*cpt′* ! *Suc i*) ⟶ (*snd* (*cpt′* ! *i*), *snd* (*cpt′* ! *Suc i*)) ∈ *?rely*›
proof
assume ‹*Suc i* < *length cpt′*›
with *len-eq* **have** ‹*Suc i* < *length cpt*› **by** *simp*
show ‹*fst* (*cpt′* ! *i*) = *fst* (*cpt′* ! *Suc i*) ⟶ (*snd* (*cpt′* ! *i*), *snd* (*cpt′* ! *Suc i*)) ∈ *?rely*›
proof(*cases* ‹*i*<*m*›)
case *True*
from *same-state* ‹*Suc i* < *length cpt′*› *len-eq* **have**
‹*snd* (*cpt′*!*i*) = *snd* (*cpt*!*i*)› **and** ‹*snd* (*cpt′*!*Suc i*) = *snd* (*cpt*!*Suc i*)› **by** *simp-all*
then show *?thesis*
using *cpt-assume* ‹*Suc i* < *length cpt*› *all-etran True* **by** (*auto simp add*: *assume-def*)
next
case *False*
have *eq-Suc-i*: ‹*cpt′*!*Suc i* = *cpt*!*Suc i*› **using** *cpt′ False Suc-m-lt*
by (*metis* (*no-types*, *lifting*) *Suc-less-SucD append-take-drop-id length-map length-take min-less-iff-conj nth-append*)
show *?thesis*
proof(*cases* ‹*i*=*m*›)
case *True*
have ‹*fst* (*cpt′*!*i*) ≠ *fst* (*cpt′*!*Suc i*)› **using** *True eq-Suc-i cpt′-m cpt-Suc-m ctran-from-Q no-estran-to-self surjective-pairing* **by** *metis*
then show *?thesis* **by** *blast*
next
case *False*
with ‹¬ *i* < *m*› **have** ‹*m*<*i*› **by** *simp*
then have *eq-i*: ‹*cpt′*!*i* = *cpt*!*i*› **using** *cpt′ Suc-m-lt*

```
                  by (metis (no-types, lifting) ‹¬ i < m› append-take-drop-id
length-map length-take less-SucE min-less-iff-conj nth-append)
                from eq-i eq-Suc-i cpt-assume ‹Suc i < length cpt›
                show ?thesis by (auto simp add: assume-def)
              qed
            qed
          qed
        qed
      qed
    qed
    with h2' have cpt'-commit: ‹cpt' ∈ commit (estran Γ) {fin} ?guar ?post›
by blast
    show ‹cpt ∈ commit (estran Γ) {fin} ?guar ?post›
      apply(simp add: commit-def)
    proof
      show ‹∀ i. Suc i < length cpt ⟶ (cpt ! i, cpt ! Suc i) ∈ estran Γ ⟶
(snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?guar›
        (is ‹∀ i. ?P i›)
      proof
        fix i
        show ‹?P i›
        proof(cases i<m)
          case True
          then show ?thesis
            apply clarify
            apply(insert all-etran[THEN spec[where x=i]])
            apply auto
            using no-estran-to-self'' apply blast
            done
        next
          case False
          have eq-Suc-i: ‹cpt'!Suc i = cpt!Suc i› using cpt' False Suc-m-lt
              by (metis (no-types, lifting) Suc-less-SucD append-take-drop-id
length-map length-take min-less-iff-conj nth-append)
          show ?thesis
          proof(cases i=m)
            case True
            with eq-Suc-i have eq-Suc-m: ‹cpt'!Suc m = cpt!Suc m› by simp
            have snd-cpt-m-eq: ‹snd (cpt!m) = s› using cpt-m by simp
            from True show ?thesis using cpt'-commit
              apply(simp add: commit-def)
              apply clarify
              apply(erule allE[where x=i])
          apply (simp add: cpt'-m eq-Suc-m cpt-Suc-m estran-def snd-cpt-m-eq
len-eq)
              using ctran-from-Q by blast
          next
            case False
            with ‹¬ i < m› have ‹m<i› by simp
```

```
                then have eq-i: ‹cpt'!i = cpt!i› using cpt' Suc-m-lt
                    by (metis (no-types, lifting) ‹¬ i < m› append-take-drop-id
length-map length-take less-SucE min-less-iff-conj nth-append)
                from False show ?thesis using cpt'-commit
                  apply(simp add: commit-def)
                  apply clarify
                  apply(erule allE[where x=i])
                  apply(simp add: eq-i eq-Suc-i len-eq)
                  done
              qed
            qed
          qed
        next
          have eq-last: ‹last cpt = last cpt'› using cpt' Suc-m-lt by simp
          show ‹fst (last cpt) = fin ⟶ snd (last cpt) ∈ ?post›
            using cpt'-commit
            by (simp add: commit-def eq-last)
        qed
      qed
    qed
  qed
qed
then show ?thesis by simp
qed


lemma join-sound-aux2:
  assumes cpt-from-assume: ‹cpt ∈ cpts-from (estran Γ) (P ⋈ Q, s0) ∩ assume
pre rely›
    and valid1: ‹∀ s0. cpts-from (estran Γ) (P, s0) ∩ assume pre1 rely1 ⊆ commit
(estran Γ) {fin} guar1 post1›
    and valid2: ‹∀ s0. cpts-from (estran Γ) (Q, s0) ∩ assume pre2 rely2 ⊆ commit
(estran Γ) {fin} guar2 post2›
    and pre: ‹pre ⊆ pre1 ∩ pre2›
    and rely1: ‹rely ∪ guar2 ⊆ rely1›
    and rely2: ‹rely ∪ guar1 ⊆ rely2›
  shows
    ‹∀ i. Suc i < length (fst (split cpt)) ∧ Suc i < length (snd (split cpt)) ⟶
     ((fst (split cpt)!i, fst (split cpt)!Suc i) ∈ estran Γ ⟶ (snd (fst (split cpt)!i),
snd (fst (split cpt)!Suc i)) ∈ guar1) ∧
    ((snd (split cpt)!i, snd (split cpt)!Suc i) ∈ estran Γ ⟶ (snd (snd (split cpt)!i),
snd (snd (split cpt)!Suc i)) ∈ guar2)›
proof−
  let ?cpt1 = ‹fst (split cpt)›
  let ?cpt2 = ‹snd (split cpt)›
  have cpt1-from: ‹?cpt1 ∈ cpts-from (estran Γ) (P,s0)›
    using cpt-from-assume split-cpt by blast
  have cpt2-from: ‹?cpt2 ∈ cpts-from (estran Γ) (Q,s0)›
```

using *cpt-from-assume split-cpt* **by** *blast*
from *cpt-from-assume* **have** *cpt-from*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0)$›
and *cpt-assume*: $cpt \in assume\ pre\ rely$ **by** *auto*
from *cpt-from* **have** *cpt*: ‹$cpt \in cpts\ (estran\ \Gamma)$› **and** *fst-hd-cpt*: ‹$fst\ (hd\ cpt) = P \bowtie Q$› **by** *auto*
from *cpts-nonnil*[*OF cpt*] **have** ‹$cpt \neq []$› **.**
show *?thesis*
proof(*rule ccontr*, *simp*, *erule exE*)
fix *k*
assume
‹$Suc\ k < length\ ?cpt1 \land Suc\ k < length\ ?cpt2 \land$
$((?cpt1\ !\ k,\ ?cpt1\ !\ Suc\ k) \in estran\ \Gamma \land (snd\ (?cpt1\ !\ k),\ snd\ (?cpt1\ !\ Suc\ k)) \notin guar1 \lor$
$(?cpt2\ !\ k,\ ?cpt2\ !\ Suc\ k) \in estran\ \Gamma \land (snd\ (?cpt2\ !\ k),\ snd\ (?cpt2\ !\ Suc\ k)) \notin guar2)$›
(**is** *?P k*)
from *exists-least*[*of ?P k*, *OF this*] **obtain** *m* **where** ‹$?P\ m \land (\forall i<m.\ \neg ?P\ i)$› **by** *blast*
then show *False*
proof(*auto*)
assume *Suc-m-lt1*: ‹$Suc\ m < length\ ?cpt1$›
assume *Suc-m-lt2*: ‹$Suc\ m < length\ ?cpt2$›
from *Suc-m-lt1 split-length-le1*[*of cpt*] **have** *Suc-m-lt*: ‹$Suc\ m < length\ cpt$› **by** *simp*
assume *h*:
‹$\forall i<m.\ ((?cpt1\ !\ i,\ ?cpt1\ !\ Suc\ i) \in estran\ \Gamma \longrightarrow (snd\ (?cpt1\ !\ i),\ snd\ (?cpt1\ !\ Suc\ i)) \in guar1) \land$
$((?cpt2\ !\ i,\ ?cpt2\ !\ Suc\ i) \in estran\ \Gamma \longrightarrow (snd\ (?cpt2\ !\ i),\ snd\ (?cpt2\ !\ Suc\ i)) \in guar2)$›
assume *ctran*: ‹$(?cpt1\ !\ m,\ ?cpt1\ !\ Suc\ m) \in estran\ \Gamma$›
assume *not-guar*: ‹$(snd\ (?cpt1\ !\ m),\ snd\ (?cpt1\ !\ Suc\ m)) \notin guar1$›
let $?cpt1' =$ ‹$take\ (Suc\ (Suc\ m))\ ?cpt1$›
from *cpt1-from* **have** *cpt1'-from*: ‹$?cpt1' \in cpts\text{-}from\ (estran\ \Gamma)\ (P,s0)$›
by (*metis Zero-not-Suc cpts-from-take*)
then have *cpt1'*: ‹$?cpt1' \in cpts\ (estran\ \Gamma)$› **by** *simp*
from *ctran* **have** *ctran'*: ‹$(?cpt1'!m,\ ?cpt1'!Suc\ m) \in estran\ \Gamma$› **by** *auto*
from *split-ctran1-aux*[*OF Suc-m-lt1*]
have *Suc-m-not-fin*: ‹$fst\ (cpt\ !\ Suc\ m) \neq fin$› **.**
have ‹$\forall i.\ Suc\ i < length\ ?cpt1' \longrightarrow ?cpt1'!i\ -e\rightarrow\ ?cpt1'!Suc\ i \longrightarrow (snd\ (?cpt1'!i),\ snd\ (?cpt1'!Suc\ i)) \in rely \cup guar2$›
proof
fix *i*
show ‹$Suc\ i < length\ ?cpt1' \longrightarrow ?cpt1'!i\ -e\rightarrow\ ?cpt1'!Suc\ i \longrightarrow (snd\ (?cpt1'!i),\ snd\ (?cpt1'!Suc\ i)) \in rely \cup guar2$›
proof(*rule impI*, *rule impI*)
assume *Suc-i-lt'*: ‹$Suc\ i < length\ ?cpt1'$›
with *Suc-m-lt1* **have** ‹$i \le m$› **by** *simp*
from *Suc-i-lt'* **have** *Suc-i-lt1*: ‹$Suc\ i < length\ ?cpt1$› **by** *simp*
with *split-same-length*[*of cpt*] **have** *Suc-i-lt2*: ‹$Suc\ i < length\ ?cpt2$› **by**

simp
        from *no-fin-before-non-fin*[*OF cpt Suc-m-lt Suc-m-not-fin*] ‹$i \le m$›
        have *Suc-i-not-fin*: ‹*fst* (*cpt*!*Suc i*) ≠ *fin*› **by** *fast*
        from *Suc-i-lt'* *split-length-le1*[*of cpt*] **have** *Suc-i-lt*: ‹*Suc i* < *length cpt*›
by *simp*
        assume *etran'*: ‹*?cpt1'*!*i* −*e*→ *?cpt1'*!*Suc i*›
        then have *etran*: ‹*?cpt1*!*i* −*e*→ *?cpt1*!*Suc i*› **using** *Suc-m-lt Suc-i-lt'* **by**
(*simp add*: *split-def*)
        show ‹(*snd* (*?cpt1'*!*i*), *snd* (*?cpt1'*!*Suc i*)) ∈ *rely* ∪ *guar2*›
        proof−
          from *split-etran1*[*OF cpt fst-hd-cpt Suc-i-lt Suc-i-not-fin etran*]
          have ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i* ∨ (*?cpt2* ! *i*, *?cpt2* ! *Suc i*) ∈ *estran* Γ› **.**
          then show *?thesis*
          proof
            assume *etran*: ‹*cpt*!*i* −*e*→ *cpt*!*Suc i*›
            with *cpt-assume Suc-i-lt* **have** ‹(*snd* (*cpt*!*i*), *snd* (*cpt*!*Suc i*)) ∈ *rely*›
              by (*simp add*: *assume-def*)
            then have ‹(*snd* (*?cpt1*!*i*), *snd* (*?cpt1*!*Suc i*)) ∈ *rely*›
          using *split-same-state1*[*OF Suc-i-lt1*] *split-same-state1*[*OF Suc-i-lt1*[*THEN*
Suc-lessD]] **by** *argo*
            then have ‹(*snd* (*?cpt1'*!*i*), *snd* (*?cpt1'*!*Suc i*)) ∈ *rely*› **using** ‹$i \le m$›
by *simp*
            then show ‹(*snd* (*?cpt1'*!*i*), *snd* (*?cpt1'*!*Suc i*)) ∈ *rely* ∪ *guar2*› **by**
simp
          next
            assume *ctran2*: ‹(*?cpt2*!*i*, *?cpt2*!*Suc i*) ∈ *estran* Γ›
            have ‹(*snd* (*?cpt2*!*i*), *snd* (*?cpt2*!*Suc i*)) ∈ *guar2*›
            proof(*cases* ‹$i=m$›)
              case *True*
              with *ctran etran ctran-imp-not-etran* **show** *?thesis* **by** *blast*
            next
              case *False*
              with ‹$i \le m$› **have** ‹$i<m$› **by** *linarith*
              show *?thesis* **using** *ctran2 h*[*THEN spec*[**where** $x=i$], *rule-format*,
OF ‹$i<m$›] **by** *blast*
            qed
            thm *split-same-state2*
            then have ‹(*snd* (*cpt*!*i*), *snd*(*cpt*!*Suc i*)) ∈ *guar2*›
              using *Suc-i-lt2* **by** (*simp add*: *split-same-state2*)
            then have ‹(*snd* (*?cpt1*!*i*), *snd* (*?cpt1*!*Suc i*)) ∈ *guar2*›
          using *split-same-state1*[*OF Suc-i-lt1*] *split-same-state1*[*OF Suc-i-lt1*[*THEN*
Suc-lessD]] **by** *argo*
          then have ‹(*snd* (*?cpt1'*!*i*), *snd* (*?cpt1'*!*Suc i*)) ∈ *guar2*› **using** ‹$i \le m$›
by *simp*
            then show ‹(*snd* (*?cpt1'*!*i*), *snd* (*?cpt1'*!*Suc i*)) ∈ *rely* ∪ *guar2*› **by**
simp
          qed
        qed
      qed

```
    qed
    moreover have ‹snd (hd ?cpt1') ∈ pre›
    proof−
      have ‹snd (hd cpt) ∈ pre› using cpt-assume by (simp add: assume-def)
      then have ‹snd (hd ?cpt1) ∈ pre› using split-same-state1
          by (metis ‹cpt ≠ []› cpt1' cpts-def' hd-conv-nth length-greater-0-conv
take-eq-Nil)
      then show ?thesis by simp
    qed
    ultimately have ‹?cpt1' ∈ assume pre1 rely1› using rely1 pre
      by (auto simp add: assume-def)
     with cpt1'-from pre have ‹?cpt1' ∈ cpts-from (estran Γ) (P,s0) ∩ assume
pre1 rely1› by blast
    with valid1 have ‹?cpt1' ∈ commit (estran Γ) {fin} guar1 post1› by blast
    then have ‹(snd (?cpt1' ! m), snd (?cpt1' ! Suc m)) ∈ guar1›
      apply(simp add: commit-def)
      apply clarify
      apply(erule allE[where x=m])
      using Suc-m-lt1 ctran' by simp
    with not-guar Suc-m-lt show False by (simp add: Suc-m-lt Suc-lessD)
  next
    assume Suc-m-lt1: ‹Suc m < length ?cpt1›
    assume Suc-m-lt2: ‹Suc m < length ?cpt2›
     from Suc-m-lt1 split-length-le1[of cpt] have Suc-m-lt: ‹Suc m < length cpt›
by simp
    assume h:
       ‹∀ i<m. ((?cpt1 ! i, ?cpt1 ! Suc i) ∈ estran Γ ⟶ (snd (?cpt1 ! i), snd
(?cpt1 ! Suc i)) ∈ guar1) ∧
            ((?cpt2 ! i, ?cpt2 ! Suc i) ∈ estran Γ ⟶ (snd (?cpt2 ! i), snd (?cpt2
! Suc i)) ∈ guar2)›
    assume ctran: ‹(?cpt2 ! m, ?cpt2 ! Suc m) ∈ estran Γ›
    assume not-guar: ‹(snd (?cpt2 ! m), snd (?cpt2 ! Suc m)) ∉ guar2›
    let ?cpt2' = ‹take (Suc (Suc m)) ?cpt2›
    from cpt2-from have cpt2'-from: ‹?cpt2' ∈ cpts-from (estran Γ) (Q,s0)›
      by (metis Zero-not-Suc cpts-from-take)
    then have cpt2': ‹?cpt2' ∈ cpts (estran Γ)› by simp
    from ctran have ctran': ‹(?cpt2'!m, ?cpt2'!Suc m) ∈ estran Γ› by fastforce
    from split-ctran2-aux[OF Suc-m-lt2]
    have Suc-m-not-fin: ‹fst (cpt ! Suc m) ≠ fin› .
     have ‹∀ i. Suc i < length ?cpt2' ⟶ ?cpt2'!i −e→ ?cpt2'!Suc i ⟶ (snd
(?cpt2'!i), snd (?cpt2'!Suc i)) ∈ rely ∪ guar1›
    proof
      fix i
        show ‹Suc i < length ?cpt2' ⟶ ?cpt2'!i −e→ ?cpt2'!Suc i ⟶ (snd
(?cpt2'!i), snd (?cpt2'!Suc i)) ∈ rely ∪ guar1›
      proof(rule impI, rule impI)
        assume Suc-i-lt': ‹Suc i < length ?cpt2'›
        with Suc-m-lt have ‹i≤m› by simp
        from Suc-i-lt' have Suc-i-lt2: ‹Suc i < length ?cpt2› by simp
```

with *split-same-length*[*of cpt*] **have** *Suc-i-lt1*: ‹*Suc i* < *length ?cpt1*› **by** *simp*
from *no-fin-before-non-fin*[*OF cpt Suc-m-lt Suc-m-not-fin*] ‹*i*≤*m*›**have**
Suc-i-not-fin: ‹*fst* (*cpt*!*Suc i*) ≠ *fin*› **by** *fast*
from *Suc-i-lt′ split-length-le2*[*of cpt*] **have** *Suc-i-lt*: ‹*Suc i* < *length cpt*› **by** *simp*
assume *etran′*: ‹*?cpt2′*!*i* −*e*→ *?cpt2′*!*Suc i*›
then have *etran*: ‹*?cpt2*!*i* −*e*→ *?cpt2*!*Suc i*› **using** *Suc-m-lt Suc-i-lt′* **by** (*simp add*: *split-def*)
show ‹(*snd* (*?cpt2′*!*i*), *snd* (*?cpt2′*!*Suc i*)) ∈ *rely* ∪ *guar1*›
proof−
have ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i* ∨ (*?cpt1* ! *i*, *?cpt1* ! *Suc i*) ∈ *estran* Γ›
by (*rule split-etran2*[*OF cpt fst-hd-cpt Suc-i-lt Suc-i-not-fin etran*])
then show *?thesis*
proof
assume *etran*: ‹*cpt*!*i* −*e*→ *cpt*!*Suc i*›
with *cpt-assume Suc-i-lt* **have** ‹(*snd* (*cpt*!*i*), *snd* (*cpt*!*Suc i*)) ∈ *rely*›
by (*simp add*: *assume-def*)
then have ‹(*snd* (*?cpt2*!*i*), *snd* (*?cpt2*!*Suc i*)) ∈ *rely*›
using *split-same-state2*[*OF Suc-i-lt2*] *split-same-state2*[*OF Suc-i-lt2*[*THEN Suc-lessD*]] **by** *argo*
then have ‹(*snd* (*?cpt2′*!*i*), *snd* (*?cpt2′*!*Suc i*)) ∈ *rely*› **using** ‹*i*≤*m*› **by** *simp*
then show ‹(*snd* (*?cpt2′*!*i*), *snd* (*?cpt2′*!*Suc i*)) ∈ *rely* ∪ *guar1*› **by** *simp*
next
assume *ctran1*: ‹(*?cpt1*!*i*, *?cpt1*!*Suc i*) ∈ *estran* Γ›
then have ‹(*snd* (*?cpt1*!*i*), *snd* (*?cpt1*!*Suc i*)) ∈ *guar1*›
proof(*cases* ‹*i*=*m*›)
case *True*
with *ctran etran ctran-imp-not-etran* **show** *?thesis* **by** *blast*
next
case *False*
with ‹*i*≤*m*› **have** ‹*i*<*m*› **by** *simp*
show *?thesis* **using** *ctran1 h*[*THEN spec*[**where** *x*=*i*], *rule-format*, *OF* ‹*i*<*m*›] **by** *blast*
qed
then have ‹(*snd* (*cpt*!*i*), *snd*(*cpt*!*Suc i*)) ∈ *guar1*›
using *Suc-i-lt1* **by** (*simp add*: *split-same-state1*)
then have ‹(*snd* (*?cpt2*!*i*), *snd* (*?cpt2*!*Suc i*)) ∈ *guar1*›
using *split-same-state2*[*OF Suc-i-lt2*] *split-same-state2*[*OF Suc-i-lt2*[*THEN Suc-lessD*]] **by** *argo*
then have ‹(*snd* (*?cpt2′*!*i*), *snd* (*?cpt2′*!*Suc i*)) ∈ *guar1*› **using** ‹*i*≤*m*› **by** *simp*
then show ‹(*snd* (*?cpt2′*!*i*), *snd* (*?cpt2′*!*Suc i*)) ∈ *rely* ∪ *guar1*› **by** *simp*
qed
qed
qed

qed
moreover have ‹*snd* (*hd* ?*cpt2′*) ∈ *pre*›
proof−
have ‹*snd* (*hd cpt*) ∈ *pre*› **using** *cpt-assume* **by** (*simp add*: *assume-def*)
then have ‹*snd* (*hd* ?*cpt2*) ∈ *pre*› **using** *split-same-state2*
by (*metis* ‹*cpt* ≠ []› *cpt2′ cpts-def′ hd-conv-nth length-greater-0-conv take-eq-Nil*)
then show ?*thesis* **by** *simp*
qed
ultimately have ‹?*cpt2′* ∈ *assume pre2 rely2*› **using** *rely2 pre*
by (*auto simp add*: *assume-def*)
with *cpt2′-from* **have** ‹?*cpt2′* ∈ *cpts-from* (*estran* Γ) (*Q*,*s0*) ∩ *assume pre2 rely2*› **by** *blast*
with *valid2* **have** ‹?*cpt2′* ∈ *commit* (*estran* Γ) {*fin*} *guar2 post2*› **by** *blast*
then have ‹(*snd* (?*cpt2′* ! *m*), *snd* (?*cpt2′* ! *Suc m*)) ∈ *guar2*›
apply(*simp add*: *commit-def*)
apply *clarify*
apply(*erule allE*[**where** *x*=*m*])
using *Suc-m-lt2 ctran′* **by** *simp*
with *not-guar Suc-m-lt* **show** *False* **by** (*simp add*: *Suc-m-lt Suc-lessD*)
qed
qed
qed

lemma *join-sound-aux3a*:
‹(*c1*, *c2*) ∈ *estran* Γ ⟹ ∃ *P′ Q′*. *fst c1* = *P′* ⋈ *Q′* ⟹ *fst c2* = *fin* ⟹ ∀ *s*. (*s*,*s*)∈*guar* ⟹ (*snd c1*, *snd c2*) ∈ *guar*›
apply(*subst* (*asm*) *surjective-pairing*[*of c1*])
apply(*subst* (*asm*) *surjective-pairing*[*of c2*])
apply(*erule exE*, *erule exE*)
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
done

lemma *split-assume-pre*:
assumes *cpt*: *cpt* ∈ *cpts* (*estran* Γ)
assumes *fst-hd-cpt*: *fst* (*hd cpt*) = *P* ⋈ *Q*
assumes *cpt-assume*: *cpt* ∈ *assume pre rely*
shows
snd (*hd* (*fst* (*split cpt*))) ∈ *pre* ∧
snd (*hd* (*snd* (*split cpt*))) ∈ *pre*
proof−
from *cpt-assume* **have** *pre*: ‹*snd* (*hd cpt*) ∈ *pre*› **using** *assume-def* **by** *blast*
from *cpt cpts-nonnil* **have** *cpt*≠[] **by** *blast*
from *pre hd-conv-nth*[*OF* ‹*cpt*≠[]›] **have** ‹*snd* (*cpt*!*0*) ∈ *pre*› **by** *simp*
obtain *s* **where** *hd-cpt-conv*: ‹*hd cpt* = (*P* ⋈ *Q*, *s*)› **using** *fst-hd-cpt surjective-pairing*

by *metis*
  from ‹$cpt \neq []$› **have** *1*:
    ‹$snd\ (fst\ (split\ cpt)!0) \in pre$›
    apply−
    apply(*subst hd-Cons-tl*[*symmetric*, *of cpt*]) **apply** *assumption*
    using *pre hd-cpt-conv* **by** *auto*
  from ‹$cpt \neq []$› **have** *2*:
    ‹$snd\ (snd\ (split\ cpt)!0) \in pre$›
    apply−
    apply(*subst hd-Cons-tl*[*symmetric*, *of cpt*]) **apply** *assumption*
    using *pre hd-cpt-conv* **by** *auto*
  from *cpt fst-hd-cpt* **have** ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ snd\ (hd\ cpt))$›
    using *cpts-from-def′* **by** (*metis surjective-pairing*)
  from *split-cpt*[*OF this*] **have** *cpt1*:
    $fst\ (split\ cpt) \in cpts\ (estran\ \Gamma)$
    and *cpt2*:
    $snd\ (split\ cpt) \in cpts\ (estran\ \Gamma)$ **by** *auto*
  from *cpt1 cpts-nonnil* **have** *cpt1-nonnil*: ‹$fst(split\ cpt) \neq []$› **by** *blast*
  from *cpt2 cpts-nonnil* **have** *cpt2-nonnil*: ‹$snd(split\ cpt) \neq []$› **by** *blast*
  from *1 2 hd-conv-nth*[*OF cpt1-nonnil*] *hd-conv-nth*[*OF cpt2-nonnil*] **show** *?thesis*
by *simp*
qed

lemma *join-sound-aux3-1*:
  ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0) \cap assume\ pre\ rely \Longrightarrow$
    $\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (P,\ s0) \cap assume\ pre1\ rely1 \subseteq commit\ (estran\ \Gamma)$
$\{fin\}\ guar1\ post1 \Longrightarrow$
    $\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (Q,\ s0) \cap assume\ pre2\ rely2 \subseteq commit\ (estran\ \Gamma)$
$\{fin\}\ guar2\ post2 \Longrightarrow$
    $pre \subseteq pre1 \cap pre2 \Longrightarrow$
    $rely \cup guar2 \subseteq rely1 \Longrightarrow$
    $rely \cup guar1 \subseteq rely2 \Longrightarrow$
    $Suc\ i < length\ (fst\ (split\ cpt)) \Longrightarrow$
    $fst\ (split\ cpt)!i\ {-e\rightarrow}\ fst\ (split\ cpt)!Suc\ i \Longrightarrow$
    $(snd\ (fst\ (split\ cpt)!i),\ snd\ (fst\ (split\ cpt)!Suc\ i)) \in rely \cup guar2$›
proof−
  assume *cpt-from-assume*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0) \cap assume$
$pre\ rely$›
  then have *cpt-from*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0)$›
    and *cpt-assume*: ‹$cpt \in assume\ pre\ rely$›
    and ‹$cpt \neq []$› **apply** *auto* **using** *cpts-nonnil* **by** *blast*
  from *cpt-from* **have** *cpt*: ‹$cpt \in cpts\ (estran\ \Gamma)$› **and** *hd-cpt*: ‹$hd\ cpt = (P \bowtie Q,$
$s0)$› **by** *auto*
  from *hd-cpt* **have** *fst-hd-cpt*: ‹$fst\ (hd\ cpt) = P \bowtie Q$› **by** *simp*
  assume *valid1*: ‹$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (P,\ s0) \cap assume\ pre1\ rely1 \subseteq commit$
$(estran\ \Gamma)\ \{fin\}\ guar1\ post1$›
  assume *valid2*: ‹$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (Q,\ s0) \cap assume\ pre2\ rely2 \subseteq commit$
$(estran\ \Gamma)\ \{fin\}\ guar2\ post2$›
  assume *pre*: ‹$pre \subseteq pre1 \cap pre2$›

assume *rely1*: ‹*rely* ∪ *guar2* ⊆ *rely1*›
assume *rely2*: ‹*rely* ∪ *guar1* ⊆ *rely2*›
let *?cpt1* = ‹*fst* (*split cpt*)›
let *?cpt2* = ‹*snd* (*split cpt*)›
assume *Suc-i-lt1*: ‹*Suc i* < *length ?cpt1*›
from *Suc-i-lt1 split-same-length* **have** *Suc-i-lt2*: ‹*Suc i* < *length ?cpt2*› **by** *metis*
from *Suc-i-lt1 split-length-le1*[*of cpt*] **have** *Suc-i-lt*: ‹*Suc i* < *length cpt*› **by** *simp*
assume *etran1*: ‹*?cpt1*!*i* −*e*→ *?cpt1*!*Suc i*›
from *split-cpt*[*OF cpt-from*, *THEN conjunct1*] **have** *cpt1-from*: ‹*?cpt1* ∈ *cpts-from* (*estran* Γ) (*P*, *s0*)› **.**
from *split-cpt*[*OF cpt-from*, *THEN conjunct2*] **have** *cpt2-from*: ‹*?cpt2* ∈ *cpts-from* (*estran* Γ) (*Q*, *s0*)› **.**
from *cpt1-from* **have** *cpt1*: ‹*?cpt1* ∈ *cpts* (*estran* Γ)› **by** *auto*
from *cpt2-from* **have** *cpt2*: ‹*?cpt2* ∈ *cpts* (*estran* Γ)› **by** *auto*
from *cpts-nonnil*[*OF cpt1*] **have** ‹*?cpt1*≠[]› **.**
from *cpts-nonnil*[*OF cpt2*] **have** ‹*?cpt2*≠[]› **.**
from *ctran-or-etran*[*OF cpt Suc-i-lt*]
show ‹(*snd* (*?cpt1*!*i*), *snd*(*?cpt1*!*Suc i*)) ∈ *rely* ∪ *guar2*›
proof
assume *ctran-no-etran*: ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ ∧ ¬ *cpt* ! *i* −*e*→ *cpt* ! *Suc i*›
from *split-ctran1-aux*[*OF Suc-i-lt1*] **have** *Suc-i-not-fin*: ‹*fst* (*cpt* ! *Suc i*) ≠ *fin*› **.**
from *split-ctran*[*OF cpt fst-hd-cpt Suc-i-not-fin Suc-i-lt ctran-no-etran*[*THEN conjunct1*]] **show** *?thesis*
proof
assume ‹(*fst* (*split cpt*) ! *i*, *fst* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *snd* (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i*›
with *ctran-or-etran*[*OF cpt1 Suc-i-lt1*] *etran1* **have** *False* **by** *blast*
then show *?thesis* **by** *blast*
next
assume ‹(*snd* (*split cpt*) ! *i*, *snd* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *fst* (*split cpt*) ! *i* −*e*→ *fst* (*split cpt*) ! *Suc i*›
from *join-sound-aux2*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2*, *rule-format*, *OF conjI*[*OF Suc-i-lt1 Suc-i-lt2*], *THEN conjunct2*, *rule-format*, *OF this*[*THEN conjunct1*]]
have ‹(*snd* (*snd* (*split cpt*) ! *i*), *snd* (*snd* (*split cpt*) ! *Suc i*)) ∈ *guar2*› **.**
with *split-same-state1*[*OF Suc-i-lt1*] *split-same-state1*[*OF Suc-i-lt1*[*THEN Suc-lessD*]] *split-same-state2*[*OF Suc-i-lt2*] *split-same-state2*[*OF Suc-i-lt2*[*THEN Suc-lessD*]]
have ‹(*snd* (*fst* (*split cpt*) ! *i*), *snd* (*fst* (*split cpt*) ! *Suc i*)) ∈ *guar2*› **by** *simp*
then show *?thesis* **by** *blast*
qed
next
assume ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i* ∧ (*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ›
from *this*[*THEN conjunct1*] *cpt-assume* **have** ‹(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *rely*›
apply(*auto simp add*: *assume-def*)
apply(*erule allE*[**where** *x*=*i*])

