An Isabelle/HOL Formalization of the Modular Assembly Kit for Security Properties

Oliver Bračevac, Richard Gay, Sylvia Grewe,
Heiko Mantel, Henning Sudbrock, Markus Tasch

Abstract

The "Modular Assembly Kit for Security Properties" (MAKS) is a framework for both the definition and verification of possibilistic information-flow security properties at the specification-level. MAKS supports the uniform representation of a wide range of possibilistic information-flow properties and provides support for the verification of such properties via unwinding results and compositionality results. We provide a formalization of this framework in Isabelle/HOL.

Contents

1 Introduction

This is a formalization of the Modular Assembly Kit for Security Properties (MAKS) [2, 3] in its version from [3]. We provide a more detailed explanation on how key concepts of MAKS are formalized in Isabelle/HOL in [1].

2 Basic Definitions

In the following, we define the notion of prefixes and the notion of projection. These definitions are preliminaries for the remaining parts of the Isabelle/HOL formalization of MAKS.

theory *Prefix*
imports *Main*
begin

definition *prefix* :: *'e list* ⇒ *'e list* ⇒ *bool* (**infixl** ‹⪯› *100*)
where
(*l1* ⪯ *l2*) ≡ (∃ *l3*. *l1* @ *l3* = *l2*)

definition *prefixclosed* :: (*'e list*) *set* ⇒ *bool*
where
prefixclosed tr ≡ (∀ *l1* ∈ *tr*. ∀ *l2*. *l2* ⪯ *l1* ⟶ *l2* ∈ *tr*)

lemma *empty-prefix-of-all*: [] ⪯ *l*
using *prefix-def* [*of* [] *l*] **by** *simp*

lemma *empty-trace-contained*: ⟦ *prefixclosed tr* ; *tr* ≠ {} ⟧ ⟹ [] ∈ *tr*
proof −
assume *1*: *prefixclosed tr* **and**
2: *tr* ≠ {}
then obtain *l1* **where** *l1* ∈ *tr*
by *auto*
with *1* **have** ∀ *l2*. *l2* ⪯ *l1* ⟶ *l2* ∈ *tr*
by (*simp add*: *prefixclosed-def*)
thus [] ∈ *tr*
by (*simp add*: *empty-prefix-of-all*)
qed

lemma *transitive-prefix*: ⟦ *l1* ⪯ *l2* ; *l2* ⪯ *l3* ⟧ ⟹ *l1* ⪯ *l3*
by (*auto simp add*: *prefix-def*)

end
theory *Projection*
imports *Main*
begin

definition *projection*:: *'e list* ⇒ *'e set* ⇒ *'e list* (**infixl** ‹↾› *100*)
where
$l \restriction E \equiv$ *filter* $(\lambda x\ .\ x \in E)\ l$

lemma *projection-on-union*:
$l \restriction Y = [] \Longrightarrow l \restriction (X \cup Y) = l \restriction X$
proof (*induct l*)
case *Nil* **show** *?case* **by** (*simp add*: *projection-def*)
next
case (*Cons a b*) **show** *?case*
proof (*cases* $a \in Y$)
case *True* **from** *Cons* **show** $a \in Y \Longrightarrow (a \# b) \restriction (X \cup Y) = (a \# b) \restriction X$
by (*simp add*: *projection-def*)
next
case *False* **from** *Cons* **show** $a \notin Y \Longrightarrow (a \# b) \restriction (X \cup Y) = (a \# b) \restriction X$
by (*simp add*: *projection-def*)
qed
qed

lemma *projection-on-empty-trace*: $[] \restriction X = []$ **by** (*simp add*: *projection-def*)

lemma *projection-to-emptyset-is-empty-trace*: $l \restriction \{\} = []$ **by** (*simp add*: *projection-def*)

lemma *projection-idempotent*: $l \restriction X = (l \restriction X) \restriction X$ **by** (*simp add*: *projection-def*)

lemma *projection-empty-implies-absence-of-events*: $l \restriction X = [] \Longrightarrow X \cap (set\ l) = \{\}$
by (*metis empty-set inter-set-filter projection-def*)

lemma *disjoint-projection*: $X \cap Y = \{\} \Longrightarrow (l \restriction X) \restriction Y = []$
proof −
assume *X-Y-disjoint*: $X \cap Y = \{\}$
show $(l \restriction X) \restriction Y = []$ **unfolding** *projection-def*
proof (*induct l*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons x xs*) **show** *?case*
proof (*cases* $x \in X$)
case *True*
with *X-Y-disjoint* **have** $x \notin Y$ **by** *auto*
thus $[x \leftarrow [x \leftarrow x \# xs\ .\ x \in X]\ .\ x \in Y] = []$ **using** *Cons.hyps* **by** *auto*
next
case *False* **show** $[x \leftarrow [x \leftarrow x \# xs\ .\ x \in X]\ .\ x \in Y] = []$ **using** *Cons.hyps False* **by** *auto*
qed
qed
qed

lemma *projection-concatenation-commute*:
(*l1* @ *l2*) ↾ *X* = (*l1* ↾ *X*) @ (*l2* ↾ *X*)
by (*unfold projection-def*, *auto*)

lemma *projection-subset-eq-from-superset-eq*:
((*xs* ↾ (*X* ∪ *Y*)) = (*ys* ↾ (*X* ∪ *Y*))) ⟹ ((*xs* ↾ *X*) = (*ys* ↾ *X*))
(**is** (*?L1* = *?L2*) ⟹ (*?L3* = *?L4*))
proof −
assume *prem*: *?L1* = *?L2*
have *?L1* ↾ *X* = *?L3* ∧ *?L2* ↾ *X* = *?L4*
proof −
have ⋀ *a*. ((*a* ∈ *X* ∨ *a* ∈ *Y*) ∧ *a* ∈ *X*) = (*a* ∈ *X*)
by *auto*
thus *?thesis*
by (*simp add*: *projection-def*)
qed
with *prem* **show** *?thesis*
by *auto*
qed

lemma *list-subset-iff-projection-neutral*: (*set l* ⊆ *X*) = ((*l* ↾ *X*) = *l*)
(**is** *?A* = *?B*)
proof −
have *?A* ⟹ *?B*
proof −
assume *?A*
hence ⋀*x*. *x* ∈ (*set l*) ⟹ *x* ∈ *X*
by *auto*
thus *?thesis*
by (*simp add*: *projection-def*)
qed
moreover
have *?B* ⟹ *?A*
proof −
assume *?B*
hence (*set* (*l* ↾ *X*)) = *set l*
by (*simp add*: *projection-def*)
thus *?thesis*
by (*simp add*: *projection-def*, *auto*)
qed
ultimately show *?thesis* **..**
qed

lemma *projection-split-last*: *Suc n* = *length* (*τ* ↾ *X*) ⟹
∃ *β x α*. (*x* ∈ *X* ∧ *τ* = *β* @ [*x*] @ *α* ∧ *α* ↾ *X* = [] ∧ *n* = *length* ((*β* @ *α*) ↾ *X*))
proof −
assume *Suc-n-is-len-τX*: *Suc n* = *length* (*τ* ↾ *X*)

let *?L* = $\tau \restriction X$
let *?RL* = *filter* $(\lambda x\ .\ x \in X)$ (*rev* τ)

have *Suc n* = *length ?RL*
proof −
  have *rev ?L* = *?RL*
    by (*simp add*: *projection-def*, *rule rev-filter*)
  hence *rev* (*rev ?L*) = *rev ?RL* **..**
  hence *?L* = *rev ?RL*
    by *auto*
  with *Suc-n-is-len-*τ*X* **show** *?thesis*
    by *auto*
qed
with *Suc-length-conv*[*of n ?RL*] **obtain** *x xs*
  where *?RL* = *x* # *xs*
  by *auto*
hence *x* # *xs* = *?RL*
  by *auto*

from *Cons-eq-filterD*[*OF this*] **obtain** *rev*α *rev*β
  where (*rev* τ) = *rev*α @ *x* # *rev*β
  and *rev*α*-no-x*: $\forall\, a \in$ *set rev*α. $a \notin X$
  and *x-in-X*: $x \in X$
  by *auto*
hence *rev* (*rev* τ) = *rev* (*rev*α @ *x* # *rev*β)
  by *auto*
hence τ = (*rev rev*β) @ [*x*] @ (*rev rev*α)
  by *auto*
then obtain β α
  where τ*-is-*$\beta x\alpha$: $\tau = \beta$ @ [*x*] @ α
  and α*-is-rev*revα: α = (*rev rev*α)
  and β*-is-rev*revβ: β = (*rev rev*β)
  by *auto*
hence α*-no-x*: $\alpha \restriction X = []$
proof −
  from α*-is-rev*revα *rev*α*-no-x* **have** $\forall\, a \in$ *set* α. $a \notin X$
    by *auto*
  thus *?thesis*
    by (*simp add*: *projection-def*)
qed

have *n* = *length* $((\beta$ @ $\alpha) \restriction X)$
proof −
  from α*-no-x* **have** α*X-zero-len*: *length* $(\alpha \restriction X) = 0$
    by *auto*

  from *x-in-X* **have** *xX-one-len*: *length* $([x] \restriction X) = 1$
    by (*simp add*: *projection-def*)

  from τ*-is-*$\beta x\alpha$ **have** *length ?L* = *length* $(\beta \restriction X)$ + *length* $([x] \restriction X)$ + *length* $(\alpha \restriction X)$
    by (*simp add*: *projection-def*)

```
    with αX-zero-len have length ?L = length (β ↾ X) + length ([x] ↾ X)
      by auto
    with xX-one-len Suc-n-is-len-τX have n = length (β ↾ X)
      by auto
    with αX-zero-len show ?thesis
      by (simp add: projection-def)
  qed
  with x-in-X τ-is-βxα α-no-x show ?thesis
    by auto
qed

lemma projection-rev-commute:
  rev (l ↾ X) = (rev l) ↾ X
  by (induct l, simp add: projection-def, simp add: projection-def)


lemma projection-split-first: ⟦ (τ ↾ X) = x # xs ⟧ ⟹ ∃ α β. (τ = α @ [x] @ β ∧ α ↾ X = [])
proof −
  assume τX-is-x-xs: (τ ↾ X) = x # xs
  hence 0 ≠ length (τ ↾ X)
    by auto
  hence 0 ≠ length (rev (τ ↾ X))
    by auto
  hence 0 ≠ length ((rev τ) ↾ X)
    by (simp add: projection-rev-commute)
  then obtain n where Suc n = length ((rev τ) ↾ X)
    by (auto, metis Suc-pred length-greater-0-conv that)
  from projection-split-last[OF this] obtain β' x' α'
    where x'-in-X: x' ∈ X
    and revτ-is-β'x'α': rev τ = β' @ [x'] @ α'
    and α'X-empty: α' ↾ X = []
    by auto

  from revτ-is-β'x'α' have rev (rev τ) = rev (β' @ [x'] @ α') ..
  hence τ-is-revα'-x'-revβ':τ = rev α' @ [x'] @ rev β'
    by auto
  moreover
  from α'X-empty have revα'X-empty: rev α' ↾ X = []
    by (metis projection-rev-commute rev-is-Nil-conv)
  moreover
  note x'-in-X
  ultimately have (τ ↾ X) = x' # ((rev β') ↾ X)
    by (simp only: projection-concatenation-commute projection-def, auto)
  with τX-is-x-xs have x = x'
    by auto
  with τ-is-revα'-x'-revβ' have τ-is-revα'-x-revβ': τ = rev α' @ [x] @ rev β'
    by auto
  with revα'X-empty show ?thesis
    by auto
qed
```

lemma *projection-split-first-with-suffix*:
⟦ (τ ↾ X) = x # xs ⟧ ⟹ ∃ α β. (τ = α @ [x] @ β ∧ α ↾ X = [] ∧ β ↾ X = xs)
proof −
assume *tau-proj-X*: (τ ↾ X) = x # xs
show *?thesis*
proof −
from *tau-proj-X* **have** *x-in-X*: x ∈ X
by (*metis IntE inter-set-filter list.set-intros*(1) *projection-def*)
from *tau-proj-X* **have** ∃ α β. τ = α @ [x] @ β ∧ α ↾ X = []
using *projection-split-first* **by** *auto*
then obtain α β **where** *tau-split*: τ = α @ [x] @ β
and *X-empty-prefix*:α ↾ X = []
by *auto*
from *tau-split tau-proj-X* **have** (α @ [x] @ β) ↾ X =x # xs
by *auto*
with *X-empty-prefix* **have** ([x] @ β) ↾ X =x # xs
by (*simp add*: *projection-concatenation-commute*)
hence (x # β) ↾ X =x # xs
by *auto*
with *x-in-X* **have** β ↾ X = xs
unfolding *projection-def* **by** *simp*
with *tau-split X-empty-prefix* **show** *?thesis*
by *auto*
qed
qed

lemma *projection-split-arbitrary-element*:
⟦τ ↾ X = (α @ [x] @ β) ↾ X; x ∈ X ⟧
⟹ ∃ α′ β′. (τ = α′ @ [x] @ β′ ∧ α′ ↾ X = α ↾ X ∧ β′ ↾ X = β ↾ X)
proof −
assume τ ↾ X = (α @ [x] @ β) ↾ X
and x ∈ X
{
fix n
have ⟦τ ↾ X = (α @ [x] @ β) ↾ X; x ∈ X; n = *length*(α↾X) ⟧
⟹ ∃ α′ β′. (τ = α′ @ [x] @ β′ ∧ α′ ↾ X = α ↾ X ∧ β′ ↾ X = β ↾ X)
proof (*induct n arbitrary*: τ α)
case *0*
hence α↾X = []
unfolding *projection-def* **by** *simp*
with *0.prems*(1) *0.prems*(2) **have** τ↾X = x # β↾X
unfolding *projection-def* **by** *simp*
with ‹α↾X = []› **show** *?case*
using *projection-split-first-with-suffix* **by** *fastforce*
next
case (*Suc n*)
from *Suc.prems*(1) **have** τ↾X=α↾X @ ([x] @ β) ↾X
using *projection-concatenation-commute* **by** *auto*
from *Suc.prems*(3) **obtain** x′ xs′ **where** α ↾X= x′ #xs′

```
                                    and x' ∈ X
      by (metis filter-eq-ConsD length-Suc-conv projection-def)
    then obtain a1 a2 where α = a1 @ [x'] @ a2
                        and a1↑X = []
                        and a2↑X = xs'
      using projection-split-first-with-suffix by metis
    with ‹x' ∈ X› Suc.prems(1) have τ↑X= x' # (a2 @ [x] @ β) ↑X
      unfolding projection-def by simp
    then obtain t1 t2 where τ= t1 @ [x'] @ t2
                        and t1↑X = []
                        and t2↑X = (a2 @ [x] @ β) ↑X
      using projection-split-first-with-suffix by metis
    from Suc.prems(3) ‹α ↑X= x' # xs'› ‹α = a1 @ [x'] @ a2› ‹a1↑X = []› ‹a2↑X = xs'›
    have n=length(a2↑X)
      by auto
    with Suc.hyps(1) Suc.prems(2) ‹t2↑X = (a2 @ [x] @ β) ↑X›
      obtain t2' t3' where t2=t2' @ [x] @ t3'
                        and t2'↑X = a2↑X
                        and t3'↑X = β↑X
        using projection-concatenation-commute by blast

    let ?α'=t1 @ [x'] @ t2' and ?β'=t3'
    from ‹τ= t1 @ [x'] @ t2› ‹t2=t2' @ [x] @ t3'› have τ=?α'@[x]@?β'
      by auto
    moreover
    from ‹α ↑X= x' # xs'› ‹t1↑X = []› ‹x' ∈ X› ‹t2'↑X = a2↑X› ‹a2↑X = xs'›
    have ?α'↑X = α↑X
      using projection-concatenation-commute unfolding projection-def by simp
    ultimately
    show ?case using ‹t3'↑X = β↑X›
      by blast
  qed
 }
 with ‹τ ↑ X = (α @ [x] @ β) ↑ X› ‹ x ∈ X› show ?thesis
  by simp
qed


lemma projection-on-intersection: l ↑ X = [] ⟹ l ↑ (X ∩ Y) = []
(is ?L1 = [] ⟹ ?L2 = [])
proof −
 assume ?L1 = []
 hence set ?L1 = {}
  by simp
 moreover
 have set ?L2 ⊆ set ?L1
  by (simp add: projection-def, auto)
 ultimately have set ?L2 = {}
  by auto
 thus ?thesis
  by auto
qed
```

lemma *projection-on-subset*: $[\![\ Y \subseteq X;\ l \restriction X = []\ ]\!] \Longrightarrow l \restriction Y = []$
proof −
  assume *subset*: $Y \subseteq X$
  assume *proj-empty*: $l \restriction X = []$
  hence $l \restriction (X \cap Y) = []$
    by (*rule projection-on-intersection*)
  moreover
  from *subset* **have** $X \cap Y = Y$
    by *auto*
  ultimately show *?thesis*
    by *auto*
qed