using *Suc-i-lt* **by** *blast*
with *split-same-state1*[*OF Suc-i-lt1*] *split-same-state1*[*OF Suc-i-lt1*[*THEN Suc-lessD*]]
have ‹$(snd\ (?cpt1!i),\ snd\ (?cpt1!Suc\ i)) \in rely$› **by** *simp*
then show *?thesis* **by** *blast*
qed
qed

lemma *join-sound-aux3-2*:
‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0) \cap assume\ pre\ rely \Longrightarrow$
$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (P,\ s0) \cap assume\ pre1\ rely1 \subseteq commit\ (estran\ \Gamma)$ $\{fin\}\ guar1\ post1 \Longrightarrow$
$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (Q,\ s0) \cap assume\ pre2\ rely2 \subseteq commit\ (estran\ \Gamma)$ $\{fin\}\ guar2\ post2 \Longrightarrow$
$pre \subseteq pre1 \cap pre2 \Longrightarrow$
$rely \cup guar2 \subseteq rely1 \Longrightarrow$
$rely \cup guar1 \subseteq rely2 \Longrightarrow$
$Suc\ i < length\ (snd\ (split\ cpt)) \Longrightarrow$
$snd\ (split\ cpt)!i\ -e\rightarrow\ snd\ (split\ cpt)!Suc\ i \Longrightarrow$
$(snd\ (snd\ (split\ cpt)!i),\ snd\ (snd\ (split\ cpt)!Suc\ i)) \in rely \cup guar1$›
proof−
assume *cpt-from-assume*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0) \cap assume$ $pre\ rely$›
then have *cpt-from*: ‹$cpt \in cpts\text{-}from\ (estran\ \Gamma)\ (P \bowtie Q,\ s0)$›
and *cpt-assume*: ‹$cpt \in assume\ pre\ rely$›
and ‹$cpt \neq []$› **apply** *auto* **using** *cpts-nonnil* **by** *blast*
from *cpt-from* **have** *cpt*: ‹$cpt \in cpts\ (estran\ \Gamma)$› **and** *hd-cpt*: ‹$hd\ cpt = (P \bowtie Q,$ $s0)$› **by** *auto*
from *hd-cpt* **have** *fst-hd-cpt*: ‹$fst\ (hd\ cpt) = P \bowtie Q$› **by** *simp*
assume *valid1*: ‹$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (P,\ s0) \cap assume\ pre1\ rely1 \subseteq commit$ $(estran\ \Gamma)\ \{fin\}\ guar1\ post1$›
assume *valid2*: ‹$\forall s0.\ cpts\text{-}from\ (estran\ \Gamma)\ (Q,\ s0) \cap assume\ pre2\ rely2 \subseteq commit$ $(estran\ \Gamma)\ \{fin\}\ guar2\ post2$›
assume *pre*: ‹$pre \subseteq pre1 \cap pre2$›
assume *rely1*: ‹$rely \cup guar2 \subseteq rely1$›
assume *rely2*: ‹$rely \cup guar1 \subseteq rely2$›
let *?cpt1* = ‹$fst\ (split\ cpt)$›
let *?cpt2* = ‹$snd\ (split\ cpt)$›
assume *Suc-i-lt2*: ‹$Suc\ i < length\ ?cpt2$›
from *Suc-i-lt2 split-same-length* **have** *Suc-i-lt1*: ‹$Suc\ i < length\ ?cpt1$› **by** *metis*
from *Suc-i-lt2 split-length-le2*[*of cpt*] **have** *Suc-i-lt*: ‹$Suc\ i < length\ cpt$› **by** *simp*
assume *etran2*: ‹$?cpt2!i\ -e\rightarrow\ ?cpt2!Suc\ i$›
from *split-cpt*[*OF cpt-from*, *THEN conjunct1*] **have** *cpt1-from*: ‹$?cpt1 \in cpts\text{-}from$ $(estran\ \Gamma)\ (P,\ s0)$› **.**
from *split-cpt*[*OF cpt-from*, *THEN conjunct2*] **have** *cpt2-from*: ‹$?cpt2 \in cpts\text{-}from$ $(estran\ \Gamma)\ (Q,\ s0)$› **.**
from *cpt1-from* **have** *cpt1*: ‹$?cpt1 \in cpts\ (estran\ \Gamma)$› **by** *auto*
from *cpt2-from* **have** *cpt2*: ‹$?cpt2 \in cpts\ (estran\ \Gamma)$› **by** *auto*
from *cpts-nonnil*[*OF cpt1*] **have** ‹$?cpt1 \neq []$› **.**
from *cpts-nonnil*[*OF cpt2*] **have** ‹$?cpt2 \neq []$› **.**

from *ctran-or-etran*[*OF cpt Suc-i-lt*]
show ‹(*snd* (*?cpt2*!*i*), *snd*(*?cpt2*!*Suc i*)) ∈ *rely* ∪ *guar1*›
proof
assume *ctran-no-etran*: ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ ∧ ¬ *cpt* ! *i* −*e*→ *cpt* ! *Suc i*›
from *split-ctran1-aux*[*OF Suc-i-lt1*] **have** *Suc-i-not-fin*: ‹*fst* (*cpt* ! *Suc i*) ≠ *fin*›
.
from *split-ctran*[*OF cpt fst-hd-cpt Suc-i-not-fin Suc-i-lt ctran-no-etran*[*THEN conjunct1*]] **show** *?thesis*
proof
assume ‹(*fst* (*split cpt*) ! *i*, *fst* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *snd* (*split cpt*) ! *i* −*e*→ *snd* (*split cpt*) ! *Suc i*›
from *join-sound-aux2*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2*, *rule-format*, *OF conjI*[*OF Suc-i-lt1 Suc-i-lt2*], *THEN conjunct1*, *rule-format*, *OF this*[*THEN conjunct1*]]
have ‹(*snd* (*fst* (*split cpt*) ! *i*), *snd* (*fst* (*split cpt*) ! *Suc i*)) ∈ *guar1*› **.**
with *split-same-state1*[*OF Suc-i-lt1*] *split-same-state1*[*OF Suc-i-lt1*[*THEN Suc-lessD*]] *split-same-state2*[*OF Suc-i-lt2*] *split-same-state2*[*OF Suc-i-lt2*[*THEN Suc-lessD*]]
have ‹(*snd* (*snd* (*split cpt*) ! *i*), *snd* (*snd* (*split cpt*) ! *Suc i*)) ∈ *guar1*› **by** *simp*
then show *?thesis* **by** *blast*
next
assume ‹(*snd* (*split cpt*) ! *i*, *snd* (*split cpt*) ! *Suc i*) ∈ *estran* Γ ∧ *fst* (*split cpt*) ! *i* −*e*→ *fst* (*split cpt*) ! *Suc i*›
with *ctran-or-etran*[*OF cpt2 Suc-i-lt2*] *etran2* **have** *False* **by** *blast*
then show *?thesis* **by** *blast*
qed
next
assume ‹*cpt* ! *i* −*e*→ *cpt* ! *Suc i* ∧ (*cpt* ! *i*, *cpt* ! *Suc i*) ∉ *estran* Γ›
from *this*[*THEN conjunct1*] *cpt-assume* **have** ‹(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *rely*›
apply(*auto simp add*: *assume-def*)
apply(*erule allE*[**where** *x*=*i*])
using *Suc-i-lt* **by** *blast*
with *split-same-state2*[*OF Suc-i-lt2*] *split-same-state2*[*OF Suc-i-lt2*[*THEN Suc-lessD*]]
have ‹(*snd* (*?cpt2*!*i*), *snd* (*?cpt2*!*Suc i*)) ∈ *rely*› **by** *simp*
then show *?thesis* **by** *blast*
qed
qed

lemma *split-cpt-nonnil*:
‹*cpt* ≠ [] ⟹ *fst* (*hd cpt*) = *P* ⋈ *Q* ⟹ *fst* (*split cpt*) ≠ [] ∧ *snd* (*split cpt*) ≠ []›
apply(*rule conjI*)
apply(*subst hd-Cons-tl*[*of cpt*, *symmetric*]) **apply** *assumption*
apply(*subst surjective-pairing*[*of* ‹*hd cpt*›])
apply *simp*
apply(*subst hd-Cons-tl*[*of cpt*, *symmetric*]) **apply** *assumption*
apply(*subst surjective-pairing*[*of* ‹*hd cpt*›])

apply *simp*
done

lemma *join-sound-aux5*:
‹$cpt \in$ *cpts-from* (*estran* Γ) ($P \bowtie Q$, *S0*) $\cap$ *assume pre rely* $\Longrightarrow$
$\forall$ *S0*. *cpts-from* (*estran* Γ) (*P*, *S0*) $\cap$ *assume pre1 rely1* $\subseteq$ *commit* (*estran* Γ) {*fin*} *guar1 post1* $\Longrightarrow$
$\forall$ *S0*. *cpts-from* (*estran* Γ) (*Q*, *S0*) $\cap$ *assume pre2 rely2* $\subseteq$ *commit* (*estran* Γ) {*fin*} *guar2 post2* $\Longrightarrow$
pre $\subseteq$ *pre1* $\cap$ *pre2* $\Longrightarrow$
rely $\cup$ *guar2* $\subseteq$ *rely1* $\Longrightarrow$
rely $\cup$ *guar1* $\subseteq$ *rely2* $\Longrightarrow$
fst (*last cpt*) $\in$ {*fin*} $\longrightarrow$ *snd* (*last cpt*) $\in$ *post1* $\cap$ *post2*›
proof−
assume *cpt-from-assume*: ‹*cpt* $\in$ *cpts-from* (*estran* Γ) ($P \bowtie Q$, *S0*) $\cap$ *assume pre rely*›
then have *cpt*: ‹*cpt* $\in$ *cpts* (*estran* Γ)›
and *fst-hd-cpt*: ‹*fst* (*hd cpt*) = $P \bowtie Q$›
and *cpt-assume*: ‹*cpt* $\in$ *assume pre rely*›
and *cpt-from*: ‹*cpt* $\in$ *cpts-from* (*estran* Γ) ($P \bowtie Q$, *S0*)›
by *auto*
assume *valid1*: ‹$\forall$ *S0*. *cpts-from* (*estran* Γ) (*P*, *S0*) $\cap$ *assume pre1 rely1* $\subseteq$ *commit* (*estran* Γ) {*fin*} *guar1 post1*›
assume *valid2*: ‹$\forall$ *S0*. *cpts-from* (*estran* Γ) (*Q*, *S0*) $\cap$ *assume pre2 rely2* $\subseteq$ *commit* (*estran* Γ) {*fin*} *guar2 post2*›
assume *pre*: ‹*pre* $\subseteq$ *pre1* $\cap$ *pre2*›
assume *rely1*: ‹*rely* $\cup$ *guar2* $\subseteq$ *rely1*›
assume *rely2*: ‹*rely* $\cup$ *guar1* $\subseteq$ *rely2*›
let *?cpt1* = ‹*fst* (*split cpt*)›
let *?cpt2* = ‹*snd* (*split cpt*)›
from *cpts-nonnil*[*OF cpt*] **have** ‹*cpt*≠[]› **.**
from *split-cpt-nonnil*[*OF* ‹*cpt*≠[]› *fst-hd-cpt*, *THEN conjunct1*] **have** ‹*?cpt1*≠[]› **.**
from *split-cpt-nonnil*[*OF* ‹*cpt*≠[]› *fst-hd-cpt*, *THEN conjunct2*] **have** ‹*?cpt2*≠[]› **.**
show *?thesis*
proof(*cases* ‹*fst* (*last cpt*) = *fin*›)
case *True*
with *last-conv-nth*[*OF* ‹*cpt*≠[]›] **have** ‹*fst* (*cpt* ! (*length cpt* − *1*)) = *fin*› **by** *simp*
from *exists-least*[**where** *P*=‹λ*i*. *fst* (*cpt*!*i*) = *fin*›, *OF this*]
obtain *m* **where** *m*: ‹*fst* (*cpt* ! *m*) = *fin* $\wedge$ ($\forall$ *i*<*m*. *fst* (*cpt* ! *i*) ≠ *fin*)› **by** *blast*
note *m-fin* = *m*[*THEN conjunct1*]
have ‹*m*≠*0*›
apply(*rule ccontr*)
apply(*insert m*)
apply(*insert* ‹*fst* (*hd cpt*) = $P \bowtie Q$›)
apply(*subst* (*asm*) *hd-conv-nth*) **apply**(*rule* ‹*cpt*≠[]›)

apply *simp*
done
then obtain *m′* **where** *m′*: ‹*m* = *Suc m′*› **using** *not0-implies-Suc* **by** *blast*
have *m-lt*: ‹*m* < *length cpt*›
proof(*rule ccontr*)
assume *h*: ‹¬ *m* < *length cpt*›
from *m*[*THEN conjunct2*] **have** ‹∀ *i*<*m*. *fst* (*cpt* ! *i*) ≠ *fin*› **.**
then have ‹*fst* (*cpt* ! (*length cpt* − *1*)) ≠ *fin*›
apply−
apply(*erule allE*[**where** *x*=‹*length cpt* − *1*›])
using *h* **by** (*metis* ‹*cpt* ≠ []› *diff-less length-greater-0-conv less-imp-diff-less linorder-neqE-nat zero-less-one*)
with *last-conv-nth*[*OF* ‹*cpt*≠[]›] **have** ‹*fst* (*last cpt*) ≠ *fin*› **by** *simp*
with ‹*fst* (*last cpt*) = *fin*› **show** *False* **by** *blast*
qed
with *m′* **have** *Suc-m′-lt*: ‹*Suc m′* < *length cpt*› **by** *simp*
from *m m′* **have** *m1*: ‹*fst* (*cpt* ! *Suc m′*) = *fin* ∧ (∀ *i*<*Suc m′*. *fst* (*cpt* ! *i*) ≠ *fin*)› **by** *simp*
from *m1*[*THEN conjunct1*] **obtain** *s* **where** *cpt-Suc-m′*: ‹*cpt*!*Suc m′* = (*fin*, *s*)› **using** *surjective-pairing* **by** *metis*
from *m1* **have** *m′-not-fin*: ‹*fst* (*cpt*!*m′*) ≠ *fin*›
apply *clarify*
apply(*erule allE*[**where** *x*=*m′*])
by *fast*
have ‹*fst* (*cpt*!*m′*) = *fin* ⋈ *fin*›
proof−
from *ctran-or-etran*[*OF cpt Suc-m′-lt*]
have ‹(*cpt* ! *m′*, *cpt* ! *Suc m′*) ∈ *estran* Γ ∧ ¬ *cpt* ! *m′* −*e*→ *cpt* ! *Suc m′* ∨ *cpt* ! *m′* −*e*→ *cpt* ! *Suc m′* ∧ (*cpt* ! *m′*, *cpt* ! *Suc m′*) ∉ *estran* Γ› **.**
moreover have ‹¬ *cpt* ! *m′* −*e*→ *cpt* ! *Suc m′*›
proof(*rule ccontr*, *simp*)
assume *h*: ‹*fst* (*cpt* ! *m′*) = *fst* (*cpt* ! *Suc m′*)›
from *m1*[*THEN conjunct1*] *m′-not-fin h* **show** *False* **by** *simp*
qed
ultimately have *ctran*: ‹(*cpt* ! *m′*, *cpt* ! *Suc m′*) ∈ *estran* Γ› **by** *blast*
with *cpt-Suc-m′* **show** *?thesis*
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*insert all-join*[*OF cpt fst-hd-cpt Suc-m′-lt*[*THEN Suc-lessD*] *m′-not-fin*, *rule-format*, *of m′*])
apply(*erule estran-p.cases*, *auto*)
done
qed
have ‹*length ?cpt1* = *m* ∧ *length ?cpt2* = *m*›
using *split-length*[*OF cpt fst-hd-cpt Suc-m′-lt m′-not-fin m1*[*THEN conjunct1*]] *m′* **by** *simp*
then have ‹*length ?cpt1* = *m*› **and** ‹*length ?cpt2* = *m*› **by** *auto*

from ‹*length ?cpt1* = *m*› *m-lt* **have** *cpt1-shorter*: ‹*length ?cpt1* < *length cpt*›

by *simp*

from ‹*length ?cpt2* = *m*› *m-lt* **have** *cpt2-shorter*: ‹*length ?cpt2* < *length cpt*› **by** *simp*

have ‹*m′* < *length ?cpt1*› **using** ‹*length ?cpt1* = *m*› *m′* **by** *simp*
from *split-prog1*[*OF this* ‹*fst* (*cpt*!*m′*) = *fin* ⋈ *fin*›]
have ‹*fst* (*fst* (*split cpt*) ! *m′*) = *fin*› **.**
moreover have ‹*last ?cpt1* = *?cpt1* ! *m′*›
apply(*subst last-conv-nth*[*OF* ‹*?cpt1*≠[]›])
using *m′* ‹*length ?cpt1* = *m*› **by** *simp*
ultimately have ‹*fst* (*last* (*fst* (*split cpt*))) = *fin*› **by** *simp*

have ‹*m′* < *length ?cpt2*› **using** ‹*length ?cpt2* = *m*› *m′* **by** *simp*
from *split-prog2*[*OF this* ‹*fst* (*cpt*!*m′*) = *fin* ⋈ *fin*›]
have ‹*fst* (*snd* (*split cpt*) ! *m′*) = *fin*› **.**
moreover have ‹*last ?cpt2* = *?cpt2* ! *m′*›
apply(*subst last-conv-nth*[*OF* ‹*?cpt2*≠[]›])
using *m′* ‹*length ?cpt2* = *m*› **by** *simp*
ultimately have ‹*fst* (*last* (*snd* (*split cpt*))) = *fin*› **by** *simp*

let *?cpt1′* = ‹*?cpt1* @ *drop* (*Suc m*) *cpt*›
let *?cpt2′* = ‹*?cpt2* @ *drop* (*Suc m*) *cpt*›

from *split-cpt*[*OF cpt-from*, *THEN conjunct1*, *simplified*, *THEN conjunct2*]
have ‹*hd* (*fst* (*split cpt*)) = (*P*, *S0*)› **.**
with *hd-Cons-tl*[*OF* ‹*?cpt1*≠[]›]
have ‹*?cpt1* = (*P*,*S0*) # *tl ?cpt1*› **by** *simp*
from *split-cpt*[*OF cpt-from*, *THEN conjunct2*, *simplified*, *THEN conjunct2*]
have ‹*hd* (*snd* (*split cpt*)) = (*Q*, *S0*)› **.**
with *hd-Cons-tl*[*OF* ‹*?cpt2*≠[]›]
have ‹*?cpt2* = (*Q*,*S0*) # *tl ?cpt2*› **by** *simp*

have *cpt′-from*: ‹*?cpt1′* ∈ *cpts-from* (*estran* Γ) (*P*,*S0*) ∧ *?cpt2′* ∈ *cpts-from* (*estran* Γ) (*Q*,*S0*)›
proof(*cases* ‹*Suc m* < *length cpt*›)
case *True*
then have ‹*m* < *length cpt*› **by** *simp*
have ‹*m* < *Suc m*› **by** *simp*
from *all-fin-after-fin″*[*OF cpt* ‹*m* < *length cpt*› *m-fin*, *rule-format*, *OF* ‹*m* < *Suc m*› *True*]
have ‹*fst* (*cpt* ! *Suc m*) = *fin*› **.**
then have ‹*fst* (*hd* (*drop* (*Suc m*) *cpt*)) = *fin*› **by** (*simp add*: *True hd-drop-conv-nth*)
show *?thesis*
apply *auto*
apply(*rule cpts-append-env*)
using *split-cpt cpt-from-assume* **apply** *fastforce*
apply(*rule cpts-drop*[*OF cpt True*])
apply(*simp add*: ‹*fst* (*last* (*fst* (*split cpt*))) = *fin*› ‹*fst* (*hd* (*drop* (*Suc m*) *cpt*)) = *fin*›)

apply(*subst* ‹?cpt1 = (P,S0) # tl (fst (split cpt))›)
apply *simp*
apply(*rule cpts-append-env*)
using *split-cpt cpt-from-assume* **apply** *fastforce*
apply(*rule cpts-drop*[*OF cpt True*])
apply(*simp add*: ‹fst (last (snd (split cpt))) = fin› ‹fst (hd (drop (Suc m) cpt)) = fin›)
apply(*subst* ‹?cpt2 = (Q,S0) # tl ?cpt2›)
apply *simp*
done
next
case *False*
then have ‹length cpt ≤ Suc m› **by** *simp*
from *drop-all*[*OF this*]
show *?thesis*
apply *auto*
using *split-cpt cpt-from-assume* **apply** *fastforce*
apply(*rule* ‹hd (fst (split cpt)) = (P, S0)›)
using *split-cpt cpt-from-assume* **apply** *fastforce*
apply(*rule* ‹hd (snd (split cpt)) = (Q, S0)›)
done
qed

from *cpt-from*[*simplified*, *THEN conjunct2*] **have** ‹hd cpt = (P ⋈ Q, S0)› **.**
have ‹S0 ∈ pre›
using *cpt-assume* **apply**(*simp add*: *assume-def*)
apply(*drule conjunct1*)
by (*simp add*: ‹hd cpt = (P ⋈ Q, S0)›)
have *cpt′-assume*: ‹?cpt1′ ∈ assume pre1 rely1 ∧ ?cpt2′ ∈ assume pre2 rely2›
proof(*auto simp add*: *assume-def*)
show ‹snd (hd (fst (split cpt) @ drop (Suc m) cpt)) ∈ pre1›
apply(*subst* ‹?cpt1 = (P,S0) # tl ?cpt1›)
apply *simp*
using ‹S0∈pre› *pre* **by** *blast*
next
fix *i*
assume ‹Suc i < length ?cpt1 + (length cpt − Suc m)›
with ‹length ?cpt1 = m› *Suc-leI*[*OF m-lt*] **have** ‹Suc (Suc i) < length cpt› **by** *linarith*
then have ‹Suc i < length cpt› **by** *simp*
assume ‹fst (?cpt1′!i) = fst (?cpt1′!Suc i)›
show ‹(snd (?cpt1′!i), snd (?cpt1′!Suc i)) ∈ rely1›
proof(*cases* ‹Suc i < length ?cpt1›)
case *True*
from *True* **have** ‹?cpt1′!i = ?cpt1!i›
by (*simp add*: *Suc-lessD nth-append*)
from *True* **have** ‹?cpt1′!Suc i = ?cpt1!Suc i›
by (*simp add*: *nth-append*)
from ‹fst (?cpt1′!i) = fst (?cpt1′!Suc i)› ‹?cpt1′!i = ?cpt1!i› ‹?cpt1′!Suc i

= *?cpt1*!*Suc i*›
have ‹*?cpt1*!*i* −*e*→ *?cpt1*!*Suc i*› **by** *simp*
have ‹(*snd* (*fst* (*split cpt*) ! *i*), *snd* (*fst* (*split cpt*) ! *Suc i*)) ∈ *rely1*›
using *join-sound-aux3-1*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2 True* ‹*?cpt1*!*i* −*e*→ *?cpt1*!*Suc i*›] *rely1* **by** *blast*
then show *?thesis*
by (*simp add*: ‹*?cpt1*′!*i* = *?cpt1*!*i*› ‹*?cpt1*′!*Suc i* = *?cpt1*!*Suc i*›)
next
case *False*
then have *Suc-i-ge*: ‹*Suc i* ≥ *length ?cpt1*› **by** *simp*
show *?thesis*
proof(*cases* ‹*Suc i* = *length ?cpt1*›)
case *True*
then have ‹*i* < *length ?cpt1*› **by** *linarith*
from *cpt1-shorter True* **have** ‹*Suc i* < *length cpt*› **by** *simp*
from *True* ‹*length ?cpt1* = *m*› **have** ‹*Suc i* = *m*› **by** *simp*
with *m*′ **have** ‹*i* = *m*′› **by** *simp*
with ‹*fst* (*cpt*!*m*′) = *fin* ⋈ *fin*› **have** ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› **by** *simp*
from ‹*Suc i* < *length ?cpt1* + (*length cpt* − *Suc m*)› ‹*Suc i* = *m*› ‹*length ?cpt1* = *m*›
have ‹*Suc m* < *length cpt*› **by** *simp*
from ‹*Suc i* = *m*› *m-fin* **have** ‹*fst* (*cpt*!*Suc i*) = *fin*› **by** *simp*
have *conv1*: ‹*snd* (*?cpt1*′ ! *i*) = *snd* (*cpt* ! *Suc i*)›
proof−
have ‹*snd* (*?cpt1*′!*i*) = *snd* (*?cpt1*!*i*)› **using** *True* **by** (*simp add*: *nth-append*)
moreover have ‹*snd* (*?cpt1*!*i*) = *snd* (*cpt*!*i*)›
using *split-same-state1*[*OF* ‹*i* < *length ?cpt1*›] **.**
moreover have ‹*snd* (*cpt*!*i*) = *snd* (*cpt*!*Suc i*)›
proof−
from *ctran-or-etran*[*OF cpt* ‹*Suc i* < *length cpt*›] ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› ‹*fst* (*cpt*!*Suc i*) = *fin*›
have ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ› **by** *fastforce*
then show *?thesis*
apply(*subst* (*asm*) *surjective-pairing*[*of* ‹*cpt*!*i*›])
apply(*subst* (*asm*) *surjective-pairing*[*of* ‹*cpt*!*Suc i*›])
apply(*simp add*: ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› ‹*fst* (*cpt*!*Suc i*) = *fin*› *estran-def*)
apply(*erule exE*)
apply(*erule estran-p.cases*, *auto*)
done
qed
ultimately show *?thesis* **by** *simp*
qed
have *conv2*: ‹*snd* (*?cpt1*′ ! *Suc i*) = *snd* (*cpt* ! *Suc* (*Suc i*))›
apply(*simp add*: *nth-append True*)
apply(*subst nth-drop*) **apply**(*rule Suc-leI*[*OF m-lt*])
apply(*simp add*: ‹*length ?cpt1* = *m*›)
done

```
          have ‹(snd (cpt ! Suc i), snd (cpt ! Suc (Suc i))) ∈ rely›
          proof−
            have ‹m<Suc m› by simp
             from all-fin-after-fin''[OF cpt m-lt m-fin, rule-format, OF this ‹Suc m
< length cpt›]
            have Suc-m-fin: ‹fst (cpt ! Suc m) = fin› .
            from cpt-assume show ?thesis
              apply(simp add: assume-def)
              apply(drule conjunct2)
              apply(erule allE[where x=m])
              using ‹Suc m < length cpt› m-fin Suc-m-fin ‹Suc i = m› by argo
          qed
          then show ?thesis
            apply(simp add: conv1 conv2) using rely1 by blast
        next
          case False
          with Suc-i-ge have Suc-i-gt: ‹Suc i > length ?cpt1› by linarith
          with ‹length ?cpt1 = m› have ‹¬ i < m› by simp
          then have ‹m < Suc i› by simp
          then have ‹m < Suc (Suc i)› by simp
          have conv1: ‹?cpt1'!i = cpt!Suc i›
            apply(simp add: nth-append Suc-i-gt ‹length ?cpt1 = m› ‹¬ i < m›)
            apply(subst nth-drop) apply(rule Suc-leI[OF m-lt])
            using ‹¬i<m› by simp
          have conv2: ‹?cpt1'!Suc i = cpt!Suc(Suc i)›
            using Suc-i-gt apply(simp add: nth-append)
            apply(subst nth-drop) apply(rule Suc-leI[OF m-lt])
            by (simp add: ‹length ?cpt1 = m›)
           from all-fin-after-fin''[OF cpt m-lt m-fin, rule-format, OF ‹m < Suc i›
‹Suc i < length cpt›]
          have ‹fst (cpt ! Suc i) = fin› .
         from all-fin-after-fin''[OF cpt m-lt m-fin, rule-format, OF ‹m < Suc (Suc
i)› ‹Suc (Suc i) < length cpt›]
          have ‹fst (cpt ! Suc (Suc i)) = fin› .
          from cpt-assume show ?thesis
            apply(simp add: assume-def conv1 conv2)
            apply(drule conjunct2)
            apply(erule allE[where x=‹Suc i›])
            using ‹Suc (Suc i) < length cpt› ‹fst (cpt ! Suc i) = fin› ‹fst (cpt ! Suc
(Suc i)) = fin› rely1 by auto
        qed
      qed
    next
      show ‹snd (hd (snd (split cpt) @ drop (Suc m) cpt)) ∈ pre2›
        apply(subst ‹?cpt2 = (Q,S0) # tl ?cpt2›)
        apply simp
        using ‹S0∈pre› pre by blast
    next
      fix i
```

assume ‹*Suc i* < *length ?cpt2* + (*length cpt* − *Suc m*)›
with ‹*length ?cpt2* = *m*› *Suc-leI*[*OF m-lt*] **have** ‹*Suc* (*Suc i*) < *length cpt*› **by** *linarith*
then have ‹*Suc i* < *length cpt*› **by** *simp*
assume ‹*fst* (*?cpt2'*!*i*) = *fst* (*?cpt2'*!*Suc i*)›
show ‹(*snd* (*?cpt2'*!*i*), *snd* (*?cpt2'*!*Suc i*)) ∈ *rely2*›
proof(*cases* ‹*Suc i* < *length ?cpt2*›)
case *True*
from *True* **have** *conv1*: ‹*?cpt2'*!*i* = *?cpt2*!*i*›
by (*simp add*: *Suc-lessD nth-append*)
from *True* **have** *conv2*: ‹*?cpt2'*!*Suc i* = *?cpt2*!*Suc i*›
by (*simp add*: *nth-append*)
from ‹*fst* (*?cpt2'*!*i*) = *fst* (*?cpt2'*!*Suc i*)› *conv1 conv2*
have ‹*?cpt2*!*i* −*e*→ *?cpt2*!*Suc i*› **by** *simp*
have ‹(*snd* (*snd* (*split cpt*) ! *i*), *snd* (*snd* (*split cpt*) ! *Suc i*)) ∈ *rely2*›
using *join-sound-aux3-2*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2 True* ‹*?cpt2*!*i* −*e*→ *?cpt2*!*Suc i*›] *rely2* **by** *blast*
then show *?thesis*
by (*simp add*: *conv1 conv2*)
next
case *False*
then have *Suc-i-ge*: ‹*Suc i* ≥ *length ?cpt2*› **by** *simp*
show *?thesis*
proof(*cases* ‹*Suc i* = *length ?cpt2*›)
case *True*
then have ‹*i* < *length ?cpt2*› **by** *linarith*
from *cpt2-shorter True* **have** ‹*Suc i* < *length cpt*› **by** *simp*
from *True* ‹*length ?cpt2* = *m*› **have** ‹*Suc i* = *m*› **by** *simp*
with *m'* **have** ‹*i* = *m'*› **by** *simp*
with ‹*fst* (*cpt*!*m'*) = *fin* ⋈ *fin*› **have** ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› **by** *simp*
from ‹*Suc i* < *length ?cpt2* + (*length cpt* − *Suc m*)› ‹*Suc i* = *m*› ‹*length ?cpt2* = *m*›
have ‹*Suc m* < *length cpt*› **by** *simp*
from ‹*Suc i* = *m*› *m-fin* **have** ‹*fst* (*cpt*!*Suc i*) = *fin*› **by** *simp*
have *conv1*: ‹*snd* (*?cpt2'* ! *i*) = *snd* (*cpt* ! *Suc i*)›
proof−
have ‹*snd* (*?cpt2'*!*i*) = *snd* (*?cpt2*!*i*)› **using** *True* **by** (*simp add*: *nth-append*)
moreover have ‹*snd* (*?cpt2*!*i*) = *snd* (*cpt*!*i*)›
using *split-same-state2*[*OF* ‹*i* < *length ?cpt2*›] **.**
moreover have ‹*snd* (*cpt*!*i*) = *snd* (*cpt*!*Suc i*)›
proof−
from *ctran-or-etran*[*OF cpt* ‹*Suc i* < *length cpt*›] ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› ‹*fst* (*cpt*!*Suc i*) = *fin*›
have ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ› **by** *fastforce*
then show *?thesis*
apply(*subst* (*asm*) *surjective-pairing*[*of* ‹*cpt*!*i*›])
apply(*subst* (*asm*) *surjective-pairing*[*of* ‹*cpt*!*Suc i*›])
apply(*simp add*: ‹*fst* (*cpt*!*i*) = *fin* ⋈ *fin*› ‹*fst* (*cpt*!*Suc i*) = *fin*›

estran-def)
        apply(*erule exE*)
        apply(*erule estran-p.cases*, *auto*)
        done
      qed
      ultimately show *?thesis* **by** *simp*
    qed
    have *conv2*: ‹*snd* (*?cpt2′* ! *Suc i*) = *snd* (*cpt* ! *Suc* (*Suc i*))›
      apply(*simp add*: *nth-append True*)
      apply(*subst nth-drop*) **apply**(*rule Suc-leI*[*OF m-lt*])
      apply(*simp add*: ‹*length ?cpt2* = *m*›)
      done
    have ‹(*snd* (*cpt* ! *Suc i*), *snd* (*cpt* ! *Suc* (*Suc i*))) ∈ *rely*›
    proof−
      have ‹*m*<*Suc m*› **by** *simp*
       from *all-fin-after-fin″*[*OF cpt m-lt m-fin*, *rule-format*, *OF this* ‹*Suc m* < *length cpt*›]
      have *Suc-m-fin*: ‹*fst* (*cpt* ! *Suc m*) = *fin*› **.**
      from *cpt-assume* **show** *?thesis*
        apply(*simp add*: *assume-def*)
        apply(*drule conjunct2*)
        apply(*erule allE*[**where** *x*=*m*])
        using ‹*Suc m* < *length cpt*› *m-fin Suc-m-fin* ‹*Suc i* = *m*› **by** *argo*
    qed
    then show *?thesis*
      apply(*simp add*: *conv1 conv2*) **using** *rely2* **by** *blast*
  next
    case *False*
    with *Suc-i-ge* **have** *Suc-i-gt*: ‹*Suc i* > *length ?cpt2*› **by** *linarith*
    with ‹*length ?cpt2* = *m*› **have** ‹¬ *i* < *m*› **by** *simp*
    then have ‹*m* < *Suc i*› **by** *simp*
    then have ‹*m* < *Suc* (*Suc i*)› **by** *simp*
    have *conv1*: ‹*?cpt2′*!*i* = *cpt*!*Suc i*›
      apply(*simp add*: *nth-append Suc-i-gt* ‹*length ?cpt2* = *m*› ‹¬ *i* < *m*›)
      apply(*subst nth-drop*) **apply**(*rule Suc-leI*[*OF m-lt*])
      using ‹¬*i*<*m*› **by** *simp*
    have *conv2*: ‹*?cpt2′*!*Suc i* = *cpt*!*Suc*(*Suc i*)›
      using *Suc-i-gt* **apply**(*simp add*: *nth-append*)
      apply(*subst nth-drop*) **apply**(*rule Suc-leI*[*OF m-lt*])
      by (*simp add*: ‹*length ?cpt2* = *m*›)
     from *all-fin-after-fin″*[*OF cpt m-lt m-fin*, *rule-format*, *OF* ‹*m* < *Suc i*› ‹*Suc i* < *length cpt*›]
    have ‹*fst* (*cpt* ! *Suc i*) = *fin*› **.**
   from *all-fin-after-fin″*[*OF cpt m-lt m-fin*, *rule-format*, *OF* ‹*m* < *Suc* (*Suc i*)› ‹*Suc* (*Suc i*) < *length cpt*›]
    have ‹*fst* (*cpt* ! *Suc* (*Suc i*)) = *fin*› **.**
    from *cpt-assume* **show** *?thesis*
      apply(*simp add*: *assume-def conv1 conv2*)
      apply(*drule conjunct2*)

apply(*erule allE*[**where** *x*=‹*Suc i*›])
using ‹*Suc* (*Suc i*) < *length cpt*› ‹*fst* (*cpt* ! *Suc i*) = *fin*› ‹*fst* (*cpt* ! *Suc* (*Suc i*)) = *fin*› *rely2* **by** *auto*
qed
qed
qed

from *cpt'-from cpt'-assume valid1 valid2*
have
commit1: ‹*?cpt1'* ∈ *commit* (*estran* Γ) {*fin*} *guar1 post1*› **and**
commit2: ‹*?cpt2'* ∈ *commit* (*estran* Γ) {*fin*} *guar2 post2*› **by** *blast+*

from *ctran-or-etran*[*OF cpt Suc-m'-lt*] ‹*fst* (*cpt*!*m'*) = *fin* ⋈ *fin*› ‹*fst* (*cpt*!*Suc m'*) = *fin*›
have ‹(*cpt* ! *m'*, *cpt* ! *Suc m'*) ∈ *estran* Γ› **by** *fastforce*
then have ‹*snd* (*cpt*!*m'*) = *snd* (*cpt*!*m*)›
apply(*subst* ‹*m* = *Suc m'*›)
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*insert* ‹*fst* (*cpt*!*m'*) = *fin* ⋈ *fin*›)
apply(*insert* ‹*fst* (*cpt*!*Suc m'*) = *fin*›)
apply(*erule estran-p.cases*, *auto*)
done
have *last-conv1*: ‹*last ?cpt1'* = *last cpt*›
proof(*cases* ‹*Suc m* = *length cpt*›)
case *True*
then have ‹*m* = *length cpt* − *1*› **by** *linarith*
have ‹*snd* (*last ?cpt1*) = *snd* (*cpt* ! *m'*)›
apply(*simp add*: ‹*last ?cpt1* = *?cpt1* ! *m'*›)
by (*rule split-same-state1*[*OF* ‹*m'* < *length ?cpt1*›])
moreover have ‹*cpt*!*m* = *last cpt*›
apply(*subst last-conv-nth*[*OF* ‹*cpt*≠[]›])
using ‹*m* = *length cpt* − *1*› **by** *simp*
ultimately have ‹*snd* (*last ?cpt1*) = *snd* (*last cpt*)› **using** ‹*snd* (*cpt*!*m'*) = *snd* (*cpt*!*m*)› **by** *argo*
with ‹*fst* (*last ?cpt1*) = *fin*› ‹*fst* (*last cpt*) = *fin*› **show** *?thesis*
apply(*simp add*: *True*)
using *surjective-pairing* **by** *metis*
next
case *False*
with ‹*m* < *length cpt*› **have** ‹*Suc m* < *length cpt*› **by** *linarith*
then show *?thesis* **by** *simp*
qed

have *last-conv2*: ‹*last ?cpt2'* = *last cpt*›
proof(*cases* ‹*Suc m* = *length cpt*›)
case *True*
then have ‹*m* = *length cpt* − *1*› **by** *linarith*
have ‹*snd* (*last ?cpt2*) = *snd* (*cpt* ! *m'*)›

apply(*simp add*: ‹*last ?cpt2* = *?cpt2* ! *m*′›)
by (*rule split-same-state2*[*OF* ‹*m*′ < *length ?cpt2*›])
moreover have ‹*cpt*!*m* = *last cpt*›
apply(*subst last-conv-nth*[*OF* ‹*cpt*≠[]›])
using ‹*m* = *length cpt* − *1*› **by** *simp*
ultimately have ‹*snd* (*last ?cpt2*) = *snd* (*last cpt*)› **using** ‹*snd* (*cpt*!*m*′) = *snd* (*cpt*!*m*)› **by** *argo*
with ‹*fst* (*last ?cpt2*) = *fin*› ‹*fst* (*last cpt*) = *fin*› **show** *?thesis*
apply(*simp add*: *True*)
using *surjective-pairing* **by** *metis*
next
case *False*
with ‹*m* < *length cpt*› **have** ‹*Suc m* < *length cpt*› **by** *linarith*
then show *?thesis* **by** *simp*
qed

from *commit1 commit2*
show *?thesis* **apply**(*simp add*: *commit-def*)
apply(*drule conjunct2*)
apply(*drule conjunct2*)
using *last-conv1 last-conv2* **by** *argo*
next
case *False*
have ‹*?cpt1* ∈ *cpts-from* (*estran* Γ) (*P*,*S0*)› **using** *cpt-from-assume split-cpt* **by** *blast*
moreover have ‹*?cpt1* ∈ *assume pre1 rely1*›
proof(*auto simp add*: *assume-def*)
from *split-assume-pre*[*OF cpt fst-hd-cpt cpt-assume*, *THEN conjunct1*] *pre*
show ‹*snd* (*hd* (*fst* (*split cpt*))) ∈ *pre1*› **by** *blast*
next
fix *i*
assume *etran*: ‹*fst* (*fst* (*split cpt*) ! *i*) = *fst* (*fst* (*split cpt*) ! *Suc i*)›
assume *Suc-i-lt1*: ‹*Suc i* < *length* (*fst* (*split cpt*))›
from *join-sound-aux3-1*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2 Suc-i-lt1*] *etran*
have ‹(*snd* (*fst* (*split cpt*) ! *i*), *snd* (*fst* (*split cpt*) ! *Suc i*)) ∈ *rely* ∪ *guar2*› **by** *force*
then show ‹(*snd* (*fst* (*split cpt*) ! *i*), *snd* (*fst* (*split cpt*) ! *Suc i*)) ∈ *rely1*› **using** *rely1* **by** *blast*
qed
ultimately have *cpt1-commit*: ‹*?cpt1* ∈ *commit* (*estran* Γ) {*fin*} *guar1 post1*› **using** *valid1* **by** *blast*
have ‹*?cpt2* ∈ *cpts-from* (*estran* Γ) (*Q*,*S0*)› **using** *cpt-from-assume split-cpt* **by** *blast*
moreover have ‹*?cpt2* ∈ *assume pre2 rely2*›
proof(*auto simp add*: *assume-def*)
show ‹*snd* (*hd* (*snd* (*split cpt*))) ∈ *pre2*›
using *split-assume-pre*[*OF cpt fst-hd-cpt cpt-assume*] *pre* **by** *blast*
next

fix *i*
assume *etran*: ⟨*fst* (*?cpt2*!*i*) = *fst* (*?cpt2*!*Suc i*)⟩
assume *Suc-i-lt2*: ⟨*Suc i* < *length ?cpt2*⟩
from *join-sound-aux3-2*[*OF cpt-from-assume valid1 valid2 pre rely1 rely2 Suc-i-lt2*] *etran*
have ⟨(*snd* (*snd* (*split cpt*) ! *i*), *snd* (*snd* (*split cpt*) ! *Suc i*)) ∈ *rely* ∪ *guar1*⟩ **by** *force*
then show ⟨(*snd* (*?cpt2*!*i*), *snd* (*?cpt2*!*Suc i*)) ∈ *rely2*⟩ **using** *rely2* **by** *blast*
qed
ultimately have *cpt2-commit*: ⟨*?cpt2* ∈ *commit* (*estran* Γ) {*fin*} *guar2 post2*⟩ **using** *valid2* **by** *blast*
from *cpt1-commit commit-def* **have**
⟨*fst* (*last ?cpt1*) ∈ {*fin*} ⟶ *snd* (*last ?cpt1*) ∈ *post1*⟩ **by** *fastforce*
moreover from *cpt2-commit commit-def* **have**
⟨*fst* (*last ?cpt2*) ∈ {*fin*} ⟶ *snd* (*last ?cpt2*) ∈ *post2*⟩ **by** *fastforce*
ultimately show ⟨*fst* (*last cpt*) ∈ {*fin*} ⟶ *snd* (*last cpt*) ∈ *post1* ∩ *post2*⟩
using *False* **by** *blast*
qed
qed

lemma *split-length-gt*:
assumes *cpt*: ⟨*cpt* ∈ *cpts* (*estran* Γ)⟩
and *fst-hd-cpt*: ⟨*fst* (*hd cpt*) = *P* ⋈ *Q*⟩
and *i-lt*: ⟨*i* < *length cpt*⟩
and *not-fin*: ⟨*fst* (*cpt*!*i*) ≠ *fin*⟩
shows ⟨*length* (*fst* (*split cpt*)) > *i* ∧ *length* (*snd* (*split cpt*)) > *i*⟩
proof−
from *all-join*[*OF cpt fst-hd-cpt i-lt not-fin*]
have *1*: ⟨∀ *ia*≤*i*. ∃ *P′ Q′*. *fst* (*cpt* ! *ia*) = *P′* ⋈ *Q′*⟩ **.**
from *cpt fst-hd-cpt i-lt not-fin 1*
show *?thesis*
proof(*induct cpt arbitrary*:*P Q i rule*:*split.induct*; *simp*; *case-tac ia*; *simp*)
fix *s Pa Qa ia nat*
fix *rest*
assume *IH*:
⟨⋀*P Q i*.
rest ∈ *cpts* (*estran* Γ) ⟹
fst (*hd rest*) = *P* ⋈ *Q* ⟹
i < *length rest* ⟹
fst (*rest* ! *i*) ≠ *fin* ⟹
∀ *ia*≤*i*. ∃ *P′ Q′*. *fst* (*rest* ! *ia*) = *P′* ⋈ *Q′* ⟹
i < *length* (*fst* (*split rest*)) ∧ *i* < *length* (*snd* (*split rest*))⟩
assume *a1*: ⟨(*Pa* ⋈ *Qa*, *s*) # *rest* ∈ *cpts* (*estran* Γ)⟩
assume *a2*: ⟨*nat* < *length rest*⟩
assume *a3*: ⟨*fst* (*rest* ! *nat*) ≠ *fin*⟩
assume *a4*: ⟨∀ *ia*≤*Suc nat*. ∃ *P′ Q′*. *fst* (((*Pa* ⋈ *Qa*, *s*) # *rest*) ! *ia*) = *P′* ⋈ *Q′*⟩
from *a2* **have** *rest*≠[] **by** *fastforce*
from *cpts-tl*[*OF a1*, *simplified*, *OF* ⟨*rest*≠[]⟩] **have** *1*: ⟨*rest* ∈ *cpts* (*estran* Γ)⟩ **.**

from *a4* **have** *5*: ⟨$\forall$ *ia*$\leq$*nat*. $\exists$ *P'* *Q'*. *fst* (*rest* ! *ia*) = *P'* ⋈ *Q'*⟩ **by** *auto*
from *a4*[*THEN spec*[**where** *x=1*]] **have** ⟨$\exists$ *P'* *Q'*. *fst* (((*Pa* ⋈ *Qa*, *s*) # *rest*) ! *1*) = *P'* ⋈ *Q'*⟩ **by** *force*
then have ⟨$\exists$ *P'* *Q'*. *fst* (*hd rest*) = *P'* ⋈ *Q'*⟩
apply *simp*
apply(*subst hd-conv-nth*) **apply**(*rule* ⟨*rest*$\neq$[]⟩) **apply** *assumption* **done**
then obtain *P' Q'* **where** *2*: ⟨*fst* (*hd rest*) = *P'* ⋈ *Q'*⟩ **by** *blast*
from *IH*[*OF 1 2 a2 a3 5*]
show ⟨*nat* < *length* (*fst* (*split rest*)) $\wedge$ *nat* < *length* (*snd* (*split rest*))⟩ **.**
qed
qed

lemma *Join-sound-aux*:
assumes *h1*:
⟨$\Gamma \models$ *P* sat_e [*pre1*, *rely1*, *guar1*, *post1*]⟩
assumes *h2*:
⟨$\Gamma \models$ *Q* sat_e [*pre2*, *rely2*, *guar2*, *post2*]⟩
and *rely1*: ⟨*rely* $\cup$ *guar2* $\subseteq$ *rely1*⟩
and *rely2*: ⟨*rely* $\cup$ *guar1* $\subseteq$ *rely2*⟩
and *guar-refl*: ⟨$\forall$ *s*. (*s*,*s*)$\in$*guar*⟩
and *guar*: ⟨*guar1* $\cup$ *guar2* $\subseteq$ *guar*⟩
shows
⟨$\Gamma \models$ *EJoin P Q* sat_e [*pre1* $\cap$ *pre2*, *rely*, *guar*, *post1* $\cap$ *post2*]⟩
using *h1 h2*
proof(*unfold es-validity-def validity-def*)
let *?pre1* = ⟨*lift-state-set pre1*⟩
let *?pre2* = ⟨*lift-state-set pre2*⟩
let *?rely* = ⟨*lift-state-pair-set rely*⟩
let *?rely1* = ⟨*lift-state-pair-set rely1*⟩
let *?rely2* = ⟨*lift-state-pair-set rely2*⟩
let *?guar* = ⟨*lift-state-pair-set guar*⟩
let *?guar1* = ⟨*lift-state-pair-set guar1*⟩
let *?guar2* = ⟨*lift-state-pair-set guar2*⟩
let *?post1* = ⟨*lift-state-set post1*⟩
let *?post2* = ⟨*lift-state-set post2*⟩
let *?inter-pre* = ⟨*lift-state-set* (*pre1*$\cap$*pre2*)⟩
let *?inter-post* = ⟨*lift-state-set* (*post1*$\cap$*post2*)⟩

have *rely1'*: ⟨*?rely* $\cup$ *?guar2* $\subseteq$ *?rely1*⟩
apply *standard*
apply(*simp add*: *lift-state-pair-set-def case-prod-unfold*)
using *rely1* **by** *blast*
have *rely2'*: ⟨*?rely* $\cup$ *?guar1* $\subseteq$ *?rely2*⟩
apply *standard*
apply(*simp add*: *lift-state-pair-set-def case-prod-unfold*)
using *rely2* **by** *blast*
have *guar-refl'*: ⟨$\forall$ *S*. (*S*,*S*)$\in$*?guar*⟩ **using** *guar-refl lift-state-pair-set-def* **by** *blast*

have *guar'*: ‹*?guar1* ∪ *?guar2* ⊆ *?guar*›
apply *standard*
apply(*simp add*: *lift-state-pair-set-def case-prod-unfold*)
using *guar* **by** *blast*

assume *h1'*: ‹∀ *s0*. *cpts-from* (*estran* Γ) (*P*, *s0*) ∩ *assume ?pre1 ?rely1* ⊆ *commit* (*estran* Γ) {*fin*} *?guar1 ?post1*›
assume *h2'*: ‹∀ *s0*. *cpts-from* (*estran* Γ) (*Q*, *s0*) ∩ *assume ?pre2 ?rely2* ⊆ *commit* (*estran* Γ) {*fin*} *?guar2 ?post2*›
show ‹∀ *s0*. *cpts-from* (*estran* Γ) (*P* ⋈ *Q*, *s0*) ∩ *assume ?inter-pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?inter-post*›
proof
fix *s0*
show ‹*cpts-from* (*estran* Γ) (*P* ⋈ *Q*, *s0*) ∩ *assume ?inter-pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?inter-post*›
proof
fix *cpt*
assume *cpt-from-assume*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*P* ⋈ *Q*, *s0*) ∩ *assume ?inter-pre ?rely*›
then have
cpt-from: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*P* ⋈ *Q*, *s0*)› **and**
cpt: ‹*cpt* ∈ *cpts* (*estran* Γ)› **and**
fst-hd-cpt: ‹*fst* (*hd cpt*) = *P* ⋈ *Q*› **and**
cpt-assume: ‹*cpt* ∈ *assume ?inter-pre ?rely*› **by** *auto*
show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?inter-post*›
proof−
let *?cpt1* = ‹*fst* (*split cpt*)›
let *?cpt2* = ‹*snd* (*split cpt*)›
from *split-cpt*[*OF cpt-from*, *THEN conjunct1*] **have** *?cpt1* ∈ *cpts-from* (*estran* Γ) (*P*, *s0*) **.**
then have ‹*?cpt1*≠[]› **using** *cpts-nonnil* **by** *auto*
from *split-cpt*[*OF cpt-from*, *THEN conjunct2*] **have** *?cpt2* ∈ *cpts-from* (*estran* Γ) (*Q*, *s0*) **.**
then have ‹*?cpt2*≠[]› **using** *cpts-nonnil* **by** *auto*
from *cpts-nonnil*[*OF cpt*] **have** ‹*cpt*≠[]› **.**
from *join-sound-aux2*[*OF cpt-from-assume h1' h2' - rely1' rely2'*]
have *2*:
‹∀ *i*. *Suc i* < *length ?cpt1* ∧ *Suc i* < *length ?cpt2* ⟶
((*?cpt1* ! *i*, *?cpt1* ! *Suc i*) ∈ *estran* Γ ⟶
(*snd* (*?cpt1* ! *i*), *snd* (*?cpt1* ! *Suc i*)) ∈ *?guar1*) ∧
((*?cpt2* ! *i*, *?cpt2* ! *Suc i*) ∈ *estran* Γ ⟶
(*snd* (*?cpt2* ! *i*), *snd* (*?cpt2* ! *Suc i*)) ∈ *?guar2*)› **unfolding** *lift-state-set-def* **by** *blast*
show *?thesis* **using** *cpt-from-assume*
proof(*auto simp add*: *assume-def commit-def*)
fix *i*
assume *Suc-i-lt*: ‹*Suc i* < *length cpt*›
assume *ctran*: ‹(*cpt* ! *i*, *cpt* ! *Suc i*) ∈ *estran* Γ›
show ‹(*snd* (*cpt* ! *i*), *snd* (*cpt* ! *Suc i*)) ∈ *?guar*›

```
    proof(cases ‹fst (cpt!Suc i) = fin›)
      case True
      have ‹fst (cpt ! i) ≠ fin› by (rule no-estran-from-fin'[OF ctran])
       from all-join[OF cpt fst-hd-cpt Suc-i-lt[THEN Suc-lessD] this, THEN
spec[where x=i]] have
        ‹∃ P' Q'. fst (cpt ! i) = P' ⋈ Q'› by simp
      from join-sound-aux3a[OF ctran this True guar-refl'] show ?thesis .
    next
      case False
      from split-length-gt[OF cpt fst-hd-cpt Suc-i-lt False]
      have
        Suc-i-lt1: ‹Suc i < length ?cpt1› and
        Suc-i-lt2: ‹Suc i < length ?cpt2› by auto
      from split-ctran[OF cpt fst-hd-cpt False Suc-i-lt ctran] have
        (?cpt1!i, ?cpt1!Suc i) ∈ estran Γ ∨
         (?cpt2!i, ?cpt2!Suc i) ∈ estran Γ by fast
      then show ?thesis
      proof
        assume ‹(?cpt1 ! i, ?cpt1 ! Suc i) ∈ estran Γ›
         with 2 Suc-i-lt1 Suc-i-lt2 have ‹(snd (?cpt1!i), snd (?cpt1!Suc i)) ∈
?guar1› by blast
    with split-same-state1[OF Suc-i-lt1[THEN Suc-lessD]] split-same-state1[OF
Suc-i-lt1]
        have ‹(snd (cpt!i), snd (cpt!Suc i)) ∈ ?guar1› by argo
        with guar' show ‹(snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?guar› by blast
      next
        assume ‹(?cpt2 ! i, ?cpt2 ! Suc i) ∈ estran Γ›
         with 2 Suc-i-lt1 Suc-i-lt2 have ‹(snd (?cpt2!i), snd (?cpt2!Suc i)) ∈
?guar2› by blast
    with split-same-state2[OF Suc-i-lt2[THEN Suc-lessD]] split-same-state2[OF
Suc-i-lt2]
        have ‹(snd (cpt!i), snd (cpt!Suc i)) ∈ ?guar2› by argo
        with guar' show ‹(snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?guar› by blast
      qed
    qed
  next
    have 1: ‹fst (last cpt) = fin ⟹ snd (last cpt) ∈ ?post1›
         using join-sound-aux5[OF cpt-from-assume h1' h2' - rely1' rely2']
unfolding lift-state-set-def by fastforce
    have 2: ‹fst (last cpt) = fin ⟹ snd (last cpt) ∈ ?post2›
         using join-sound-aux5[OF cpt-from-assume h1' h2' - rely1' rely2']
unfolding lift-state-set-def by fastforce
    from 1 2
      show ‹fst (last cpt) = fin ⟹ snd (last cpt) ∈ lift-state-set (post1 ∩
post2)›
      by (simp add: lift-state-set-def case-prod-unfold)
  qed
 qed
qed
```

qed
qed

lemma *post-after-fin*:
‹(*fin*, *s*) # *cs* ∈ *cpts* (*estran* Γ) ⟹
(*fin*, *s*) # *cs* ∈ *assume pre rely* ⟹
s ∈ *post* ⟹
stable post rely ⟹
snd (*last* ((*fin*, *s*) # *cs*)) ∈ *post*›
proof−
assume *1*: ‹(*fin*, *s*) # *cs* ∈ *cpts* (*estran* Γ)›
assume *asm*: ‹(*fin*, *s*) # *cs* ∈ *assume pre rely*›
assume ‹*s* ∈ *post*›
assume *stable*: ‹*stable post rely*›
obtain *cpt* **where** *cpt*: ‹*cpt* = (*fin*, *s*) # *cs*› **by** *simp*
with *asm* **have** ‹*cpt* ∈ *assume pre rely*› **by** *simp*
have *all-etran*: ‹∀ *i*. *Suc i* < *length cpt* ⟶ *cpt*!*i* −*e*→ *cpt*!*Suc i*›
apply(*rule allI*)
apply(*case-tac i*; *simp*)
using *cpt all-fin-after-fin*[*OF 1*] **by** *simp*+
from *asm* **have** *all-rely*: ‹∀ *i*. *Suc i* < *length cpt* ⟶ (*snd* (*cpt*!*i*), *snd* (*cpt*!*Suc i*))∈*rely*›
apply (*auto simp add*: *assume-def*)
using *all-etran* **by** (*simp add*: *cpt*)
from *cpt* **have** *fst-hd-cpt*: ‹*fst* (*hd cpt*) = *fin*› **by** *simp*
have *aux*: ‹∀ *i*. *i* < *length cpt* ⟶ *snd* (*cpt*!*i*) ∈ *post*›
apply(*rule allI*)
apply(*induct-tac i*)
using *cpt* **apply** *simp* **apply** (*rule* ‹*s*∈*post*›)
apply *clarify*
proof−
fix *n*
assume *h*: ‹*n* < *length cpt* ⟶ *snd* (*cpt* ! *n*) ∈ *post*›
assume *lt*: ‹*Suc n* < *length cpt*›
with *h* **have** ‹*snd* (*cpt*!*n*) ∈ *post*› **by** *fastforce*
moreover **have** ‹(*snd* (*cpt*!*n*), *snd*(*cpt*!*Suc n*))∈*rely*› **using** *all-rely lt* **by** *simp*
ultimately show ‹*snd* (*cpt*!*Suc n*) ∈ *post*› **using** *stable stable-def* **by** *fast*
qed
then have ‹*snd* (*last cpt*) ∈ *post*›
apply(*subst last-conv-nth*)
using *cpt* **apply** *simp*
using *aux*[*THEN spec*[**where** *x*=‹*length cpt*−*1*›]] *cpt* **by** *force*
then show *?thesis* **using** *cpt* **by** *simp*
qed

lemma *unlift-seq-assume*:
‹*map* (*lift-seq-esconf Q*) ((*P*, *s*) # *cs*) ∈ *assume pre rely* ⟹ (*P*,*s*)#*cs* ∈ *assume pre rely*›
apply(*auto simp add*: *assume-def lift-seq-esconf-def case-prod-unfold*)

apply(*erule-tac x=i* **in** *allE*)
apply *auto*
apply (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-lessD fst-conv linorder-neqE-nat nth-Cons′ nth-map zero-order*(*3*))
by (*metis* (*no-types*, *lifting*) *Suc-diff-1 Suc-lessD linorder-neqE-nat nth-Cons′ nth-map snd-conv zero-order*(*3*))

lemma *lift-seq-commit-aux*:
‹((*P NEXT Q*, *S*), *fst c NEXT Q*, *snd c*) ∈ *estran* Γ ⟹ ((*P*, *S*), *c*) ∈ *estran* Γ›
apply(*simp add*: *estran-def*, *erule exE*)
apply(*erule estran-p.cases*, *auto*)
using *surjective-pairing* **apply** *metis*
using *seq-neq2* **by** *fast*

lemma *nth-length-last*:
‹((*P*, *s*) # *cs* @ *cs′*) ! *length cs* = *last* ((*P*, *s*) # *cs*)›
by (*induct cs*) *auto*

lemma *while-sound-aux1*:
‹(*Q*,*t*)#*cs′* ∈ *commit* (*estran* Γ) {*fin*} *guar post* ⟹
(*P*,*s*)#*cs* ∈ *commit* (*estran* Γ) {*f*} *guar p* ⟹
(*last* ((*P*,*s*)#*cs*), (*Q*,*t*)) ∈ *estran* Γ ⟹
snd (*last* ((*P*,*s*)#*cs*)) = *t* ⟹
∀ *s*. (*s*,*s*)∈*guar* ⟹
(*P*,*s*) # *cs* @ (*Q*,*t*) # *cs′* ∈ *commit* (*estran* Γ) {*fin*} *guar post*›
proof−
assume *commit2*: ‹(*Q*,*t*)#*cs′* ∈ *commit* (*estran* Γ) {*fin*} *guar post*›
assume *commit1*: ‹(*P*,*s*)#*cs* ∈ *commit* (*estran* Γ) {*f*} *guar p*›
assume *tran*: ‹(*last* ((*P*,*s*)#*cs*), (*Q*,*t*)) ∈ *estran* Γ›
assume *last-state1*: ‹*snd* (*last* ((*P*,*s*)#*cs*)) = *t*›
assume *guar-refl*: ‹∀ *s*. (*s*,*s*)∈*guar*›
show ‹(*P*,*s*) # *cs* @ (*Q*,*t*) # *cs′* ∈ *commit* (*estran* Γ) {*fin*} *guar post*›
apply(*auto simp add*: *commit-def*)
apply(*case-tac* ‹*i*<*length cs*›)
apply *simp*
using *commit1* **apply**(*simp add*: *commit-def*)
apply *clarify*
apply(*erule-tac x=i* **in** *allE*)
apply (*smt append-is-Nil-conv butlast.simps*(*2*) *butlast-snoc length-Cons less-SucI nth-butlast*)
apply(*subgoal-tac* ‹*i* = *length cs*›)
prefer *2*
apply *linarith*
apply(*thin-tac* ‹*i* < *Suc* (*length cs*)›)
apply(*thin-tac* ‹¬ *i* < *length cs*›)
apply *simp*
apply(*thin-tac* ‹*i* = *length cs*›)

```
    apply(unfold nth-length-last)
    using tran last-state1 guar-refl apply simp using guar-refl apply blast
    using commit2 apply(simp add: commit-def)
       apply(case-tac ‹i<length cs›)
        apply simp
    using commit1 apply(simp add: commit-def)
    apply clarify
      apply(erule-tac x=i in allE)
     apply (metis (no-types, lifting) Suc-diff-1 Suc-lessD linorder-neqE-nat nth-Cons'
nth-append zero-order(3))
     apply(case-tac ‹i = length cs›)
      apply simp
    apply(unfold nth-length-last)
    using tran last-state1 guar-refl apply simp using guar-refl apply blast
       apply(subgoal-tac ‹i > length cs›)
        prefer 2
        apply linarith
    apply(thin-tac ‹¬ i < length cs›)
     apply(thin-tac ‹i ≠ length cs›)
     apply(case-tac i; simp)
    apply(rename-tac i')
    using commit2 apply(simp add: commit-def)
     apply(subgoal-tac ‹∃ j. i' = length cs + j›)
      prefer 2
    using le-Suc-ex apply simp
    apply(erule exE)
     apply simp
    apply clarify
     apply(erule-tac x=j in allE)
   apply (metis (no-types, hide-lams) add-Suc-right nth-Cons-Suc nth-append-length-plus)
    using commit2 apply(simp add: commit-def)
    done
qed

lemma while-sound-aux2:
  assumes ‹stable post rely›
    and ‹s ∈ post›
    and ‹∀ i. Suc i < length ((P,s)#cs) ⟶ ((P,s)#cs)!i −e→ ((P,s)#cs)!Suc i›
     and ‹∀ i. Suc i < length ((P,s)#cs) ⟶ ((P,s)#cs)!i −e→ ((P,s)#cs)!Suc i
⟶ (snd(((P,s)#cs)!i), snd(((P,s)#cs)!Suc i))∈rely›
  shows ‹snd (last ((P,s)#cs)) ∈ post›
  using assms(2−4)
proof(induct cs arbitrary:P s)
  case Nil
  then show ?case by simp
next
  case (Cons c cs)
  obtain P' s' where c: ‹c=(P',s')› by fastforce
  have 1: ‹s'∈post›
```