lemma *projection-on-subset2*: $[\![\ set\ l \subseteq L;\ l \restriction X' = []; X \cap L \subseteq X'\ ]\!] \Longrightarrow l \restriction X = []$
proof −
  assume *setl-subset-L*: $set\ l \subseteq L$
  assume *l-no-X'*: $l \restriction X' = []$
  assume *X-inter-L-subset-X'*: $X \cap L \subseteq X'$

  from *X-inter-L-subset-X' l-no-X'* **have** $l \restriction (X \cap L) = []$
    by (*rule projection-on-subset*)
  moreover
  have $l \restriction (X \cap L) = (l \restriction L) \restriction X$
    by (*simp add*: *Int-commute projection-def*)
  moreover
  note *setl-subset-L*
  ultimately show *?thesis*
    by (*simp add*: *list-subset-iff-projection-neutral*)
qed

lemma *non-empty-projection-on-subset*: $X \subseteq Y \wedge l_1 \restriction Y = l_2 \restriction Y \Longrightarrow\ l_1 \restriction X = l_2 \restriction X$
  by (*metis projection-subset-eq-from-superset-eq subset-Un-eq*)

lemma *projection-intersection-neutral*: $(set\ l \subseteq X) \Longrightarrow (l \restriction (X \cap Y) = l \restriction Y)$
proof −
  assume $set\ l \subseteq X$
  hence $(l \restriction X) = l$
    by (*simp add*: *list-subset-iff-projection-neutral*)
  hence $(l \restriction X) \restriction Y = l \restriction Y$
    by *simp*
  moreover
  have $(l \restriction X) \restriction Y = l \restriction (X \cap Y)$
    by (*simp add*: *projection-def*)
  ultimately show *?thesis*
    by *simp*
qed

lemma *projection-commute*:
(*l* ↾ *X*) ↾ *Y* = (*l* ↾ *Y*) ↾ *X*
by (*simp add*: *projection-def conj-commute*)

lemma *projection-subset-elim*: *Y* ⊆ *X* ⟹ (*l* ↾ *X*) ↾ *Y* = *l* ↾ *Y*
by (*simp only*: *projection-def*, *metis Diff-subset list-subset-iff-projection-neutral minus-coset-filter order-trans projection-commute projection-def*)

lemma *projection-sequence*: (*xs* ↾ *X*) ↾ *Y* = (*xs* ↾ (*X* ∩ *Y*))
by (*metis Int-absorb inf-sup-ord*(*1*) *list-subset-iff-projection-neutral projection-intersection-neutral projection-subset-elim*)

fun *merge* :: *'e set* ⇒ *'e set* ⇒ *'e list* ⇒ *'e list* ⇒ *'e list*
where
merge A B [] *t2* = *t2* |
merge A B t1 [] = *t1* |
merge A B (*e1* # *t1'*) (*e2* # *t2'*) = (*if e1* = *e2 then*
e1 # (*merge A B t1' t2'*)
else (*if e1* ∈ (*A* ∩ *B*) *then*
e2 # (*merge A B* (*e1* # *t1'*) *t2'*)
else e1 # (*merge A B t1'* (*e2* # *t2'*))))

lemma *merge-property*: ⟦*set t1* ⊆ *A*; *set t2* ⊆ *B*; *t1* ↾ *B* = *t2* ↾ *A* ⟧
⟹ *let t* = (*merge A B t1 t2*) *in* (*t* ↾ *A* = *t1* ∧ *t* ↾ *B* = *t2* ∧ *set t* ⊆ ((*set t1*) ∪ (*set t2*)))
unfolding *Let-def*
proof (*induct A B t1 t2 rule*: *merge.induct*)
case (*1 A B t2*) **thus** *?case*
by (*metis Un-empty-left empty-subsetI list-subset-iff-projection-neutral merge.simps*(*1*) *set-empty subset-iff-psubset-eq*)
next
case (*2 A B t1*) **thus** *?case*
by (*metis Un-empty-right empty-subsetI list-subset-iff-projection-neutral merge.simps*(*2*) *set-empty subset-refl*)
next
case (*3 A B e1 t1' e2 t2'*) **thus** *?case*
proof (*cases*)
assume *e1-is-e2*: *e1* = *e2*

note *e1-is-e2*
moreover
from *3*(*4*) **have** *set t1'* ⊆ *A*
by *auto*
moreover
from *3*(*5*) **have** *set t2'* ⊆ *B*
by *auto*
moreover
from *e1-is-e2 3*(*4*−*6*) **have** *t1'* ↾ *B* = *t2'* ↾ *A*

```
    by (simp add: projection-def)
  moreover
  note 3(1)
  ultimately have ind1: merge A B t1' t2' ⌈ A = t1'
    and ind2: merge A B t1' t2' ⌈ B = t2'
    and ind3: set (merge A B t1' t2') ⊆ (set t1') ∪ (set t2')
    by auto

  from e1-is-e2 have merge-eq:
    merge A B (e1 # t1') (e2 # t2') = e1 # (merge A B t1' t2')
    by auto

  from 3(4) ind1 have goal1:
    merge A B (e1 # t1') (e2 # t2') ⌈ A = e1 # t1'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from e1-is-e2 3(5) ind2 have goal2:
    merge A B (e1 # t1') (e2 # t2') ⌈ B = e2 # t2'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from ind3 have goal3:
    set (merge A B (e1 # t1') (e2 # t2')) ⊆ set (e1 # t1') ∪ set (e2 # t2')
    by (simp only: merge-eq, auto)
  ultimately show ?thesis
    by auto
next
  assume e1-isnot-e2: e1 ≠ e2
  show ?thesis
  proof (cases)
    assume e1-in-A-inter-B: e1 ∈ A ∩ B

    from 3(6) e1-isnot-e2 e1-in-A-inter-B have e2-notin-A: e2 ∉ A
      by (simp add: projection-def, auto)

    note e1-isnot-e2 e1-in-A-inter-B 3(4)
    moreover
    from 3(5) have set t2' ⊆ B
      by auto
    moreover
    from 3(6) e1-isnot-e2 e1-in-A-inter-B have (e1 # t1') ⌈ B = t2' ⌈ A
      by (simp add: projection-def, auto)
    moreover
    note 3(2)
    ultimately have ind1: merge A B (e1 # t1') t2' ⌈ A = (e1 # t1')
      and ind2: merge A B (e1 # t1') t2' ⌈ B = t2'
      and ind3: set (merge A B (e1 # t1') t2') ⊆ set (e1 # t1') ∪ set t2'
      by auto

    from e1-isnot-e2 e1-in-A-inter-B
    have merge-eq:
      merge A B (e1 # t1') (e2 # t2') = e2 # (merge A B (e1 # t1') t2')
      by auto
```

```
  from e1-isnot-e2 ind1 e2-notin-A have goal1:
    merge A B (e1 # t1') (e2 # t2') ↾ A = e1 # t1'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from 3(5) ind2 have goal2: merge A B (e1 # t1') (e2 # t2') ↾ B = e2 # t2'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from 3(5) ind3 have goal3:
    set (merge A B (e1 # t1') (e2 # t2')) ⊆ set (e1 # t1') ∪ set (e2 # t2')
    by (simp only: merge-eq, auto)
  ultimately show ?thesis
    by auto
next
  assume e1-notin-A-inter-B: e1 ∉ A ∩ B

  from 3(4) e1-notin-A-inter-B have e1-notin-B: e1 ∉ B
    by auto

  note e1-isnot-e2 e1-notin-A-inter-B
  moreover
  from 3(4) have set t1' ⊆ A
    by auto
  moreover
  note 3(5)
  moreover
  from 3(6) e1-notin-B have t1' ↾ B = (e2 # t2') ↾ A
    by (simp add: projection-def)
  moreover
  note 3(3)
  ultimately have ind1: merge A B t1' (e2 # t2') ↾ A = t1'
    and ind2: merge A B t1' (e2 # t2') ↾ B = (e2 # t2')
    and ind3: set (merge A B t1' (e2 # t2')) ⊆ set t1' ∪ set (e2 # t2')
    by auto

  from e1-isnot-e2 e1-notin-A-inter-B
  have merge-eq: merge A B (e1 # t1') (e2 # t2') = e1 # (merge A B t1' (e2 # t2'))
    by auto

  from 3(4) ind1 have goal1: merge A B (e1 # t1') (e2 # t2') ↾ A = e1 # t1'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from ind2 e1-notin-B have goal2:
    merge A B (e1 # t1') (e2 # t2') ↾ B = e2 # t2'
    by (simp only: merge-eq projection-def, auto)
  moreover
  from 3(4) ind3 have goal3:
    set (merge A B (e1 # t1') (e2 # t2')) ⊆ set (e1 # t1') ∪ set (e2 # t2')
    by (simp only: merge-eq, auto)
  ultimately show ?thesis
    by auto
qed
```

qed
qed

end

3 System Specification

3.1 Event Systems

We define the system model of event systems as well as the parallel composition operator for event systems provided as part of MAKS in [3].

theory *EventSystems*
imports *../Basics/Prefix ../Basics/Projection*
begin

record *'e ES-rec* =
E-ES :: *'e set*
I-ES :: *'e set*
O-ES :: *'e set*
Tr-ES :: (*'e list*) *set*

abbreviation *ESrecEES* :: *'e ES-rec* ⇒ *'e set*
(‹*E-*› [*1000*] *1000*)
where
E_{ES} ≡ (*E-ES ES*)

abbreviation *ESrecIES* :: *'e ES-rec* ⇒ *'e set*
(‹*I-*› [*1000*] *1000*)
where
I_{ES} ≡ (*I-ES ES*)

abbreviation *ESrecOES* :: *'e ES-rec* ⇒ *'e set*
(‹*O-*› [*1000*] *1000*)
where
O_{ES} ≡ (*O-ES ES*)

abbreviation *ESrecTrES* :: *'e ES-rec* ⇒ (*'e list*) *set*
(‹*Tr-*› [*1000*] *1000*)
where
Tr_{ES} ≡ (*Tr-ES ES*)

definition *es-inputs-are-events* :: *'e ES-rec* ⇒ *bool*
where
es-inputs-are-events ES ≡ $I_{ES} \subseteq E_{ES}$

definition *es-outputs-are-events* :: *'e ES-rec* ⇒ *bool*
where

es-outputs-are-events ES $\equiv O_{ES} \subseteq E_{ES}$

definition *es-inputs-outputs-disjoint* :: *'e ES-rec* ⇒ *bool*
where
es-inputs-outputs-disjoint ES $\equiv I_{ES} \cap O_{ES} = \{\}$

definition *traces-contain-events* :: *'e ES-rec* ⇒ *bool*
where
traces-contain-events ES $\equiv \forall\, l \in Tr_{ES}.\ \forall\, e \in (set\ l).\ e \in E_{ES}$

definition *traces-prefixclosed* :: *'e ES-rec* ⇒ *bool*
where
traces-prefixclosed ES $\equiv$ *prefixclosed* Tr_{ES}

definition *ES-valid* :: *'e ES-rec* ⇒ *bool*
where
ES-valid ES ≡
  es-inputs-are-events ES ∧ *es-outputs-are-events ES*
  ∧ *es-inputs-outputs-disjoint ES* ∧ *traces-contain-events ES*
  ∧ *traces-prefixclosed ES*

definition *total* :: *'e ES-rec* ⇒ *'e set* ⇒ *bool*
where
total ES E $\equiv E \subseteq E_{ES} \wedge (\forall \tau \in Tr_{ES}.\ \forall\, e \in E.\ \tau\ @\ [e] \in Tr_{ES})$

lemma *totality*: ⟦ *total ES E*; $t \in Tr_{ES}$; *set* $t' \subseteq E$ ⟧ ⟹ $t\ @\ t' \in Tr_{ES}$
  by (*induct* t' *rule*: *rev-induct*, *force*, *simp only*: *total-def*, *auto*)

definition *composeES* :: *'e ES-rec* ⇒ *'e ES-rec* ⇒ *'e ES-rec*
where
composeES ES1 ES2 ≡
  (|
    E-ES $= E_{ES1} \cup E_{ES2}$,
    I-ES $= (I_{ES1} - O_{ES2}) \cup (I_{ES2} - O_{ES1})$,
    O-ES $= (O_{ES1} - I_{ES2}) \cup (O_{ES2} - I_{ES1})$,
    Tr-ES $= \{\tau\ .\ (\tau \restriction E_{ES1}) \in Tr_{ES1} \wedge (\tau \restriction E_{ES2}) \in Tr_{ES2}$
                $\wedge\ (set\ \tau \subseteq E_{ES1} \cup E_{ES2})\}$
  |)

abbreviation *composeESAbbrv* :: *'e ES-rec* ⇒ *'e ES-rec* ⇒ *'e ES-rec*
(‹- ‖ -›[*1000*] *1000*)
where
ES1 ‖ *ES2* ≡ (*composeES ES1 ES2*)

definition *composable* :: *'e ES-rec* ⇒ *'e ES-rec* ⇒ *bool*
where
composable ES1 ES2 $\equiv (E_{ES1} \cap E_{ES2}) \subseteq ((O_{ES1} \cap I_{ES2}) \cup (O_{ES2} \cap I_{ES1}))$

lemma *composeES-yields-ES*:
 ⟦ *ES-valid ES1*; *ES-valid ES2* ⟧ ⟹ *ES-valid* (*ES1* ∥ *ES2*)
 unfolding *ES-valid-def*
proof (*auto*)
 assume *ES1-inputs-are-events*: *es-inputs-are-events ES1*
 assume *ES2-inputs-are-events*: *es-inputs-are-events ES2*
 show *es-inputs-are-events* (*ES1* ∥ *ES2*) **unfolding** *composeES-def es-inputs-are-events-def*
  proof (*simp*)
   have *subgoal11*: $I_{ES1} - O_{ES2} \subseteq E_{ES1} \cup E_{ES2}$
   proof (*auto*)
    fix *x*
    assume $x \in I_{ES1}$
    with *ES1-inputs-are-events* **show** $x \in E_{ES1}$
     by (*auto simp add*: *es-inputs-are-events-def*)
   qed
   have *subgoal12*: $I_{ES2} - O_{ES1} \subseteq E_{ES1} \cup E_{ES2}$
   proof (*rule subsetI*, *rule UnI2*, *auto*)
    fix *x*
    assume $x \in I_{ES2}$
    with *ES2-inputs-are-events* **show** $x \in E_{ES2}$
     by (*auto simp add*: *es-inputs-are-events-def*)
   qed
   from *subgoal11 subgoal12*
   show $I_{ES1} - O_{ES2} \subseteq E_{ES1} \cup E_{ES2} \wedge I_{ES2} - O_{ES1} \subseteq E_{ES1} \cup E_{ES2}$ **..**
 qed
next
 assume *ES1-outputs-are-events*: *es-outputs-are-events ES1*
 assume *ES2-outputs-are-events*: *es-outputs-are-events ES2*
 show *es-outputs-are-events* (*ES1* ∥ *ES2*)
  unfolding *composeES-def es-outputs-are-events-def*
  proof (*simp*)
   have *subgoal21*: $O_{ES1} - I_{ES2} \subseteq E_{ES1} \cup E_{ES2}$
   proof (*auto*)
    fix *x*
    assume $x \in O_{ES1}$
    with *ES1-outputs-are-events* **show** $x \in E_{ES1}$
     by (*auto simp add*: *es-outputs-are-events-def*)
   qed
   have *subgoal22*: $O_{ES2} - I_{ES1} \subseteq E_{ES1} \cup E_{ES2}$
   proof (*rule subsetI*, *rule UnI2*, *auto*)
    fix *x*
    assume $x \in O_{ES2}$
    with *ES2-outputs-are-events* **show** $x \in E_{ES2}$
     by (*auto simp add*: *es-outputs-are-events-def*)
   qed
   from *subgoal21 subgoal22*
   show $O_{ES1} - I_{ES2} \subseteq E_{ES1} \cup E_{ES2} \wedge O_{ES2} - I_{ES1} \subseteq E_{ES1} \cup E_{ES2}$ **..**
 qed
next