proof−
have *rely*: ‹$(s,s')\in rely$›
using *Cons*(*3*)[*THEN spec*[**where** *x*=*0*]] *Cons*(*4*)[*THEN spec*[**where** *x*=*0*]] *c*
by (*simp add*: *assume-def*)
show *?thesis* **using** *assms*(*1*) ‹$s\in post$› *rely*
by (*simp add*: *stable-def*)
qed
from *Cons*(*3*) *c*
have *2*: ‹$\forall i.\ Suc\ i < length\ ((P', s') \# cs) \longrightarrow ((P', s') \# cs) ! i -e\rightarrow ((P', s') \# cs) ! Suc\ i$› **by** *fastforce*
from *Cons*(*4*) *c*
have *3*: ‹$\forall i.\ Suc\ i < length\ ((P', s') \# cs) \longrightarrow ((P', s') \# cs) ! i -e\rightarrow ((P', s') \# cs) ! Suc\ i \longrightarrow (snd\ (((P', s') \# cs) ! i),\ snd\ (((P',s') \# cs) ! Suc\ i)) \in rely$› **by** *fastforce*
show *?case* **using** *Cons*(*1*)[*OF 1 2 3*] *c* **by** *fastforce*
qed

lemma *seq-tran-inv*:
assumes ‹$((P\ NEXT\ Q,S),\ (P'\ NEXT\ Q,T))\in estran\ \Gamma$›
shows ‹$((P,S),\ (P',T))\in estran\ \Gamma$›
using *assms*
apply (*simp add*: *estran-def*)
apply(*erule exE*) **apply**(*rule exI*) **apply**(*erule estran-p.cases*, *auto*)
using *seq-neq2* **by** *blast*

lemma *seq-tran-inv-fin*:
assumes ‹$((P\ NEXT\ Q,S),\ (Q,T))\in estran\ \Gamma$›
shows ‹$((P,S),\ (fin,T))\in estran\ \Gamma$›
using *assms*
apply (*simp add*: *estran-def*)
apply(*erule exE*) **apply**(*rule exI*) **apply**(*erule estran-p.cases*, *auto*)
using *seq-neq2*[*symmetric*] **by** *blast*

lemma *lift-seq-commit*:
assumes ‹$cpt \in commit\ (estran\ \Gamma)\ \{fin\}\ guar\ post$›
and ‹$cpt\neq[]$›
shows ‹$map\ (lift\text{-}seq\text{-}esconf\ Q)\ cpt \in commit\ (estran\ \Gamma)\ \{fin\}\ guar\ post$›
using *assms*(*1*)
apply(*simp add*: *commit-def lift-seq-esconf-def case-prod-unfold*)
apply(*rule conjI*)
apply(*rule allI*)
apply *clarify*
apply(*erule-tac x*=*i* **in** *allE*)
apply(*drule seq-tran-inv*)
apply *force*
apply *clarify*
by (*simp add*: *last-map*[*OF* ‹$cpt\neq[]$›])

lemma *while-sound-aux3*:
  assumes ‹$cs \in commit\ (estran\ \Gamma)\ \{fin\}\ guar\ post$›
    and ‹$cs \neq []$›
  shows ‹$map\ (lift\text{-}seq\text{-}esconf\ Q)\ cs \in commit\ (estran\ \Gamma)\ \{fin\}\ guar\ post'$›
  using *assms*
  apply(*auto simp add*: *commit-def lift-seq-esconf-def case-prod-unfold*)
  subgoal for *i*
  proof−
    assume *a*: ‹$\forall i.\ Suc\ i < length\ cs \longrightarrow (cs\ !\ i,\ cs\ !\ Suc\ i) \in estran\ \Gamma \longrightarrow (snd\ (cs\ !\ i),\ snd\ (cs\ !\ Suc\ i)) \in guar$›
    assume *1*: ‹$Suc\ i < length\ cs$›
    assume ‹$((fst\ (cs\ !\ i)\ \ NEXT\ \ Q,\ snd\ (cs\ !\ i)),\ fst\ (cs\ !\ Suc\ i)\ \ NEXT\ \ Q,\ snd\ (cs\ !\ Suc\ i)) \in estran\ \Gamma$›
   then have *2*: ‹$(cs\ !\ i,\ cs\ !\ Suc\ i) \in estran\ \Gamma$› **using** *seq-tran-inv surjective-pairing* **by** *metis*
    from *a*[*rule-format*, *OF 1 2*] **show** *?thesis* **.**
  qed
  subgoal
  proof−
    assume *1*: ‹$fst\ (last\ cs) \neq fin$›
    assume *2*: ‹$fst\ (last\ (map\ (\lambda uu.\ (fst\ uu\ \ NEXT\ \ Q,\ snd\ uu))\ cs)) = fin$›
    from *1 2* **have** *False*
      by (*metis* (*no-types*, *lifting*) *esys.distinct*(*5*) *fst-conv last-map list.simps*(*8*))
    then show *?thesis* **by** *blast*
  qed
  subgoal for *i*
  proof−
    assume *a*: ‹$\forall i.\ Suc\ i < length\ cs \longrightarrow (cs\ !\ i,\ cs\ !\ Suc\ i) \in estran\ \Gamma \longrightarrow (snd\ (cs\ !\ i),\ snd\ (cs\ !\ Suc\ i)) \in guar$›
    assume *1*: ‹$Suc\ i < length\ cs$›
    assume ‹$((fst\ (cs\ !\ i)\ \ NEXT\ \ Q,\ snd\ (cs\ !\ i)),\ fst\ (cs\ !\ Suc\ i)\ \ NEXT\ \ Q,\ snd\ (cs\ !\ Suc\ i)) \in estran\ \Gamma$›
   then have *2*: ‹$(cs\ !\ i,\ cs\ !\ Suc\ i) \in estran\ \Gamma$› **using** *seq-tran-inv surjective-pairing* **by** *metis*
    from *a*[*rule-format*, *OF 1 2*] **show** *?thesis* **.**
  qed
  subgoal
  proof−
    assume ‹$fst\ (last\ (map\ (\lambda uu.\ (fst\ uu\ \ NEXT\ \ Q,\ snd\ uu))\ cs)) = fin$›
    with ‹$cs \neq []$› **have** *False* **by** (*simp add*: *last-conv-nth*)
    then show *?thesis* **by** *blast*
  qed
  .

lemma *no-fin-in-unfinished*:
  assumes ‹$cpt \in cpts\ (estran\ \Gamma)$›
    and ‹$\Gamma \vdash last\ cpt\ -es[a]\rightarrow\ c$›
  shows ‹$\forall i.\ i<length\ cpt \longrightarrow fst\ (cpt!i) \neq fin$›
proof(*rule allI*, *rule impI*)

fix *i*
assume ‹*i*<*length cpt*›
show ‹*fst* (*cpt*!*i*) ≠ *fin*›
proof
assume *fin*: ‹*fst* (*cpt*!*i*) = *fin*›
let *?cpt* = ‹*drop i cpt*›
have *drop-cpt*: ‹*?cpt* ∈ *cpts* (*estran* Γ)› **using** *cpts-drop*[*OF assms*(*1*) ‹*i*<*length cpt*›] **.**
obtain *S* **where** ‹*cpt*!*i* = (*fin*,*S*)› **using** *surjective-pairing fin* **by** *metis*
have *drop-cpt-dest*: ‹*drop i cpt* = (*fin*,*S*) # *tl* (*drop i cpt*)›
using ‹*i*<*length cpt*› ‹*cpt*!*i* = (*fin*,*S*)›
by (*metis cpts-def′ drop-cpt hd-Cons-tl hd-drop-conv-nth*)
have ‹(*fin*,*S*) # *tl* (*drop i cpt*) ∈ *cpts* (*estran* Γ)› **using** *drop-cpt drop-cpt-dest* **by** *argo*
from *all-fin-after-fin*[*OF this*] **have** ‹*fst* (*last cpt*) = *fin*›
by (*metis* (*no-types*, *lifting*) ‹*cpt* ! *i* = (*fin*, *S*)› ‹*i* < *length cpt*› *drop-cpt-dest fin last-ConsL last-ConsR last-drop last-in-set*)
with *assms*(*2*) *no-estran-from-fin* **show** *False*
by (*metis prod.collapse*)
qed
qed

lemma *while-sound-aux*:
assumes ‹*cpt* ∈ *cpts-es-mod* Γ›
and ‹*preL* = *lift-state-set pre*›
and ‹*relyL* = *lift-state-pair-set rely*›
and ‹*guarL* = *lift-state-pair-set guar*›
and ‹*postL* = *lift-state-set post*›
and ‹*pre* ∩ − *b* ⊆ *post*›
and ‹∀ *S0*. *cpts-from* (*estran* Γ) (*P*,*S0*) ∩ *assume* (*lift-state-set* (*pre*∩*b*)) *relyL* ⊆ *commit* (*estran* Γ) {*fin*} *guarL preL*›
and ‹∀ *s*. (*s*, *s*) ∈ *guar*›
and ‹*stable pre rely*›
and ‹*stable post rely*›
shows ‹∀ *S cs*. *cpt* = (*EWhile b P*, *S*)#*cs* ⟶ *cpt* ∈ *assume preL relyL* ⟶ *cpt* ∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
using *assms*
proof(*induct*)
case (*CptsModOne P s x*)
then show *?case* **by** (*simp add*: *commit-def*)
next
case (*CptsModEnv P t y cs s x*)
have *1*: ‹∀ *P s t*. ((*P*, *s*), *P*, *t*) ∉ *estran* Γ› **using** *no-estran-to-self′* **by** *blast*
have *2*: ‹*stable preL relyL*› **using** *stable-lift*[*OF* ‹*stable pre rely*›] *CptsModEnv*(*3*,*4*) **by** *simp*
show *?case*
apply *clarify*
apply(*rule commit-Cons-env*)
apply(*rule 1*)

apply(*insert CptsModEnv*(*2*)[*OF CptsModEnv*(*3*−*11*)])
apply *clarify*
apply(*erule allE*[**where** *x*=‹(*t*,*y*)›])
apply(*erule allE*[**where** *x*=*cs*])
apply(*drule assume-tl-comp*[*OF* - *2*])
by *blast*
next
case (*CptsModAnon P s Q t x cs*)
then show *?case* **by** *simp*
next
case (*CptsModAnon-fin P s Q t x cs*)
then show *?case* **by** *simp*
next
case (*CptsModBasic P e s y x k cs*)
then show *?case* **by** *simp*
next
case (*CptsModAtom P e s t x cs*)
then show *?case* **by** *simp*
next
case (*CptsModSeq P s x a Q t y R cs*)
then show *?case* **by** *simp*
next
case (*CptsModSeq-fin P s x a t y Q cs*)
then show *?case* **by** *simp*
next
case (*CptsModChc1 P s x a Q t y cs R*)
then show *?case* **by** *simp*
next
case (*CptsModChc2 P s x a Q t y cs R*)
then show *?case* **by** *simp*
next
case (*CptsModJoin1 P s x a Q t y R cs*)
then show *?case* **by** *simp*
next
case (*CptsModJoin2 P s x a Q t y R cs*)
then show *?case* **by** *simp*
next
case (*CptsModJoin-fin t y cs*)
then show *?case* **by** *simp*
next
case (*CptsModWhileTMore s b1 P1 x cs a t y cs'*)
show *?case*
proof(*rule allI*, *rule allI*, *clarify*)
assume ‹*P1*=*P*› ‹*b1*=*b*›
assume *a*: ‹(*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) @ (*EWhile b P*, *t*, *y*) # *cs'* ∈ *assume preL relyL*›

let *?part1* = ‹(*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*)›
have *part2-assume*: ‹(*EWhile b P*, *t*, *y*) # *cs'* ∈ *assume preL relyL*›
proof(*simp add*: *assume-def*, *rule conjI*)
let *?c* = ‹(*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]›
have ‹*?c* ∈ *cpts-from* (*estran* Γ) (*P1*,*s*,*x*) ∩ *assume* (*lift-state-set* (*pre*∩*b*)) *relyL*›
proof
show ‹(*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)] ∈ *cpts-from* (*estran* Γ) (*P1*, *s*, *x*)›
proof(*simp*)
from *CptsModWhileTMore*(*3*) **have** *tran*: ‹(*last* ((*P1*, *s*, *x*) # *cs*), (*fin*, *t*, *y*)) ∈ *estran* Γ›
apply(*simp only*: *estran-def*) **by** *blast*
from *cpts-snoc-comp*[*OF CptsModWhileTMore*(*2*) *tran*]
show ‹*?c* ∈ *cpts* (*estran* Γ)› **by** *simp*
qed
next
from *a*
show ‹(*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)] ∈ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*›
proof(*auto simp add*: *assume-def*)
assume ‹(*s*, *x*) ∈ *preL*›
then show ‹(*s*, *x*) ∈ *lift-state-set* (*pre* ∩ *b*)›
using ‹*preL* = *lift-state-set pre*› ‹*s*∈*b1*›
by (*simp add*: *lift-state-set-def* ‹*b1*=*b*›)
next
fix *i*
assume *a2*[*rule-format*]: ‹∀ *i*<*Suc* (*Suc* (*length cs* + *length cs'*)).
fst (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs'*) ! *i*) =
fst (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs'*) ! *i*) ⟶
(*snd* (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs'*) ! *i*),
snd (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs'*) ! *i*)) ∈ *relyL*›
let *?j* = ‹*Suc i*›
assume *i-lt*: ‹*i* < *Suc* (*length cs*)›
assume *etran*: ‹*fst* (((*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*) = *fst* ((*cs* @ [(*fin*, *t*, *y*)]) ! *i*)›
show ‹(*snd* (((*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*), *snd* ((*cs* @ [(*fin*, *t*, *y*)]) ! *i*)) ∈ *relyL*›
proof(*cases* ‹*i*=*length cs*›)
case *True*
from *CptsModWhileTMore*(*3*) **have** *ctran*: ‹(*last* ((*P1*, *s*, *x*) # *cs*), (*fin*, *t*, *y*)) ∈ *estran* Γ›

apply(*simp only*: *estran-def*) **by** *blast*
have *1*: ‹((*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i* = *last* ((*P1*,*s*,*x*)#*cs*)› **using**

True **by** (*simp add*: *nth-length-last*)
have *2*: ‹(*cs* @ [(*fin*, *t*, *y*)]) ! *i* = (*fin*, *t*, *y*)› **using** *True* **by** (*simp add*: *nth-append*)
from *ctran-imp-not-etran*[*OF ctran*] *etran 1 2* **have** *False* **by** *force*
then show *?thesis* **by** *blast*
next
case *False*
with *i-lt* **have** ‹*i*<*length cs*› **by** *simp*
have
‹*fst* (*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) ! *i*) =
fst (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* ! *i*)›
proof−
have ∗: ‹*i* < *length* ((*P1*,*s*,*x*)#*cs*)› **using** ‹*i*<*length cs*› **by** *simp*
have ∗∗: ‹*i* < *length* ((*P*,*s*,*x*)#*cs*)› **using** ‹*i*<*length cs*› **by** *simp*
have ‹(((*P1*, *s*, *x*) # *cs*) @ [(*fin*, *t*, *y*)]) ! *i* = ((*P1*,*s*,*x*)#*cs*) ! *i*›
using ∗ **apply**(*simp only*: *nth-append*) **by** *simp*
then have *eq1*: ‹((*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i* = ((*P1*,*s*,*x*)#*cs*) ! *i*› **by** *simp*
have *eq2*: ‹(*cs* @ [(*fin*, *t*, *y*)]) ! *i* = *cs*!*i*›
using ‹*i*<*length cs*› **by** (*simp add*: *nth-append*)
show *?thesis*
apply(*simp only*: *nth-map*[*OF* ∗∗] *nth-map*[*OF* ‹*i*<*length cs*›])
using *etran* **apply**(*simp add*: *eq1 eq2 lift-seq-esconf-def case-prod-unfold*)
using ‹*P1*=*P*› **by** *simp*
qed
then have
‹*fst* ((*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *i*) =
fst ((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *i*)›
by (*metis* (*no-types*, *lifting*) *One-nat-def* ‹*i* < *length cs*› *add.commute* *i-lt length-map list.size*(*4*) *nth-append plus-1-eq-Suc*)
then have *2*:
‹*fst* (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *?j*) =
fst (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *?j*)›
by *simp*
have *1*: ‹*?j* < *Suc* (*Suc* (*length cs* + *length cs*′))› **using** ‹*i*<*length cs*› **by** *simp*
from *a2*[*OF 1 2*] **have** *rely*:
‹(*snd* (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *Suc i*),
snd (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *Suc i*))
∈ *relyL*› .
have *eq1*: ‹*snd* (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *Suc i*) = *snd* (((*P1*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*)›

proof−
have ∗∗: ‹i < *length* ((*P*,*s*,*x*)#*cs*)› **using** ‹*i*<*length cs*› **by** *simp*
have ‹*snd* ((*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*)) ! *i*) = *snd* (((*P1*, *s*, *x*) # *cs*) ! *i*)›
apply(*subst nth-map*[*OF* ∗∗])
by (*simp add*: *lift-seq-esconf-def case-prod-unfold* ‹*P1*=*P*›)
then have ‹*snd* ((*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) @ ((*EWhile b P*, *t*, *y*) # *cs*′)) ! *i*) = *snd* ((((*P1*, *s*, *x*) # *cs*)@[(*fin*,*t*,*y*)]) ! *i*)›
apply−
apply(*subst nth-append*) **apply**(*subst nth-append*)
using ‹*i*<*length cs*› **by** *simp*
then show *?thesis* **by** *simp*
qed
have *eq2*: ‹*snd* (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! *Suc i*) = *snd* ((*cs* @ [(*fin*, *t*, *y*)]) ! *i*)›
proof−
have ‹*snd* ((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) ! *i*) = *snd* (*cs* ! *i*)›
apply(*subst nth-map*[*OF* ‹*i*<*length cs*›])
by (*simp add*: *lift-seq-esconf-def case-prod-unfold* ‹*P1*=*P*›)
then have ‹*snd* ((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ ((*EWhile b P*, *t*, *y*) # *cs*′)) ! *i*) = *snd* ((*cs*@[(*fin*,*t*,*y*)]) ! *i*)›
apply−
apply(*subst nth-append*) **apply**(*subst nth-append*)
using ‹*i*<*length cs*› **by** *simp*
then show *?thesis* **by** *simp*
qed
from *rely* **show** *?thesis* **by** (*simp only*: *eq1 eq2*)
qed
qed
qed
with *CptsModWhileTMore*(*11*) ‹*P1*=*P*› **have** ‹*?c* ∈ *commit* (*estran* Γ) {*fin*} *guarL preL*› **by** *blast*
then show ‹(*t*,*y*)∈*preL*› **by** (*simp add*: *commit-def*)
next
show ‹∀ *i*<*length cs*′. *fst* (((*EWhile b P*, *t*, *y*) # *cs*′) ! *i*) = *fst* (*cs*′ ! *i*) ⟶ (*snd* (((*EWhile b P*, *t*, *y*) # *cs*′) ! *i*), *snd* (*cs*′ ! *i*)) ∈ *relyL*›
apply(*rule allI*)
using *a* **apply**(*auto simp add*: *assume-def*)
apply(*erule-tac x*=‹*Suc*(*Suc*(*length cs*)) + *i*› **in** *allE*)
subgoal for *i*
proof−
assume *h*[*rule-format*]:
‹*Suc* (*Suc* (*length cs*)) + *i* < *Suc* (*Suc* (*length cs* + *length cs*′)) ⟶
fst (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! (*Suc* (*Suc* (*length cs*)) + *i*)) =
fst (((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *cs*′) ! (*Suc* (*Suc* (*length cs*)) + *i*)) ⟶
(*snd* (((*EWhile b P*, *s*, *x*) # (*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf*

$(EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i)),$
snd $(((P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i)))\ \in\ relyL$›
assume *i-lt*: ‹$i < length\ cs'$›
assume *etran*: ‹$fst\ (((EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ i) = fst\ (cs'\ !\ i)$›
have *eq1*:
‹$((EWhile\ b\ P,\ s,\ x)\ \#\ (P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i) =$
$((EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ i$›
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI One-nat-def add.commute length-map list.size*(*4*) *nth-append-length-plus plus-1-eq-Suc*)
have *eq2*:
‹$((P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i) =$
$cs'!i$›
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI One-nat-def add.commute add-Suc-shift length-map list.size*(*4*) *nth-Cons-Suc nth-append-length-plus plus-1-eq-Suc*)
from *i-lt* **have** *i-lt'*: ‹$Suc\ (Suc\ (length\ cs))\ +\ i < Suc\ (Suc\ (length\ cs\ +\ length\ cs'))$› **by** *simp*
from *etran* **have** *etran'*:
‹$fst\ (((EWhile\ b\ P,\ s,\ x)\ \#\ (P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i)) =$
$fst\ (((P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i))$›
using *eq1 eq2* **by** *simp*
from *h*[*OF i-lt' etran'*] **have**
‹$(snd\ (((EWhile\ b\ P,\ s,\ x)\ \#\ (P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i)),$
$snd\ (((P\ \ NEXT\ \ EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ cs\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs')\ !\ (Suc\ (Suc\ (length\ cs))\ +\ i)))$
$\in relyL$› **.**
then show *?thesis*
using *eq1 eq2* **by** *simp*
qed
done
qed
show ‹$(EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ ((P,\ s,\ x)\ \#\ cs)\ @\ (EWhile\ b\ P,\ t,\ y)\ \#\ cs' \in commit\ (estran\ \Gamma)\ \{fin\}\ guarL\ postL$›
proof−
from *CptsModWhileTMore*(*5*)[*OF CptsModWhileTMore*(*6*−*14*), *rule-format*, *of* ‹(*t*,*y*)› *cs'*] ‹*P1*=*P*› ‹*b1*=*b*› *part2-assume*
have *part2-commit*: ‹$(EWhile\ b\ P,\ t,\ y)\ \#\ cs' \in commit\ (estran\ \Gamma)\ \{fin\}\ guarL\ postL$› **by** *simp*
have *part1-commit*: ‹$(EWhile\ b\ P,\ s,\ x)\ \#\ map\ (lift\text{-}seq\text{-}esconf\ (EWhile\ b\ P))\ ((P,\ s,\ x)\ \#\ cs) \in commit\ (estran\ \Gamma)\ \{fin\}\ guarL\ preL$›
proof−
have *1*: ‹$(P,s,x)\#cs \in cpts\text{-}from\ (estran\ \Gamma)\ (P,s,x) \cap assume\ (lift\text{-}state\text{-}set$

(*pre* ∩ *b*)) *relyL*›
    proof
      show ‹(*P*, *s*, *x*) # *cs* ∈ *cpts-from* (*estran* Γ) (*P*, *s*, *x*)›
      proof(*simp*)
        show ‹(*P*,*s*,*x*)#*cs* ∈ *cpts* (*estran* Γ)›
          using *CptsModWhileTMore*(*2*) ‹*P1*=*P*› **by** *simp*
      qed
    next
      from *assume-tl-env*[*OF a*[*simplified*]] *assume-appendD*
      have ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*› **by** *simp*
      from *unlift-seq-assume*[*OF this*] **have** ‹(*P*, *s*, *x*) # *cs* ∈ *assume preL relyL*›
.
      then show ‹(*P*, *s*, *x*) # *cs* ∈ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*› **using** ‹*s*∈*b1*›
        by (*auto simp add*: *assume-def lift-state-set-def* ‹*preL* = *lift-state-set pre*› ‹*b1*=*b*›)
    qed
      from ‹∀ *s*. (*s*, *s*) ∈ *guar*› ‹*guarL* = *lift-state-pair-set guar*› **have** ‹∀ *S*. (*S*,*S*)∈*guarL*›
      using *lift-state-pair-set-def* **by** *blast*
    from *CptsModWhileTMore*(*11*) *1* **have** ‹(*P*, *s*, *x*) # *cs* ∈ *commit* (*estran* Γ) {*fin*} *guarL preL*› **by** *blast*
    from *lift-seq-commit*[*OF this*]
    have *2*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *commit* (*estran* Γ) {*fin*} *guarL preL*› **by** *blast*
    have ‹*P*≠*fin*›
    proof
      assume ‹*P*=*fin*›
        with ‹*P1*=*P*› *CptsModWhileTMore*(*2*) **have** ‹(*fin*, *s*, *x*) # *cs* ∈ *cpts* (*estran* Γ)› **by** *simp*
        from *all-fin-after-fin*[*OF this*] **have** ‹*fst* (*last* ((*fin*,*s*,*x*)#*cs*)) = *fin*› **by** *simp*
      with *CptsModWhileTMore*(*3*) *no-estran-from-fin* **show** *False*
        by (*metis* ‹*P* = *fin*› ‹*P1* = *P*› *prod.collapse*)
    qed
    show *?thesis*
      apply *simp*
      apply(*rule commit-Cons-comp*)
        apply(*rule 2*[*simplified*])
      apply(*simp add*: *estran-def*)
       apply(*rule exI*)
       apply(*rule EWhileT*)
      using ‹*s*∈*b1*› **apply**(*simp add*: ‹*b1*=*b*›)
       apply(*rule* ‹*P*≠*fin*›)
      using ‹∀ *S*. (*S*,*S*)∈*guarL*› **by** *blast*
  qed
  have *guar*: ‹(*snd* (*last* ((*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*))), *snd* (*EWhile b P*, *t*, *y*)) ∈ *guarL*›

proof−
from *CptsModWhileTMore*(*3*)
have *tran*: ‹(*last* ((*P1*, *s*, *x*) # *cs*), (*fin*, *t*, *y*)) ∈ *estran* Γ›
apply(*simp only*: *estran-def*) **by** *blast*
thm *CptsModWhileTMore*
have *1*: ‹(*P*,*s*,*x*)#*cs*@[(*fin*,*t*,*y*)] ∈ *cpts-from* (*estran* Γ) (*P*,*s*,*x*) ∩ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*›
proof
show ‹(*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)] ∈ *cpts-from* (*estran* Γ) (*P*, *s*, *x*)›
proof(*simp*)
show ‹(*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)] ∈ *cpts* (*estran* Γ)›
using *CptsModWhileTMore*(*2*) **apply**(*auto simp add*: ‹*P1*=*P*› *cpts-def'*)
apply(*erule-tac x*=*i* **in** *allE*)
apply(*case-tac* ‹*i*=*length cs*›; *simp*)
using *tran* ‹*P1*=*P*› **apply**(*simp add*: *nth-length-last*)
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI One-nat-def add.commute less-antisym list.size*(*4*) *nth-append plus-1-eq-Suc*)
qed
next
have *1*: ‹*fst* (((*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *length cs*) ≠ *fst* ((*cs* @ [(*fin*, *t*, *y*)]) ! *length cs*)›
apply(*subst append-Cons*[*symmetric*])
apply(*subst nth-append*)
apply *simp*
using *no-fin-in-unfinished*[*OF CptsModWhileTMore*(*2*,*3*)] ‹*P1*=*P*› **by** *simp*
from *a* **have** ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) @ (*EWhile b P*, *t*, *y*) # *cs'* ∈ *assume preL relyL*›
using *assume-tl-env* **by** *fastforce*
then have ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*›
using *assume-appendD* **by** *fastforce*
then have ‹((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*›
using *unlift-seq-assume* **by** *fast*
then show ‹(*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)] ∈ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*›
apply(*auto simp add*: *assume-def*)
using ‹*s*∈*b1*› **apply**(*simp add*: *lift-state-set-def* ‹*preL* = *lift-state-set pre*› ‹*b1*=*b*›)
apply(*case-tac* ‹*i*=*length cs*›)
using *1* **apply** *blast*
apply(*erule-tac x*=*i* **in** *allE*)
apply(*subst append-Cons*[*symmetric*])
apply(*subst nth-append*) **apply**(*subst nth-append*)
apply *simp*
apply(*subst*(*asm*) *append-Cons*[*symmetric*])
apply(*subst*(*asm*) *nth-append*) **apply**(*subst*(*asm*) *nth-append*)
apply *simp*
done

qed
with *CptsModWhileTMore(11)* **have** ‹$(P,s,x)\#cs@[(fin,t,y)] \in$ *commit* (*estran* Γ) {*fin*} *guarL preL*› **by** *blast*
then show *?thesis*
apply(*auto simp add*: *commit-def*)
using *tran* ‹*P1=P*› **apply** *simp*
apply(*erule allE*[**where** *x*=‹*length cs*›])
using *tran* **by** (*simp add*: *nth-append last-map lift-seq-esconf-def case-prod-unfold last-conv-nth*)
qed
have ‹((*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*)) @ (*EWhile b P*, *t*, *y*) # *cs'* ∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
using *commit-append*[*OF part1-commit guar part2-commit*] **.**
then show *?thesis* **by** *simp*
qed
qed
next
case (*CptsModWhileTOnePartial s b1 P1 x cs*)
have *guar-refl'*: ‹∀ *S*. (*S*,*S*)∈*guarL*›
using ‹∀ *s*. (*s*,*s*)∈*guar*› ‹*guarL* = *lift-state-pair-set guar*› *lift-state-pair-set-def* **by** *auto*
show *?case*
proof(*rule allI*, *rule allI*, *clarify*)
assume ‹*P1=P*› ‹*b1=b*›
assume *a*: ‹(*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*›
have *1*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
proof−
have ‹((*P*, *s*, *x*) # *cs*) ∈ *commit* (*estran* Γ) {*fin*} *guarL preL*›
proof−
have ‹((*P*, *s*, *x*) # *cs*) ∈ *cpts-from* (*estran* Γ) (*P*, *s*, *x*) ∩ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*›
proof
show ‹(*P*, *s*, *x*) # *cs* ∈ *cpts-from* (*estran* Γ) (*P*, *s*, *x*)› **using** ‹(*P1*, *s*, *x*) # *cs* ∈ *cpts* (*estran* Γ)› ‹*P1=P*› **by** *simp*
next
show ‹(*P*, *s*, *x*) # *cs* ∈ *assume* (*lift-state-set* (*pre* ∩ *b*)) *relyL*›
proof−
from *a* **have** ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*›
by (*auto simp add*: *assume-def*)
from *unlift-seq-assume*[*OF this*] **have** ‹((*P*, *s*, *x*) # *cs*) ∈ *assume preL relyL*› **.**
then show *?thesis*
proof(*auto simp add*: *assume-def lift-state-set-def* ‹*preL* = *lift-state-set pre*›)
show ‹*s*∈*b*› **using** ‹*s*∈*b1*› ‹*b1=b*› **by** *simp*
qed

```
        qed
       qed
        with ‹ ∀S0. cpts-from (estran Γ) (P, S0) ∩ assume (lift-state-set (pre ∩
b)) relyL ⊆ commit (estran Γ) {fin} guarL preL›
       show ?thesis by blast
      qed
      then show ?thesis using while-sound-aux3 by blast
    qed
    show ‹(EWhile b P, s, x) # map (lift-seq-esconf (EWhile b P)) ((P, s, x) #
cs) ∈ commit (estran Γ) {fin} guarL postL›
      apply(auto simp add: commit-def)
      using guar-refl' apply blast
       apply(case-tac i; simp)
      using guar-refl' apply blast
      using 1 apply(simp add: commit-def)
      apply(simp add: last-conv-nth lift-seq-esconf-def case-prod-unfold) .
  qed
next
  case (CptsModWhileTOneFull s b1 P1 x cs a t y cs')
  have guar-refl': ‹∀S. (S,S)∈guarL›
    using ‹∀s. (s,s)∈guar› ‹guarL = lift-state-pair-set guar› lift-state-pair-set-def
by auto
  show ?case
  proof(rule allI, rule allI, clarify)
    assume ‹P1=P› ‹b1=b›
    assume a: ‹(EWhile b P, s, x) # map (lift-seq-esconf (EWhile b P)) ((P, s,
x) # cs) @ map (λ(-, s, x). (EWhile b P, s, x)) ((fin, t, y) # cs') ∈ assume preL
relyL›
    have 1: ‹map (lift-seq-esconf (EWhile b P)) ((P, s, x) # cs) @ map (λ(-, s,
x). (EWhile b P, s, x)) ((fin, t, y) # cs')
      ∈ commit (estran Γ) {fin} guarL postL›
    proof-
      have 1: ‹((P, s, x) # cs) @ ((fin, t, y) # cs') ∈ commit (estran Γ) {fin}
guarL preL›
      proof-
        let ?c = ‹((P, s, x) # cs) @ ((fin, t, y) # cs')›
        have ‹?c ∈ cpts-from (estran Γ) (P,s,x) ∩ assume (lift-state-set (pre ∩ b))
relyL›
        proof
          show ‹((P, s, x) # cs) @ (fin, t, y) # cs' ∈ cpts-from (estran Γ) (P, s,
x)›
          proof(simp)
            note part1 = CptsModWhileTOneFull(2)
            from CptsModWhileTOneFull(4) cpts-es-mod-equiv
            have part2: ‹(fin, t, y) # cs' ∈ cpts (estran Γ)› by blast
            from CptsModWhileTOneFull(3)
            have tran: ‹(last ((P1, s, x) # cs), (fin, t, y)) ∈ estran Γ›
              apply(subst estran-def) by blast
            show ‹(P, s, x) # cs @ (fin, t, y) # cs' ∈ cpts (estran Γ)›
```

```
        using cpts-append-comp[OF part1 part2] tran ‹P1=P› by force
      qed
    next
      from assume-appendD[OF assume-tl-env[OF a[simplified]]]
        have ‹map (lift-seq-esconf (EWhile b P)) ((P,s,x)#cs) ∈ assume preL
relyL› by simp
        from unlift-seq-assume[OF this] have part1: ‹(P, s, x) # cs ∈ assume
preL relyL› .
      have part2: ‹∀ i. Suc i < length ((fin,t,y)#cs') ⟶ (snd (((fin,t,y)#cs')!i),
snd (((fin,t,y)#cs')!Suc i)) ∈ relyL›
      proof−
        from CptsModWhileTOneFull(4) cpts-es-mod-equiv
        have part2-cpt: ‹(fin, t, y) # cs' ∈ cpts (estran Γ)› by blast
        let ?c2 = ‹map (λ(-, s, x). (EWhile b P, s, x)) ((fin, t, y) # cs')›
        from assume-appendD2[OF a[simplified append-Cons[symmetric]]]
      have 1: ‹∀ i. Suc i < length ?c2 ⟶ (snd (?c2!i), snd (?c2!Suc i))∈relyL›
          apply(auto simp add: assume-def case-prod-unfold)
          apply(erule-tac x=i in allE)
          by (simp add: nth-Cons')
        show ?thesis
        proof(rule allI, rule impI)
          fix i
          assume a1: ‹Suc i < length ((fin, t, y) # cs')›
          then have ‹i<length cs'› by simp
          from 1 have ‹∀ i. i < length cs' ⟶
    (snd (map (λ(-, s, x). (EWhile b P, s, x)) ((fin, t, y) # cs') ! i), snd (map
(λ(-, s, x). (EWhile b P, s, x)) ((fin, t, y) # cs') ! Suc i)) ∈ relyL›
            by simp
          from this[rule-format, OF ‹i<length cs'›]
          show ‹(snd (((fin, t, y) # cs') ! i), snd (((fin, t, y) # cs') ! Suc i)) ∈
relyL›
            apply(simp only: nth-map[OF ‹i<length cs'›] nth-map[OF a1[THEN
Suc-lessD]] nth-map[OF a1] case-prod-unfold)
            by simp
        qed
      qed
      from CptsModWhileTOneFull(3)
      have tran: ‹(last ((P1, s, x) # cs), (fin, t, y)) ∈ estran Γ›
        apply(subst estran-def) by blast
    from assume-append[OF part1] part2 ctran-imp-not-etran[OF tran[simplified
‹P1=P›]]
      have ‹((P, s, x) # cs) @ (fin,t,y) # cs' ∈ assume preL relyL› by blast
      then show ‹((P, s, x) # cs) @ (fin, t, y) # cs' ∈ assume (lift-state-set
(pre ∩ b)) relyL›
          using ‹s∈b1› by (simp add: assume-def lift-state-set-def ‹preL =
lift-state-set pre› ‹b1=b›)
    qed
    with CptsModWhileTOneFull(11) show ?thesis by blast
  qed
```