assume *ES1-inputs-outputs-disjoint*: *es-inputs-outputs-disjoint ES1*
assume *ES2-inputs-outputs-disjoint*: *es-inputs-outputs-disjoint ES2*
show *es-inputs-outputs-disjoint* (*ES1* ∥ *ES2*)
unfolding *composeES-def es-inputs-outputs-disjoint-def*
proof (*simp*)
have *subgoal31*:
$\{\} \subseteq (I_{ES1} - O_{ES2} \cup (I_{ES2} - O_{ES1})) \cap (O_{ES1} - I_{ES2} \cup (O_{ES2} - I_{ES1}))$
by *auto*
have *subgoal32*:
$(I_{ES1} - O_{ES2} \cup (I_{ES2} - O_{ES1})) \cap (O_{ES1} - I_{ES2} \cup (O_{ES2} - I_{ES1})) \subseteq \{\}$
proof (*rule subsetI*, *erule IntE*)
fix x
assume *ass1*: $x \in I_{ES1} - O_{ES2} \cup (I_{ES2} - O_{ES1})$
then have *ass1'*: $x \in I_{ES1} - O_{ES2} \vee x \in (I_{ES2} - O_{ES1})$
by *auto*
assume *ass2*: $x \in O_{ES1} - I_{ES2} \cup (O_{ES2} - I_{ES1})$
then have *ass2'*:$x \in O_{ES1} - I_{ES2} \vee x \in (O_{ES2} - I_{ES1})$
by *auto*
note *ass1'*
moreover {
assume *left1*: $x \in I_{ES1} - O_{ES2}$
note *ass2'*
moreover {
assume *left2*: $x \in O_{ES1} - I_{ES2}$
with *left1* **have** $x \in (I_{ES1}) \cap (O_{ES1})$
by (*auto*)
with *ES1-inputs-outputs-disjoint* **have** $x \in \{\}$
by (*auto simp add*: *es-inputs-outputs-disjoint-def*)
}
moreover {
assume *right2*: $x \in (O_{ES2} - I_{ES1})$
with *left1* **have** $x \in (I_{ES1} - I_{ES1})$
by *auto*
hence $x \in \{\}$
by *auto*
}
ultimately have $x \in \{\}$ **..**
}
moreover {
assume *right1*: $x \in I_{ES2} - O_{ES1}$
note *ass2'*
moreover {
assume *left2*: $x \in O_{ES1} - I_{ES2}$
with *right1* **have** $x \in (I_{ES2} - I_{ES2})$
by *auto*
hence $x \in \{\}$
by *auto*
}
moreover {
assume *right2*: $x \in (O_{ES2} - I_{ES1})$
with *right1* **have** $x \in (I_{ES2} \cap O_{ES2})$
by *auto*

```
        with ES2-inputs-outputs-disjoint have x∈{}
          by (auto simp add: es-inputs-outputs-disjoint-def)
      }
      ultimately have x∈{} ..
    }
    ultimately show x∈{} ..
  qed

  from subgoal31 subgoal32
  show (I_ES1 − O_ES2 ∪ (I_ES2 − O_ES1)) ∩ (O_ES1 − I_ES2 ∪ (O_ES2 − I_ES1)) = {}
    by auto
 qed
next
 show traces-contain-events (ES1 ∥ ES2) unfolding composeES-def traces-contain-events-def
  proof (clarsimp)
    fix l e
    assume e ∈ set l
      and set l ⊆ E_ES1 ∪ E_ES2
    then have e-in-union: e ∈ E_ES1 ∪ E_ES2
      by auto
    assume e ∉ E_ES2
    with e-in-union show e ∈ E_ES1
      by auto
  qed
next
 assume ES1-traces-prefixclosed: traces-prefixclosed ES1
 assume ES2-traces-prefixclosed: traces-prefixclosed ES2
 show traces-prefixclosed (ES1 ∥ ES2)
  unfolding composeES-def traces-prefixclosed-def prefixclosed-def prefix-def
 proof (clarsimp)
  fix l2 l3
  have l2l3split: (l2 @ l3) ↾ E_ES1 = (l2 ↾ E_ES1) @ (l3 ↾ E_ES1)
    by (rule projection-concatenation-commute)
  assume (l2 @ l3) ↾ E_ES1 ∈ Tr_ES1
  with l2l3split have l2l3cattrace: (l2 ↾ E_ES1) @ (l3 ↾ E_ES1) ∈ Tr_ES1
    by auto
  have theprefix: (l2 ↾ E_ES1) ⪯ ((l2 ↾ E_ES1) @ (l3 ↾ E_ES1))
    by (simp add: prefix-def)
  have prefixclosure: ∀ es1 ∈ (Tr_ES1). ∀ es2. es2 ⪯ es1 ⟶ es2 ∈ (Tr_ES1)
    by (clarsimp, insert ES1-traces-prefixclosed, unfold traces-prefixclosed-def prefixclosed-def,
      erule-tac x=es1 in ballE, erule-tac x=es2 in allE, erule impE, auto)
  hence
    ((l2 ↾ E_ES1) @ (l3 ↾ E_ES1)) ∈ Tr_ES1 ⟹ ∀ es2. es2 ⪯ ((l2 ↾ E_ES1) @ (l3 ↾ E_ES1))
      ⟶ es2 ∈ Tr_ES1 ..
  with l2l3cattrace have ∀ es2. es2 ⪯ ((l2 ↾ E_ES1) @ (l3 ↾ E_ES1)) ⟶ es2 ∈ Tr_ES1
    by auto
  hence (l2 ↾ E_ES1) ⪯ ((l2 ↾ E_ES1) @ (l3 ↾ E_ES1)) ⟶ (l2 ↾ E_ES1) ∈ Tr_ES1 ..
  with theprefix have goal51: (l2 ↾ E_ES1) ∈ Tr_ES1
    by simp
  have l2l3split: (l2 @ l3) ↾ E_ES2 = (l2 ↾ E_ES2) @ (l3 ↾ E_ES2)
    by (rule projection-concatenation-commute)
  assume (l2 @ l3) ↾ E_ES2 ∈ Tr_ES2
```

with *l2l3split* **have** *l2l3cattrace*: $(l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2}) \in Tr_{ES2}$
by *auto*
have *theprefix*: $(l2 \restriction E_{ES2}) \preceq ((l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2}))$
by (*simp add*: *prefix-def*)
have *prefixclosure*: $\forall\ es1 \in Tr_{ES2}.\ \forall\, es2.\ es2 \preceq es1 \longrightarrow es2 \in Tr_{ES2}$
by (*clarsimp*, *insert ES2-traces-prefixclosed*,
unfold traces-prefixclosed-def prefixclosed-def,
erule-tac x=es1 **in** *ballE*, *erule-tac x=es2* **in** *allE*, *erule impE*, *auto*)
hence $((l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2})) \in Tr_{ES2}$
$\Longrightarrow \forall\ es2.\ es2 \preceq ((l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2})) \longrightarrow es2 \in Tr_{ES2}$ **..**
with *l2l3cattrace* **have** $\forall\ es2.\ es2 \preceq ((l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2})) \longrightarrow es2 \in Tr_{ES2}$
by *auto*
hence $(l2 \restriction E_{ES2}) \preceq ((l2 \restriction E_{ES2}) \mathbin{@} (l3 \restriction E_{ES2})) \longrightarrow (l2 \restriction E_{ES2}) \in Tr_{ES2}$ **..**
with *theprefix* **have** *goal52*: $(l2 \restriction E_{ES2}) \in Tr_{ES2}$
by *simp*
from *goal51 goal52* **show** *goal5*: $l2 \restriction E_{ES1} \in Tr_{ES1} \wedge l2 \restriction E_{ES2} \in Tr_{ES2}$ **..**
qed
qed

end

3.2 State-Event Systems

We define the system model of state-event systems as well as the translation from state-event systems to event systems provided as part of MAKS in [3]. State-event systems are the basis for the unwinding theorems that we prove later in this entry.

theory *StateEventSystems*
imports *EventSystems*
begin

record $('s, 'e)$ *SES-rec* =
S-SES :: $'s$ *set*
s0-SES :: $'s$
E-SES :: $'e$ *set*
I-SES :: $'e$ *set*
O-SES :: $'e$ *set*
T-SES :: $'s \Rightarrow 'e \rightharpoonup 's$

abbreviation *SESrecSSES* :: $('s, 'e)$ *SES-rec* $\Rightarrow$ $'s$ *set*
$(\langle S\_ \rangle\ [1000]\ 1000)$
where
$S_{SES} \equiv$ (*S-SES SES*)

abbreviation *SESrecs0SES* :: $('s, 'e)$ *SES-rec* $\Rightarrow$ $'s$
$(\langle s0\_ \rangle\ [1000]\ 1000)$
where
$s0_{SES} \equiv$ (*s0-SES SES*)

abbreviation *SESrecESES* :: ('*s*, '*e*) *SES-rec* ⇒ '*e set*
(‹*E-*› [*1000*] *1000*)
where
$E_{SES} \equiv$ (*E-SES SES*)

abbreviation *SESrecISES* :: ('*s*, '*e*) *SES-rec* ⇒ '*e set*
(‹*I-*› [*1000*] *1000*)
where
$I_{SES} \equiv$ (*I-SES SES*)

abbreviation *SESrecOSES* :: ('*s*, '*e*) *SES-rec* ⇒ '*e set*
(‹*O-*› [*1000*] *1000*)
where
$O_{SES} \equiv$ (*O-SES SES*)

abbreviation *SESrecTSES* :: ('*s*, '*e*) *SES-rec* ⇒ ('*s* ⇒ '*e* ⇀ '*s*)
(‹*T-*› [*1000*] *1000*)
where
$T_{SES} \equiv$ (*T-SES SES*)

abbreviation *TSESpred* :: '*s* ⇒ '*e* ⇒ ('*s*, '*e*) *SES-rec* ⇒ '*s* ⇒ *bool*
(‹- -⟶- -› [*100*,*100*,*100*,*100*] *100*)
where
$s\ e \longrightarrow_{SES} s' \equiv (T_{SES}\ s\ e = Some\ s')$

definition *s0-is-state* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
$s0\text{-}is\text{-}state\ SES \equiv s0_{SES} \in S_{SES}$

definition *ses-inputs-are-events* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
$ses\text{-}inputs\text{-}are\text{-}events\ SES \equiv I_{SES} \subseteq E_{SES}$

definition *ses-outputs-are-events* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
$ses\text{-}outputs\text{-}are\text{-}events\ SES \equiv O_{SES} \subseteq E_{SES}$

definition *ses-inputs-outputs-disjoint* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
$ses\text{-}inputs\text{-}outputs\text{-}disjoint\ SES \equiv I_{SES} \cap O_{SES} = \{\}$

definition *correct-transition-relation* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
correct-transition-relation SES ≡
$\forall x\ y\ z.\ x\ y \longrightarrow_{SES} z \longrightarrow ((x \in S_{SES}) \wedge (y \in E_{SES}) \wedge (z \in S_{SES}))$

definition *SES-valid* :: ('*s*, '*e*) *SES-rec* ⇒ *bool*
where
SES-valid SES ≡
  $s0\text{-}is\text{-}state\ SES \wedge ses\text{-}inputs\text{-}are\text{-}events\ SES$

$\wedge$ *ses-outputs-are-events SES* $\wedge$ *ses-inputs-outputs-disjoint SES* $\wedge$
correct-transition-relation SES

primrec *path* :: ('*s*, '*e*) *SES-rec* $\Rightarrow$ '*s* $\Rightarrow$ '*e list* $\rightharpoonup$ '*s*
where
path-empt: *path SES s1* [] = (*Some s1*) |
path-nonempt: *path SES s1* (*e* # *t*) =
(*if* ($\exists$ *s2*. *s1* $e \longrightarrow_{SES}$ *s2*)
then (*path SES* (*the* (T_{SES} *s1 e*)) *t*)
else None)

abbreviation *pathpred* :: '*s* $\Rightarrow$ '*e list* $\Rightarrow$ ('*s*, '*e*) *SES-rec* $\Rightarrow$ '*s* $\Rightarrow$ *bool*
($\langle$- -$\Longrightarrow_-$ -$\rangle$ [100, 100, 100, 100] 100)
where
s t$\Longrightarrow_{SES}$ *s'* $\equiv$ *path SES s t* = *Some s'*

definition *reachable* :: ('*s*, '*e*) *SES-rec* $\Rightarrow$ '*s* $\Rightarrow$ *bool*
where
reachable SES s $\equiv$ ($\exists$ *t*. $s0_{SES}$ *t*$\Longrightarrow_{SES}$ *s*)

definition *enabled* :: ('*s*, '*e*) *SES-rec* $\Rightarrow$ '*s* $\Rightarrow$ '*e list* $\Rightarrow$ *bool*
where
enabled SES s t $\equiv$ ($\exists$ *s'*. *s t*$\Longrightarrow_{SES}$ *s'*)

definition *possible-traces* :: ('*s*, '*e*) *SES-rec* $\Rightarrow$ ('*e list*) *set*
where
possible-traces SES $\equiv$ {*t*. (*enabled SES* $s0_{SES}$ *t*)}

definition *induceES* :: ('*s*, '*e*) *SES-rec* $\Rightarrow$ '*e ES-rec*
where
induceES SES $\equiv$
(|
E-ES = E_{SES},
I-ES = I_{SES},
O-ES = O_{SES},
Tr-ES = *possible-traces SES*
|)

lemma *none-remains-none* : $\bigwedge$ *s e*. (*path SES s t*) = *None*
$\Longrightarrow$ (*path SES s* (*t* @ [*e*])) = *None*
by (*induct t*, *auto*)

lemma *path-trans-single-neg*: $\bigwedge$ *s1*. $[\![$*s1* $t \Longrightarrow_{SES}$ *s2*; $\neg$ (*s2* $e \longrightarrow_{SES}$ *sn*)$]\!]$
$\Longrightarrow \neg$ (*s1* (*t* @ [*e*])$\Longrightarrow_{SES}$ *sn*)
by (*induct t*, *auto*)

lemma *path-split-single*: *s1* (*t*@[*e*])$\Longrightarrow_{SES}$ *sn*
$\Longrightarrow \exists s'$. *s1* *t*$\Longrightarrow_{SES}$ *s'* $\wedge$ *s'* *e*$\longrightarrow_{SES}$ *sn*
by (*cases path SES s1 t*, *simp add*: *none-remains-none*,
simp, *rule ccontr*, *auto simp add*: *path-trans-single-neg*)

lemma *path-trans-single*: $\bigwedge$*s*. $[\![$ *s* *t*$\Longrightarrow_{SES}$ *s'*; *s'* *e*$\longrightarrow_{SES}$ *sn* $]\!]$
$\Longrightarrow$ *s* (*t* @ [*e*])$\Longrightarrow_{SES}$ *sn*
proof (*induct t*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*Cons a t*) **thus** *?case*
proof −
from *Cons* **obtain** *s1'* **where** *trans-s-a-s1'*: *s* *a*$\longrightarrow_{SES}$ *s1'*
by (*simp*, *split if-split-asm*, *auto*)
with *Cons* **have** *s1'* (*t* @ [*e*])$\Longrightarrow_{SES}$ *sn*
by *auto*
with *trans-s-a-s1'* **show** *?thesis*
by *auto*
qed
qed

lemma *path-split*: $\bigwedge$ *sn*. $[\![$ *s1* (*t1* @ *t2*)$\Longrightarrow_{SES}$ *sn* $]\!]$
$\Longrightarrow$ ($\exists$ *s2*. (*s1* *t1*$\Longrightarrow_{SES}$ *s2* $\wedge$ *s2* *t2*$\Longrightarrow_{SES}$ *sn*))
proof (*induct t2 rule*: *rev-induct*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*snoc a t*) **thus** *?case*
proof −
from *snoc* **have** *s1* (*t1* @ *t* @ [*a*])$\Longrightarrow_{SES}$ *sn*
by *auto*
hence $\exists$ *sn'*. *s1* (*t1* @ *t*)$\Longrightarrow_{SES}$ *sn'* $\wedge$ *sn'* *a*$\longrightarrow_{SES}$ *sn*
by (*simp add*: *path-split-single*)
then obtain *sn'* **where** *path-t1-t-trans-a*:
s1 (*t1* @ *t*)$\Longrightarrow_{SES}$ *sn'* $\wedge$ *sn'* *a*$\longrightarrow_{SES}$ *sn*
by *auto*
with *snoc* **obtain** *s2* **where** *path-t1-t*:
s1 *t1*$\Longrightarrow_{SES}$ *s2* $\wedge$ *s2* *t*$\Longrightarrow_{SES}$ *sn'*
by *auto*
with *path-t1-t-trans-a* **have** *s2* (*t* @ [*a*])$\Longrightarrow_{SES}$ *sn*
by (*simp add*: *path-trans-single*)
with *path-t1-t* **show** *?thesis* **by** *auto*
qed
qed

lemma *path-trans*:
$\bigwedge sn.\ [\![\ s1\ l1 \Longrightarrow_{SES} s2;\ s2\ l2 \Longrightarrow_{SES} sn\ ]\!] \Longrightarrow s1\ (l1\ @\ l2) \Longrightarrow_{SES} sn$
proof (*induct l2 rule*: *rev-induct*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*snoc a l*) **thus** *?case*
proof −
assume *path-l1*: $s1\ l1 \Longrightarrow_{SES} s2$
assume $s2\ (l@[a]) \Longrightarrow_{SES} sn$
hence $\exists sn'.\ s2\ l \Longrightarrow_{SES} sn' \wedge sn'\ [a] \Longrightarrow_{SES} sn$
by (*simp add*: *path-split del*: *path-nonempt*)
then obtain sn' **where** *path-l-a*: $s2\ l \Longrightarrow_{SES} sn' \wedge sn'\ [a] \Longrightarrow_{SES} sn$
by *auto*
with *snoc path-l1* **have** *path-l1-l*: $s1\ (l1@l) \Longrightarrow_{SES} sn'$
by *auto*
with *path-l-a* **have** $sn'\ a \longrightarrow_{SES} sn$
by (*simp*, *split if-split-asm*, *auto*)
with *path-l1-l* **show** $s1\ (l1\ @\ l\ @\ [a]) \Longrightarrow_{SES} sn$
by (*subst append-assoc*[*symmetric*], *rule-tac* $s'{=}sn'$ **in** *path-trans-single*, *auto*)
qed
qed