show *?thesis*
apply(*auto simp add*: *commit-def*)
using *1* **apply**(*simp add*: *commit-def*)
apply *clarify*
apply(*erule-tac x=i* **in** *allE*)
subgoal for *i*
proof−
assume *a*: ‹*i* < *Suc* (*length cs*) ⟶ (((*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*, (*cs* @ [(*fin*, *t*, *y*)]) ! *i*) ∈ *estran* Γ ⟶ (*snd* (((*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*), *snd* ((*cs* @ [(*fin*, *t*, *y*)]) ! *i*)) ∈ *guarL*›
assume *1*: ‹*i* < *Suc* (*length cs*)›
assume *a3*: ‹(((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]) ! *i*, (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]) ! *i*)
∈ *estran* Γ›
have *2*: ‹(((*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*, (*cs* @ [(*fin*, *t*, *y*)]) ! *i*) ∈ *estran* Γ›
proof−
from *a3* **have** *a3'*: ‹((*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) @ [(*EWhile b P*, *t*, *y*)]) ! *i*, (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]) ! *i*)
∈ *estran* Γ› **by** *simp*
have *eq1*:
‹(*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) @ [(*EWhile b P*, *t*, *y*)]) ! *i* =
(*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*)) ! *i*›
using *1* **by** (*simp add*: *nth-append del*: *list.map*)
show *?thesis*
proof(*cases* ‹*i*=*length cs*›)
case *True*
let *?c* = ‹((*P*, *s*, *x*) # *cs*) ! *length cs*›
from *a3'* **show** *?thesis*
apply(*simp add*: *eq1 nth-append True del*: *list.map*)
apply(*subst append-Cons*[*symmetric*])
apply(*simp add*: *nth-append del*: *append-Cons*)
apply(*simp add*: *lift-seq-esconf-def case-prod-unfold*)
apply(*simp add*: *estran-def*)
apply(*erule exE*)
apply(*rule exI*)
apply(*erule estran-p.cases*, *auto*)[]
apply(*subst surjective-pairing*[*of ?c*])
by *auto*
next
case *False*
with ‹*i*<*Suc* (*length cs*)› **have** ‹*i* < *length cs*› **by** *simp*
have *eq2*:
‹(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]) ! *i* =
(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) ! *i*›
using ‹*i*<*length cs*› **by** (*simp add*: *nth-append*)

from *a3'* **show** *?thesis*
using ‹*i*<*length cs*› **apply**(*simp add*: *eq1 eq2 nth-append del*: *list.map*)
apply(*subst append-Cons*[*symmetric*])
apply(*simp add*: *nth-append del*: *append-Cons*)
apply(*simp add*: *lift-seq-esconf-def case-prod-unfold*)
using *seq-tran-inv* **by** *fastforce*
qed
qed
from *a*[*rule-format*, *OF 1 2*] **have**
‹(*snd* (((*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]) ! *i*), *snd* ((*cs* @ [(*fin*, *t*, *y*)]) ! *i*)) ∈ *guarL*› **.**
then have
‹(((*s*,*x*) # *map snd cs* @ [(*t*,*y*)])!*i*, (*map snd cs* @ [(*t*,*y*)])!*i*) ∈ *guarL*›
using *1 nth-map*[*of i* ‹(*P*, *s*, *x*) # *cs* @ [(*fin*, *t*, *y*)]› *snd*] *nth-map*[*of i* ‹*cs* @ [(*fin*, *t*, *y*)]› *snd*] **by** *simp*
then have
‹(((*s*,*x*) # *map snd* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) @ [(*t*,*y*)])!*i*, (*map snd* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) @ [(*t*,*y*)])!*i*) ∈ *guarL*›
proof−
assume *a*: ‹(((*s*, *x*) # *map snd cs* @ [(*t*, *y*)]) ! *i*, (*map snd cs* @ [(*t*, *y*)]) ! *i*) ∈ *guarL*›
have *aux*[*rule-format*]: ‹∀ *f*. *map* (*snd* ∘ (λ*uu*. (*f uu*, *snd uu*))) *cs* = *map snd cs*› **by** *simp*
from *a* **show** *?thesis* **by** (*simp add*: *lift-seq-esconf-def case-prod-unfold aux*)
qed
then show *?thesis*
using *1 nth-map*[*of i* ‹(*P* *NEXT* *EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]› *snd*]
nth-map[*of i* ‹*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ [(*EWhile b P*, *t*, *y*)]› *snd*]
by *simp*
qed
using *1* **apply**(*simp add*: *commit-def*)
apply *clarify*
apply(*erule-tac x*=*i* **in** *allE*)
subgoal for *i*
proof−
assume *a*: ‹*i* < *Suc* (*length cs* + *length cs'*) ⟶ (((*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i*, (*cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i*) ∈ *estran* Γ ⟶
(*snd* (((*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i*), *snd* ((*cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i*)) ∈ *guarL*›
assume *1*: ‹*i* < *Suc* (*length cs* + *length cs'*)›
assume ‹(((*P* *NEXT* *EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i*,
(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i*)
∈ *estran* Γ›
then have *2*: ‹(((*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i*, (*cs* @ (*fin*, *t*, *y*)

cs') ! *i*) ∈ *estran* Γ›
apply(*cases* ‹*i* < *length cs*›; *simp*)
subgoal
proof−
assume *a1*: ‹*i* < *length cs*›
assume *a2*: ‹(((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i*,
(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i*)
∈ *estran* Γ›
have *aux*[*rule-format*]: ‹∀ *x xs y ys*. *i* < *length xs* ⟶ (*x*#*xs*@*y*#*ys*)!*i* = (*x*#*xs*)!*i*›
by (*metis add-diff-cancel-left' less-SucI less-Suc-eq-0-disj nth-Cons' nth-append plus-1-eq-Suc*)
from *a1* **have** *a1'*: ‹*i* < *length* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*)› **by** *simp*
have *a2'*: ‹(((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs*)!*i*, (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*)!*i*) ∈ *estran* Γ›
proof−
have *1*: ‹((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i*
=
((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) ! *i*› **using** *aux*[*OF a1'*] **.**
have *2*: ‹(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *i* =
map (*lift-seq-esconf* (*EWhile b P*)) *cs* ! *i*› **using** *a1'* **by** (*simp add*: *nth-append*)
from *a2* **show** *?thesis* **by** (*simp add*: *1 2*)
qed
thm *seq-tran-inv*
have ‹(((*P*, *s*, *x*) # *cs*) ! *i*, *cs* ! *i*) ∈ *estran* Γ›
proof−
from *a2'* **have** *a2''*: ‹((*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*)) ! *i*, *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* ! *i*) ∈ *estran* Γ› **by** *simp*
obtain *P1 S1* **where** *1*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*) ! *i* = (*P1 NEXT EWhile b P*, *S1*)›
proof−
assume *a*: ‹⋀*P1 S1*. *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) ! *i* = (*P1 NEXT EWhile b P*, *S1*) ⟹ *thesis*›
have *a1'*: ‹*i* < *length* ((*P*,*s*,*x*)#*cs*)› **using** *a1* **by** *auto*
show *thesis* **apply**(*rule a*) **apply**(*subst nth-map*[*OF a1'*]) **by** (*simp add*: *lift-seq-esconf-def case-prod-unfold*)
qed
obtain *P2 S2* **where** *2*: ‹*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* ! *i* = (*P2 NEXT EWhile b P*, *S2*)›
proof−
assume *a*: ‹⋀*P2 S2*. *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* ! *i* = (*P2 NEXT EWhile b P*, *S2*) ⟹ *thesis*›
show *thesis* **apply**(*rule a*) **apply**(*subst nth-map*[*OF a1*]) **by** (*simp*

add: *lift-seq-esconf-def case-prod-unfold*)
qed
have *tran*: ‹$((P1,S1),(P2,S2)) \in$ *estran* Γ› **using** *seq-tran-inv a2'' 1 2* **by** *metis*
have *aux*[*rule-format*]: ‹$\forall$ *Q P S cs i*. *map* (*lift-seq-esconf Q*) *cs* ! *i* = (*P NEXT Q*,*S*) $\longrightarrow$ *i* < *length cs* $\longrightarrow$ *cs*!*i* = (*P*,*S*)›
apply(*rule allI*)+ **apply** *clarify* **apply**(*simp add*: *lift-seq-esconf-def case-prod-unfold nth-map*[*OF a1*])
using *surjective-pairing* **by** *metis*
have *3*: ‹((*P*, *s*, *x*) # *cs*) ! *i* = (*P1*,*S1*)› **using** *aux*[*OF 1*] *a1* **by** *auto*
have *4*: ‹*cs*!*i* = (*P2*,*S2*)› **using** *aux*[*OF 2 a1*] **.**
show *?thesis* **using** *tran 3 4* **by** *argo*
qed
moreover have ‹((*P*, *s*, *x*) # *cs*) ! *i* = (((*P*, *s*, *x*) # *cs*) @ (*fin*, *t*, *y*) # *cs'*) ! *i*› **using** *a1* **by** (*simp add*: *aux*)
moreover have ‹(*cs* @ (*fin*, *t*, *y*) # *cs'*) ! *i* = *cs*!*i*› **using** *a1* **by** (*simp add*: *nth-append*)
ultimately show *?thesis* **by** *simp*
qed
apply(*cases* ‹*i* = *length cs*›; *simp*)
subgoal
proof−
assume *a*: ‹(((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *length cs*,
(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *length cs*)
$\in$ *estran* Γ›
have *1*: ‹((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *length cs* = ((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) ! *length cs*›
by (*metis append-Nil2 length-map nth-length-last*)
have *2*: ‹(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs'*) ! *length cs* =
(*EWhile b P*, *t*, *y*)›
by (*metis* (*no-types*, *lifting*) *map-eq-imp-length-eq map-ident nth-append-length*)
from *a* **have** *a'*: ‹(((*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) ! *length cs*, (*EWhile b P*, *t*, *y*)) $\in$ *estran* Γ›
by (*simp add*: *1 2*)
obtain *P1 S1* **where** *3*: ‹(*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*)) ! *length cs* = (*P1 NEXT EWhile b P*,*S1*)›
proof−
assume *a*: ‹$\bigwedge$*P1 S1*. (*map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*,*s*,*x*)#*cs*)) ! *length cs* = (*P1 NEXT EWhile b P*, *S1*) $\Longrightarrow$ *thesis*›
have *1*: ‹*length cs* < *length* ((*P*,*s*,*x*)#*cs*)› **by** *simp*
show *thesis* **apply**(*rule a*) **apply**(*subst nth-map*[*OF 1*]) **by** (*simp*

add: *lift-seq-esconf-def case-prod-unfold*)
qed
from *a′ seq-tran-inv-fin 3* **have** ‹((*P1 NEXT EWhile b P*,*S1*),(*EWhile b P*,*t*,*y*))∈*estran* Γ› **by** *auto*
moreover have ‹((*P*,*s*,*x*)#*cs*) ! *length cs* = (*P1*,*S1*)›
proof−
have ∗: ‹*length cs* < *length* ((*P*,*s*,*x*)#*cs*)› **by** *simp*
show *?thesis* **using** *3*
apply(*simp only*: *lift-seq-esconf-def case-prod-unfold*)
apply(*subst* (*asm*) *nth-map*[*OF* ∗])
by *auto*
qed
moreover have ‹((*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs′*) ! *length cs* = ((*P*, *s*, *x*) # *cs*) ! *length cs*›
by (*metis append-Nil2 nth-length-last*)
ultimately show *?thesis* **using** *seq-tran-inv-fin* **by** *metis*
qed
subgoal
proof−
assume *a1*: ‹¬ *i* < *length cs*›
assume *a2*: ‹((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *Suc 0*),
(*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! *i*)
∈ *estran* Γ›
assume *a3*: ‹*i* ≠ *length cs*›
from *a1 a3* **have** ‹*i*>*length cs*› **by** *simp*
have *1*: ‹((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *Suc 0*)) =
((*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *Suc 0* − *length cs*)›
by (*metis* (*no-types*, *lifting*) *Suc-pred* ‹*length cs* < *i*› *a1 length-map less-Suc-eq-0-disj less-antisym nth-append*)
have *2*: ‹((*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! *i*) =
((*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *length cs*)›
by (*simp add*: *a1 nth-append*)
from *a2* **have** *a2′*: ‹((((*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *Suc 0* − *length cs*)), (((*EWhile b P*, *t*, *y*) # *map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*) ! (*i* − *length cs*))) ∈ *estran* Γ›
by (*simp add*: *1 2*)
note *i-lt* = ‹*i* < *Suc* (*length cs* + *length cs′*)›
obtain *S1* **where** *3*: ‹((*map* (λ(-, *y*). (*EWhile b P*, *y*)) ((*fin*,*t*,*y*)#*cs′*)) ! (*i* − *Suc 0* − *length cs*)) = (*EWhile b P*, *S1*)›
proof−
assume *a*: ‹⋀*S1*. *map* (λ(-, *y*). (*EWhile b P*, *y*)) ((*fin*, *t*, *y*) # *cs′*) ! (*i* − *Suc 0* − *length cs*) = (*EWhile b P*, *S1*) ⟹ *thesis*›
have ∗: ‹*i* − *Suc 0* − *length cs* < *length* ((*fin*,*t*,*y*)#*cs′*)› **using** *i-lt* **by** *simp*

show *thesis* **apply**(*rule a*) **apply**(*subst nth-map*[*OF* ∗]) **by** (*simp add*: *case-prod-unfold*)
qed
obtain *S2* **where** *4*: ‹(*map* (λ(-, *y*). (*EWhile b P*, *y*)) ((*fin*,*t*,*y*)#*cs*′)) ! (*i* − *length cs*) = (*EWhile b P*, *S2*)›
proof−
assume *a*: ‹⋀*S2*. (*map* (λ(-, *y*). (*EWhile b P*, *y*)) ((*fin*, *t*, *y*) # *cs*′)) ! (*i* − *length cs*) = (*EWhile b P*, *S2*) ⟹ *thesis*›
have ∗: ‹*i* − *length cs* < *length* ((*fin*,*t*,*y*)#*cs*′)› **using** *i-lt* **by** *simp*
show *thesis* **apply**(*rule a*) **apply**(*subst nth-map*[*OF* ∗]) **by** (*simp add*: *case-prod-unfold*)
qed
from *no-estran-to-self*′ *a2*′ *3 4* **have** *False* **by** *fastforce*
then show *?thesis* **by** (*rule FalseE*)
qed
done
from *a*[*rule-format*, *OF 1 2*] **have** ‹(*snd* (((*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs*′) ! *i*), *snd* ((*cs* @ (*fin*, *t*, *y*) # *cs*′) ! *i*)) ∈ *guarL*› **.**

then have ‹(((*s*,*x*) # *map snd cs* @ (*t*,*y*) # *map snd cs*′)!*i*, (*map snd cs* @ (*t*,*y*) # *map snd cs*′)!*i*) ∈ *guarL*›
using *1 nth-map*[*of i* ‹(*P*, *s*, *x*) # *cs* @ (*fin*, *t*, *y*) # *cs*′› *snd*] *nth-map*[*of i* ‹*cs* @ (*fin*, *t*, *y*) # *cs*′› *snd*] **by** *simp*
then have ‹(((*s*,*x*) # *map snd* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) @ (*t*,*y*) # *map snd* (*map* (λ(-,*S*). (*EWhile b P*, *S*)) *cs*′))!*i*, (*map snd* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) @ (*t*,*y*) # *map snd* (*map* (λ(-,*S*). (*EWhile b P*, *S*)) *cs*′))!*i*) ∈ *guarL*›
proof−
assume ‹(((*s*,*x*) # *map snd cs* @ (*t*,*y*) # *map snd cs*′)!*i*, (*map snd cs* @ (*t*,*y*) # *map snd cs*′)!*i*) ∈ *guarL*›
moreover have ‹*map snd* (*map* (*lift-seq-esconf* (*EWhile b P*)) *cs*) = *map snd cs*› **by** *auto*
moreover have ‹*map snd* (*map* (λ(-, *S*). (*EWhile b P*, *S*)) *cs*′) = *map snd cs*′› **by** *auto*
ultimately show *?thesis* **by** *metis*
qed
then show *?thesis*
using *1 nth-map*[*of i* ‹(*P NEXT EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-,*S*). (*EWhile b P*, *S*)) *cs*′› *snd*] *nth-map*[*of i* ‹*map* (*lift-seq-esconf* (*EWhile b P*)) *cs* @ (*EWhile b P*, *t*, *y*) # *map* (λ(-,*S*). (*EWhile b P*, *S*)) *cs*′› *snd*]
by *simp*
qed
apply(*rule FalseE*) **by** (*simp add*: *last-conv-nth case-prod-unfold*)
qed
show ‹(*EWhile b P*, *s*, *x*) # *map* (*lift-seq-esconf* (*EWhile b P*)) ((*P*, *s*, *x*) # *cs*) @ *map* (λ(-, *s*, *x*). (*EWhile b P*, *s*, *x*)) ((*fin*, *t*, *y*) # *cs*′)

∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
apply(*auto simp add*: *commit-def*)
apply(*case-tac i*; *simp*)
using *guar-refl′* **apply** *blast*
using *1* **apply**(*simp add*: *commit-def*)
apply(*case-tac i*; *simp*)
using *1* **apply**(*simp add*: *commit-def*)
using *guar-refl′* **apply** *blast*
using *1* **apply**(*simp add*: *commit-def*)
subgoal
proof−
assume ‹*cs′*≠[]› ‹*fst* (*last* (*map* (λ(-, *y*). (*EWhile b P*, *y*)) *cs′*)) = *fin*›
then have *False* **by** (*simp add*: *last-conv-nth case-prod-unfold*)
then show *?thesis* **by** *blast*
qed.
qed
next
case (*CptsModWhileF s b1 x cs P1*)
have *cpt*: ‹((*fin*, *s*, *x*) # *cs*) ∈ *cpts* (*estran* Γ)› **using** ‹((*fin*, *s*, *x*) # *cs*) ∈ *cpts-es-mod* Γ› *cpts-es-mod-equiv* **by** *blast*

show *?case*
proof(*rule allI*, *rule allI*, *clarify*)
assume ‹*P1*=*P*› ‹*b1*=*b*›
assume *a*: ‹(*EWhile b P*, *s*, *x*) # (*fin*, *s*, *x*) # *cs* ∈ *assume preL relyL*›
then have ‹*s*∈*pre*› **by** (*simp add*: *assume-def lift-state-set-def* ‹*preL* = *lift-state-set pre*›)

show ‹(*EWhile b P*, *s*, *x*) # (*fin*, *s*, *x*) # *cs* ∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
proof−
have *1*: ‹(*fin*, *s*, *x*) # *cs* ∈ *commit* (*estran* Γ) {*fin*} *guarL postL*›
proof−
have *1*: ‹(*s*,*x*)∈*postL*›
proof−
have ‹*s*∈*post*› **using** ‹*s*∈*pre*› ‹*pre*∩−*b*⊆*post*› ‹*s*∉*b1*› ‹*b1*=*b*› **by** *blast*
then show *?thesis* **using** ‹*postL* = *lift-state-set post*› **by** (*simp add*: *lift-state-set-def*)
qed
have *guar-refl′*: ‹∀ *S*. (*S*,*S*)∈*guarL*›
using ‹∀ *s*. (*s*,*s*)∈*guar*› ‹*guarL* = *lift-state-pair-set guar*› *lift-state-pair-set-def* **by** *auto*
have *all-etran*: ‹∀ *i*. *Suc i* < *length* ((*fin*, *s*, *x*) # *cs*) ⟶ ((*fin*, *s*, *x*) # *cs*) ! *i* −*e*→ ((*fin*, *s*, *x*) # *cs*) ! *Suc i*›
using *all-etran-from-fin*[*OF cpt*] **by** *blast*
show *?thesis*
proof(*auto simp add*: *commit-def 1*)
fix *i*
assume ‹*i*<*length cs*›

```
        assume a: ‹(((fin, s, x) # cs) ! i, cs ! i) ∈ estran Γ›
        have False
        proof−
          from ctran-or-etran[OF cpt] ‹i<length cs› a all-etran
          show False by simp
        qed
        then show ‹(snd (((fin, s, x) # cs) ! i), snd (cs ! i)) ∈ guarL› by blast
      next
        assume ‹cs≠[]›
        thm while-sound-aux2
        show ‹snd (last cs) ∈ postL›
        proof−
         have 1: ‹stable postL relyL› using ‹stable post rely› ‹postL = lift-state-set
post› ‹relyL = lift-state-pair-set rely›
            by (simp add: lift-state-set-def lift-state-pair-set-def stable-def)
          have 2: ‹∀ i. Suc i < length ((fin, s, x) # cs) ⟶
    ((fin, s, x) # cs) ! i −e→ ((fin, s, x) # cs) ! Suc i ⟶ (snd (((fin, s, x) #
cs) ! i), snd (((fin, s, x) # cs) ! Suc i)) ∈ relyL›
            using a
            apply(simp add: assume-def)
            apply(rule allI)
            apply(erule conjE)
            apply(erule-tac x=‹Suc i› in allE)
            by simp
          have ‹snd (last ((fin, s, x) # cs)) ∈ postL› using while-sound-aux2[OF
1 ‹(s,x)∈postL› all-etran 2] .
          then show ?thesis using ‹cs≠[]› by simp
        qed
      qed
    qed
    have 2: ‹((EWhile b P, s, x), (fin, s, x)) ∈ estran Γ›
      apply(simp add: estran-def)
      apply(rule exI)
      apply(rule EWhileF)
      using ‹s∉b1› ‹b1=b› by simp
      from ‹∀ s. (s, s) ∈ guar› ‹guarL = lift-state-pair-set guar› have 3: ‹∀ S.
(S,S)∈guarL›
      using lift-state-pair-set-def by auto
    from commit-Cons-comp[OF 1 2 3[rule-format]] show ?thesis .
  qed
 qed
qed
```

```
theorem While-sound:
  ‹⟦ stable pre rely; (pre ∩ −b) ⊆ post; stable post rely;
   Γ ⊨ P sat_e [pre ∩ b, rely, guar, pre]; ∀ s. (s,s)∈guar ⟧ ⟹
   Γ ⊨ EWhile b P sat_e [pre, rely, guar, post]›
 apply(unfold es-validity-def validity-def)
```

proof−
  let *?pre* = ‹*lift-state-set pre*›
  let *?rely* = ‹*lift-state-pair-set rely*›
  let *?guar* = ‹*lift-state-pair-set guar*›
  let *?post* = ‹*lift-state-set post*›

  assume *stable-pre*: ‹*stable pre rely*›
  assume *pre-post*: ‹*pre* ∩ −*b* ⊆ *post*›
  assume *stable-post*: ‹*stable post rely*›
  assume *P-valid*: ‹∀ *S0*. *cpts-from* (*estran* Γ) (*P*, *S0*) ∩ *assume* (*lift-state-set* (*pre* ∩ *b*)) *?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?pre*›
  assume *guar-refl*: ‹∀ *s*. (*s*,*s*)∈*guar*›
  show ‹∀ *S0*. *cpts-from* (*estran* Γ) (*EWhile b P*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
  proof
    fix *S0*
    show ‹*cpts-from* (*estran* Γ) (*EWhile b P*, *S0*) ∩ *assume ?pre ?rely* ⊆ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
    proof
      fix *cpt*
      assume *cpt-from-assume*: ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*EWhile b P*, *S0*) ∩ *assume ?pre ?rely*›
      then have *cpt*:
        ‹*cpt* ∈ *cpts* (*estran* Γ)› **and** *cpt-assume*:
        ‹*cpt* ∈ *assume ?pre ?rely*› **by** *auto*
      from *cpt-from-assume* **have** ‹*cpt* ∈ *cpts-from* (*estran* Γ) (*EWhile b P*, *S0*)› **by** *blast*
      then have ‹*hd cpt* = (*EWhile b P*, *S0*)› **by** *simp*
      moreover from *cpt cpts-nonnil* **have** ‹*cpt*≠[]› **by** *blast*
      ultimately obtain *cs* **where** *1*: ‹*cpt* = (*EWhile b P*, *S0*) # *cs*› **by** (*metis hd-Cons-tl*)
      from *cpt cpts-es-mod-equiv* **have** *cpt-mod*:
        ‹*cpt* ∈ *cpts-es-mod* Γ› **by** *blast*
      obtain *preL* :: ‹('*s* × ('*a*,'*b*,'*s*,'*prog*) *ectx*) *set*› **where** *preL*: ‹*preL* = *?pre*› **by** *simp*
      obtain *relyL* :: ‹('*s* × ('*a*,'*b*,'*s*,'*prog*) *ectx*) *tran set*› **where** *relyL*: ‹*relyL* = *?rely*› **by** *simp*
      obtain *guarL* :: ‹('*s* × ('*a*,'*b*,'*s*,'*prog*) *ectx*) *tran set*› **where** *guarL*: ‹*guarL* = *?guar*› **by** *simp*
      obtain *postL* :: ‹('*s* × ('*a*,'*b*,'*s*,'*prog*) *ectx*) *set*› **where** *postL*: ‹*postL* = *?post*› **by** *simp*
      show ‹*cpt* ∈ *commit* (*estran* Γ) {*fin*} *?guar ?post*›
      using *while-sound-aux*[*OF cpt-mod preL relyL guarL postL pre-post - guar-refl stable-pre stable-post*, *THEN spec*[**where** *x*=*S0*], *THEN spec*[**where** *x*=*cs*], *rule-format*] *P-valid 1 cpt-assume preL relyL guarL postL* **by** *blast*
    qed
  qed
qed

lemma *lift-seq-assume*:
  ‹$cs \neq [] \Longrightarrow cs \in assume\ pre\ rely \longleftrightarrow lift\text{-}seq\text{-}cpt\ P\ cs \in assume\ pre\ rely$›
  by (*auto simp add*: *assume-def lift-seq-esconf-def case-prod-unfold hd-map*)

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

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

| *Evt-Atom*:
  ‹$[\![\ \forall V.\ \Gamma \vdash body\ ev\ sat_p\ [pre \cap guard\ ev \cap \{V\},\ Id,\ UNIV,\ \{s.\ (V,s) \in guar\} \cap post];$
   $stable\ pre\ rely;\ stable\ post\ rely\ ]\!] \Longrightarrow$
   $\Gamma \vdash EAtom\ ev\ sat_e\ [pre,\ rely,\ guar,\ post]$›

| *Evt-Seq*:

  ‹$[\![\ \Gamma \vdash es1\ sat_e\ [pre,\ rely,\ guar,\ mid];\ \Gamma \vdash es2\ sat_e\ [mid,\ rely,\ guar,\ post]\ ]\!] \Longrightarrow$
   $\Gamma \vdash ESeq\ es1\ es2\ sat_e\ [pre,\ rely,\ guar,\ post]$›

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

| *Evt-Choice*:
  ‹$\Gamma \vdash P\ sat_e\ [pre,\ rely,\ guar,\ post] \Longrightarrow$
   $\Gamma \vdash Q\ sat_e\ [pre,\ rely,\ guar,\ post] \Longrightarrow$
   $\Gamma \vdash P\ OR\ Q\ sat_e\ [pre,\ rely,\ guar,\ post]$›

| *Evt-Join*:
  ‹$\Gamma \vdash P\ sat_e\ [pre1,\ rely1,\ guar1,\ post1] \Longrightarrow$
   $\Gamma \vdash Q\ sat_e\ [pre2,\ rely2,\ guar2,\ post2] \Longrightarrow$
    $pre \subseteq pre1 \cap pre2 \Longrightarrow$
    $rely \cup guar2 \subseteq rely1 \Longrightarrow$
    $rely \cup guar1 \subseteq rely2 \Longrightarrow$
    $\forall s.\ (s,s) \in guar \Longrightarrow$
    $guar1 \cup guar2 \subseteq guar \Longrightarrow$
    $post1 \cap post2 \subseteq post \Longrightarrow$
    $\Gamma \vdash EJoin\ P\ Q\ sat_e\ [pre,\ rely,\ guar,\ post]$›

| *Evt-While*:
  ‹$[\![\ stable\ pre\ rely;\ (pre \cap -b) \subseteq post;\ stable\ post\ rely;$
   $\Gamma \vdash P\ sat_e\ [pre \cap b,\ rely,\ guar,\ pre];\ \forall s.\ (s,s) \in guar\ ]\!] \Longrightarrow$

$\Gamma \vdash$ *EWhile b P* sat_e [*pre*, *rely*, *guar*, *post*]›

theorem *rghoare-es-sound*:
assumes *h*: $\Gamma \vdash$ *es* sat_e [*pre*, *rely*, *guar*, *post*]
shows $\Gamma \models$ *es* sat_e [*pre*, *rely*, *guar*, *post*]
using *h*
proof(*induct*)
case (*Evt-Anon* Γ *P pre rely guar post*)
then show *?case* **by**(*rule Anon-sound*)
next
case (*Evt-Basic* Γ *ev pre rely guar post*)
then show *?case* **using** *Basic-sound* **by** *blast*
next
case (*Evt-Atom* Γ *ev pre guar post rely*)
then show *?case* **using** *Atom-sound* **by** *blast*
next
case (*Evt-Seq* Γ *es1 pre rely guar mid es2 post*)
then show *?case* **using** *Seq-sound* **by** *blast*
next
case (*Evt-conseq pre pre′ rely rely′ guar′ guar post′ post* Γ *ev*)
then show *?case* **using** *conseq-sound* **by** *blast*
next
case *Evt-Choice*
then show *?case* **using** *Choice-sound* **by** *blast*
next
case (*Evt-Join* Γ *P pre1 rely1 guar1 post1 Q pre2 rely2 guar2 post2 pre rely guar post*)
then show *?case* **apply**−
apply(*rule conseq-sound*[*of* Γ - ‹*pre1*∩*pre2*› *rely guar* ‹*post1*∩*post2*›])
using *Join-sound-aux* **apply** *blast*
by *auto*
next
case *Evt-While*
then show *?case* **using** *While-sound* **by** *blast*
qed

inductive *rghoare-pes* :: [′*Env*, ′*k* ⇒ ((′*l*,′*k*,′*s*,′*prog*)*esys*,′*s*) *rgformula*, ′*s set*, (′*s* × ′*s*) *set*, (′*s* × ′*s*) *set*, ′*s set*] ⇒ *bool*
(- $\vdash$ - SAT_e [-, -, -, -] [*60,0,0,0,0,0*] *45*)
where
Par:
⟦ ∀*k*. $\Gamma \vdash$ *Com* (*prgf k*) sat_e [*Pre* (*prgf k*), *Rely* (*prgf k*), *Guar* (*prgf k*), *Post* (*prgf k*)];
∀*k*. *pre* ⊆ *Pre* (*prgf k*);
∀*k*. *rely* ⊆ *Rely* (*prgf k*);
∀*k j*. *j*≠*k* ⟶ *Guar* (*prgf j*) ⊆ *Rely* (*prgf k*);
∀*k*. *Guar* (*prgf k*) ⊆ *guar*;
(⋂*k*. (*Post* (*prgf k*))) ⊆ *post* ⟧ ⟹

$\Gamma \vdash prgf\ SAT_e\ [pre,\ rely,\ guar,\ post]$

lemma *Par-conseq*:
$[\![\ pre \subseteq pre';\ rely \subseteq rely';\ guar' \subseteq guar;\ post' \subseteq post;$
$\Gamma \vdash prgf\ SAT_e\ [pre',\ rely',\ guar',\ post']\ ]\!] \Longrightarrow$
$\Gamma \vdash prgf\ SAT_e\ [pre,\ rely,\ guar,\ post]$
apply(*erule rghoare-pes.cases*, *auto*)
apply(*rule Par*)
apply *auto*
by *blast+*

lemma *par-sound-aux2*:
assumes *pc*: ‹$pc \in cpts\text{-}from\ (pestran\ \Gamma)\ ((\lambda k.\ Com\ (prgf\ k)),\ S0) \cap assume\ pre\ rely$›
and *valid*: ‹$\forall k\ S0.\ cpts\text{-}from\ (estran\ \Gamma)\ (Com\ (prgf\ k),\ S0) \cap assume\ pre\ (Rely\ (prgf\ k)) \subseteq commit\ (estran\ \Gamma)\ \{fin\}\ (Guar\ (prgf\ k))\ (Post\ (prgf\ k))$›
and *rely1*: ‹$\forall k.\ rely \subseteq Rely\ (prgf\ k)$›
and *rely2*: ‹$\forall k\ k'.\ k' \neq k \longrightarrow Guar\ (prgf\ k') \subseteq Rely\ (prgf\ k)$›
and *guar*: ‹$\forall k.\ Guar\ (prgf\ k) \subseteq guar$›
and *conjoin*: ‹$pc \propto cs$›
shows
‹$\forall i\ k.\ Suc\ i < length\ pc \longrightarrow (cs\ k\ !\ i,\ cs\ k\ !\ Suc\ i) \in estran\ \Gamma \longrightarrow (snd\ (cs\ k\ !\ i),\ snd\ (cs\ k\ !\ Suc\ i)) \in Guar\ (prgf\ k)$›
proof(*rule ccontr*, *simp*, *erule exE*)
from *pc* **have** *pc-cpts-from*: ‹$pc \in cpts\text{-}from\ (pestran\ \Gamma)\ ((\lambda k.\ Com\ (prgf\ k)),\ S0)$› **by** *blast*
then have *pc-cpt*: ‹$pc \in cpts\ (pestran\ \Gamma)$› **by** *simp*
from *pc* **have** *pc-assume*: ‹$pc \in assume\ pre\ rely$› **by** *blast*
fix *l*
assume ‹$Suc\ l < length\ pc \wedge (\exists k.\ (cs\ k\ !\ l,\ cs\ k\ !\ Suc\ l) \in estran\ \Gamma \wedge (snd\ (cs\ k\ !\ l),\ snd\ (cs\ k\ !\ Suc\ l)) \notin Guar\ (prgf\ k))$›
(**is** ‹$?P\ l$›)
from *exists-least*[*of ?P*, *OF this*] **obtain** *m* **where** *contra*:
‹$(Suc\ m < length\ pc \wedge (\exists k.\ (cs\ k\ !\ m,\ cs\ k\ !\ Suc\ m) \in estran\ \Gamma \wedge (snd\ (cs\ k\ !\ m),\ snd\ (cs\ k\ !\ Suc\ m)) \notin Guar\ (prgf\ k))) \wedge$
$(\forall i<m.\ \neg\ (Suc\ i < length\ pc \wedge (\exists k.\ (cs\ k\ !\ i,\ cs\ k\ !\ Suc\ i) \in estran\ \Gamma \wedge (snd\ (cs\ k\ !\ i),\ snd\ (cs\ k\ !\ Suc\ i)) \notin Guar\ (prgf\ k))))$›
by *blast*
then have *Suc-m-lt*: ‹$Suc\ m < length\ pc$› **by** *argo*
from *contra* **obtain** *k* **where** ‹$(cs\ k\ !\ m,\ cs\ k\ !\ Suc\ m) \in estran\ \Gamma \wedge (snd\ (cs\ k\ !\ m),\ snd\ (cs\ k\ !\ Suc\ m)) \notin Guar\ (prgf\ k)$›
by *blast*
then have *ctran*: ‹$(cs\ k\ !\ m,\ cs\ k\ !\ Suc\ m) \in estran\ \Gamma$› **and** *not-guar*: ‹$(snd\ (cs\ k\ !\ m),\ snd\ (cs\ k\ !\ Suc\ m)) \notin Guar\ (prgf\ k)$›
by *auto*
from *contra* **have** ‹$\forall i<m.\ \neg\ (Suc\ i < length\ pc \wedge (\exists k.\ (cs\ k\ !\ i,\ cs\ k\ !\ Suc\ i) \in estran\ \Gamma \wedge (snd\ (cs\ k\ !\ i),\ snd\ (cs\ k\ !\ Suc\ i)) \notin Guar\ (prgf\ k)))$›
by *argo*
then have *forall-i-lt-m*: ‹$\forall i<m.\ Suc\ i < length\ pc \longrightarrow (\forall k.\ (cs\ k\ !\ i,\ cs\ k\ !\ Suc$

i) $\in$ *estran* Γ $\longrightarrow$ (*snd* (*cs k* ! *i*), *snd* (*cs k* ! *Suc i*)) $\in$ *Guar* (*prgf k*))›
by *simp*
from *Suc-m-lt* **have** ‹*Suc m* < *length* (*cs k*)› **using** *conjoin*
by (*simp add*: *conjoin-def same-length-def*)
let *?c* = ‹*take* (*Suc* (*Suc m*)) (*cs k*)›
have ‹*cs k* $\in$ *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *S0*)› **using** *conjoin-cpt*′[*OF pc-cpts-from conjoin*] **.**
then have *c-from*: ‹*?c* $\in$ *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *S0*)›
by (*metis Zero-not-Suc cpts-from-take*)
have ‹$\forall$ *i*. *Suc i* < *length ?c* $\longrightarrow$ *?c*!*i* −*e*→ *?c*!*Suc i* $\longrightarrow$ (*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) $\in$ *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* $\neq$ *k*}. *Guar* (*prgf j*))›
proof(*rule allI*, *rule impI*, *rule impI*)
fix *i*
assume *Suc-i-lt*′: ‹*Suc i* < *length ?c*›
then have ‹*i*$\leq$*m*› **using** *Suc-m-lt* **by** *simp*
then have *Suc-i-lt*: ‹*Suc i* < *length pc*› **using** *Suc-m-lt* **by** *simp*
assume *etran*′: ‹*?c*!*i* −*e*→ *?c*!*Suc i*›
then have *etran*: ‹*cs k*!*i* −*e*→ *cs k*!*Suc i*› **using** ‹*i*$\leq$*m*› **by** *simp*
from *conjoin-etran-k*[*OF pc-cpt conjoin Suc-i-lt etran*]
have ‹(*pc*!*i* −*e*→ *pc*!*Suc i*) $\vee$ ($\exists$ *k*′. *k*′$\neq$*k* $\wedge$ (*cs k*′!*i*, *cs k*′!*Suc i*) $\in$ *estran* Γ)› **.**
then show ‹(*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) $\in$ *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* $\neq$ *k*}. *Guar* (*prgf j*))›
proof
assume ‹*pc*!*i* −*e*→ *pc*!*Suc i*›
then have ‹(*snd* (*pc*!*i*), *snd* (*pc*!*Suc i*)) $\in$ *rely*› **using** *pc-assume Suc-i-lt*
by (*simp add*: *assume-def*)
then have ‹(*snd* (*cs k*!*i*), *snd* (*cs k*!*Suc i*)) $\in$ *rely*› **using** *conjoin Suc-i-lt*
by (*simp add*: *conjoin-def same-state-def*)
then have ‹(*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) $\in$ *rely*› **using** ‹*i*$\leq$*m*› **by** *simp*
then show ‹(*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) $\in$ *rely* $\cup$ ($\bigcup$*j*$\in${*j*. *j* $\neq$ *k*}. *Guar* (*prgf j*))› **by** *blast*
next
assume ‹$\exists$ *k*′. *k*′ $\neq$ *k* $\wedge$ (*cs k*′ ! *i*, *cs k*′ ! *Suc i*) $\in$ *estran* Γ›
then obtain *k*′ **where** *k*′: ‹*k*′ $\neq$ *k* $\wedge$ (*cs k*′ ! *i*, *cs k*′ ! *Suc i*) $\in$ *estran* Γ› **by** *blast*
then have *ctran-k*′: ‹(*cs k*′ ! *i*, *cs k*′ ! *Suc i*) $\in$ *estran* Γ› **by** *argo*
have ‹(*snd* (*cs k*′!*i*), *snd* (*cs k*′!*Suc i*)) $\in$ *Guar* (*prgf k*′)›
proof(*cases i*=*m*)
case *True*
with *ctran etran ctran-imp-not-etran* **show** *?thesis* **by** *blast*
next
case *False*
with ‹*i*$\leq$*m*› **have** ‹*i*<*m*› **by** *linarith*
with *forall-i-lt-m Suc-i-lt ctran-k*′ **show** *?thesis* **by** *blast*
qed
then have ‹(*snd* (*cs k*!*i*), *snd* (*cs k*!*Suc i*)) $\in$ *Guar* (*prgf k*′)› **using** *conjoin Suc-i-lt*
by (*simp add*: *conjoin-def same-state-def*)
then have ‹(*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) $\in$ *Guar* (*prgf k*′)› **using** ‹*i*$\leq$*m*› **by**

fastforce
then show ‹(*snd* (*?c*!*i*), *snd* (*?c*!*Suc i*)) ∈ *rely* ∪ (⋃*j*∈{*j*. *j* ≠ *k*}. *Guar* (*prgf j*))›
using *k'* **by** *blast*
qed
qed
moreover have ‹*snd* (*hd ?c*) ∈ *pre*›
proof−
from *pc-cpt cpts-nonnil* **have** ‹*pc*≠[]› **by** *blast*
then have *length pc* ≠ *0* **by** *simp*
then have ‹*length* (*cs k*) ≠ *0*› **using** *conjoin* **by** (*simp add*: *conjoin-def same-length-def*)
then have ‹*cs k* ≠ []› **by** *simp*
have ‹*snd* (*hd pc*) ∈ *pre*› **using** *pc-assume* **by** (*simp add*: *assume-def*)
then have ‹*snd* (*pc*!*0*) ∈ *pre*› **by** (*simp add*: *hd-conv-nth* ‹*pc*≠[]›)
then have ‹*snd* (*cs k* ! *0*) ∈ *pre*› **using** *conjoin*
by (*simp add*: *conjoin-def same-state-def* ‹*pc* ≠ []›)
then have ‹*snd* (*hd* (*cs k*)) ∈ *pre*› **by** (*simp add*: *hd-conv-nth* ‹*cs k*≠[]›)
then show ‹*snd* (*hd ?c*) ∈ *pre*› **by** *simp*
qed
ultimately have ‹*?c* ∈ *assume pre* (*Rely* (*prgf k*))› **using** *rely1 rely2*
apply(*auto simp add*: *assume-def*) **by** *blast*
with *c-from* **have** ‹*?c* ∈ *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *S0*) ∩ *assume pre* (*Rely* (*prgf k*))› **by** *blast*
with *valid* **have** ‹*?c* ∈ *commit* (*estran* Γ) {*fin*} (*Guar* (*prgf k*)) (*Post* (*prgf k*))› **by** *blast*
then have ‹(*snd* (*?c*!*m*), *snd* (*?c*!*Suc m*)) ∈ *Guar* (*prgf k*)›
apply(*simp add*: *commit-def*)
apply *clarify*
apply(*erule allE*[**where** *x*=*m*])
using *ctran* ‹*Suc m* < *length* (*cs k*)› **by** *blast*
with *not-guar* ‹*Suc m* < *length* (*cs k*)› **show** *False* **by** *simp*
qed

lemma *par-sound-aux3*:
assumes *pc*: ‹*pc* ∈ *cpts-from* (*pestran* Γ) ((λ*k*. *Com* (*prgf k*)), *s0*) ∩ *assume pre rely*›
and *valid*: ‹∀ *k s0*. *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *s0*) ∩ *assume pre* (*Rely* (*prgf k*)) ⊆ *commit* (*estran* Γ) {*fin*} (*Guar* (*prgf k*)) (*Post* (*prgf k*))›
and *rely1*: ‹∀ *k*. *rely* ⊆ *Rely* (*prgf k*)›
and *rely2*: ‹∀ *k k'*. *k'* ≠ *k* ⟶ *Guar* (*prgf k'*) ⊆ *Rely* (*prgf k*)›
and *guar*: ‹∀ *k*. *Guar* (*prgf k*) ⊆ *guar*›
and *conjoin*: ‹*pc* ∝ *cs*›
and *Suc-i-lt*: ‹*Suc i* < *length pc*›
and *etran*: ‹(*cs k* ! *i* −*e*→ *cs k* ! *Suc i*)›
shows ‹(*snd* (*cs k*!*i*), *snd* (*cs k*!*Suc i*)) ∈ *Rely* (*prgf k*)›
proof−

from *pc* **have** *pc-cpt*: ‹*pc* ∈ *cpts* (*pestran* Γ)› **by** *fastforce*

from *conjoin-etran-k*[*OF pc-cpt conjoin Suc-i-lt etran*]
have ‹*pc* ! *i* −*e*→ *pc* ! *Suc i* ∨ (∃ *k'*. *k'* ≠ *k* ∧ (*cs k'* ! *i*, *cs k'* ! *Suc i*) ∈ *estran* Γ)› **.**
then show *?thesis*
proof
assume ‹*pc* ! *i* −*e*→ *pc* ! *Suc i*›
moreover from *pc* **have** ‹*pc* ∈ *assume pre rely*› **by** *blast*
ultimately have ‹(*snd* (*pc*!*i*), *snd* (*pc*!*Suc i*)) ∈ *rely*› **using** *Suc-i-lt*
by (*simp add*: *assume-def*)
with *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*]] *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*] *rely1*
show ‹(*snd* (*cs k* ! *i*), *snd* (*cs k* ! *Suc i*)) ∈ *Rely* (*prgf k*)›
by *auto*
next
assume ‹∃ *k'*. *k'* ≠ *k* ∧ (*cs k'* ! *i*, *cs k'* ! *Suc i*) ∈ *estran* Γ›
then obtain *k''* **where** *k''*: ‹*k''* ≠ *k* ∧ (*cs k''* ! *i*, *cs k''* ! *Suc i*) ∈ *estran* Γ› **by** *blast*
then have ‹(*cs k''* ! *i*, *cs k''* ! *Suc i*) ∈ *estran* Γ› **by** (*rule conjunct2*)
from *par-sound-aux2*[*OF pc valid rely1 rely2 guar conjoin*, *rule-format*, *OF Suc-i-lt*, *OF this*]
have *1*: ‹(*snd* (*cs k''* ! *i*), *snd* (*cs k''* ! *Suc i*)) ∈ *Guar* (*prgf k''*)› **.**
show ‹(*snd* (*cs k* ! *i*), *snd* (*cs k* ! *Suc i*)) ∈ *Rely* (*prgf k*)›
proof−
from *1 conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*]] *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*]
have ‹(*snd* (*pc* ! *i*), *snd* (*pc* ! *Suc i*)) ∈ *Guar* (*prgf k''*)› **by** *simp*
with *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*]] *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*]
have ‹(*snd* (*cs k* ! *i*), *snd* (*cs k* ! *Suc i*)) ∈ *Guar* (*prgf k''*)› **by** *simp*
moreover from *k''* **have** ‹*k''*≠*k*› **by** (*rule conjunct1*)
ultimately show *?thesis* **using** *rely2*[*rule-format*, *OF* ‹*k''*≠*k*›] **by** *blast*
qed
qed
qed

lemma *par-sound-aux5*:
assumes *pc*: ‹*pc* ∈ *cpts-from* (*pestran* Γ) ((λ*k*. *Com* (*prgf k*)), *s0*) ∩ *assume pre rely*›
and *valid*: ‹∀ *k s0*. *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *s0*) ∩ *assume pre* (*Rely* (*prgf k*)) ⊆ *commit* (*estran* Γ) {*fin*} (*Guar* (*prgf k*)) (*Post* (*prgf k*))›
and *rely1*: ‹∀ *k*. *rely* ⊆ *Rely* (*prgf k*)›
and *rely2*: ‹∀ *k k'*. *k'* ≠ *k* ⟶ *Guar* (*prgf k'*) ⊆ *Rely* (*prgf k*)›
and *guar*: ‹∀ *k*. *Guar* (*prgf k*) ⊆ *guar*›
and *conjoin*: ‹*pc* ∝ *cs*›
and *fin*: ‹*fst* (*last pc*) ∈ *par-fin*›
shows ‹*snd* (*last pc*) ∈ (⋂ *k*. *Post* (*prgf k*))›
proof−
have ‹∀ *k*. *cs k* ∈ *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *s0*) ∩ *assume pre* (*Rely* (*prgf k*))›

```
  proof
    fix k
    show ‹cs k ∈ cpts-from (estran Γ) (Com (prgf k), s0) ∩ assume pre (Rely (prgf
k))›
    proof
      from pc have pc': ‹pc ∈ cpts-from (pestran Γ) ((λk. Com (prgf k)), s0)› by
blast
      show ‹cs k ∈ cpts-from (estran Γ) (Com (prgf k), s0)›
        using conjoin-cpt'[OF pc' conjoin] .
    next
      show ‹cs k ∈ assume pre (Rely (prgf k))›
      proof(auto simp add: assume-def)
        from pc have pc-cpt: ‹pc ∈ cpts (pestran Γ)› by simp
        from pc have pc-assume: ‹pc ∈ assume pre rely› by blast
        from pc-cpt cpts-nonnil have ‹pc≠[]› by blast
        then have length pc ≠ 0 by simp
        then have ‹length (cs k) ≠ 0› using conjoin by (simp add: conjoin-def
same-length-def)
        then have ‹cs k ≠ []› by simp
        have ‹snd (hd pc) ∈ pre› using pc-assume by (simp add: assume-def)
        then have ‹snd (pc!0) ∈ pre› by (simp add: hd-conv-nth ‹pc≠[]›)
        then have ‹snd (cs k ! 0) ∈ pre› using conjoin
          by (simp add: conjoin-def same-state-def ‹pc ≠ []›)
        then show ‹snd (hd (cs k)) ∈ pre› by (simp add: hd-conv-nth ‹cs k≠[]›)
      next
        fix i
        show ‹Suc i < length (cs k) ⟹ fst (cs k ! i) = fst (cs k ! Suc i) ⟹ (snd
(cs k ! i), snd (cs k ! Suc i)) ∈ Rely (prgf k)›
        proof−
          assume ‹Suc i < length (cs k)›
          with conjoin-same-length[OF conjoin] have ‹Suc i < length pc› by simp
          assume ‹fst (cs k ! i) = fst (cs k ! Suc i)›
          then have etran: ‹(cs k ! i) −e→ (cs k ! Suc i)› by simp
          show ‹(snd (cs k ! i), snd (cs k ! Suc i)) ∈ Rely (prgf k)›
            using par-sound-aux3[OF pc valid rely1 rely2 guar conjoin ‹Suc i <
length pc› etran] .
        qed
      qed
    qed
  qed
  with valid have commit: ‹∀ k. cs k ∈ commit (estran Γ) {fin} (Guar (prgf k))
(Post (prgf k))› by blast
  from pc have pc-cpt: ‹pc ∈ cpts (pestran Γ)› by fastforce
  with cpts-nonnil have ‹pc≠[]› by blast
  have ‹∀ k. fst (last (cs k)) = fin›
  proof
    fix k
    from conjoin-cpt[OF pc-cpt conjoin] have ‹cs k ∈ cpts (estran Γ)› .
    with cpts-nonnil have ‹cs k ≠ []› by blast
```

from *fin* **have** ‹∀ *k*. *fst* (*last pc*) *k* = *fin*› **by** *blast*
moreover have ‹*fst* (*last pc*) *k* = *fst* (*last* (*cs k*))› **using** *conjoin-same-spec*[*OF conjoin*]
apply(*subst last-conv-nth*)
apply(*rule* ‹*pc*≠[]›)
apply(*subst last-conv-nth*)
apply(*rule* ‹*cs k*≠[]›)
apply(*subst conjoin-same-length*[*OF conjoin*, *of k*])
apply(*erule allE*[**where** *x*=*k*])
apply(*erule allE*[**where** *x*=‹*length* (*cs k*) − *1*›])
apply(*subst* (*asm*) *conjoin-same-length*[*OF conjoin*, *of k*])
using ‹*cs k* ≠ []› **by** *force*
ultimately show ‹*fst* (*last* (*cs k*)) = *fin*› **using** *fin conjoin-same-spec*[*OF conjoin*] **by** *simp*
qed
then have ‹∀ *k*. *snd* (*last* (*cs k*)) ∈ *Post* (*prgf k*)› **using** *commit*
by (*simp add*: *commit-def*)
moreover have ‹∀ *k*. *snd* (*last* (*cs k*)) = *snd* (*last pc*)›
proof
fix *k*
from *conjoin-cpt*[*OF pc-cpt conjoin*] **have** ‹*cs k* ∈ *cpts* (*estran* Γ)› **.**
with *cpts-nonnil* **have** ‹*cs k* ≠ []› **by** *blast*
show ‹*snd* (*last* (*cs k*)) = *snd* (*last pc*)› **using** *conjoin-same-state*[*OF conjoin*]
apply−
apply(*subst last-conv-nth*)
apply(*rule* ‹*cs k* ≠ []›)
apply(*subst last-conv-nth*)
apply(*rule* ‹*pc* ≠ []›)
apply(*subst conjoin-same-length*[*OF conjoin*, *of k*])
apply(*erule allE*[**where** *x*=*k*])
apply(*erule allE*[**where** *x*=‹*length* (*cs k*) − *1*›])
apply(*subst* (*asm*) *conjoin-same-length*[*OF conjoin*, *of k*])
using ‹*cs k* ≠ []› **by** *force*
qed
ultimately show *?thesis* **by** *fastforce*
qed

definition ‹*split-par pc* ≡ λ*k*. *map* (λ(*Ps*,*s*). (*Ps k*, *s*)) *pc*›

lemma *split-par-conjoin*:
‹*pc* ∈ *cpts* (*pestran* Γ) ⟹ *pc* ∝ *split-par pc*›
proof(*unfold conjoin-def*, *auto*)
show ‹*same-length pc* (*split-par pc*)›
by (*simp add*: *same-length-def split-par-def*)
next
show ‹*same-state pc* (*split-par pc*)›
by (*simp add*: *same-state-def split-par-def case-prod-unfold*)
next
show ‹*same-spec pc* (*split-par pc*)›

by (*simp add*: *same-spec-def split-par-def case-prod-unfold*)
next
assume ‹$pc \in cpts$ (*pestran* Γ)›
then show ‹*compat-tran pc* (*split-par pc*)›
proof(*auto simp add*: *compat-tran-def split-par-def case-prod-unfold*)
fix j
assume *cpt*: ‹$pc \in cpts$ (*pestran* Γ)›
assume *Suc-j-lt*: ‹*Suc* $j <$ *length pc*›
assume *not-etran*: ‹*fst* (pc ! j) $\neq$ *fst* (pc ! *Suc* j)›
from *ctran-or-etran-par*[*OF cpt Suc-j-lt*] *not-etran*
have ‹(pc ! j, pc ! *Suc* j) $\in$ *pestran* Γ› **by** *fastforce*
then show ‹$\exists\, t\ k\ \Gamma.\ \Gamma \vdash pc\ !\ j\ -pes[t\sharp k]\rightarrow pc\ !\ Suc\ j$›
by (*auto simp add*: *pestran-def*)
next
fix $j\ k\ t\ \Gamma'$
assume *ctran*: ‹$\Gamma' \vdash pc\ !\ j\ -pes[t\sharp k]\rightarrow pc\ !\ Suc\ j$›
then show ‹$\Gamma' \vdash$ (*fst* (pc ! j) k, *snd* (pc ! j)) $-es[t\sharp k]\rightarrow$ (*fst* (pc ! *Suc* j) k, *snd* (pc ! *Suc* j))›
apply−
by (*erule pestran-p.cases*, *auto*)
next
fix $j\ k\ t\ \Gamma'\ k'$
assume ‹$\Gamma' \vdash pc\ !\ j\ -pes[t\sharp k]\rightarrow pc\ !\ Suc\ j$›
moreover assume ‹$k' \neq k$›
ultimately show ‹*fst* (pc ! j) k' = *fst* (pc ! *Suc* j) k'›
apply−
by (*erule pestran-p.cases*, *auto*)
next
fix $j\ k$
assume *cpt*: ‹$pc \in cpts$ (*pestran* Γ)›
assume *Suc-j-lt*: ‹*Suc* $j <$ *length pc*›
assume ‹*fst* (pc ! j) $k \neq$ *fst* (pc ! *Suc* j) k›
then have ‹*fst* ($pc!j$) $\neq$ *fst* ($pc!Suc\ j$)› **by** *force*
with *ctran-or-etran-par*[*OF cpt Suc-j-lt*] **have** ‹(pc ! j, pc ! *Suc* j) $\in$ *pestran* Γ› **by** *fastforce*
then show ‹$\exists\, t\ k\ \Gamma.\ \Gamma \vdash pc\ !\ j\ -pes[t\sharp k]\rightarrow pc\ !\ Suc\ j$› **by** (*auto simp add*: *pestran-def*)
next
fix $j\ k\ ka\ t\ \Gamma'$
assume ‹$\Gamma' \vdash pc\ !\ j\ -pes[t\sharp ka]\rightarrow pc\ !\ Suc\ j$›
then show ‹$\Gamma' \vdash$ (*fst* (pc ! j) ka, *snd* (pc ! j)) $-es[t\sharp ka]\rightarrow$ (*fst* (pc ! *Suc* j) ka, *snd* (pc ! *Suc* j))›
apply−
by (*erule pestran-p.cases*, *auto*)
next
fix $j\ k\ ka\ t\ \Gamma'\ k'$
assume ‹$\Gamma' \vdash pc\ !\ j\ -pes[t\sharp ka]\rightarrow pc\ !\ Suc\ j$›
moreover assume ‹$k' \neq ka$›
ultimately show ‹*fst* (pc ! j) k' = *fst* (pc ! *Suc* j) k'›

apply−
by (*erule pestran-p.cases*, *auto*)
qed
qed

theorem *par-sound*:
assumes *h*: ‹$\forall k.\ \Gamma \vdash$ *Com* (*prgf k*) sat_e [*Pre* (*prgf k*), *Rely* (*prgf k*), *Guar* (*prgf k*), *Post* (*prgf k*)]›
assumes *pre*: ‹$\forall k.$ *pre* $\subseteq$ *Pre* (*prgf k*)›
assumes *rely1*: ‹$\forall k.$ *rely* $\subseteq$ *Rely* (*prgf k*)›
assumes *rely2*: ‹$\forall k\ j.\ j \neq k \longrightarrow$ *Guar* (*prgf j*) $\subseteq$ *Rely* (*prgf k*)›
assumes *guar*: ‹$\forall k.$ *Guar* (*prgf k*) $\subseteq$ *guar*›
assumes *post*: ‹($\bigcap k.$ *Post* (*prgf k*)) $\subseteq$ *post*›
shows
‹$\Gamma \models$ *par-com prgf* SAT_e [*pre*, *rely*, *guar*, *post*]›
proof(*simp*)
let *?pre* = ‹*lift-state-set pre*›
let *?rely* = ‹*lift-state-pair-set rely*›
let *?guar* = ‹*lift-state-pair-set guar*›
let *?post* = ‹*lift-state-set post*›
obtain *prgf'* :: ‹$'a \Rightarrow (('b, 'a, 's, 'prog)$ *esys*, $'s \times ('a \Rightarrow ('b \times 's$ *set* $\times$ $'prog)$ *option*)) *rgformula*›
where *prgf'-def*: ‹*prgf'* = ($\lambda k.$ ⦇ *Com* = *Com* (*prgf k*), *Pre* = *lift-state-set* (*Pre* (*prgf k*)), *Rely* = *lift-state-pair-set* (*Rely* (*prgf k*)),
Guar = *lift-state-pair-set* (*Guar* (*prgf k*)), *Post* = *lift-state-set* (*Post* (*prgf k*)) ⦈)›
by *simp*

from *rely1* **have** *rely1'*: ‹$\forall k.$ *lift-state-pair-set rely* $\subseteq$ *lift-state-pair-set* (*Rely* (*prgf k*))›
apply(*simp add*: *lift-state-pair-set-def*) **by** *blast*
from *rely2* **have** *rely2'*: ‹$\forall k\ k'.\ k' \neq k \longrightarrow$ *lift-state-pair-set* (*Guar* (*prgf k'*)) $\subseteq$ *lift-state-pair-set* (*Rely* (*prgf k*))›
apply(*simp add*: *lift-state-pair-set-def*) **by** *blast*
from *guar* **have** *guar'*: ‹$\forall k.$ *lift-state-pair-set* (*Guar* (*prgf k*)) $\subseteq$ *?guar*›
apply(*simp add*: *lift-state-pair-set-def*) **by** *blast*
from *post* **have** *post'*: ‹$\bigcap$(*lift-state-set* ‘ (*Post* ‘ (*prgf* ‘ *UNIV*))) $\subseteq$ *?post*›
apply(*simp add*: *lift-state-set-def*) **by** *fast*

have *valid*: ‹$\forall k$ *s0*. *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *s0*) $\cap$ *assume ?pre* (*lift-state-pair-set* (*Rely* (*prgf k*))) $\subseteq$ *commit* (*estran* Γ) {*fin*} (*lift-state-pair-set* (*Guar* (*prgf k*))) (*lift-state-set* (*Post* (*prgf k*)))›
proof
fix *k*
from *rghoare-es-sound*[*OF h*[*rule-format*, *of k*]] *pre*[*rule-format*, *of k*]
show ‹$\forall$ *s0*. *cpts-from* (*estran* Γ) (*Com* (*prgf k*), *s0*) $\cap$ *assume ?pre* (*lift-state-pair-set* (*Rely* (*prgf k*))) $\subseteq$ *commit* (*estran* Γ) {*fin*} (*lift-state-pair-set* (*Guar* (*prgf k*))) (*lift-state-set* (*Post* (*prgf k*)))›
by (*auto simp add*: *assume-def lift-state-set-def lift-state-pair-set-def case-prod-unfold*)
qed