lemma *enabledPrefixSingle* : $[\![\ enabled\ SES\ s\ (t@[e])\ ]\!] \Longrightarrow enabled\ SES\ s\ t$
unfolding *enabled-def*
proof −
assume *ass*: $\exists s'.\ s\ (t\ @\ [e]) \Longrightarrow_{SES} s'$
from *ass* **obtain** s' **where** $s\ (t\ @\ [e]) \Longrightarrow_{SES} s'$ **..**
hence $\exists t'.\ (s\ t \Longrightarrow_{SES} t') \wedge (t'\ e \longrightarrow_{SES} s')$
by (*rule path-split-single*)
then obtain t' **where** $s\ t \Longrightarrow_{SES} t'$
by (*auto*)
thus $\exists s'.\ s\ t \Longrightarrow_{SES} s'$ **..**
qed

lemma *enabledPrefix* : $[\![\ enabled\ SES\ s\ (t1\ @\ t2)\ ]\!] \Longrightarrow enabled\ SES\ s\ t1$
unfolding *enabled-def*
proof −
assume *ass*: $\exists s'.\ s\ (t1\ @\ t2) \Longrightarrow_{SES} s'$
from *ass* **obtain** s' **where** $s\ (t1\ @\ t2) \Longrightarrow_{SES} s'$ **..**
hence $\exists t.\ (s\ t1 \Longrightarrow_{SES} t \wedge t\ t2 \Longrightarrow_{SES} s')$
by (*rule path-split*)
then obtain t **where** $s\ t1 \Longrightarrow_{SES} t$
by (*auto*)
then show $\exists s'.\ s\ t1 \Longrightarrow_{SES} s'$ **..**
qed

lemma *enabledPrefixSingleFinalStep* : $[\![$ *enabled SES s* (*t*@[*e*]) $]\!] \Longrightarrow \exists\ t'\ t''.\ t'\ e \longrightarrow_{SES} t''$
unfolding *enabled-def*
proof −
assume *ass*: $\exists\, s'.\ s\ (t\ @\ [e]) \Longrightarrow_{SES} s'$
from *ass* **obtain** s' **where** $s\ (t\ @\ [e]) \Longrightarrow_{SES} s'$ **..**
hence $\exists\, t'.\ (s\ t \Longrightarrow_{SES} t')\ \wedge\ (t'\ e \longrightarrow_{SES} s')$
by (*rule path-split-single*)
then obtain t' **where** $t'\ e \longrightarrow_{SES} s'$
by (*auto*)
thus $\exists\, t'\ t''.\ t'\ e \longrightarrow_{SES} t''$
by (*auto*)
qed

lemma *induceES-yields-ES*:
SES-valid SES $\Longrightarrow$ *ES-valid* (*induceES SES*)
proof (*simp add*: *SES-valid-def ES-valid-def*, *auto*)
assume *SES-inputs-are-events*: *ses-inputs-are-events SES*
thus *es-inputs-are-events* (*induceES SES*)
by (*simp add*: *induceES-def ses-inputs-are-events-def es-inputs-are-events-def*)
next
assume *SES-outputs-are-events*: *ses-outputs-are-events SES*
thus *es-outputs-are-events* (*induceES SES*)
by (*simp add*: *induceES-def ses-outputs-are-events-def es-outputs-are-events-def*)
next
assume *SES-inputs-outputs-disjoint*: *ses-inputs-outputs-disjoint SES*
thus *es-inputs-outputs-disjoint* (*induceES SES*)
by (*simp add*: *induceES-def ses-inputs-outputs-disjoint-def es-inputs-outputs-disjoint-def*)
next
assume *SES-correct-transition-relation*: *correct-transition-relation SES*
thus *traces-contain-events* (*induceES SES*)
unfolding *induceES-def traces-contain-events-def possible-traces-def*
proof (*auto*)
fix *l e*
assume *enabled-l*: *enabled SES s0* $_{SES}$ *l*
assume *e-in-l*: *e* ∈ *set l*
from *enabled-l e-in-l* **show** $e \in E_{SES}$
proof (*induct l rule*: *rev-induct*)
case *Nil*
assume *e-in-empty-list*: *e* ∈ *set* []
hence *f*: *False*
by (*auto*)
thus *?case*
by *auto*
next
case (*snoc a l*)
from *snoc.prems* **have** *l-enabled*: *enabled SES s0* $_{SES}$ *l*
by (*simp add*: *enabledPrefixSingle*)
show *?case*
proof (*cases e* ∈ (*set l*))

```
        from snoc.hyps l-enabled show e ∈ set l ⟹ e ∈ E_SES
          by auto
        show e ∉ set l ⟹ e ∈ E_SES
          proof −
            assume e ∉ set l
            with snoc.prems have e-eq-a : e=a
              by auto
            from snoc.prems have ∃ t t'. t a⟶_SES t'
              by (auto simp add: enabledPrefixSingleFinalStep)
            then obtain t t' where t a⟶_SES t'
              by auto
            with e-eq-a SES-correct-transition-relation show e ∈ E_SES
              by (simp add: correct-transition-relation-def)
          qed
      qed
    qed
  qed
next
  show traces-prefixclosed (induceES SES)
    unfolding traces-prefixclosed-def prefixclosed-def induceES-def possible-traces-def prefix-def
    by (clarsimp simp add: enabledPrefix)
qed

end
```

4 Security Specification

4.1 Views & Flow Policies

We define views, flow policies and how views can be derived from a given flow policy.

```
theory Views
imports Main
begin


record 'e V-rec =
  V :: 'e set
  N :: 'e set
  C :: 'e set


abbreviation VrecV :: 'e V-rec ⇒ 'e set
(‹V-› [100] 1000)
where
V_v ≡ (V v)

abbreviation VrecN :: 'e V-rec ⇒ 'e set
(‹N-› [100] 1000)
where
N_v ≡ (N v)
```

abbreviation *VrecC* :: *'e V-rec* ⇒ *'e set*
(‹*C*-› [*100*] *1000*)
where
$C_v \equiv (C\ v)$

definition *VN-disjoint* :: *'e V-rec* ⇒ *bool*
where
VN-disjoint $v \equiv V_v \cap N_v = \{\}$

definition *VC-disjoint* :: *'e V-rec* ⇒ *bool*
where
VC-disjoint $v \equiv V_v \cap C_v = \{\}$

definition *NC-disjoint* :: *'e V-rec* ⇒ *bool*
where
NC-disjoint $v \equiv N_v \cap C_v = \{\}$

definition *V-valid* :: *'e V-rec* ⇒ *bool*
where
V-valid $v \equiv$ *VN-disjoint* $v \wedge$ *VC-disjoint* $v \wedge$ *NC-disjoint* v

definition *isViewOn* :: *'e V-rec* ⇒ *'e set* ⇒ *bool*
where
isViewOn $\mathcal{V}\ E \equiv$ *V-valid* $\mathcal{V} \wedge V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}} = E$

end
theory *FlowPolicies*
imports *Views*
begin

record *'domain FlowPolicy-rec* =
  D :: *'domain set*
  v-rel :: (*'domain* × *'domain*) *set*
  n-rel :: (*'domain* × *'domain*) *set*
  c-rel :: (*'domain* × *'domain*) *set*

definition *FlowPolicy* :: *'domain FlowPolicy-rec* ⇒ *bool*
where
FlowPolicy fp ≡
  ((*v-rel fp*) ∪ (*n-rel fp*) ∪ (*c-rel fp*) = ((*D fp*) × (*D fp*)))
∧ (*v-rel fp*) ∩ (*n-rel fp*) = {}
∧ (*v-rel fp*) ∩ (*c-rel fp*) = {}
∧ (*n-rel fp*) ∩ (*c-rel fp*) = {}
∧ (∀ *d* ∈ (*D fp*). (*d*, *d*) ∈ (*v-rel fp*))

type-synonym (*'e*, *'domain*) *dom-type* = *'e* ⇀ *'domain*

definition *dom* :: (*'e*, *'domain*) *dom-type* ⇒ *'domain set* ⇒ *'e set* ⇒ *bool*
where
dom domas dset es ≡
(∀ *e*. ∀ *d*. ((*domas e* = *Some d*) ⟶ (*e* ∈ *es* ∧ *d* ∈ *dset*)))

definition *view-dom* :: *'domain FlowPolicy-rec* ⇒ *'domain* ⇒ (*'e*, *'domain*) *dom-type* ⇒ *'e V-rec*
where
view-dom fp d domas ≡
(| *V* = {*e*. ∃ *d'*. (*domas e* = *Some d'* ∧ (*d'*, *d*) ∈ (*v-rel fp*))},
N = {*e*. ∃ *d'*. (*domas e* = *Some d'* ∧ (*d'*, *d*) ∈ (*n-rel fp*))},
C = {*e*. ∃ *d'*. (*domas e* = *Some d'* ∧ (*d'*, *d*) ∈ (*c-rel fp*))} |)

end

4.2 Basic Security Predicates

We define all 14 basic security predicates provided as part of MAKS in [3].

theory *BasicSecurityPredicates*
imports *Views ../Basics/Projection*
begin

definition *areTracesOver* :: (*'e list*) *set* ⇒ *'e set* ⇒ *bool*
where
areTracesOver Tr E ≡
∀ *τ* ∈ *Tr*. (*set τ*) ⊆ *E*

type-synonym *'e BSP* = *'e V-rec* ⇒ ((*'e list*) *set*) ⇒ *bool*

definition *BSP-valid* :: *'e BSP* ⇒ *bool*
where
BSP-valid bsp ≡
∀ *V Tr E*. (*isViewOn V E* ∧ *areTracesOver Tr E*)
⟶ (∃ *Tr'*. *Tr'* ⊒ *Tr* ∧ *bsp V Tr'*)

definition *R* :: *'e BSP*
where
R V Tr ≡
∀ *τ*∈*Tr*. ∃ *τ'*∈*Tr*. *τ'* ↾ $C_{\mathcal{V}}$ = [] ∧ *τ'* ↾ $V_{\mathcal{V}}$ = *τ* ↾ $V_{\mathcal{V}}$

lemma *BSP-valid-R*: *BSP-valid R*
proof −
{

```
    fix 𝒱::('e V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr' ⊇ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have R 𝒱 ?Tr'
      proof −
        {
          fix τ
          assume τ ∈ {t. (set t) ⊆ E}
          let ?τ'=τ↾(V_𝒱)
          have ?τ' ↾ C_𝒱 = [] ∧ ?τ' ↾ V_𝒱 = τ ↾ V_𝒱
            using ‹isViewOn 𝒱 E› disjoint-projection projection-idempotent
            unfolding isViewOn-def V-valid-def VC-disjoint-def by metis
          moreover
          from ‹τ ∈ {t. (set t) ⊆ E}› have ?τ' ∈ ?Tr' using ‹isViewOn 𝒱 E›
            unfolding isViewOn-def
            by (simp add: list-subset-iff-projection-neutral projection-commute)
          ultimately
          have ∃τ'∈{t. set t ⊆ E}. τ' ↾ C_𝒱 = [] ∧ τ' ↾ V_𝒱 = τ ↾ V_𝒱
            by auto
        }
        thus ?thesis unfolding R-def
          by auto
      qed
    ultimately
    have ∃ Tr'. Tr' ⊇ Tr ∧ R 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition D :: 'e BSP
where
D 𝒱 Tr ≡
  ∀α β. ∀c∈C_𝒱. ((β @ [c] @ α) ∈ Tr ∧ α↾C_𝒱 = [])
    ⟶ (∃α' β'. ((β' @ α') ∈ Tr ∧ α'↾V_𝒱 = α↾V_𝒱 ∧ α'↾C_𝒱 = []
                ∧ β'↾(V_𝒱 ∪ C_𝒱) = β↾(V_𝒱 ∪ C_𝒱)))

lemma BSP-valid-D: BSP-valid D
proof −
  {
    fix 𝒱::('e V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
```

```
    have ?Tr'⊒ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have D 𝒱 ?Tr'
      unfolding D-def by auto
    ultimately
    have ∃ Tr'. Tr' ⊒ Tr ∧ D 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition I :: 'e BSP
where
I 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α' β'. ((β' @ [c] @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []
                  ∧ β'↿(V𝒱 ∪ C𝒱) = β↿(V𝒱 ∪ C𝒱)))

lemma BSP-valid-I: BSP-valid I
proof −
  {
    fix 𝒱::('e V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr'⊒ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have I 𝒱 ?Tr' using ‹isViewOn 𝒱 E›
      unfolding isViewOn-def I-def by auto
    ultimately
    have ∃ Tr'. Tr' ⊒ Tr ∧ I 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed



type-synonym 'e Rho = 'e V-rec ⇒ 'e set

definition
Adm :: 'e V-rec ⇒ 'e Rho ⇒ ('e list) set ⇒ 'e list ⇒ 'e ⇒ bool
where
Adm 𝒱 ϱ Tr β e ≡
  ∃γ. ((γ @ [e]) ∈ Tr ∧ γ↿(ϱ 𝒱) = β↿(ϱ 𝒱))
```

definition *IA* :: *'e Rho* ⇒ *'e BSP*
where
IA ϱ 𝒱 *Tr* ≡
∀ α β. ∀ c∈$C_{\mathcal{V}}$. ((β @ α) ∈ *Tr* ∧ α↿$C_{\mathcal{V}}$ = [] ∧ (*Adm* 𝒱 ϱ *Tr* β *c*))
⟶ (∃ α′ β′. ((β′ @ [*c*] @ α′) ∈ *Tr*) ∧ α′↿$V_{\mathcal{V}}$ = α↿$V_{\mathcal{V}}$
∧ α′↿$C_{\mathcal{V}}$ = [] ∧ β′↿($V_{\mathcal{V}}$ ∪ $C_{\mathcal{V}}$) = β↿($V_{\mathcal{V}}$ ∪ $C_{\mathcal{V}}$))

lemma *BSP-valid-IA*: *BSP-valid* (*IA* ϱ)
proof −
{
fix 𝒱 :: (*'a V-rec*)
fix *Tr E*
assume *isViewOn* 𝒱 *E*
and *areTracesOver Tr E*
let *?Tr′*={*t*. (*set t*) ⊆ *E*}
have *?Tr′*⊇ *Tr*
by (*meson Ball-Collect* ‹*areTracesOver Tr E*› *areTracesOver-def*)
moreover
have *IA* ϱ 𝒱 *?Tr′* **using** ‹*isViewOn* 𝒱 *E*›
unfolding *isViewOn-def IA-def* **by** *auto*
ultimately
have ∃ *Tr′*. *Tr′* ⊇ *Tr* ∧ *IA* ϱ 𝒱 *Tr′*
by *auto*
}
thus *?thesis*
unfolding *BSP-valid-def* **by** *auto*
qed

definition *BSD* :: *'e BSP*
where
BSD 𝒱 *Tr* ≡
∀ α β. ∀ c∈$C_{\mathcal{V}}$. ((β @ [*c*] @ α) ∈ *Tr* ∧ α↿$C_{\mathcal{V}}$ = [])
⟶ (∃ α′. ((β @ α′) ∈ *Tr* ∧ α′↿$V_{\mathcal{V}}$ = α↿$V_{\mathcal{V}}$ ∧ α′↿$C_{\mathcal{V}}$ = []))

lemma *BSP-valid-BSD*: *BSP-valid BSD*
proof −
{
fix 𝒱::(*'e V-rec*)
fix *Tr E*
assume *isViewOn* 𝒱 *E*
and *areTracesOver Tr E*
let *?Tr′*={*t*. (*set t*) ⊆ *E*}
have *?Tr′*⊇ *Tr*
by (*meson Ball-Collect* ‹*areTracesOver Tr E*› *areTracesOver-def*)
moreover
have *BSD* 𝒱 *?Tr′*
unfolding *BSD-def* **by** *auto*
ultimately
have ∃ *Tr′*. *Tr′* ⊇ *Tr* ∧ *BSD* 𝒱 *Tr′*

```
    by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition BSI :: 'e BSP
where
BSI 𝒱 Tr ≡
  ∀ α β. ∀ c∈C𝒱. ((β @ α) ∈ Tr ∧ α↑C𝒱 = [])
    ⟶ (∃ α'. ((β @ [c] @ α') ∈ Tr ∧ α'↑V𝒱 = α↑V𝒱 ∧ α'↑C𝒱 = []))

lemma BSP-valid-BSI: BSP-valid BSI
proof −
  {
    fix 𝒱::('e V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr' ⊇ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have BSI 𝒱 ?Tr' using ‹isViewOn 𝒱 E›
      unfolding isViewOn-def BSI-def by auto
    ultimately
    have ∃ Tr'. Tr' ⊇ Tr ∧ BSI 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition BSIA :: 'e Rho ⇒ 'e BSP
where
BSIA ϱ 𝒱 Tr ≡
  ∀ α β. ∀ c∈C𝒱. ((β @ α) ∈ Tr ∧ α↑C𝒱 = [] ∧ (Adm 𝒱 ϱ Tr β c))
    ⟶ (∃ α'. ((β @ [c] @ α') ∈ Tr ∧ α'↑V𝒱 = α↑V𝒱 ∧ α'↑C𝒱 = []))

lemma BSP-valid-BSIA: BSP-valid (BSIA ϱ)
proof −
  {
    fix 𝒱 :: ('a V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr' ⊇ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
```

```
    have BSIA ϱ V ?Tr' using ‹isViewOn V E›
      unfolding isViewOn-def BSIA-def by auto
    ultimately
    have ∃ Tr'. Tr' ⊒ Tr ∧ BSIA ϱ V Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


record 'e Gamma =
  Nabla :: 'e set
  Delta :: 'e set
  Upsilon :: 'e set


abbreviation GammaNabla :: 'e Gamma ⇒ 'e set
(‹∇_› [100] 1000)
where
∇Γ ≡ (Nabla Γ)

abbreviation GammaDelta :: 'e Gamma ⇒ 'e set
(‹Δ_› [100] 1000)
where
ΔΓ ≡ (Delta Γ)

abbreviation GammaUpsilon :: 'e Gamma ⇒ 'e set
(‹ϒ_› [100] 1000)
where
ϒΓ ≡ (Upsilon Γ)



definition FCD :: 'e Gamma ⇒ 'e BSP
where
FCD Γ V Tr ≡
  ∀ α β. ∀ c∈(CV ∩ ϒΓ). ∀ v∈(VV ∩ ∇Γ).
    ((β @ [c,v] @ α) ∈ Tr ∧ α ↾ CV = [])
      ⟶ (∃ α'. ∃ δ'. (set δ') ⊆ (NV ∩ ΔΓ)
                  ∧ ((β @ δ' @ [v] @ α') ∈ Tr
                  ∧ α'↾VV = α↾VV ∧ α'↾CV = []))

lemma BSP-valid-FCD: BSP-valid (FCD Γ)
proof −
  {
    fix V::('a V-rec)
    fix Tr E
    assume isViewOn V E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr' ⊒ Tr
```

```
    by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
  moreover
  have FCD Γ 𝒱 ?Tr'
    proof −
      {
        fix α β c v
        assume  c ∈ C_𝒱 ∩ ϒ_Γ
          and  v ∈ V_𝒱 ∩ ∇_Γ
          and  β @ [c ,v] @ α ∈ ?Tr'
          and  α ↾ C_𝒱 = []
        let ?α'=α and ?δ'=[]
        from ‹β @ [c ,v] @ α ∈ ?Tr'› have β @ ?δ' @ [v] @ ?α' ∈ ?Tr'
          by auto
        hence  (set ?δ') ⊆ (N_𝒱 ∩ Δ_Γ) ∧ ((β @ ?δ' @ [v] @ ?α') ∈ ?Tr'
                  ∧ ?α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ ?α' ↾ C_𝒱 = [])
          using ‹isViewOn 𝒱 E› ‹α ↾ C_𝒱 = []›
          unfolding isViewOn-def ‹α ↾ C_𝒱 = []› by auto
        hence ∃α'. ∃δ'. (set δ') ⊆ (N_𝒱 ∩ Δ_Γ) ∧ ((β @ δ' @ [v] @ α') ∈ ?Tr'
          ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = [])
          by blast
      }
      thus ?thesis
        unfolding FCD-def by auto
    qed
  ultimately
  have  ∃  Tr'. Tr' ⊇ Tr  ∧ FCD Γ 𝒱 Tr'
    by auto
 }
 thus ?thesis
   unfolding BSP-valid-def by auto
qed


definition FCI :: 'e Gamma ⇒ 'e BSP
where
FCI Γ 𝒱 Tr ≡
 ∀α β. ∀c∈(C_𝒱 ∩ ϒ_Γ). ∀v∈(V_𝒱 ∩ ∇_Γ).
   ((β @ [v] @ α) ∈ Tr ∧ α↾C_𝒱 = [])
     ⟶ (∃α'. ∃δ'. (set δ') ⊆ (N_𝒱 ∩ Δ_Γ)
                  ∧ ((β @ [c] @ δ' @ [v] @ α') ∈ Tr
                  ∧ α'↾V_𝒱 = α↾V_𝒱 ∧ α'↾C_𝒱 = []))

lemma BSP-valid-FCI: BSP-valid (FCI Γ)
proof −
 {
   fix 𝒱::('a V-rec)
   fix Tr E
   assume isViewOn 𝒱 E
   and areTracesOver Tr E
   let ?Tr'={t. (set t) ⊆ E}
   have ?Tr'⊇ Tr
     by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
```

```
    moreover
    have FCI Γ 𝒱 ?Tr'
      proof −
        {
          fix α β c v
          assume  c ∈ C_𝒱 ∩ ϒ_Γ
             and  v ∈ V_𝒱 ∩ ∇_Γ
             and  β @ [v] @ α ∈ ?Tr'
             and  α ↾ C_𝒱 = []
          let ?α'=α and ?δ'=[]
          from ‹c ∈ C_𝒱 ∩ ϒ_Γ› have c ∈ E
            using ‹isViewOn 𝒱 E›
            unfolding isViewOn-def by auto
          with  ‹β @ [v] @ α ∈ ?Tr'› have β @ [c] @ ?δ' @ [v] @ ?α' ∈ ?Tr'
            by auto
          hence  (set ?δ') ⊆ (N_𝒱 ∩ Δ_Γ) ∧ ((β @ [c] @ ?δ' @ [v] @ ?α') ∈ ?Tr'
                    ∧ ?α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ ?α' ↾ C_𝒱 = [])
           using ‹isViewOn 𝒱 E› ‹α ↾ C_𝒱 = []›  unfolding isViewOn-def ‹α ↾ C_𝒱 = []› by auto
         hence
           ∃α'. ∃δ'. (set δ') ⊆ (N_𝒱 ∩ Δ_Γ) ∧ ((β @ [c] @ δ' @ [v] @ α') ∈ ?Tr'
            ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = [])
            by blast
        }
        thus ?thesis
          unfolding FCI-def by auto
      qed
    ultimately
    have  ∃ Tr'. Tr' ⊒ Tr  ∧ FCI Γ 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition FCIA :: 'e Rho ⇒ 'e Gamma ⇒ 'e BSP
where
FCIA ϱ Γ 𝒱 Tr ≡
  ∀α β. ∀c∈(C_𝒱 ∩ ϒ_Γ). ∀v∈(V_𝒱 ∩ ∇_Γ).
    ((β @ [v] @ α) ∈ Tr ∧ α↾C_𝒱 = [] ∧ (Adm 𝒱 ϱ Tr β c))
      ⟶ (∃α'. ∃δ'. (set δ') ⊆ (N_𝒱 ∩ Δ_Γ)
                  ∧ ((β @ [c] @ δ' @ [v] @ α') ∈ Tr
                  ∧ α'↾V_𝒱 = α↾V_𝒱 ∧ α'↾C_𝒱 = []))

lemma BSP-valid-FCIA: BSP-valid (FCIA ϱ Γ)
proof −
  {
    fix 𝒱 :: ('a V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
```

```
    have ?Tr'⊇ Tr
      by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have FCIA ϱ Γ 𝒱 ?Tr'
    proof −
       {
        fix α β c v
        assume  c ∈ C𝒱 ∩ ϒΓ
          and  v ∈V𝒱 ∩ ∇Γ
          and  β @ [v] @ α ∈ ?Tr'
          and  α ↾ C𝒱 = []
        let ?α'=α and ?δ'=[]
        from ‹c ∈ C𝒱 ∩ ϒΓ› have c ∈ E
          using ‹isViewOn 𝒱 E› unfolding isViewOn-def by auto
        with  ‹β @ [v] @ α ∈ ?Tr'› have β @ [c] @ ?δ' @ [v] @ ?α' ∈ ?Tr'
          by auto
        hence  (set ?δ') ⊆ (N𝒱 ∩ ΔΓ) ∧ ((β @ [c] @ ?δ' @ [v] @ ?α') ∈ ?Tr'
                  ∧ ?α' ↾ V𝒱 = α ↾ V𝒱 ∧ ?α' ↾ C𝒱 = [])
          using ‹isViewOn 𝒱 E› ‹α ↾ C𝒱 = []›
          unfolding isViewOn-def ‹α ↾ C𝒱 = []› by auto
        hence
          ∃α'. ∃δ'. (set δ') ⊆ (N𝒱 ∩ ΔΓ) ∧ ((β @ [c] @ δ' @ [v] @ α') ∈ ?Tr'
          ∧ α' ↾ V𝒱 = α ↾ V𝒱 ∧ α' ↾ C𝒱 = [])
          by blast
       }
       thus ?thesis
         unfolding FCIA-def by auto
     qed
    ultimately
    have  ∃ Tr'. Tr' ⊇ Tr  ∧ FCIA ϱ Γ 𝒱 Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition SR :: 'e BSP
where
SR 𝒱 Tr ≡ ∀τ∈Tr. τ ↾ (V𝒱 ∪ N𝒱) ∈ Tr

lemma BSP-valid SR
proof −
  {
    fix 𝒱::('e V-rec)
    fix Tr E
    assume isViewOn 𝒱 E
    and areTracesOver Tr E
    let ?Tr'={t. ∃τ ∈ Tr. t=τ↾(V𝒱 ∪ N𝒱)} ∪ Tr
    have ?Tr'⊇ Tr
      by blast
    moreover
```

```
    have SR V ?Tr' unfolding SR-def
     proof
       fix τ
       assume τ ∈ ?Tr'
       {
         from ‹τ ∈ ?Tr'› have (∃ t∈Tr. τ = t ↾ (V_V ∪ N_V)) ∨ τ ∈ Tr
           by auto
         hence τ ↾ (V_V ∪ N_V) ∈ ?Tr'
           proof
             assume ∃ t∈Tr. τ = t ↾(V_V ∪ N_V)
             hence ∃ t∈Tr. τ ↾ (V_V ∪ N_V)= t ↾(V_V ∪ N_V)
               using projection-idempotent by metis
             thus ?thesis
               by auto
           next
             assume τ ∈ Tr
             thus ?thesis
               by auto
           qed
       }
       thus τ ↾ (V_V ∪ N_V) ∈ ?Tr'
         by auto
     qed
    ultimately
    have ∃ Tr'. Tr' ⊇ Tr ∧ SR V Tr'
      by auto
  }
  thus ?thesis
    unfolding BSP-valid-def by auto
qed


definition SD :: 'e BSP
where
SD V Tr ≡
  ∀ α β. ∀ c∈C_V. ((β @ [c] @ α) ∈ Tr ∧ α↾C_V = []) ⟶ β @ α ∈ Tr

lemma BSP-valid SD
proof −
  {
    fix V::('e V-rec)
    fix Tr E
    assume isViewOn V E
    and areTracesOver Tr E
    let ?Tr'={t. (set t) ⊆ E}
    have ?Tr' ⊇ Tr by (meson Ball-Collect ‹areTracesOver Tr E› areTracesOver-def)
    moreover
    have SD V ?Tr' unfolding SD-def by auto
    ultimately
    have ∃ Tr'. Tr' ⊇ Tr ∧ SD V Tr' by auto
  }
  thus ?thesis unfolding BSP-valid-def by auto
```

qed

definition *SI* :: *'e BSP*
where
SI $\mathcal{V}$ *Tr* ≡
$\forall\,\alpha\ \beta.\ \forall\, c{\in}C_{\mathcal{V}}.\ ((\beta\ @\ \alpha) \in Tr \wedge \alpha \restriction C_{\mathcal{V}} = []) \longrightarrow \beta\ @\ [c]\ @\ \alpha \in Tr$

lemma *BSP-valid SI*
proof −
{
fix $\mathcal{V}$::(*'a V-rec*)
fix *Tr E*
assume *isViewOn* $\mathcal{V}$ *E*
and *areTracesOver Tr E*
let *?Tr'*={*t*. (*set t*) ⊆ *E*}
have *?Tr'* ⊇ *Tr*
by (*meson Ball-Collect* ‹*areTracesOver Tr E*› *areTracesOver-def*)
moreover
have *SI* $\mathcal{V}$ *?Tr'*
using ‹*isViewOn* $\mathcal{V}$ *E*›
unfolding *isViewOn-def SI-def* **by** *auto*
ultimately
have ∃ *Tr'*. *Tr'* ⊇ *Tr* ∧ *SI* $\mathcal{V}$ *Tr'*
by *auto*
}
thus *?thesis*
unfolding *BSP-valid-def* **by** *auto*
qed

definition *SIA* :: *'e Rho* ⇒ *'e BSP*
where
SIA ϱ $\mathcal{V}$ *Tr* ≡
$\forall\,\alpha\ \beta.\ \forall\, c{\in}C_{\mathcal{V}}.\ ((\beta\ @\ \alpha) \in Tr \wedge \alpha \restriction C_{\mathcal{V}} = [] \wedge (Adm\ \mathcal{V}\ \varrho\ Tr\ \beta\ c))$
$\longrightarrow (\beta\ @\ [c]\ @\ \alpha) \in Tr$

lemma *BSP-valid* (*SIA* ϱ)
proof −
{
fix $\mathcal{V}$ :: (*'a V-rec*)
fix *Tr E*
assume *isViewOn* $\mathcal{V}$ *E*
and *areTracesOver Tr E*
let *?Tr'*={*t*. (*set t*) ⊆ *E*}
have *?Tr'* ⊇ *Tr*
by (*meson Ball-Collect* ‹*areTracesOver Tr E*› *areTracesOver-def*)
moreover
have *SIA* ϱ $\mathcal{V}$ *?Tr'*
using ‹*isViewOn* $\mathcal{V}$ *E*›
unfolding *isViewOn-def SIA-def* **by** *auto*
ultimately

have $\exists\ Tr'.\ Tr' \supseteq Tr\ \wedge SIA\ \varrho\ \mathcal{V}\ Tr'$
by *auto*
}
thus *?thesis*
unfolding *BSP-valid-def* **by** *auto*
qed

end

4.3 Information-Flow Properties

We define the notion of information-flow properties from [3].

theory *InformationFlowProperties*
imports *BasicSecurityPredicates*
begin

type-synonym $'e\ SP = ('e\ BSP)\ set$

type-synonym $'e\ IFP\text{-}type = ('e\ V\text{-}rec\ set) \times 'e\ SP$

definition *IFP-valid* :: $'e\ set \Rightarrow 'e\ IFP\text{-}type \Rightarrow bool$
where
IFP-valid E ifp $\equiv$
$\forall \mathcal{V} \in (fst\ ifp).\ isViewOn\ \mathcal{V}\ E$
$\wedge\ (\forall BSP \in (snd\ ifp).\ BSP\text{-}valid\ BSP)$

definition *IFPIsSatisfied* :: $'e\ IFP\text{-}type \Rightarrow ('e\ list)\ set \Rightarrow bool$
where
IFPIsSatisfied ifp Tr $\equiv$
$\forall\ \mathcal{V} \in (fst\ ifp).\ \forall\ BSP \in (snd\ ifp).\ BSP\ \mathcal{V}\ Tr$

end

4.4 Property Library

We define the representations of several possibilistic information-flow properties from the literature that are provided as part of MAKS in [3].

theory *PropertyLibrary*
imports *InformationFlowProperties ../SystemSpecification/EventSystems ../Verification/Basics/BSPTaxonomy*
begin

definition
HighInputsConfidential :: $'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ V\text{-}rec$
where

HighInputsConfidential L H IE ≡ (| *V=L*, *N=H−IE*, *C=H* ∩ *IE* |)

definition *HighConfidential* :: *'e set* ⇒ *'e set* ⇒ *'e V-rec*
where
HighConfidential L H ≡ (| *V=L*, *N={}*, *C=H* |)

fun *interleaving* :: *'e list* ⇒ *'e list* ⇒ (*'e list*) *set*
where
interleaving t1 [] = {*t1*} |
interleaving [] *t2* = {*t2*} |
interleaving (*e1* # *t1*) (*e2* # *t2*) =
{*t*. (∃ *t'*. *t*=(*e1* # *t'*) ∧ *t'* ∈ *interleaving t1* (*e2* #*t2*))}
∪ {*t*. (∃ *t'*. *t*=(*e2* # *t'*) ∧ *t'* ∈ *interleaving* (*e1* # *t1*) *t2*)}

definition *GNI* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
GNI L H IE ≡ ({*HighInputsConfidential L H IE*}, {*BSD*, *BSI*})

lemma *GNI-valid*: *L* ∩ *H* = {} ⟹ *IFP-valid* (*L* ∪ *H*) (*GNI L H IE*)
unfolding *IFP-valid-def GNI-def HighInputsConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-BSD BasicSecurityPredicates.BSP-valid-BSI*
by *auto*

definition *litGNI* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litGNI L H IE Tr ≡
∀ *t1 t2 t3*.
t1 @ *t2* ∈ *Tr* ∧ *t3* ↾ (*L* ∪ (*H* − *IE*)) = *t2* ↾ (*L* ∪ (*H* − *IE*))
⟶ (∃ *t4*. *t1* @ *t4* ∈ *Tr* ∧ *t4*↾(*L* ∪ (*H* ∩ *IE*)) = *t3*↾(*L* ∪ (*H* ∩ *IE*)))

definition *IBGNI* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where *IBGNI L H IE* ≡ ({*HighInputsConfidential L H IE*}, {*D*, *I*})

lemma *IBGNI-valid*: *L* ∩ *H* = {} ⟹ *IFP-valid* (*L* ∪ *H*) (*IBGNI L H IE*)
unfolding *IFP-valid-def IBGNI-def HighInputsConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-D BasicSecurityPredicates.BSP-valid-I*
by *auto*

definition
litIBGNI :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litIBGNI L H IE Tr ≡
∀ *τ-l* ∈ *Tr*. ∀ *t-hi t*.