show ‹∀ *s0 x0*. {*cpt* ∈ *cpts* (*pestran* Γ). *hd cpt* = (*par-com prgf*, *s0*, *x0*)} ∩ *assume ?pre ?rely* ⊆ *commit* (*pestran* Γ) *par-fin ?guar ?post*›
proof(*rule allI*, *rule allI*)
fix *s0*
fix *x0*
show ‹{*cpt* ∈ *cpts* (*pestran* Γ). *hd cpt* = (*par-com prgf*, *s0*, *x0*)} ∩ *assume ?pre ?rely* ⊆ *commit* (*pestran* Γ) *par-fin ?guar ?post*›
proof(*auto*)
fix *pc*
assume *hd-pc*: ‹*hd pc* = (*par-com prgf*, *s0*, *x0*)›
assume *pc-cpt*: ‹*pc* ∈ *cpts* (*pestran* Γ)›
assume *pc-assume*: ‹*pc* ∈ *assume ?pre ?rely*›
from *hd-pc pc-cpt pc-assume*
have *pc*: ‹*pc* ∈ *cpts-from* (*pestran* Γ) (*par-com prgf*, *s0*, *x0*) ∩ *assume ?pre ?rely*› **by** *simp*
obtain *cs* **where** ‹*cs* = *split-par pc*› **by** *simp*
with *split-par-conjoin*[*OF pc-cpt*] **have** *conjoin*: ‹*pc* ∝ *cs*› **by** *simp*
show ‹*pc* ∈ *commit* (*pestran* Γ) *par-fin ?guar ?post*›
proof(*auto simp add*: *commit-def*)
fix *i*
assume *Suc-i-lt*: ‹*Suc i* < *length pc*›
assume ‹(*pc*!*i*, *pc*!*Suc i*) ∈ *pestran* Γ›
then obtain *a k* **where** ‹Γ ⊢ *pc* ! *i* −*pes*[*a*♯*k*]→ *pc* ! *Suc i*› **by** (*auto simp add*: *pestran-def*)
then show ‹(*snd* (*pc* ! *i*), *snd* (*pc* ! *Suc i*)) ∈ *?guar*› **apply** −
proof(*erule pestran-p.cases*, *auto*)
fix *pes s x es′ t y*
assume *eq1*: ‹*pc* ! *i* = (*pes*, *s*, *x*)›
assume *eq2*: ‹*pc* ! *Suc i* = (*pes*(*k* := *es′*), *t*, *y*)›
have *eq1s*: ‹*snd* (*cs k* ! *i*) = (*s*,*x*)› **using** *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*], *of k*] *eq1*
by *simp*
have *eq2s*: ‹*snd* (*cs k* ! *Suc i*) = (*t*,*y*)› **using** *conjoin-same-state*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*, *of k*] *eq2*
by *simp*
have *eq1p*: ‹*fst* (*cs k* ! *i*) = *pes k*› **using** *conjoin-same-spec*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*[*THEN Suc-lessD*], *of k*] *eq1*
by *simp*
have *eq2p*: ‹*fst* (*cs k* ! *Suc i*) = *es′*› **using** *conjoin-same-spec*[*OF conjoin*, *rule-format*, *OF Suc-i-lt*, *of k*] *eq2*
by *simp*
assume ‹Γ ⊢ (*pes k*, *s*, *x*) −*es*[*a*♯*k*]→ (*es′*, *t*, *y*)›
with *eq1s eq2s eq1p eq2p*
have ‹Γ ⊢ (*fst* (*cs k* ! *i*), *snd* (*cs k* ! *i*)) −*es*[*a*♯*k*]→ (*fst* (*cs k* ! *Suc i*), *snd* (*cs k* ! *Suc i*))› **by** *simp*
then have *estran*: ‹(*cs k*!*i*, *cs k*!*Suc i*)∈*estran* Γ› **by** (*auto simp add*: *estran-def*)
from *par-sound-aux2*[*of pc* Γ *prgf′*, *simplified prgf′-def rgformula.simps*, *OF pc valid rely1′ rely2′ guar′ conjoin*, *rule-format*, *of i k*, *OF Suc-i-lt estran*]

have ‹(*snd* (*cs k* ! *i*), *snd* (*cs k* ! *Suc i*)) ∈ *lift-state-pair-set* (*Guar* (*prgf k*))› **.**
with *eq1s eq2s* **have** ‹((*s*,*x*),(*t*,*y*)) ∈ *lift-state-pair-set* (*Guar* (*prgf k*))› **by** *simp*
with *guar′* **show** ‹((*s*, *x*), *t*, *y*) ∈ *lift-state-pair-set guar*› **by** *blast*
qed
next
assume ‹∀ *k*. *fst* (*last pc*) *k* = *fin*›
then have *fin*: ‹*fst* (*last pc*) ∈ *par-fin*› **by** *fast*
from *par-sound-aux5*[*of pc* Γ *prgf′*, *simplified prgf′-def rgformula.simps*, *OF pc valid rely1′ rely2′ guar′ conjoin fin*] *post′*
show ‹*snd* (*last pc*) ∈ *lift-state-set post*› **by** *blast*
qed
qed
qed
qed

theorem *rghoare-pes-sound*:
assumes *h*: ‹Γ ⊢ *prgf* SAT_e [*pre*, *rely*, *guar*, *post*]›
shows ‹Γ ⊨ *par-com prgf* SAT_e [*pre*, *rely*, *guar*, *post*]›
using *h*
proof(*cases*)
case *Par*
then show *?thesis* **using** *par-sound* **by** *blast*
qed

definition *Evt-sat-RG* :: ′*Env* ⇒ ((′*l*, ′*k*, ′*s*, ′*prog*) *esys*, ′*s*) *rgformula* ⇒ *bool* (- ⊢ - [*60*,*60*] *61*)
where Γ ⊢ *rg* ≡ Γ ⊢ *Com rg* sat_e [*Pre rg*, *Rely rg*, *Guar rg*, *Post rg*]

end

end

6 Rely-guarantee-based Safety Reasoning

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

type-synonym ′*s invariant* = ′*s* ⇒ *bool*

context *event-hoare*
begin

definition *invariant-presv-pares*::′*Env* ⇒ ′*s invariant* ⇒ (′*l*,′*k*,′*s*,′*prog*) *paresys* ⇒ ′*s set* ⇒ (′*s* × ′*s*) *set* ⇒ *bool*
where *invariant-presv-pares* Γ *invar pares init R* ≡
∀ *s0 x0 pesl*. *s0*∈*init* ∧ *pesl* ∈ (*cpts-from* (*pestran* Γ) (*pares*, *s0*, *x0*) ∩

assume (*lift-state-set init*) (*lift-state-pair-set R*))
$\longrightarrow$ ($\forall$ *i*<*length pesl*. *invar* (*fst* (*snd* (*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-from* (*pestran* Γ) (*pares*, *s0*, *x0*) $\cap$ *assume* (*lift-state-set init*) (*lift-state-pair-set R*))
$\longrightarrow$ ($\forall$ *i*<*length pesl*. *invar* (*fst* (*snd* (*pesl*!*i*))))

lemma *invariant-presv-pares* Γ *invar pares init R* = *invariant-presv-pares2* Γ *invar pares init R*
by (*auto simp add*:*invariant-presv-pares-def invariant-presv-pares2-def assume-def lift-state-set-def*)

theorem *invariant-theorem*:
assumes *parsys-sat-rg*: $\Gamma \vdash$ *pesf* SAT_e [*init*, *R*, *G*, *pst*]
and *stb-rely*: *stable* (*Collect invar*) *R*
and *stb-guar*: *stable* (*Collect invar*) *G*
and *init-in-invar*: *init* $\subseteq$ (*Collect invar*)
shows *invariant-presv-pares* Γ *invar* (*par-com pesf*) *init R*
proof −
let *?init* = ‹*lift-state-set init*›
let *?R* = ‹*lift-state-pair-set R*›
let *?G* = ‹*lift-state-pair-set G*›
let *?pst* = ‹*lift-state-set pst*›
from *parsys-sat-rg* **have** $\Gamma \models$ *par-com pesf* SAT_e [*init*, *R*, *G*, *pst*] **using** *rghoare-pes-sound* **by** *fast*
hence *cpts-pes*: $\forall$ *s*. (*cpts-from* (*pestran* Γ) (*par-com pesf*, *s*)) $\cap$ *assume ?init ?R* $\subseteq$ *commit* (*pestran* Γ) *par-fin ?G ?pst* **by** *simp*
show *?thesis*
proof −
{
fix *s0 x0 pesl*
assume *a0*: *s0*$\in$*init*
and *a1*: *pesl*$\in$*cpts-from* (*pestran* Γ) (*par-com pesf*, *s0*, *x0*) $\cap$ *assume ?init ?R*
from *a1* **have** *a3*: *pesl*!*0* = (*par-com pesf*, *s0*, *x0*) $\wedge$ *pesl*$\in$*cpts* (*pestran* Γ) **using** *hd-conv-nth cpts-nonnil* **by** *force*
from *a1 cpts-pes* **have** *pesl-in-comm*: *pesl* $\in$ *commit* (*pestran* Γ) *par-fin ?G ?pst* **by** *auto*
{
fix *i*
assume *b0*: *i*<*length pesl*
then have *fst* (*snd* (*pesl*!*i*)) $\in$ (*Collect invar*)
proof(*induct i*)
case *0*
with *a3* **have** *snd* (*pesl*!*0*) = (*s0*,*x0*) **by** *simp*
with *a0 init-in-invar* **show** *?case* **by** *auto*

```
      next
        case (Suc ni)
        assume c0: ni < length pesl ⟹ fst (snd (pesl ! ni)) ∈ (Collect invar)
          and  c1: Suc ni < length pesl
        then have c2: fst (snd (pesl ! ni)) ∈ (Collect invar) by auto
        from c1 have c3: ni < length pesl by simp
        with c0 have c4: fst (snd (pesl ! ni)) ∈ (Collect invar) by simp
        from a3 c1 have pesl ! ni −e→ pesl ! Suc ni ∨ (pesl ! ni, pesl ! Suc ni) ∈
pestran Γ
          using ctran-or-etran-par by blast
        then show ?case
        proof
          assume d0: pesl ! ni −e→ pesl ! Suc ni
           then show ?thesis using c3 c4 a1 c1 stb-rely by(simp add:assume-def
stable-def lift-state-set-def lift-state-pair-set-def case-prod-unfold)
        next
          assume (pesl ! ni, pesl ! Suc ni) ∈ pestran Γ
         then obtain et where d0: Γ ⊢ pesl ! ni −pes[et]→ pesl ! Suc ni by (auto
simp add: pestran-def)
          then show ?thesis using c3 c4 c1 pesl-in-comm stb-guar
         apply(simp add:commit-def stable-def lift-state-set-def lift-state-pair-set-def
case-prod-unfold)
           using ‹(pesl ! ni, pesl ! Suc ni) ∈ pestran Γ› by blast
        qed
      qed
    }
  }
  then show ?thesis using invariant-presv-pares-def by blast
  qed
qed

end

end
```

7 Extending SIMP language with new proof rules

```
theory SIMP-plus
imports HOL−Hoare-Parallel.RG-Hoare
begin
```

7.1 new proof rules

```
inductive rghoare-p :: ['a com option, 'a set, ('a × 'a) set, ('a × 'a) set, 'a set]
⇒ bool
    (⊢_I - sat_p [-, -, -, -] [60,0,0,0,0] 45)
where

  Basic: ⟦ pre ⊆ {s. f s ∈ post}; {(s,t). s ∈ pre ∧ (t=f s)} ⊆ guar;
```

stable pre rely; *stable post rely* $\rrbracket$
$\Longrightarrow \vdash_I$ *Some* (*Basic f*) sat_p [*pre*, *rely*, *guar*, *post*]

| *Seq*: $\llbracket \vdash_I$ *Some P* sat_p [*pre*, *rely*, *guar*, *mid*]; $\vdash_I$ *Some Q* sat_p [*mid*, *rely*, *guar*, *post*] $\rrbracket$
$\Longrightarrow \vdash_I$ *Some* (*Seq P Q*) sat_p [*pre*, *rely*, *guar*, *post*]

| *Cond*: $\llbracket$ *stable pre rely*; $\vdash_I$ *Some P1* sat_p [*pre* $\cap$ *b*, *rely*, *guar*, *post*];
$\vdash_I$ *Some P2* sat_p [*pre* $\cap$ $-b$, *rely*, *guar*, *post*]; $\forall s.$ (*s*,*s*)$\in$*guar* $\rrbracket$
$\Longrightarrow \vdash_I$ *Some* (*Cond b P1 P2*) sat_p [*pre*, *rely*, *guar*, *post*]

| *While*: $\llbracket$ *stable pre rely*; (*pre* $\cap$ $-b$) $\subseteq$ *post*; *stable post rely*;
$\vdash_I$ *Some P* sat_p [*pre* $\cap$ *b*, *rely*, *guar*, *pre*]; $\forall s.$ (*s*,*s*)$\in$*guar* $\rrbracket$
$\Longrightarrow \vdash_I$ *Some* (*While b P*) sat_p [*pre*, *rely*, *guar*, *post*]

| *Await*: $\llbracket$ *stable pre rely*; *stable post rely*;
$\forall V. \vdash_I$ *Some P* sat_p [*pre* $\cap$ *b* $\cap$ $\{V\}$, $\{(s, t).\ s = t\}$,
UNIV, $\{s.\ (V, s) \in guar\} \cap post$] $\rrbracket$
$\Longrightarrow \vdash_I$ *Some* (*Await b P*) sat_p [*pre*, *rely*, *guar*, *post*]

| *None-hoare*: $\llbracket$ *stable pre rely*; *pre* $\subseteq$ *post* $\rrbracket$ $\Longrightarrow \vdash_I$ *None* sat_p [*pre*, *rely*, *guar*, *post*]

| *Conseq*: $\llbracket$ *pre* $\subseteq$ *pre'*; *rely* $\subseteq$ *rely'*; *guar'* $\subseteq$ *guar*; *post'* $\subseteq$ *post*;
$\vdash_I$ *P* sat_p [*pre'*, *rely'*, *guar'*, *post'*] $\rrbracket$
$\Longrightarrow \vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]

| *Unprecond*: $\llbracket \vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]; $\vdash_I$ *P* sat_p [*pre'*, *rely*, *guar*, *post*] $\rrbracket$
$\Longrightarrow \vdash_I$ *P* sat_p [*pre* $\cup$ *pre'*, *rely*, *guar*, *post*]

| *Intpostcond*: $\llbracket \vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]; $\vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post'*] $\rrbracket$
$\Longrightarrow \vdash_I$ *P* sat_p [*pre*, *rely*, *guar*, *post* $\cap$ *post'*]

| *Allprecond*: $\forall v\in U. \vdash_I$ *P* sat_p [$\{v\}$, *rely*, *guar*, *post*]
$\Longrightarrow \vdash_I$ *P* sat_p [*U*, *rely*, *guar*, *post*]

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

definition *prog-validity* :: *'a com option* $\Rightarrow$ *'a set* $\Rightarrow$ (*'a* $\times$ *'a*) *set* $\Rightarrow$ (*'a* $\times$ *'a*) *set* $\Rightarrow$ *'a set* $\Rightarrow$ *bool*
($\models_I$ - sat_p [-, -, -, -] [*60*,*0*,*0*,*0*,*0*] *45*) **where**
$\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*] $\equiv$
$\forall s.$ *cp P s* $\cap$ *assum*(*pre*, *rely*) $\subseteq$ *comm*(*guar*, *post*)

7.2 lemmas of SIMP

lemma *etran-or-ctran2-disjI3*:

$[\![x{\in}cptn;\ Suc\ i{<}length\ x;\ \neg\ x!i\ -c\rightarrow\ x!Suc\ i]\!] \Longrightarrow x!i\ -e\rightarrow\ x!Suc\ i$
apply(*induct x arbitrary*:*i*)
 apply *simp*
apply *clarify*
apply(*rule cptn.cases*)
  apply *simp+*
  using *less-Suc-eq-0-disj etran.intros* **apply** *force*
  apply(*case-tac i*,*simp*)
  by *simp*

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

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

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

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

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

lemma *Seq2*: $[\![\vdash_I$ *Some P* sat_p [*pre*, *rely*, *guar*, *mida*]; *mida* $\subseteq$ *midb*; $\vdash_I$ *Some Q* sat_p [*midb*, *rely*, *guar*, *post*] $]\!]$
  $\Longrightarrow \vdash_I$ *Some* (*Seq P Q*) sat_p [*pre*, *rely*, *guar*, *post*]
  using *Seq*[*of P pre rely guar mida Q post*]
        Conseq[*of mida midb rely rely guar guar post post*]
  by *blast*

7.3 Soundness of the Rule of Consequence

lemma *Conseq-sound*:
  $[\![$*pre* $\subseteq$ *pre'*; *rely* $\subseteq$ *rely'*; *guar'* $\subseteq$ *guar*; *post'* $\subseteq$ *post*;
  $\models_I$ *P* sat_p [*pre'*, *rely'*, *guar'*, *post'*]$]\!]$
  $\Longrightarrow \models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
apply(*simp add*:*prog-validity-def assum-def comm-def*)
apply *clarify*
apply(*erule-tac x=s* **in** *allE*)
apply(*drule-tac c=x* **in** *subsetD*)
 apply *force*
apply *force*
done

7.4 Soundness of the Rule of Unprecond

lemma *Unprecond-sound*:
  assumes *p0*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
    and *p1*: $\models_I$ *P* sat_p [*pre'*, *rely*, *guar*, *post*]
   shows $\models_I$ *P* sat_p [*pre* $\cup$ *pre'*, *rely*, *guar*, *post*]
proof −
{
  fix *s c*
  assume $c \in cp\ P\ s \cap assum(pre \cup pre', rely)$
  hence *a1*: $c \in cp\ P\ s$ **and**
      a2: $c \in assum(pre \cup pre', rely)$ **by** *auto*
  hence $c \in assum(pre, rely) \vee c \in assum(pre', rely)$
   by (*metis* (*no-types*, *lifting*) *CollectD CollectI Un-iff assum-def prod.simps*(*2*))
  hence $c \in comm(guar, post)$
    proof
      assume $c \in assum\ (pre, rely)$
      with *p0 a1* **show** $c \in comm\ (guar, post)$
        unfolding *prog-validity-def* **by** *auto*
    next
      assume $c \in assum\ (pre', rely)$
      with *p1 a1* **show** $c \in comm\ (guar, post)$
        unfolding *prog-validity-def* **by** *auto*
    qed
}
then show *?thesis* **unfolding** *prog-validity-def* **by** *auto*
qed

7.5 Soundness of the Rule of Intpostcond

lemma *Intpostcond-sound*:
  assumes *p0*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post*]
    and *p1*: $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post'*]
   shows $\models_I$ *P* sat_p [*pre*, *rely*, *guar*, *post* $\cap$ *post'*]
proof −
{
  fix *s c*
  assume *a0*: $c \in cp\ P\ s \cap assum(pre, rely)$
  with *p0* **have** $c \in comm(guar, post)$ **unfolding** *prog-validity-def* **by** *auto*
  moreover
  from *a0 p1* **have** $c \in comm(guar, post')$ **unfolding** *prog-validity-def* **by** *auto*
  ultimately have $c \in comm(guar, post \cap post')$
    by (*simp add*: *comm-def*)
}
then show *?thesis* **unfolding** *prog-validity-def* **by** *auto*
qed

7.6 Soundness of the Rule of Allprecond

lemma *Allprecond-sound*:

assumes *p1*: $\forall v{\in}U.\ \models_I P\ sat_p\ [\{v\},\ rely,\ guar,\ post]$
shows $\models_I P\ sat_p\ [U,\ rely,\ guar,\ post]$
proof −
{
fix *s c*
assume *a0*: $c \in cp\ P\ s \cap assum(U,\ rely)$
then obtain *x* **where** *a1*: $x{\in}U \wedge snd\ (c!0) = x$
by (*metis* (*no-types*, *lifting*) *CollectD IntD2 assum-def prod.simps*(*2*))

with *p1* **have** $\models_I P\ sat_p\ [\{x\},\ rely,\ guar,\ post]$ **by** *simp*
hence *a2*: $\forall s.\ cp\ P\ s \cap assum(\{x\},\ rely) \subseteq comm(guar,\ post)$ **unfolding** *prog-validity-def* **by** *simp*

from *a0* **have** $c \in assum(U,\ rely)$ **by** *simp*
hence $snd\ (c!0) \in U \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$ **by** (*simp add*:*assum-def*)
with *a1* **have** $snd\ (c!0) \in \{x\} \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$ **by** *simp*

hence $c{\in}assum(\{x\},\ rely)$ **by** (*simp add*:*assum-def*)
with *a0 a2* **have** $c \in comm(guar,\ post)$ **by** *auto*
}
then show *?thesis* **using** *prog-validity-def* **by** *blast*
qed

7.7 Soundness of the Rule of Emptyprecond

lemma *Emptyprecond-sound*: $\models_I P\ sat_p\ [\{\},\ rely,\ guar,\ post]$
unfolding *prog-validity-def* **by**(*simp add*:*assum-def*)

7.8 Soundness of None rule

lemma *none-all-none*: $c!0{=}(None{,}s) \wedge c \in cptn \Longrightarrow \forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
proof(*induct c arbitrary*:*s*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a c*)
assume *p1*: $\bigwedge s.\ c\ !\ 0 = (None,\ s) \wedge c \in cptn \Longrightarrow \forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
and *p2*: $(a\ \#\ c)\ !\ 0 = (None,\ s) \wedge a\ \#\ c \in cptn$
hence *a0*: $a = (None{,}s)$ **by** *simp*
thus *?case*
proof(*cases* $c = []$)
case *True*
with *a0* **show** *?thesis* **by** *auto*
next
case *False*

assume $b0$: $c \neq []$
with $p2$ **have** $c\text{-}cpts$: $c \in cptn$ **using** $tl\text{-}in\text{-}cptn$ **by** $fast$
from $b0$ **obtain** c' **and** b **where** bc': $c = b \# c'$
using $list.exhaust$ **by** $blast$
from $a0$ **have** $\neg\ a\ {-c\rightarrow}\ b$ **by**($force\ elim$: $ctran.cases$)
with $p2$ **have** $a\ {-e\rightarrow}\ b$ **using** bc' $etran\text{-}or\text{-}ctran2\text{-}disjI3[of\ a\#c\ 0]$ **by** $auto$
hence $fst\ b = None$ **using** $etran.cases$
by ($metis\ a0\ prod.collapse$)
with $p1$ bc' $c\text{-}cpts$ **have** $\forall i{<}length\ c.\ fst\ (c\ !\ i) = None$
by ($metis\ nth\text{-}Cons\text{-}0\ prod.collapse$)
with $a0$ **show** $?thesis$
by ($simp\ add$: $nth\text{-}Cons'$)
qed

qed

lemma $None\text{-}sound\text{-}h$: $\forall x.\ x \in pre \longrightarrow (\forall y.\ (x, y) \in rely \longrightarrow y \in pre) \Longrightarrow$
$pre \subseteq post \Longrightarrow$
$snd\ (c\ !\ 0) \in pre \Longrightarrow$
$c \neq [] \Longrightarrow \forall i.\ Suc\ i < length\ c \longrightarrow (snd\ (c\ !\ i),\ snd\ (c\ !\ Suc\ i)) \in rely$
$\Longrightarrow i < length\ c \Longrightarrow snd\ (c\ !\ i) \in pre$
apply($induct\ i$) **by** $auto$

lemma $None\text{-}sound$:
$[\![\ stable\ pre\ rely;\ pre \subseteq post\ ]\!]$
$\Longrightarrow \models_I None\ sat_p\ [pre,\ rely,\ guar,\ post]$
proof $-$
assume $p0$: $stable\ pre\ rely$
and $p2$: $pre \subseteq post$
{
fix $s\ c$
assume $a0$: $c \in cp\ None\ s \cap assum(pre,\ rely)$
hence $c1$: $c!0{=}(None,s) \wedge c \in cptn$ **by** ($simp\ add{:}cp\text{-}def$)
from $a0$ **have** $c2$: $snd\ (c!0) \in pre \wedge (\forall i.\ Suc\ i{<}length\ c \longrightarrow$
$c!i\ {-e\rightarrow}\ c!(Suc\ i) \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in rely)$
by ($simp\ add{:}assum\text{-}def$)
from $c1$ **have** $c\text{-}ne\text{-}empty$: $c \neq []$
by $auto$
from $c1$ **have** $c\text{-}all\text{-}none$: $\forall i{<}length\ c.\ fst\ (c\ !\ i) = None$ **using** $none\text{-}all\text{-}none$
by $fast$

{
fix i
assume $suci$: $Suc\ i{<}length\ c$
and cc: $c!i\ {-c\rightarrow}\ c!(Suc\ i)$
from $suci$ $c\text{-}all\text{-}none$ **have** $c!i\ {-e\rightarrow}\ c!(Suc\ i)$
by ($metis\ Suc\text{-}lessD\ etran.intros\ prod.collapse$)
with cc **have**($snd\ (c!i),\ snd\ (c!Suc\ i)) \in guar$

using *c1 etran-or-ctran2-disjI1 suci* **by** *auto*
}
moreover
{
assume *last-none*: $fst\ (last\ c) = None$
from *c2 c-all-none* **have** $\forall\, i.\ Suc\ i{<}length\ c \longrightarrow (snd\ (c!i),\ snd\ (c!Suc\ i)) \in$ *rely*
by (*metis Suc-lessD etran.intros prod.collapse*)
with *p0 p2 c2 c-ne-empty* **have** $\forall\, i.\ i < length\ c \longrightarrow snd\ (c\ !\ i) \in pre$
apply(*simp add*: *stable-def*) **apply** *clarify* **using** *None-sound-h* **by** *blast*
with *p2 c-ne-empty* **have** $snd\ (last\ c) \in post$
using *One-nat-def c-ne-empty last-conv-nth* **by** *force*
}
ultimately have $c \in comm(guar,\ post)$ **by** (*simp add*:*comm-def*)
}
thus $\models_I None\ sat_p\ [pre,\ rely,\ guar,\ post]$ **using** *prog-validity-def* **by** *blast*
qed

7.9 Soundness of the Await rule

lemma *Await-sound*:
$[\![stable\ pre\ rely;\ stable\ post\ rely;$
$\forall\, V.\ \vdash_I Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post] \wedge$
$\models_I Some\ P\ sat_p\ [pre \cap b \cap \{s.\ s = V\},\ \{(s,\ t).\ s = t\},$
$UNIV,\ \{s.\ (V,\ s) \in guar\} \cap post]\ ]\!]$
$\Longrightarrow\ \models_I Some\ (Await\ b\ P)\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*unfold prog-validity-def*)
apply *clarify*
apply(*simp add*:*comm-def*)
apply(*rule conjI*)
apply *clarify*
apply(*simp add*:*cp-def assum-def*)
apply *clarify*
apply(*frule-tac j=0* **and** *k=i* **and** *p=pre* **in** *stability*,*simp-all*)
apply(*erule-tac x=ia* **in** *allE*,*simp*)
apply(*subgoal-tac* $x \in cp\ (Some(Await\ b\ P))\ s$)
apply(*erule-tac i=i* **in** *unique-ctran-Await*,*force*,*simp-all*)
apply(*simp add*:*cp-def*)

apply(*erule ctran.cases*,*simp-all*)
apply(*drule Star-imp-cptn*)
apply *clarify*
apply(*erule-tac x=sa* **in** *allE*)
apply *clarify*
apply(*erule-tac x=sa* **in** *allE*)
apply(*drule-tac c=l* **in** *subsetD*)
apply (*simp add*:*cp-def*)
apply *clarify*

apply(*erule-tac* x=*ia* **and** $P=\lambda i.\ H\ i \longrightarrow (J\ i,\ I\ i) \in ctran$ **for** $H\ J\ I$ **in** *allE*,*simp*)
apply(*erule etranE*,*simp*)
apply *simp*
apply *clarify*
apply(*simp add*:*cp-def*)
apply *clarify*
apply(*frule-tac* i=*length* $x - 1$ **in** *exists-ctran-Await-None*,*force*)
apply (*case-tac* x,*simp*+)
apply(*rule last-fst-esp*,*simp add*:*last-length*)
apply(*case-tac* x, *simp*+)
apply *clarify*
apply(*simp add*:*assum-def*)
apply *clarify*
apply(*frule-tac* j=*0* **and** k=j **and** p=*pre* **in** *stability*,*simp-all*)
apply(*erule-tac* x=i **in** *allE*,*simp*)
apply(*erule-tac* i=j **in** *unique-ctran-Await*,*force*,*simp-all*)
apply(*case-tac* $x!j$)
apply *clarify*
apply *simp*
apply(*drule-tac* s=*Some* (*Await* b P) **in** *sym*,*simp*)
apply(*case-tac* $x!Suc\ j$,*simp*)
apply(*rule ctran.cases*,*simp*)
apply(*simp-all*)
apply(*drule Star-imp-cptn*)
apply *clarify*
apply(*erule-tac* x=*sa* **in** *allE*)
apply *clarify*
apply(*erule-tac* x=*sa* **in** *allE*)
apply(*drule-tac* c=l **in** *subsetD*)
apply (*simp add*:*cp-def*)
apply *clarify*
apply(*erule-tac* x=i **and** $P=\lambda i.\ H\ i \longrightarrow (J\ i,\ I\ i) \in ctran$ **for** $H\ J\ I$ **in** *allE*,*simp*)
apply(*erule etranE*,*simp*)
apply *simp*
apply *clarify*
apply(*frule-tac* j=*Suc* j **and** k=*length* $x - 1$ **and** p=*post* **in** *stability*,*simp-all*)
apply(*case-tac* x,*simp*+)
apply(*erule-tac* x=i **in** *allE*)
apply(*erule-tac* i=j **in** *unique-ctran-Await*,*force*,*simp-all*)
apply *arith*+
apply(*case-tac* x)
apply(*simp add*:*last-length*)+
done

theorem *rgsound-p*:
$\vdash_I P\ sat_p\ [pre,\ rely,\ guar,\ post] \Longrightarrow \models_I P\ sat_p\ [pre,\ rely,\ guar,\ post]$
apply(*erule rghoare-p.induct*)
using *RG-Hoare.Basic-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*

using *RG-Hoare.Seq-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*) **apply** *blast*
using *RG-Hoare.Cond-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*
using *RG-Hoare.While-sound* **apply**(*simp add*:*prog-validity-def com-validity-def*)
apply *blast*
using *Await-sound* **apply** *fastforce*
apply(*force elim*:*None-sound*)
apply(*erule Conseq-sound*,*simp*+)
apply(*erule Unprecond-sound*,*simp*+)
apply(*erule Intpostcond-sound*,*simp*+)
using *Allprecond-sound* **apply** *force*
using *Emptyprecond-sound* **apply** *force*
done

end

8 Rely-guarantee-based Safety Reasoning

theory *PiCore-ext*
  imports *PiCore-Hoare*
begin

definition *list-of-set aset* ≡ (*SOME l. set l* = *aset*)

lemma *set-of-list-of-set*:
  assumes *fin*: *finite aset*
  shows *set* (*list-of-set aset*) = *aset*
proof(*simp add*: *list-of-set-def*)
  from *fin* **obtain** *l* **where** *set l* = *aset* **using** *finite-list* **by** *auto*
  then show *set* (*SOME l. set l* = *aset*) = *aset*
    by (*metis* (*mono-tags*, *lifting*) *some-eq-ex*)
qed

context *event-hoare*
begin

fun *OR-list* :: (*'l*,*'k*,*'s*,*'prog*) *esys list* ⇒ (*'l*,*'k*,*'s*,*'prog*) *esys* **where**
  OR-list [*a*] = *a* |
  OR-list (*a*#*b*#*ax*) = *a OR* (*OR-list* (*b*#*ax*)) |
  OR-list [] = *fin*

lemma *OR-list* [*a*] = *a* **by** *auto*
lemma *OR-list* [*a*,*b*] = *a OR b* **by** *auto*
lemma *OR-list* [*a*,*b*,*c*] = *a OR* (*b OR c*) **by** *auto*

lemma *Evt-OR-list*:

$ess \neq [] \Longrightarrow \forall i{<}length\ ess.\ \Gamma \vdash (ess!i)\ sat_e\ [pre,\ rely,\ guar,\ post]$
$\Longrightarrow \Gamma \vdash (OR\text{-}list\ ess)\ sat_e\ [pre,\ rely,\ guar,\ post]$
apply(*induct ess*) **apply** *simp*
apply(*case-tac ess=[]*) **apply** *auto[1]*
by (*metis Evt-Choice OR-list.simps(2) length-Cons less-Suc-eq-0-disj list.exhaust nth-Cons-0 nth-Cons-Suc*)

fun *AND-list* :: *('l,'k,'s,'prog) esys list* $\Rightarrow$ *('l,'k,'s,'prog) esys* **where**
$AND\text{-}list\ [a] = a\ |$
$AND\text{-}list\ (a\#b\#ax) = a \bowtie (AND\text{-}list\ (b\#ax))\ |$
$AND\text{-}list\ [] = fin$

lemma $AND\text{-}list\ [a] = a$ **by** *auto*
lemma $AND\text{-}list\ [a,b] = a \bowtie b$ **by** *auto*
lemma $AND\text{-}list\ [a,b,c] = a \bowtie (b \bowtie c)$ **by** *auto*

lemma *Int-list-lm*: $P\ a \cap (\bigcap i{<}length\ ess.\ P\ (ess\ !\ i)) = (\bigcap i{<}length\ (a\ \#\ ess).\ P\ ((a\ \#\ ess)\ !\ i))$
apply(*induct ess*) **apply** *auto[1]*
apply(*rule subset-antisym*)
apply *auto[1]* **apply** (*metis lessThan-iff less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc*)
apply *auto*
by (*metis Suc-leI le-imp-less-Suc lessThan-iff nth-Cons-Suc*)

lemma *Evt-AND-list*:
$ess \neq [] \Longrightarrow$
$\forall i{<}length\ ess.\ \Gamma \vdash Com\ (ess!i)\ sat_e\ [Pre\ (ess!i),\ Rely\ (ess!i),\ Guar\ (ess!i),\ Post\ (ess!i)] \Longrightarrow$
$\forall i{<}length\ ess.\ \forall s.\ (s,s){\in}Guar\ (ess!i) \Longrightarrow$
$\forall i\ j.\ i < length\ ess \wedge j < length\ ess \wedge i \neq j \longrightarrow Guar\ (ess!i) \subseteq Rely\ (ess!j) \Longrightarrow$
$\Gamma \vdash (AND\text{-}list\ (map\ Com\ ess))\ sat_e\ [\bigcap i{<}length\ ess.\ Pre\ (ess!i),\ \bigcap i{<}length\ ess.\ Rely\ (ess!i),$
$\bigcup i{<}length\ ess.\ Guar\ (ess!i),\ \bigcap i{<}length\ ess.\ Post\ (ess!i)]$
apply(*induct ess*) **apply** *simp*
apply(*case-tac ess=[]*) **apply** *auto[1]*
proof−
fix *a ess*

assume *a0*: $ess \neq [] \Longrightarrow$
$\forall i{<}length\ ess.\ \Gamma \vdash Com\ (ess\ !\ i)\ sat_e\ [Pre\ (ess\ !\ i),\ Rely\ (ess\ !\ i),\ Guar\ (ess\ !\ i),\ Post\ (ess\ !\ i)] \Longrightarrow$
$\forall i{<}length\ ess.\ \forall s.\ (s,\ s) \in Guar\ (ess\ !\ i) \Longrightarrow$
$\forall i\ j.\ i < length\ ess \wedge j < length\ ess \wedge i \neq j \longrightarrow Guar\ (ess\ !\ i) \subseteq Rely\ (ess\ !\ j) \Longrightarrow$
$\Gamma \vdash AND\text{-}list\ (map\ Com\ ess)\ sat_e\ [\bigcap i{<}length\ ess.\ Pre\ (ess\ !\ i),\ \bigcap i{<}length\ ess.\ Rely\ (ess\ !\ i),$
$\bigcup i{<}length\ ess.\ Guar\ (ess\ !\ i),\ \bigcap i{<}length\ ess.\ Post\ (ess\ !\ i)]$

and $a1$: $a \# ess \neq []$
and $a2$: $\forall i{<}length\ (a \# ess).\ \Gamma \vdash Com\ ((a \# ess)\ !\ i)\ sat_e\ [Pre\ ((a \# ess)\ !\ i),$
$Rely\ ((a \# ess)\ !\ i),\ Guar\ ((a \# ess)\ !\ i),\ Post\ ((a \# ess)\ !\ i)]$
and $a3$: $\forall i{<}length\ (a \# ess).\ \forall s.\ (s,\ s) \in Guar\ ((a \# ess)\ !\ i)$
and $a4$: $\forall i\ j.\ i < length\ (a \# ess) \wedge j < length\ (a \# ess) \wedge i \neq j$
$\longrightarrow Guar\ ((a \# ess)\ !\ i) \subseteq Rely\ ((a \# ess)\ !\ j)$
and $a5$: $ess \neq []$
let $?pre = \bigcap i{<}length\ ess.\ Pre\ (ess\ !\ i)$
let $?rely = \bigcap i{<}length\ ess.\ Rely\ (ess\ !\ i)$
let $?guar = \bigcup i{<}length\ ess.\ Guar\ (ess\ !\ i)$
let $?post = \bigcap i{<}length\ ess.\ Post\ (ess\ !\ i)$
let $?pre' = \bigcap i{<}length\ (a \# ess).\ Pre\ ((a \# ess)\ !\ i)$
let $?rely' = \bigcap i{<}length\ (a \# ess).\ Rely\ ((a \# ess)\ !\ i)$
let $?guar' = \bigcup i{<}length\ (a \# ess).\ Guar\ ((a \# ess)\ !\ i)$
let $?post' = \bigcap i{<}length\ (a \# ess).\ Post\ ((a \# ess)\ !\ i)$

from $a2$ **have** $a6$: $\forall i{<}length\ ess.\ \Gamma \vdash Com\ (ess\ !\ i)\ sat_e\ [Pre\ (ess\ !\ i),\ Rely\ (ess\ !\ i),\ Guar\ (ess\ !\ i),\ Post\ (ess\ !\ i)]$
by *auto*
moreover
from $a3$ **have** $a7$: $\forall i{<}length\ ess.\ \forall s.\ (s,\ s) \in Guar\ (ess\ !\ i)$ **by** *auto*
moreover
from $a4$ **have** $a8$: $\forall i\ j.\ i < length\ ess \wedge j < length\ ess \wedge i \neq j \longrightarrow Guar\ (ess\ !\ i) \subseteq Rely\ (ess\ !\ j)$
by *fastforce*
ultimately have $b1$: $\Gamma \vdash AND\text{-}list\ (map\ Com\ ess)\ sat_e\ [?pre,\ ?rely,\ ?guar,\ ?post]$
using $a0$ $a5$ **by** *auto*
have $b2$: $AND\text{-}list\ (map\ Com\ (a \# ess)) = Com\ a \bowtie AND\text{-}list\ (map\ Com\ ess)$
by (*metis* (*no-types*, *hide-lams*) *AND-list.simps*(*2*) *a5 list.exhaust list.simps*(*9*))
from $a2$ **have** $b3$: $\Gamma \vdash Com\ a\ sat_e\ [Pre\ a,\ Rely\ a,\ Guar\ a,\ Post\ a]$
by *fastforce*
have $b4$: $\Gamma \vdash AND\text{-}list\ (map\ Com\ ess)\ sat_e\ [?pre',\ ?rely,\ ?guar,\ ?post]$
apply(*rule Evt-conseq*[*of ?pre' ?pre ?rely ?rely ?guar ?guar ?post ?post*])
apply *fastforce* **using** $b1$ **by** *simp+*
have $b5$: $\Gamma \vdash Com\ a\ sat_e\ [?pre',\ Rely\ a,\ Guar\ a,\ Post\ a]$
apply(*rule Evt-conseq*[*of ?pre' Pre a Rely a Rely a Guar a Guar a Post a Post a*])
apply *fastforce*
using $b3$ **by** *simp+*
show $\Gamma \vdash AND\text{-}list\ (map\ Com\ (a \# ess))\ sat_e\ [?pre',\ ?rely',\ ?guar',\ ?post']$
apply(*rule subst*[**where** $t{=}AND\text{-}list\ (map\ Com\ (a \# ess))$ **and** $s{=}\ Com\ a \bowtie AND\text{-}list\ (map\ Com\ ess)$])
using $b2$ **apply** *simp*
apply(*rule subst*[**where** $s{=}Post\ a \cap ?post$ **and** $t{=}?post'$])
prefer *2*
apply(*rule Evt-Join*[*of* Γ *Com a ?pre' Rely a Guar a Post a AND-list* (*map Com ess*)

?pre' ?rely ?guar ?post ?pre' ?rely' ?guar'])
using *b5* **apply** *fast*
using *b4* **apply** *fast*
apply *blast*
apply(*rule Un-least*) **apply** *fastforce* **apply** *clarsimp* **using** *a4*
apply (*smt Suc-mono a1 drop-Suc-Cons hd-drop-conv-nth length-Cons length-greater-0-conv nat.simps*(*3*) *nth-Cons-0 set-mp*)
apply(*rule Un-least*) **apply** *fastforce* **apply** *clarsimp* **using** *a4*
apply (*smt Suc-mono a1 drop-Suc-Cons hd-drop-conv-nth length-Cons length-greater-0-conv nat.simps*(*3*) *nth-Cons-0 set-mp*)
using *a3* **apply** *force* **using** *a3 a5 a7* **apply** *auto*[*1*]
apply *auto*[*1*]
using *Int-list-lm* **by** *metis*
qed

lemma *Evt-AND-list2*:
$ess \neq [] \Longrightarrow$
$\forall\, i<length\ ess.\ \Gamma \vdash Com\ (ess!i)\ sat_e\ [Pre\ (ess!i),\ Rely\ (ess!i),\ Guar\ (ess!i),\ Post\ (ess!i)] \Longrightarrow$
$\forall\, i<length\ ess.\ \forall\, s.\ (s,s){\in}Guar\ (ess!i) \Longrightarrow$
$\forall\, i<length\ ess.\ P \subseteq Pre\ (ess!i) \Longrightarrow$
$\forall\, i<length\ ess.\ Guar\ (ess!i) \subseteq G \Longrightarrow$
$\forall\, i<length\ ess.\ R \subseteq Rely\ (ess!i) \Longrightarrow$
$\forall\, i\ j.\ i<length\ ess \wedge j < length\ ess \wedge i \neq j \longrightarrow Guar\ (ess!i) \subseteq Rely\ (ess!j) \Longrightarrow$
$\forall\, i<length\ ess.\ Post\ (ess!i) \subseteq Q \Longrightarrow$
$\Gamma \vdash (AND\text{-}list\ (map\ Com\ ess))\ sat_e\ [P,\ R,\ G,\ Q]$

apply(*rule Evt-conseq*[*of P* $\bigcap$ *i<length ess. Pre* (*ess*!*i*)
R $\bigcap$ *i<length ess. Rely* (*ess*!*i*)
$\bigcup$ *i<length ess. Guar* (*ess*!*i*) *G*
$\bigcap$ *i<length ess. Post* (*ess*!*i*) *Q*
Γ *AND-list* (*map Com ess*)])
apply *fast* **apply** *fast* **apply** *fast* **apply** *fastforce*
using *Evt-AND-list* **by** *metis*

definition ‹*react-sys l* ≡ *EWhile UNIV* (*OR-list l*)›

lemma *fin-sat*:
‹$stable\ P\ R \Longrightarrow \Gamma \models fin\ sat_e\ [P,\ R,\ G,\ P]$›
proof(*simp*, *rule allI*, *rule allI*, *standard*)
let *?P* = ‹*lift-state-set P*›
let *?R* = ‹*lift-state-pair-set R*›
let *?G* = ‹*lift-state-pair-set G*›

fix *s0 x0*
fix *cpt*
assume *stable*: ‹*stable P R*›
assume ‹$cpt \in \{cpt \in cpts\ (estran\ \Gamma).\ hd\ cpt = (fin,\ s0,\ x0)\} \cap assume\ ?P\ ?R$›

```
  then have cpt: ‹cpt ∈ cpts (estran Γ)› and hd-cpt: ‹hd cpt = (fin, s0, x0)› and
cpt-assume: ‹cpt ∈ assume ?P ?R› by auto
  from cpts-nonnil[OF cpt] have ‹cpt≠[]› .
  from hd-cpt ‹cpt≠[]› obtain cs where cpt-Cons: ‹cpt = (fin, s0, x0) # cs› by
(metis hd-Cons-tl)
  from all-etran-from-fin[OF cpt cpt-Cons] have all-etran: ‹∀ i. Suc i < length cpt
⟶ cpt ! i −e→ cpt ! Suc i› .
  show ‹cpt ∈ commit (estran Γ) {fin} ?G ?P›
  proof(auto simp add: commit-def)
    fix i
    assume Suc-i-lt: ‹Suc i < length cpt›
    assume ctran: ‹(cpt ! i, cpt ! Suc i) ∈ estran Γ›
    from all-etran[rule-format, OF Suc-i-lt] have ‹cpt ! i −e→ cpt ! Suc i› .
    from etran-imp-not-ctran[OF this] have ‹(cpt ! i, cpt ! Suc i) ∉ estran Γ› .
    with ctran show ‹(snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?G› by blast
  next
    assume ‹fst (last cpt) = fin›
    have ‹∀ i<length cpt. snd (cpt!i) ∈ ?P›
    proof(auto)
      fix i
      assume i-lt: ‹i < length cpt›
      show ‹snd (cpt ! i) ∈ ?P›
        using i-lt
      proof(induct i)
        case 0
        then show ?case
          apply(subst hd-conv-nth[symmetric])
           apply(rule ‹cpt≠[]›)
          using cpt-assume by (simp add: assume-def)
      next
        case (Suc i)
        then show ?case
        proof−
          assume 1: ‹i < length cpt ⟹ snd (cpt ! i) ∈ ?P›
          assume Suc-i-lt: ‹Suc i < length cpt›
          with 1 have ‹snd (cpt ! i) ∈ ?P› by simp
          from all-etran[rule-format, OF Suc-i-lt] have ‹cpt ! i −e→ cpt ! Suc i› .
          with cpt-assume have ‹(snd (cpt ! i), snd (cpt ! Suc i)) ∈ ?R›
            apply(auto simp add: assume-def)
            using Suc-i-lt by blast
          with stable show ‹snd (cpt ! Suc i) ∈ ?P›
            apply(simp add: stable-def)
          using ‹snd (cpt ! i) ∈ ?P› by (simp add: lift-state-set-def lift-state-pair-set-def
case-prod-unfold)
        qed
      qed
    qed
    then show ‹snd (last cpt) ∈ ?P› using ‹cpt≠[]›
```

apply−
apply(*subst last-conv-nth*)
apply *assumption*
by *simp*
qed
qed

lemma *Evt-react-list*:
‹⟦ ∀ *i*<*length* (*rgfs*::(('*l*,'*k*,'*s*,'*prog*) *esys*,'*s*) *rgformula list*). Γ ⊢ *Com* (*rgfs*!*i*) sat_e [*Pre* (*rgfs*!*i*), *Rely* (*rgfs*!*i*), *Guar* (*rgfs*!*i*), *Post* (*rgfs*!*i*)] ∧
pre ⊆ *Pre* (*rgfs*!*i*) ∧ *rely* ⊆ *Rely* (*rgfs*!*i*) ∧
Guar (*rgfs*!*i*) ⊆ *guar* ∧
Post (*rgfs*!*i*) ⊆ *pre*; *rgfs* ≠ [];
stable pre rely; ∀ *s*. (*s*, *s*)∈*guar* ⟧ ⟹
Γ ⊢ *react-sys* (*map Com rgfs*) sat_e [*pre*, *rely*, *guar*, *pre*]›
apply (*unfold react-sys-def*)
apply (*rule Evt-While*)
apply *assumption*
apply *fast*
apply *assumption*
apply (*simp add*: *list-of-set-def*)
apply(*rule Evt-OR-list*)
apply *simp*
apply *simp*
apply(*rule allI*)
apply(*rule impI*)
apply(*rule-tac pre'*=‹*Pre* (*rgfs*!*i*)› **and** *rely'*=‹*Rely* (*rgfs*!*i*)› **and** *guar'*=‹*Guar* (*rgfs*!*i*)› **and** *post'*=‹*Post* (*rgfs*!*i*)› **in** *Evt-conseq*)
apply *simp*+
done

lemma *Evt-react-set*:
‹⟦ ∀ *rgf* ∈ (*rgfs*::(('*l*,'*k*,'*s*,'*prog*) *esys*,'*s*) *rgformula set*). Γ ⊢ *Com rgf* sat_e [*Pre rgf*, *Rely rgf*, *Guar rgf*, *Post rgf*] ∧
pre ⊆ *Pre rgf* ∧ *rely* ⊆ *Rely rgf* ∧
Guar rgf ⊆ *guar* ∧
Post rgf ⊆ *pre*; *rgfs* ≠ {}; *finite rgfs*;
stable pre rely; ∀ *s*. (*s*, *s*)∈*guar* ⟧ ⟹
Γ ⊢ *react-sys* (*map Com* (*list-of-set rgfs*)) sat_e [*pre*, *rely*, *guar*, *pre*]›
apply(*rule Evt-react-list*)
apply(*simp add*: *list-of-set-def*)
apply (*smt finite-list nth-mem tfl-some*)
apply(*simp add*: *list-of-set-def*)
apply (*metis* (*mono-tags*, *lifting*) *empty-set finite-list tfl-some*)
apply *assumption*
apply *assumption*
done

lemma *Evt-react-set'*:

‹⟦ $\forall$ *rgf* ∈ (*rgfs*::(('*l*,'*k*,'*s*,'*prog*) *esys*,'*s*) *rgformula set*). Γ ⊢ *Com rgf* sat_e [*Pre rgf*, *Rely rgf*, *Guar rgf*, *Post rgf*] ∧
pre ⊆ *Pre rgf* ∧ *rely* ⊆ *Rely rgf* ∧
Guar rgf ⊆ *guar* ∧
Post rgf ⊆ *pre*; *rgfs* ≠ {}; *finite rgfs*;
stable pre rely; ∀ *s*. (*s*, *s*)∈*guar*; *pre* ⊆ *post* ⟧ ⟹
Γ ⊢ *react-sys* (*map Com* (*list-of-set rgfs*)) sat_e [*pre*, *rely*, *guar*, *post*]›
apply(*subgoal-tac* ‹Γ ⊢ *react-sys* (*map Com* (*list-of-set rgfs*)) sat_e [*pre*, *rely*, *guar*, *pre*]›)
using *Evt-conseq* **apply** *blast*
using *Evt-react-set* **apply** *blast*
done

end

end

9 Integrating the SIMP language into Picore

theory *picore-SIMP*
imports *../picore/PiCore-RG-Invariant SIMP-plus ../picore/PiCore-ext*
begin

abbreviation *ptranI* :: '*Env* ⇒ ('*a conf* × '*a conf*) *set*
where *ptranI* Γ ≡ *ctran*

abbreviation *prog-validityI* :: '*Env* ⇒ ('*a com*) *option* ⇒ '*a set* ⇒ ('*a* × '*a*) *set* ⇒ ('*a* × '*a*) *set* ⇒ '*a set* ⇒ *bool*
where *prog-validityI* Γ *P* ≡ *prog-validity P*

abbreviation *rghoare-pI* :: '*Env* ⇒ [('*a com*) *option*, '*a set*, ('*a* × '*a*) *set*, ('*a* × '*a*) *set*, '*a set*] ⇒ *bool*
(- $\vdash_I$ - sat_p [-, -, -, -] [*60*,*0*,*0*,*0*,*0*] *45*)
where *rghoare-pI* Γ ≡ *rghoare-p*

lemma *none-no-tranI'*: ((*Q*, *s*),(*P*,*t*)) ∈ *ptranI* Γ ⟹ *Q* ≠ *None*
apply (*simp*) **apply**(*rule ctran.cases*)
by *simp*+

lemma *none-no-tranI*: ((*None*, *s*),(*P*,*t*)) ∉ *ptranI* Γ
using *none-no-tranI'*
by *fast*

lemma *ptran-neqI*: ((*P*, *s*),(*P*,*t*)) ∉ *ptranI* Γ
by (*simp*)

lemma *eventI*: ‹*event ptranI None*›
apply (*rule event.intro*)
apply(*rule none-no-tranI*)

apply(*rule ptran-neqI*)
done

interpretation *event ptranI None*
by(*rule eventI*)

lemma *event-compI*: ‹*event-comp ptranI None*›
apply(*rule event-comp.intro*)
by(*rule eventI*)

interpretation *event-comp ptranI None*
by(*rule event-compI*)

lemma *rgsound-pI*: *rghoare-pI* Γ *P pre rely guar post* $\Longrightarrow$ *prog-validityI* Γ *P pre rely guar post*
using *rgsound-p* **by** *blast*

```
lemma cptn-equiv: ‹cptn = cpts ctran›
proof
  show ‹cptn ⊆ cpts ctran›
  proof
    fix cpt
    assume ‹cpt ∈ cptn›
    then show ‹cpt ∈ cpts ctran›
    proof(induct, auto)
      fix P s Q t xs
      assume ‹(P, s) −c→ (Q, t)›
      moreover assume ‹(Q, t) # xs ∈ cpts ctran›
      ultimately show ‹(P, s) # (Q, t) # xs ∈ cpts ctran›
        by (rule CptsComp)
    qed
  qed
next
  show ‹cpts ctran ⊆ cptn›
  proof
    fix cpt
    assume ‹cpt ∈ cpts ctran›
    then show ‹cpt ∈ cptn›
    proof(induct)
      case (CptsOne P s)
      then show ?case by (rule CptnOne)
    next
      case (CptsEnv P t cs s)
      then show ?case using CptnEnv by fast
    next
      case (CptsComp P s Q t cs)
      then show ?case
        apply −
        apply(rule CptnComp, assumption+)
```

```
      done
    qed
  qed
qed

lemma etran-equiv-aux: ‹(P,s) −e→ (Q,t) = (P,s) −e→ (Q,t)›
  apply auto
   apply(erule etran.cases, auto)
  apply(rule Env)
  done

lemma etran-equiv: ‹c1 −e→ c2 = c1 −e→ c2›
  using etran-equiv-aux surjective-pairing by metis

lemma cp-inter-assum-equiv: ‹cp P s ∩ assum (pre, rely) = {cpt ∈ cpts ctran. hd
cpt = (P, s)} ∩ assume pre rely›
proof
   show ‹cp P s ∩ assum (pre, rely) ⊆ {cpt ∈ cpts ctran. hd cpt = (P, s)} ∩
assume pre rely›
  proof
    fix cpt
    assume ‹cpt ∈ cp P s ∩ assum (pre, rely)›
    then show ‹cpt ∈ {cpt ∈ cpts ctran. hd cpt = (P, s)} ∩ assume pre rely›
      apply(auto simp add: cp-def cptn-equiv assum-def assume-def etran-equiv)
      by (simp add: hd-conv-nth cpts-nonnil)+
  qed
next
  show ‹{cpt ∈ cpts ctran. hd cpt = (P, s)} ∩ assume pre rely ⊆ cp P s ∩ assum
(pre, rely)›
  proof
    fix cpt
    assume ‹cpt ∈ {cpt ∈ cpts ctran. hd cpt = (P, s)} ∩ assume pre rely›
    then show ‹cpt ∈ cp P s ∩ assum (pre, rely)›
      apply(auto simp add: cp-def cptn-equiv assum-def assume-def etran-equiv)
      by (simp add: hd-conv-nth cpts-nonnil)+
  qed
qed

lemma comm-equiv: ‹comm (guar, post) = commit ctran {None} guar post›
  by (simp add: comm-def commit-def)

lemma prog-validity-defI: ‹⊨_I P sat_p [pre, rely, guar, post] ⟹ validity ctran
{None} P pre rely guar post›
  by (simp add: prog-validity-def cp-inter-assum-equiv comm-equiv)

interpretation event-hoare ptranI None prog-validityI rghoare-pI
  apply(rule event-hoare.intro)
   apply(rule event-validity.intro)
    apply(rule event-compI)
```

apply(*rule event-validity-axioms.intro*)
apply(*erule prog-validity-defI*)
apply(*rule event-hoare-axioms.intro*)
using *rgsound-pI* **by** *blast*

end

10 Concrete Syntax of PiCore-SIMP

theory *picore-SIMP-Syntax*
imports *picore-SIMP*

begin

syntax
-quote :: $'b \Rightarrow ('s \Rightarrow 'b)$ ((≪-≫) [*0*] *1000*)
-antiquote :: $('s \Rightarrow 'b) \Rightarrow 'b$ (´- [*1000*] *1000*)
-Assert :: $'s \Rightarrow 's$ *set* ((⦃-⦄) [*0*] *1000*)

translations
⦃*b*⦄ ⇀ *CONST Collect* ≪*b*≫

parse-translation ‹
let
fun quote-tr [*t*] = *Syntax-Trans.quote-tr* @{*syntax-const -antiquote*} *t*
| *quote-tr ts* = *raise TERM* (*quote-tr*, *ts*);
in [(@{*syntax-const -quote*}, *K quote-tr*)] *end*
›

definition *Skip* :: *'s com* (*SKIP*)
where *SKIP* ≡ *Basic id*

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

syntax
-Assign :: *idt* ⇒ *'b* ⇒ *'s com* ((´- :=/ -) [*70*, *65*] *61*)
-Cond :: *'s bexp* ⇒ *'s com* ⇒ *'s com* ⇒ *'s com* ((*0IF* -/ *THEN* -/ *ELSE* -/*FI*) [*0*, *0*, *0*] *61*)
-Cond2 :: *'s bexp* ⇒ *'s com* ⇒ *'s com* ((*0IF* - *THEN* - *FI*) [*0*,*0*] *62*)
-While :: *'s bexp* ⇒ *'s com* ⇒ *'s com* ((*0WHILE* - /*DO* - /*OD*) [*0*, *0*] *61*)
-Await :: *'s bexp* ⇒ *'s com* ⇒ *'s com* ((*0AWAIT* - /*THEN* /- /*END*) [*0*,*0*] *61*)
-Atom :: *'s com* ⇒ *'s com* ((*0ATOMIC* - *END*) *61*)
-Wait :: *'s bexp* ⇒ *'s com* ((*0WAIT* - *END*) *61*)

-For :: *'s com* ⇒ *'s bexp* ⇒ *'s com* ⇒ *'s com* ⇒ *'s com* ((*0FOR -;/ -;/ -/ DO -/ ROF*))
-Event :: [*'a*, *'a*, *'a*] ⇒ (*'l*,*'s*,*'s com option*) *event* ((*EVENT - WHEN - THEN - END*) [*0*,*0*,*0*] *61*)
-Event2 :: [*'a*, *'a*] ⇒ (*'l*,*'s*,*'s com option*) *event* ((*EVENT - THEN - END*) [*0*,*0*] *61*)
-Event-a :: [*'a*, *'a*, *'a*] ⇒ (*'l*,*'s*,*'s com option*) *event* (($EVENT_A$ *- WHEN - THEN - END*) [*0*,*0*,*0*] *61*)
-Event-a2 :: [*'a*, *'a*] ⇒ (*'l*,*'s*,*'s com option*) *event* (($EVENT_A$ *- THEN - END*) [*0*,*0*] *61*)

translations
´x := a ⇀ *CONST Basic* ≪*´*(*-update-name x* (λ-. *a*))≫
IF b THEN c1 ELSE c2 FI ⇀ *CONST Cond* ⦃*b*⦄ *c1 c2*
IF b THEN c FI ⇌ *IF b THEN c ELSE SKIP FI*
WHILE b DO c OD ⇀ *CONST While* ⦃*b*⦄ *c*
AWAIT b THEN c END ⇌ *CONST Await* ⦃*b*⦄ *c*

ATOMIC c END ⇌ *AWAIT CONST True THEN c END*
WAIT b END ⇌ *AWAIT b THEN SKIP END*
FOR a; *b*; *c DO p ROF* ⇀ *a*;; *WHILE b DO p*;;*c OD*
EVENT l WHEN g THEN bd END ⇀ *CONST EBasic* (*l*, ⦃*g*⦄, *CONST Some bd*)
EVENT l THEN bd END ⇌ *EVENT l WHEN CONST True THEN bd END*
$EVENT_A$ *l WHEN g THEN bd END* ⇀ *CONST EAtom* (*l*, ⦃*g*⦄, *CONST Some bd*)
$EVENT_A$ *l THEN bd END* ⇌ $EVENT_A$ *l WHEN CONST True THEN bd END*

Translations for variables before and after a transition:

syntax
-before :: *id* ⇒ *'a* (º-)
-after :: *id* ⇒ *'a* (ª-)

translations
º*x* ⇌ *x ´CONST fst*
ª*x* ⇌ *x ´CONST snd*

print-translation ‹
let
fun quote-tr' f (*t* :: *ts*) =
Term.list-comb (*f* $ *Syntax-Trans.quote-tr'* @{*syntax-const -antiquote*} *t*, *ts*)
| *quote-tr'* - - = *raise Match*;

val assert-tr' = *quote-tr'* (*Syntax.const* @{*syntax-const -Assert*});

fun bexp-tr' name ((*Const* (@{*const-syntax Collect*}, -) $ *t*) :: *ts*) =
quote-tr' (*Syntax.const name*) (*t* :: *ts*)
| *bexp-tr'* - - = *raise Match*;

fun assign-tr′ (*Abs* (*x*, *-*, *f* $ *k* $ *Bound 0*) :: *ts*) =
quote-tr′ (*Syntax.const* @{*syntax-const -Assign*} $ *Syntax-Trans.update-name-tr′* *f*)
(*Abs* (*x*, *dummyT*, *Syntax-Trans.const-abs-tr′* *k*) :: *ts*)
| *assign-tr′* *-* = *raise Match*;
in
[(@{*const-syntax Collect*}, *K assert-tr′*),
(@{*const-syntax Basic*}, *K assign-tr′*),
(@{*const-syntax Cond*}, *K* (*bexp-tr′* @{*syntax-const -Cond*})),
(@{*const-syntax While*}, *K* (*bexp-tr′* @{*syntax-const -While*}))]
end
›

lemma *colltrue-eq-univ*[*simp*]: ⦃*True*⦄ = *UNIV* **by** *auto*

end

11 Lemmas of Picore-SIMP

theory *picore-SIMP-lemma*
imports *picore-SIMP-Syntax picore-SIMP*

begin

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

lemma *allpre-eq-pre*: (∀ *v*∈*U*. $\vdash_I$ *P* sat_p [{*v*}, *rely*, *guar*, *post*]) ⟷ $\vdash_I$ *P* sat_p [*U*, *rely*, *guar*, *post*]
apply *auto* **using** *Allprecond* **apply** *blast*
using *Conseq*[*of - - rely rely guar guar post post P*] **by** *auto*

lemma *sat-pre-imp-allinpre*: $\vdash_I$ *P* sat_p [*U*, *rely*, *guar*, *post*] ⟹ *v*∈*U* ⟹ $\vdash_I$ *P* sat_p [{*v*}, *rely*, *guar*, *post*]
using *Conseq*[*of - - rely rely guar guar post post P*] **by** *auto*

lemma *stable-int-col2*: *stable* ⦃*s*⦄ *r* ⟹ *stable* ⦃*t*⦄ *r* ⟹ *stable* ⦃*s* ∧ *t*⦄ *r*
by *auto*

lemma *stable-int-col3*: *stable* ⦃*k*⦄ *r* ⟹ *stable* ⦃*s*⦄ *r* ⟹ *stable* ⦃*t*⦄ *r* ⟹ *stable* ⦃*k* ∧ *s* ∧ *t*⦄ *r*
by *auto*

lemma *stable-int-col4*: *stable* ⦃*m*⦄ *r* ⟹ *stable* ⦃*k*⦄ *r* ⟹ *stable* ⦃*s*⦄ *r*
⟹ *stable* ⦃*t*⦄ *r* ⟹ *stable* ⦃*m* ∧ *k* ∧ *s* ∧ *t*⦄ *r*
by *auto*

lemma *stable-int-col5*: *stable* ⦃*q*⦄ *r* ⟹ *stable* ⦃*m*⦄ *r* ⟹ *stable* ⦃*k*⦄ *r*

$\Longrightarrow$ *stable* $\{\!|s|\!\}$ $r \Longrightarrow$ *stable* $\{\!|t|\!\}$ $r \Longrightarrow$ *stable* $\{\!|q \wedge m \wedge k \wedge s \wedge t|\!\}$ r
by *auto*

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

lemma *stable-un-R*: *stable* s $r \Longrightarrow$ *stable* s $r' \Longrightarrow$ *stable* s $(r \cup r')$
by (*meson UnE stable-def*)

lemma *stable-un-S*: $\forall\, t.$ *stable* s $(P\ t) \Longrightarrow$ *stable* s $(\bigcup t.\ P\ t)$
apply(*simp add*:*stable-def*) **by** *auto*

lemma *stable-un-S2*: $\forall\, t\ x.$ *stable* s $(P\ t\ x) \Longrightarrow$ *stable* s $(\bigcup t\ x.\ P\ t\ x)$
apply(*simp add*:*stable-def*) **by** *auto*

lemma *pairv-IntI*:
$y \in \{\!|\acute{}(Pair\ V) \in A|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in B|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in A \cap B|\!\}$
by *auto*

lemma *pairv-rId*:
$y \in \{\!|\acute{}(Pair\ V) \in A|\!\} \Longrightarrow y \in \{\!|\acute{}(Pair\ V) \in A \cup Id|\!\}$
by *auto*

end