$(set\ t\text{-}hi) \subseteq (H \cap IE)\ \wedge t \in interleaving\ t\text{-}hi\ (\tau\text{-}l \restriction L)$
$\longrightarrow (\exists\ \tau' \in Tr.\ \tau' \restriction (L \cup (H \cap IE)) = t)$

definition *FC* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
FC L H IE ≡
({*HighInputsConfidential L H IE*},
{*BSD*, *BSI*, (*FCD* ⦇ *Nabla*=*IE*, *Delta*={}, *Upsilon*=*IE* ⦈),
(*FCI* ⦇ *Nabla*=*IE*, *Delta*={}, *Upsilon*=*IE* ⦈)})

lemma *FC-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* $(L \cup H)$ (*FC L H IE*)
unfolding *IFP-valid-def FC-def HighInputsConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-BSD BasicSecurityPredicates.BSP-valid-BSI*
BasicSecurityPredicates.BSP-valid-FCD BasicSecurityPredicates.BSP-valid-FCI
by *auto*

definition *litFC* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litFC L H IE Tr ≡
$\forall$ *t-1 t-2*. $\forall$ *hi* ∈ $(H \cap IE)$.
(
($\forall$ *li* ∈ $(L \cap IE)$.
t-1 @ [*li*] @ *t-2* ∈ *Tr* ∧ *t-2* ↾ $(H \cap IE) = []$
⟶ (∃ *t-3*. *t-1* @ [*hi*] @ [*li*] @ *t-3* ∈ *Tr*
∧ *t-3* ↾ *L* = *t-2* ↾ *L* ∧ *t-3* ↾ $(H \cap IE) = []$))
∧ (*t-1* @ *t-2* ∈ *Tr* ∧ *t-2* ↾ $(H \cap IE) = []$
⟶ (∃ *t-3*. *t-1* @ [*hi*] @ *t-3* ∈ *Tr*
∧ *t-3* ↾ *L* = *t-2* ↾ *L* ∧ *t-3* ↾ $(H \cap IE) = []$))
∧ ($\forall$ *li* ∈ $(L \cap IE)$.
t-1 @ [*hi*] @ [*li*] @ *t-2* ∈ *Tr* ∧ *t-2* ↾ $(H \cap IE) = []$
⟶ (∃ *t-3*. *t-1* @ [*li*] @ *t-3* ∈ *Tr*
∧ *t-3* ↾ *L* = *t-2* ↾ *L* ∧ *t-3* ↾ $(H \cap IE) = []$))
∧ (*t-1* @ [*hi*] @ *t-2* ∈ *Tr* ∧ *t-2* ↾ $(H \cap IE) = []$
⟶ (∃ *t-3*. *t-1* @ *t-3* ∈ *Tr*
∧ *t-3* ↾ *L* = *t-2* ↾ *L* ∧ *t-3* ↾ $(H \cap IE) = []$))
)

definition *NDO* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
NDO UI L H ≡
({*HighConfidential L H*}, {*BSD*, (*BSIA* ($\lambda\ \mathcal{V}.\ C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cap UI)$))})

lemma *NDO-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* $(L \cup H)$ (*NDO UI L H*)
unfolding *IFP-valid-def NDO-def HighConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-BSD*

BasicSecurityPredicates.BSP-valid-BSIA[*of* (λ $\mathcal{V}$. $C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cap UI)$)]
by *auto*

definition *litNDO* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litNDO UI L H Tr ≡
$\forall \tau\text{-}l \in Tr.\ \forall\ \tau\text{-}hlui \in Tr.\ \ \forall\ t.$
$t\restriction L = \tau\text{-}l\restriction L \wedge t\restriction(H \cup (L \cap UI)) = \tau\text{-}hlui\restriction(H \cup (L \cap UI)) \longrightarrow t \in Tr$

definition *NF* :: *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
NF L H ≡ ({*HighConfidential L H*}, {*R*})

lemma *NF-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* ($L \cup H$) (*NF L H*)
unfolding *IFP-valid-def NF-def HighConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-R*
by *auto*

definition *litNF* :: *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litNF L H Tr ≡ $\forall \tau \in Tr.\ \tau \restriction L \in Tr$

definition *GNF* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
GNF L H IE ≡ ({*HighInputsConfidential L H IE*}, {*R*})

lemma *GNF-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* ($L \cup H$) (*GNF L H IE*)
unfolding *IFP-valid-def GNF-def HighInputsConfidential-def isViewOn-def*
V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def
using *BasicSecurityPredicates.BSP-valid-R*
by *auto*

definition *litGNF* :: *'e set* ⇒ *'e set* ⇒ *'e set* ⇒ (*'e list*) *set* ⇒ *bool*
where
litGNF L H IE Tr ≡
$\forall \tau \in Tr.\ \exists \tau' \in Tr.\ \tau'\restriction (H \cap IE) = [] \wedge \tau'\restriction L = \tau \restriction L$

definition *SEP* :: *'e set* ⇒ *'e set* ⇒ *'e IFP-type*
where
SEP L H ≡ ({*HighConfidential L H*}, {*BSD*, (*BSIA* (λ $\mathcal{V}$. $C_{\mathcal{V}}$))})

lemma *SEP-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* $(L \cup H)$ $(SEP\ L\ H)$
unfolding *IFP-valid-def SEP-def HighConfidential-def isViewOn-def V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def*
using *BasicSecurityPredicates.BSP-valid-BSD BasicSecurityPredicates.BSP-valid-BSIA*[*of* $\lambda\ \mathcal{V}.\ C_{\mathcal{V}}$]
by *auto*

definition *litSEP* :: $'e\ set \Rightarrow 'e\ set \Rightarrow ('e\ list)\ set \Rightarrow bool$
where
litSEP L H Tr $\equiv$
$\forall \tau\text{-}l \in Tr.\ \forall\ \tau\text{-}h \in Tr.$
interleaving $(\tau\text{-}l \restriction L)\ (\tau\text{-}h \restriction H) \subseteq \{\tau \in Tr\ .\ \tau \restriction L = \tau\text{-}l \restriction L\}$

definition *PSP* :: $'e\ set \Rightarrow 'e\ set \Rightarrow 'e$ *IFP-type*
where
PSP L H $\equiv$
$(\ \{HighConfidential\ L\ H\},\ \{BSD,\ (BSIA\ (\lambda\ \mathcal{V}.\ C_{\mathcal{V}} \cup N_{\mathcal{V}} \cup V_{\mathcal{V}}))\})$

lemma *PSP-valid*: $L \cap H = \{\} \Longrightarrow$ *IFP-valid* $(L \cup H)$ $(PSP\ L\ H)$
unfolding *IFP-valid-def PSP-def HighConfidential-def isViewOn-def V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def*
using *BasicSecurityPredicates.BSP-valid-BSD BasicSecurityPredicates.BSP-valid-BSIA*[*of* $\lambda\ \mathcal{V}.\ C_{\mathcal{V}} \cup N_{\mathcal{V}} \cup V_{\mathcal{V}}$]
by *auto*

definition *litPSP* :: $'e\ set \Rightarrow 'e\ set \Rightarrow ('e\ list)\ set \Rightarrow bool$
where
litPSP L H Tr $\equiv$
$(\forall \tau \in Tr.\ \tau \restriction L \in Tr)$
$\wedge\ (\forall\ \alpha\ \beta.\ (\beta\ @\ \alpha) \in Tr \wedge (\alpha \restriction H) = [] \longrightarrow (\forall\ h \in H.\ \beta\ @\ [h] \in Tr \longrightarrow \beta\ @\ [h]\ @\ \alpha \in Tr))$

end

5 Verification

5.1 Basic Definitions

We define when an event system and a state-event system are secure given an information-flow property.

theory *SecureSystems*
imports *../../SystemSpecification/StateEventSystems*
../../SecuritySpecification/InformationFlowProperties
begin

locale *SecureESIFP* =

fixes *ES* :: *'e ES-rec*
and *IFP* :: *'e IFP-type*

assumes *validES*: *ES-valid ES*
and *validIFPES*: *IFP-valid* E_{ES} *IFP*

context *SecureESIFP*
begin

definition *ES-sat-IFP* :: *bool*
where
ES-sat-IFP ≡ *IFPIsSatisfied IFP* Tr_{ES}

end

locale *SecureSESIFP* =
fixes *SES* :: (*'s*, *'e*) *SES-rec*
and *IFP* :: *'e IFP-type*

assumes *validSES*: *SES-valid SES*
and *validIFPSES*: *IFP-valid* E_{SES} *IFP*

sublocale *SecureSESIFP* ⊆ *SecureESIFP induceES SES IFP*
by (*unfold-locales*, *rule induceES-yields-ES*, *rule validSES*,
simp add: *induceES-def*, *rule validIFPSES*)

context *SecureSESIFP*
begin

abbreviation *SES-sat-IFP*
where
SES-sat-IFP ≡ *ES-sat-IFP*

end

end

5.2 Taxonomy Results

We prove the taxonomy results from [3].

theory *BSPTaxonomy*

imports *../../SystemSpecification/EventSystems*
../../SecuritySpecification/BasicSecurityPredicates
begin

locale *BSPTaxonomyDifferentCorrections* =
fixes *ES* :: *'e ES-rec*
and $\mathcal{V}$ :: *'e V-rec*

assumes *validES*: *ES-valid ES*
and *VIsViewOnE*: *isViewOn* $\mathcal{V}$ E_{ES}

locale *BSPTaxonomyDifferentViews* =
fixes *ES* :: *'e ES-rec*
and $\mathcal{V}_1$ :: *'e V-rec*
and $\mathcal{V}_2$ :: *'e V-rec*

assumes *validES*: *ES-valid ES*
and $\mathcal{V}_1$*IsViewOnE*: *isViewOn* $\mathcal{V}_1$ E_{ES}
and $\mathcal{V}_2$*IsViewOnE*: *isViewOn* $\mathcal{V}_2$ E_{ES}

locale *BSPTaxonomyDifferentViewsFirstDim*= *BSPTaxonomyDifferentViews* +
assumes *V2-subset-V1*: $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$
and *N2-supset-N1*: $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$
and *C2-subset-C1*: $C_{\mathcal{V}_2} \subseteq C_{\mathcal{V}_1}$

sublocale *BSPTaxonomyDifferentViewsFirstDim* $\subseteq$ *BSPTaxonomyDifferentViews*
by (*unfold-locales*)

locale *BSPTaxonomyDifferentViewsSecondDim*= *BSPTaxonomyDifferentViews* +
assumes *V2-subset-V1*: $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$
and *N2-supset-N1*: $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$
and *C2-equals-C1*: $C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$

sublocale *BSPTaxonomyDifferentViewsSecondDim* $\subseteq$ *BSPTaxonomyDifferentViews*
by (*unfold-locales*)

context *BSPTaxonomyDifferentCorrections*
begin

lemma *SR-implies-R*:
SR $\mathcal{V}$ $Tr_{ES} \Longrightarrow$ *R* $\mathcal{V}$ Tr_{ES}
proof −
assume *SR*: *SR* $\mathcal{V}$ Tr_{ES}
{
fix τ
assume $\tau \in Tr_{ES}$
with *SR* **have** $\tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \in Tr_{ES}$
unfolding *SR-def* **by** *auto*
hence $\exists\ \tau'.\ \tau' \in Tr_{ES} \wedge \tau' \restriction V_{\mathcal{V}} = \tau \restriction V_{\mathcal{V}} \wedge \tau' \restriction C_{\mathcal{V}} = []$
proof −

```
    assume tau-V-N-is-trace: τ ↾ (V_𝒱 ∪ N_𝒱) ∈ Tr_ES
    show ∃ τ'. τ' ∈ Tr_ES ∧ τ' ↾ V_𝒱 = τ ↾ V_𝒱 ∧ τ' ↾ C_𝒱 = []
    proof
      let ?τ' = τ ↾ (V_𝒱 ∪ N_𝒱)
      have τ ↾ (V_𝒱 ∪ N_𝒱) ↾ V_𝒱 = τ ↾ V_𝒱
        by (simp add: projection-subset-elim)
      moreover
      from VIsViewOnE have VC-disjoint 𝒱 ∧ NC-disjoint 𝒱
        unfolding isViewOn-def V-valid-def
        by auto
      then have (V_𝒱 ∪ N_𝒱) ∩ C_𝒱 = {}
        by (simp add: NC-disjoint-def VC-disjoint-def inf-sup-distrib2)
      then have ?τ' ↾ C_𝒱 = []
        by (simp add: disjoint-projection)
      ultimately
      show ?τ' ∈ Tr_ES ∧ ?τ' ↾ V_𝒱 = τ ↾ V_𝒱 ∧ ?τ' ↾ C_𝒱 = []
        using tau-V-N-is-trace by auto
    qed
  qed
 }
 thus ?thesis
   unfolding SR-def R-def by auto
qed


lemma SD-implies-BSD :
(SD 𝒱 Tr_ES) ⟹ BSD 𝒱 Tr_ES
proof −
 assume SD: SD 𝒱 Tr_ES
 {
   fix α β c
   assume c ∈ C_𝒱
     and β @ c # α ∈ Tr_ES
     and alpha-C-empty: α ↾ C_𝒱 = []
   with SD have β @ α ∈ Tr_ES
     unfolding SD-def by auto
   hence ∃α'. β @ α' ∈ Tr_ES ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = []
     using alpha-C-empty
     by auto
  }
  thus ?thesis
    unfolding SD-def BSD-def by auto
qed


lemma BSD-implies-D:
BSD 𝒱 Tr_ES ⟹ D 𝒱 Tr_ES
proof −
 assume BSD: BSD 𝒱 Tr_ES

 {
   fix α β c
```

```
    assume α ↾ C_𝒱 = []
      and c ∈ C_𝒱
      and β @ [c] @ α ∈ Tr_ES
    with BSD obtain α'
      where β @ α' ∈ Tr_ES
      and α' ↾ V_𝒱 = α ↾ V 𝒱
      and  α' ↾ C_𝒱 = []
      by (simp add: BSD-def, auto)
    hence (∃α' β'.
      (β' @ α' ∈ Tr_ES ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = []) ∧
      β' ↾ (V_𝒱 ∪ C_𝒱) = β ↾ (V_𝒱 ∪ C_𝒱))
      by auto
  }
  thus ?thesis
    unfolding BSD-def D-def
    by auto
qed


lemma SD-implies-SR:
SD 𝒱 Tr_ES ⟹ SR 𝒱 Tr_ES
unfolding SR-def
proof
  fix τ

  assume SD: SD 𝒱 Tr_ES
  assume τ-trace: τ ∈ Tr_ES

  {
    fix  n


    have SR-via-length:  ⟦ τ ∈ Tr_ES; n = length (τ ↾ C_𝒱) ⟧
      ⟹ ∃τ' ∈ Tr_ES. τ' ↾ C_𝒱 = [] ∧ τ' ↾ (V_𝒱 ∪ N_𝒱) = τ ↾ (V_𝒱 ∪ N_𝒱)
    proof (induct n arbitrary: τ)
      case 0
      note τ-in-Tr = ‹τ ∈ Tr_ES›
        and ‹0 = length (τ ↾ C_𝒱)›
      hence τ ↾ C_𝒱 = []
        by simp
      with τ-in-Tr show ?case
        by auto
    next
      case (Suc n)
      from projection-split-last[OF Suc(3)] obtain β c α
        where c-in-C: c ∈ C_𝒱
        and τ-is-βcα: τ = β @ [c] @ α
        and α-no-c: α ↾ C_𝒱 = []
        and βα-contains-n-cs: n = length ((β @ α) ↾ C_𝒱)
      by auto
      with Suc(2) have βcα-in-Tr: β @ [c] @ α ∈ Tr_ES
        by auto
```

with *SD c-in-C* $\beta c \alpha$*-in-Tr* α*-no-c* **obtain** $\beta'\ \alpha'$
where $\beta'\alpha'$*-in-Tr*: $(\beta' @ \alpha') \in Tr_{ES}$
and α'*-V-is-*α*-V*: $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
and α'*-no-c*: $\alpha' \restriction C_{\mathcal{V}} = []$
and β'*-VC-is-*β*-VC*: $\beta' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}})$
unfolding *SD-def*
by *blast*

have $(\beta' @ \alpha') \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
proof −
from β'*-VC-is-*β*-VC* **have** $\beta' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \beta \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*rule projection-subset-eq-from-superset-eq*)
with α'*-V-is-*α*-V* **have** $(\beta' @ \alpha') \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = (\beta @ \alpha) \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*simp add*: *projection-def*)
moreover
with *VIsViewOnE c-in-C* **have** $c \notin (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def NC-disjoint-def*, *auto*)
hence $(\beta @ \alpha) \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = (\beta @ [c] @ \alpha) \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*simp add*: *projection-def*)
moreover note τ*-is-*$\beta c \alpha$
ultimately show *?thesis*
by *auto*
qed
moreover
have $n = length\ ((\beta' @ \alpha') \restriction C_{\mathcal{V}})$
proof −
have $\beta' \restriction C_{\mathcal{V}} = \beta \restriction C_{\mathcal{V}}$
proof −
have $V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}} = C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by *auto*
with β'*-VC-is-*β*-VC* **have** $\beta' \restriction (C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cup N_{\mathcal{V}})) = \beta \restriction (C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cup N_{\mathcal{V}}))$
by *auto*
thus *?thesis*
by (*rule projection-subset-eq-from-superset-eq*)
qed
with α'*-no-c* α*-no-c* **have** $(\beta' @ \alpha') \restriction C_{\mathcal{V}} = (\beta @ \alpha) \restriction C_{\mathcal{V}}$
by (*simp add*: *projection-def*)
with $\beta\alpha$*-contains-n-cs* **show** *?thesis*
by *auto*
qed
with *Suc.hyps* $\beta'\alpha'$*-in-Tr* **obtain** τ'
where $\tau' \in Tr_{ES}$
and $\tau' \restriction C_{\mathcal{V}} = []$
and $\tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = (\beta' @ \alpha') \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by *auto*
ultimately show *?case*
by *auto*
qed
}

hence $\tau \in Tr_{ES} \Longrightarrow \exists \tau'.\ \tau' \in Tr_{ES} \wedge \tau' \restriction C_{\mathcal{V}} = [] \wedge \tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$

by *auto*

from *this* τ*-trace* **obtain** τ' **where**
τ'*-trace* : $\tau' \in Tr_{ES}$
and τ'*-no-C* : $\tau' \restriction C_{\mathcal{V}} = []$
and τ'-τ*-rel* : $\tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by *auto*

from τ'*-no-C* **have** $\tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*auto simp add*: *projection-on-union*)

with *VIsViewOnE* **have** τ'*-E-eq-VN*: $\tau' \restriction E_{ES} = \tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*auto simp add*: *isViewOn-def*)

from *validES* τ'*-trace* **have** (*set* τ') $\subseteq E_{ES}$
by (*auto simp add*: *ES-valid-def traces-contain-events-def*)
hence $\tau' \restriction E_{ES} = \tau'$ **by** (*simp add*: *list-subset-iff-projection-neutral*)
with τ'*-E-eq-VN* **have** $\tau' = \tau' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$ **by** *auto*
with τ'-τ*-rel* **have** $\tau' = \tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$ **by** *auto*
with τ'*-trace* **show** $\tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \in Tr_{ES}$ **by** *auto*
qed

lemma *D-implies-R*:
$D\ \mathcal{V}\ Tr_{ES} \Longrightarrow R\ \mathcal{V}\ Tr_{ES}$
proof −
assume *D*: $D\ \mathcal{V}\ Tr_{ES}$
{
fix τ n

have *R-via-length*: $[\![\tau \in Tr_{ES};\ n = length\ (\tau \restriction C_{\mathcal{V}})]\!]$
$\Longrightarrow \exists \tau' \in Tr_{ES}.\ \tau' \restriction C_{\mathcal{V}} = [] \wedge \tau' \restriction V_{\mathcal{V}} = \tau \restriction V_{\mathcal{V}}$
proof (*induct n arbitrary*: τ)
case *0*
note τ*-in-Tr* = ‹$\tau \in Tr_{ES}$›
and ‹*0* = *length* $(\tau \restriction C_{\mathcal{V}})$›
hence $\tau \restriction C_{\mathcal{V}} = []$
by *simp*
with τ*-in-Tr* **show** *?case*
by *auto*
next
case (*Suc n*)
from *projection-split-last*[*OF Suc*(*3*)] **obtain** β c α
where *c-in-C*: $c \in C_{\mathcal{V}}$
and τ*-is-*$\beta c \alpha$: $\tau = \beta\ @\ [c]\ @\ \alpha$
and α*-no-c*: $\alpha \restriction C_{\mathcal{V}} = []$
and $\beta\alpha$*-contains-n-cs*: $n = length\ ((\beta\ @\ \alpha) \restriction C_{\mathcal{V}})$
by *auto*
with *Suc*(*2*) **have** $\beta c \alpha$*-in-Tr*: $\beta\ @\ [c]\ @\ \alpha \in Tr_{ES}$
by *auto*

```
    with D c-in-C βcα-in-Tr α-no-c obtain β' α'
      where β'α'-in-Tr: (β' @ α') ∈ Tr_ES
      and α'-V-is-α-V: α' ↾ V_𝒱 = α ↾ V_𝒱
      and α'-no-c: α' ↾ C_𝒱 = []
      and β'-VC-is-β-VC: β' ↾ (V_𝒱 ∪ C_𝒱) = β ↾ (V_𝒱 ∪ C_𝒱)
      unfolding D-def
      by blast

    have (β' @ α') ↾ V_𝒱 = τ ↾ V_𝒱
    proof −
      from β'-VC-is-β-VC have β' ↾ V_𝒱 = β ↾ V_𝒱
        by (rule projection-subset-eq-from-superset-eq)
      with α'-V-is-α-V have (β' @ α') ↾ V_𝒱 = (β @ α) ↾ V_𝒱
        by (simp add: projection-def)
      moreover
      with VIsViewOnE c-in-C have c ∉ V_𝒱
        by (simp add: isViewOn-def V-valid-def VC-disjoint-def, auto)
      hence (β @ α) ↾ V_𝒱 = (β @ [c] @ α) ↾ V_𝒱
        by (simp add: projection-def)
      moreover note τ-is-βcα
      ultimately show ?thesis
        by auto
    qed
    moreover
    have n = length ((β' @ α') ↾ C_𝒱)
    proof −
      have β' ↾ C_𝒱 = β ↾ C_𝒱
      proof −
        have V_𝒱 ∪ C_𝒱 = C_𝒱 ∪ V_𝒱
          by auto
        with β'-VC-is-β-VC have β' ↾ (C_𝒱 ∪ V_𝒱) = β ↾ (C_𝒱 ∪ V_𝒱)
          by auto
        thus ?thesis
          by (rule projection-subset-eq-from-superset-eq)
      qed
      with α'-no-c α-no-c have (β' @ α') ↾ C_𝒱 = (β @ α) ↾ C_𝒱
        by (simp add: projection-def)
      with βα-contains-n-cs show ?thesis
        by auto
    qed
    with Suc.hyps β'α'-in-Tr obtain τ'
      where τ' ∈ Tr_ES
      and τ' ↾ C_𝒱 = []
      and τ' ↾ V_𝒱 = (β' @ α') ↾ V_𝒱
      by auto
    ultimately show ?case
      by auto
  qed
 }
 thus ?thesis
   by (simp add: R-def)
qed
```

lemma *SR-implies-R-for-modified-view* :
$[\![SR\ \mathcal{V}\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)]\!] \Longrightarrow R\ \mathcal{V}'\ Tr_{ES}$
proof −
assume $SR\ \mathcal{V}\ Tr_{ES}$
and $\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$
{
from $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$ *VIsViewOnE*
have *V'IsViewOnE*: $isViewOn\ \mathcal{V}'\ E_{ES}$
unfolding *isViewOn-def V-valid-def VC-disjoint-def NC-disjoint-def VN-disjoint-def* **by** *auto*
fix τ
assume $\tau \in Tr_{ES}$
with $\langle SR\ \mathcal{V}\ Tr_{ES} \rangle$ **have** $\tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \in Tr_{ES}$
unfolding *SR-def* **by** *auto*

let $?\tau' = \tau \restriction V_{\mathcal{V}'}$

from $\langle \tau \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \in Tr_{ES} \rangle$ **have** $?\tau' \in Tr_{ES}$
using $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$ **by** *simp*
moreover
from *V'IsViewOnE* **have** $?\tau' \restriction C_{\mathcal{V}'} = []$
using *disjoint-projection*
unfolding *isViewOn-def V-valid-def VC-disjoint-def* **by** *auto*
moreover
have $?\tau' \restriction V_{\mathcal{V}'} = \tau \restriction V_{\mathcal{V}'}$
by (*simp add*: *projection-subset-elim*)
ultimately
have $\exists \tau' \in Tr_{ES}.\ \tau' \restriction C_{\mathcal{V}'} = [] \wedge \tau' \restriction V_{\mathcal{V}'} = \tau \restriction V_{\mathcal{V}'}$
by *auto*
}
with $\langle SR\ \mathcal{V}\ Tr_{ES} \rangle$ **show** *?thesis*
unfolding *R-def* **using** $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$ **by** *auto*
qed

lemma *R-implies-SR-for-modified-view* :
$[\![R\ \mathcal{V}'\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)]\!] \Longrightarrow SR\ \mathcal{V}\ Tr_{ES}$
proof −
assume $R\ \mathcal{V}'\ Tr_{ES}$
and $\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$
{
fix τ
assume $\tau \in Tr_{ES}$
from $\langle R\ \mathcal{V}'\ Tr_{ES} \rangle$ $\langle \tau \in Tr_{ES} \rangle$ **obtain** τ' **where** $\tau' \in Tr_{ES}$
and $\tau' \restriction C_{\mathcal{V}'} = []$
and $\tau' \restriction V_{\mathcal{V}'} = \tau \restriction V_{\mathcal{V}'}$
unfolding *R-def* **by** *auto*
from *VIsViewOnE* $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$ **have** $isViewOn\ \mathcal{V}'\ E_{ES}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def*
by *auto*

from $\langle \tau' \restriction V_{\mathcal{V}'} = \tau \restriction V_{\mathcal{V}'} \rangle$ $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$

```
    have τ' ↾ (V_V' ∪ N_V') = τ ↾ (V_V' ∪ N_V')
      by simp

    from ‹τ' ↾ C_V' = []› have τ' =τ' ↾ (V_V' ∪ N_V')
      using validES ‹τ' ∈ Tr_ES› ‹isViewOn V' E_ES›
      unfolding projection-def ES-valid-def isViewOn-def traces-contain-events-def
      by (metis UnE filter-True filter-empty-conv)
    hence  τ' =τ ↾ (V_V' ∪ N_V')
      using ‹τ' ↾ (V_V' ∪ N_V') = τ ↾ (V_V' ∪ N_V')›
      by simp
    with ‹τ' ∈ Tr_ES› have τ ↾ (V_V' ∪ N_V') ∈ Tr_ES
      by auto
  }
  thus ?thesis
    unfolding SR-def using ‹V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)›
    by simp
qed


lemma SD-implies-BSD-for-modified-view :
⟦SD V Tr_ES; V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)⟧ ⟹ BSD V' Tr_ES
proof −
  assume SD V Tr_ES
     and V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)
   {
    fix α β c
    assume c ∈ C_V'
       and β @ [c] @ α ∈ Tr_ES
       and α↿C_V' = []

    from ‹c ∈ C_V'›  ‹V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)›
    have c ∈ C_V
      by auto
    from ‹α↿C_V' = []› ‹V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)›
    have α↿C_V = []
      by auto

    from ‹c ∈ C_V› ‹β @ [c] @ α ∈ Tr_ES› ‹α↿C_V = []›
    have β @ α ∈ Tr_ES using ‹SD V Tr_ES›
      unfolding SD-def by auto
    hence  ∃α'. β @ α' ∈ Tr_ES ∧  α' ↾ V_V' = α ↾ V_V'  ∧ α' ↾ C_V' = []
      using ‹α ↾ C_V' = []› by blast
   }
  with ‹SD V Tr_ES› show ?thesis
    unfolding BSD-def using ‹V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)› by auto
qed

lemma BSD-implies-SD-for-modified-view :
⟦BSD V' Tr_ES; V' = (| V = V_V ∪ N_V , N ={} , C = C_V |)⟧ ⟹ SD V Tr_ES
  unfolding SD-def
  proof(clarsimp)
  fix α β c
```

assume *BSD-view′* : $BSD\ (\!|V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}|\!)\ Tr_{ES}$
assume *alpha-no-C-view* : $\alpha \restriction C_{\mathcal{V}} = []$
assume *c-C-view* : $c \in C_{\mathcal{V}}$
assume *beta-c-alpha-is-trace* : $\beta\ @\ c\ \#\ \alpha \in Tr_{ES}$

from *BSD-view′ alpha-no-C-view c-C-view beta-c-alpha-is-trace*
obtain α'
  where *beta-alpha′-is-trace*: $\beta\ @\ \alpha' \in (Tr_{ES})$
    and *alpha-alpha′*: $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
    and *alpha′-no-C-view* : $\alpha' \restriction C_{\mathcal{V}} = []$
  by (*auto simp add*: *BSD-def*)

from *beta-c-alpha-is-trace validES*
have *alpha-consists-of-events*: *set* $\alpha \subseteq E_{ES}$
   by (*auto simp add*: *ES-valid-def traces-contain-events-def*)

from *alpha-no-C-view* **have** $\alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
  by (*rule projection-on-union*)
with *VIsViewOnE* **have** *alpha-on-ES* : $\alpha \restriction E_{ES} = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
  unfolding *isViewOn-def* **by** *simp*

from *alpha-consists-of-events VIsViewOnE* **have** $\alpha \restriction E_{ES} = \alpha$
  by (*simp add*: *list-subset-iff-projection-neutral*)

with *alpha-on-ES* **have** *α-eq*: $\alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha$ **by** *auto*

from *beta-alpha′-is-trace validES*
have *alpha′-consists-of-events*: *set* $\alpha' \subseteq E_{ES}$
  by (*auto simp add*: *ES-valid-def traces-contain-events-def*)

from *alpha′-no-C-view* **have** $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
  by (*rule projection-on-union*)
with *VIsViewOnE* **have** *alpha′-on-ES* : $\alpha' \restriction E_{ES} = \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
  unfolding *isViewOn-def* **by** (*simp*)

from *alpha′-consists-of-events VIsViewOnE* **have** $\alpha' \restriction E_{ES} = \alpha'$
  by (*simp add*: *list-subset-iff-projection-neutral*)

with *alpha′-on-ES* **have** *α′-eq*: $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha'$ **by** *auto*

from *alpha-alpha′ α-eq α′-eq* **have** $\alpha = \alpha'$ **by** *auto*

with *beta-alpha′-is-trace* **show** $\beta\ @\ \alpha \in Tr_{ES}$ **by** *auto*
qed

lemma *SD-implies-FCD*:
$(SD\ \mathcal{V}\ Tr_{ES}) \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
  assume *SD*: $SD\ \mathcal{V}\ Tr_{ES}$

```
    {
    fix α β c v
    assume c ∈ C_𝒱  ∩ ϒ_Γ
      and  v ∈ V_𝒱  ∩ ∇_Γ
      and alpha-C-empty: α ↾ C_𝒱 = []
      and  β @ [c, v] @ α ∈ Tr_ES
    moreover
    with VIsViewOnE  have (v # α) ↾ C_𝒱 = []
      unfolding isViewOn-def V-valid-def VC-disjoint-def projection-def by auto
    ultimately
    have β @ (v # α) ∈ Tr_ES
      using SD unfolding SD-def by auto
    with alpha-C-empty
    have ∃α'. ∃δ'. (set δ') ⊆ (N_𝒱 ∩ Δ_Γ) ∧ ((β @ δ' @ [v] @ α') ∈  Tr_ES
          ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = [])
      by (metis append.simps(1) append.simps(2) bot-least list.set(1))
  }
  thus ?thesis
    unfolding SD-def FCD-def by auto
qed
```

```
lemma SI-implies-BSI :
(SI 𝒱 Tr_ES) ⟹ BSI 𝒱 Tr_ES
proof −
  assume SI: SI 𝒱 Tr_ES
  {
    fix α β c
    assume c ∈ C_𝒱
      and  β @  α ∈ Tr_ES
      and alpha-C-empty: α ↾ C_𝒱 = []
    with SI have β @ c # α ∈ Tr_ES
      unfolding SI-def by auto
    hence  ∃α'. β @ c # α' ∈ Tr_ES ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = []
      using alpha-C-empty  by auto
  }
  thus ?thesis
    unfolding SI-def BSI-def by auto
qed
```

```
lemma BSI-implies-I:
(BSI 𝒱 Tr_ES) ⟹ (I 𝒱 Tr_ES)
proof −
  assume BSI: BSI 𝒱 Tr_ES

  {
    fix α β c
    assume c ∈ C_𝒱
      and β @ α ∈ Tr_ES
```

and $\alpha \restriction C_{\mathcal{V}} = []$
with *BSI* **obtain** α'
where $\beta \mathbin{@} [c] \mathbin{@} \alpha' \in Tr_{ES}$
and $\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
and $\alpha' \restriction C_{\mathcal{V}} = []$
unfolding *BSI-def*
by *blast*
hence
$(\exists \alpha'\ \beta'.\ (\beta' \mathbin{@} [c] \mathbin{@} \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []) \wedge$
$\beta' \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}}))$
by *auto*
}
thus *?thesis* **unfolding** *BSI-def I-def*
by *auto*
qed

lemma *SIA-implies-BSIA*:
$(SIA\ \varrho\ \mathcal{V}\ Tr_{ES}) \Longrightarrow (BSIA\ \varrho\ \mathcal{V}\ Tr_{ES})$
proof −
assume *SIA*: $SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta \mathbin{@} \alpha \in Tr_{ES}$
and *alpha-C-empty*: $\alpha \restriction C_{\mathcal{V}} = []$
and $(Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c)$
with *SIA* **obtain** $\beta \mathbin{@} c \# \alpha \in Tr_{ES}$
unfolding *SIA-def* **by** *auto*
hence $\exists\ \alpha'.\ \beta \mathbin{@} c \# \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$
using *alpha-C-empty* **by** *auto*
}
thus *?thesis*
unfolding *SIA-def BSIA-def* **by** *auto*
qed

lemma *BSIA-implies-IA*:
$(BSIA\ \varrho\ \mathcal{V}\ Tr_{ES}) \Longrightarrow (IA\ \varrho\ \mathcal{V}\ Tr_{ES})$
proof −
assume *BSIA*: $BSIA\ \varrho\ \mathcal{V}\ Tr_{ES}$

{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta \mathbin{@} \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}} = []$
and $(Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c)$
with *BSIA* **obtain** α'
where $\beta \mathbin{@} [c] \mathbin{@} \alpha' \in Tr_{ES}$
and $\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
and $\alpha' \restriction C_{\mathcal{V}} = []$

unfolding *BSIA-def*
by *blast*
hence $(\exists\, \alpha'\ \beta'.$
$(\beta' \mathbin{@} [c] \mathbin{@} \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []) \wedge$
$\beta' \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}}))$
by *auto*
}
thus *?thesis*
unfolding *BSIA-def IA-def* **by** *auto*
qed

lemma *SI-implies-SIA*:
$SI\ \mathcal{V}\ Tr_{ES} \Longrightarrow SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$
proof −
assume *SI*: $SI\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta \mathbin{@} \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}} = []$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
with *SI* **have** $\beta \mathbin{@} (c \# \alpha) \in Tr_{ES}$
unfolding *SI-def* **by** *auto*
}
thus *?thesis* **unfolding** *SI-def SIA-def* **by** *auto*
qed

lemma *BSI-implies-BSIA*:
$BSI\ \mathcal{V}\ Tr_{ES} \Longrightarrow BSIA\ \varrho\ \mathcal{V}\ Tr_{ES}$
proof −
assume *BSI*: $BSI\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta \mathbin{@} \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}} = []$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
with *BSI* **have** $\exists\ \alpha'.\ \beta \mathbin{@} (c \# \alpha') \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}\ \wedge \alpha' \restriction C_{\mathcal{V}} = []$
unfolding *BSI-def* **by** *auto*
}
thus *?thesis*
unfolding *BSI-def BSIA-def* **by** *auto*
qed

lemma *I-implies-IA*:
$I\ \mathcal{V}\ Tr_{ES} \Longrightarrow IA\ \varrho\ \mathcal{V}\ Tr_{ES}$
proof −
assume *I*: $I\ \mathcal{V}\ Tr_{ES}$
{

fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}} = []$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
with I **have** $\exists\ \alpha'\ \beta'.\ \beta'\ @\ (c\ \#\ \alpha') \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
$\wedge\ \alpha' \restriction C_{\mathcal{V}} = []\ \wedge\ \beta' \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \restriction (V_{\mathcal{V}} \cup C_{\mathcal{V}})$
unfolding *I-def* **by** *auto*
}
thus *?thesis*
unfolding *I-def IA-def* **by** *auto*
qed

lemma *SI-implies-BSI-for-modified-view* :
$[\![SI\ \mathcal{V}\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)]\!] \Longrightarrow BSI\ \mathcal{V}'\ Tr_{ES}$
proof −
assume $SI\ \mathcal{V}\ Tr_{ES}$
and $\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}'}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}'} = []$

from $\langle c \in C_{\mathcal{V}'} \rangle\ \langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$
have $c \in C_{\mathcal{V}}$
by *auto*
from $\langle \alpha \restriction C_{\mathcal{V}'} = [] \rangle\ \langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$
have $\alpha \restriction C_{\mathcal{V}} = []$
by *auto*

from $\langle c \in C_{\mathcal{V}} \rangle\ \langle \beta\ @\ \alpha \in Tr_{ES} \rangle\ \langle \alpha \restriction C_{\mathcal{V}} = [] \rangle$
have $\beta\ @\ [c]\ @\ \alpha \in Tr_{ES}$
using $\langle SI\ \mathcal{V}\ Tr_{ES} \rangle$ **unfolding** *SI-def* **by** *auto*
hence $\exists \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge\ \alpha' \restriction V_{\mathcal{V}'} = \alpha \restriction V_{\mathcal{V}'}\ \wedge \alpha' \restriction C_{\mathcal{V}'} = []$
using $\langle \alpha \restriction C_{\mathcal{V}'} = [] \rangle$
by *blast*
}
with $\langle SI\ \mathcal{V}\ Tr_{ES} \rangle$ **show** *?thesis*
unfolding *BSI-def* **using** $\langle \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!) \rangle$ **by** *auto*
qed

lemma *BSI-implies-SI-for-modified-view* :
$[\![BSI\ \mathcal{V}'\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)]\!] \Longrightarrow SI\ \mathcal{V}\ Tr_{ES}$
unfolding *SI-def*
proof (*clarsimp*)
fix $\alpha\ \beta\ c$
assume *BSI-view'* : $BSI\ (\!|V = V_{\mathcal{V}} \cup N_{\mathcal{V}},\ N = \{\},\ C = C_{\mathcal{V}}|\!)\ Tr_{ES}$
assume *alpha-no-C-view* : $\alpha \restriction C_{\mathcal{V}} = []$
assume *c-C-view* : $c \in C_{\mathcal{V}}$
assume *beta-alpha-is-trace* : $\beta\ @\ \alpha \in Tr_{ES}$

from *BSI-view′* **have** $\forall\, c{\in}C_{\mathcal{V}}.\ \beta\ @\ \alpha \in Tr_{ES} \wedge \alpha \restriction C_{\mathcal{V}} = []$
$\longrightarrow (\exists\, \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \wedge \alpha' \restriction C_{\mathcal{V}} = [])$
by (*auto simp add*: *BSI-def*)

with *beta-alpha-is-trace alpha-no-C-view* **have** $\forall\, c{\in}C_{\mathcal{V}}.$
$(\exists\, \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \wedge \alpha' \restriction C_{\mathcal{V}} = [])$
by *auto*

with *this BSI-view′ c-C-view* **obtain** α'
where *beta-c-alpha′-is-trace*: $\beta\ @\ [c]\ @\ \alpha' \in Tr_{ES}$
and *alpha-alpha′*: $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
and *alpha′-no-C-view* : $\alpha' \restriction C_{\mathcal{V}} = []$
by *auto*

from *beta-alpha-is-trace validES*
have *alpha-consists-of-events*: *set* $\alpha \subseteq E_{ES}$
by (*auto simp add*: *ES-valid-def traces-contain-events-def*)

from *alpha-no-C-view* **have** $\alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*rule projection-on-union*)
with *VIsViewOnE* **have** *alpha-on-ES* : $\alpha \restriction E_{ES} = \alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
unfolding *isViewOn-def* **by** (*simp*)

from *alpha-consists-of-events VIsViewOnE* **have** $\alpha \restriction E_{ES} = \alpha$
by (*simp add*: *list-subset-iff-projection-neutral*)

with *alpha-on-ES* **have** *α-eq*: $\alpha \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha$ **by** *auto*

from *beta-c-alpha′-is-trace validES*
have *alpha′-consists-of-events*: *set* $\alpha' \subseteq E_{ES}$
by (*auto simp add*: *ES-valid-def traces-contain-events-def*)

from *alpha′-no-C-view* **have** $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}}) = \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
by (*rule projection-on-union*)
with *VIsViewOnE* **have** *alpha′-on-ES* : $\alpha' \restriction E_{ES} = \alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}})$
unfolding *isViewOn-def* **by** (*simp*)

from *alpha′-consists-of-events VIsViewOnE* **have** $\alpha' \restriction E_{ES} = \alpha'$
by (*simp add*: *list-subset-iff-projection-neutral*)

with *alpha′-on-ES* **have** *α′-eq*: $\alpha' \restriction (V_{\mathcal{V}} \cup N_{\mathcal{V}}) = \alpha'$ **by** *auto*

from *alpha-alpha′ α-eq α′-eq* **have** $\alpha = \alpha'$ **by** *auto*

with *beta-c-alpha′-is-trace* **show** $\beta\ @\ c\ \#\ \alpha \in Tr_{ES}$ **by** *auto*
qed

lemma *SIA-implies-BSIA-for-modified-view* :
$[\![SIA\ \varrho\ \mathcal{V}\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)\ ;\ \varrho\ \mathcal{V} = \varrho'\ \mathcal{V}']\!] \Longrightarrow BSIA\ \varrho'\ \mathcal{V}'\ Tr_{ES}$

proof −
assume *SIA* $\varrho\ \mathcal{V}\ Tr_{ES}$
and $\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$
and $\varrho\ \mathcal{V} = \varrho'\ \mathcal{V}'$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}'}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}'} = []$
and $Adm\ \mathcal{V}'\ \varrho'\ Tr_{ES}\ \beta\ c$

from ‹$c \in C_{\mathcal{V}'}$› ‹$\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$›
have $c \in C_{\mathcal{V}}$
by *auto*
from ‹$\alpha{\restriction}C_{\mathcal{V}'} = []$› ‹$\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$›
have $\alpha{\restriction}C_{\mathcal{V}} = []$
by *auto*
from ‹$Adm\ \mathcal{V}'\ \varrho'\ Tr_{ES}\ \beta\ c$› ‹$\varrho\ \mathcal{V} = \varrho'\ \mathcal{V}'$›
have $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
by (*simp add*: *Adm-def*)

from ‹$c \in C_{\mathcal{V}}$› ‹$\beta\ @\ \alpha \in Tr_{ES}$› ‹$\alpha{\restriction}C_{\mathcal{V}} = []$› ‹$Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$›
have $\beta\ @\ [c]\ @\ \alpha \in Tr_{ES}$
using ‹$SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$› **unfolding** *SIA-def* **by** *auto*
hence $\exists \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge\ \alpha'{\restriction}\ V_{\mathcal{V}'} = \alpha{\restriction}\ V_{\mathcal{V}'}\ \wedge \alpha'{\restriction}\ C_{\mathcal{V}'} = []$
using ‹$\alpha{\restriction}\ C_{\mathcal{V}'} = []$› **by** *blast*
}
with ‹$SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$› **show** *?thesis*
unfolding *BSIA-def* **using** ‹$\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$›
by *auto*
qed

lemma *BSIA-implies-SIA-for-modified-view* :
$[\![BSIA\ \varrho'\ \mathcal{V}'\ Tr_{ES};\ \mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!);\ \varrho\ \mathcal{V} = \varrho'\ \mathcal{V}']\!] \Longrightarrow SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$
proof −
assume *BSIA* $\varrho'\ \mathcal{V}'\ Tr_{ES}$
and $\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$
and $\varrho\ \mathcal{V} = \varrho'\ \mathcal{V}'$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}} = []$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$

from ‹$c \in C_{\mathcal{V}}$› ‹$\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$›
have $c \in C_{\mathcal{V}'}$
by *auto*
from ‹$\alpha{\restriction}C_{\mathcal{V}} = []$› ‹$\mathcal{V}' = (\!|\ V = V_{\mathcal{V}} \cup N_{\mathcal{V}}\ ,\ N = \{\}\ ,\ C = C_{\mathcal{V}}\ |\!)$›
have $\alpha{\restriction}C_{\mathcal{V}'} = []$
by *auto*
from ‹$Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$› ‹$\varrho\ \mathcal{V} = \varrho'\ \mathcal{V}'$›

```
    have Adm 𝒱' ϱ' Tr_ES β c
      by (simp add: Adm-def)

    from ‹c ∈ C_𝒱'› ‹β @ α ∈ Tr_ES› ‹α↼C_𝒱' = []› ‹Adm 𝒱' ϱ' Tr_ES β c›
    obtain α' where β @ [c] @ α' ∈ Tr_ES
                and α' ↼ V_𝒱' = α ↼ V_𝒱'
                and α' ↼ C_𝒱' = []
      using ‹BSIA ϱ' 𝒱' Tr_ES› unfolding BSIA-def by blast


    from ‹β @ α ∈ Tr_ES› validES
    have alpha-consists-of-events: set α ⊆ E_ES
      by (auto simp add: ES-valid-def traces-contain-events-def)

    from ‹β @ [c] @ α' ∈ Tr_ES› validES
    have alpha'-consists-of-events: set α' ⊆ E_ES
      by (auto simp add: ES-valid-def traces-contain-events-def)

    from ‹α' ↼ V_𝒱' = α ↼ V_𝒱'› ‹𝒱' = (| V = V_𝒱 ∪ N_𝒱 , N = {} , C = C_𝒱 |)›
    have α'↼(V_𝒱 ∪ N_𝒱)=α↼(V_𝒱 ∪ N_𝒱) by auto
    with ‹α' ↼ C_𝒱' = []› ‹α↼C_𝒱 = []› ‹𝒱' = (| V = V_𝒱 ∪ N_𝒱 , N = {} , C = C_𝒱 |)›
    have α'↼(V_𝒱 ∪ N_𝒱 ∪ C_𝒱)=α↼(V_𝒱 ∪ N_𝒱 ∪ C_𝒱)
      by (simp add: projection-on-union)
    with VIsViewOnE alpha-consists-of-events alpha'-consists-of-events
    have α'=α unfolding isViewOn-def
      by (simp add: list-subset-iff-projection-neutral)

    hence β @ [c] @ α ∈ Tr_ES
      using ‹β @ [c] @ α' ∈ Tr_ES› by blast
  }
  with ‹BSIA ϱ' 𝒱' Tr_ES› show ?thesis
    unfolding SIA-def using ‹𝒱' = (| V = V_𝒱 ∪ N_𝒱 , N ={} , C = C_𝒱 |)› by auto
qed
end


lemma Adm-implies-Adm-for-modified-rho:
⟦ Adm 𝒱_2 ϱ_2 Tr α e;ϱ_2(𝒱_2) ⊇ ϱ_1(𝒱_1)⟧ ⟹ Adm 𝒱_1 ϱ_1 Tr α e
proof −
  assume Adm 𝒱_2 ϱ_2 Tr α e
    and ϱ_2(𝒱_2) ⊇ ϱ_1(𝒱_1)
  then obtain γ
    where γ @ [e] ∈ Tr
      and γ ↼ ϱ_2 𝒱_2 = α ↼ ϱ_2 𝒱_2
    unfolding Adm-def by auto
  thus Adm 𝒱_1 ϱ_1 Tr α e
    unfolding Adm-def
    using ‹ϱ_1 𝒱_1 ⊆ ϱ_2 𝒱_2› non-empty-projection-on-subset
    by blast
qed

context BSPTaxonomyDifferentCorrections
```

begin

lemma *SI-implies-FCI*:
$(SI\ \mathcal{V}\ Tr_{ES}) \Longrightarrow FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume *SI*: $SI\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c\ v$
assume $c \in C_{\mathcal{V}}\ \cap \Upsilon_{\Gamma}$
and $v \in V_{\mathcal{V}}\ \cap \nabla_{\Gamma}$
and $\beta\ @\ [v]\ @\ \alpha \in Tr_{ES}$
and *alpha-C-empty*: $\alpha \restriction C_{\mathcal{V}} = []$
moreover
with *VIsViewOnE* **have** $(v\ \#\ \alpha) \restriction C_{\mathcal{V}} = []$
unfolding *isViewOn-def V-valid-def VC-disjoint-def projection-def* **by** *auto*
ultimately
have $\beta\ @\ [c\ ,\ v]\ @\ \alpha \in Tr_{ES}$ **using** *SI* **unfolding** *SI-def* **by** *auto*
with *alpha-C-empty*
have $\exists \alpha'.\ \exists \delta'.$
$(set\ \delta') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma}) \wedge ((\beta\ @\ [c]\ @\ \delta'\ @\ [v]\ @\ \alpha') \in\ Tr_{ES}$
$\wedge\ \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = [])$
by (*metis append.simps*(*1*) *append.simps*(*2*) *bot-least list.set*(*1*))
}
thus *?thesis*
unfolding *SI-def FCI-def* **by** *auto*
qed

lemma *SIA-implies-FCIA*:
$(SIA\ \varrho\ \mathcal{V}\ Tr_{ES}) \Longrightarrow FCIA\ \varrho\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume *SIA*: $SIA\ \varrho\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c\ v$
assume $c \in C_{\mathcal{V}}\ \cap \Upsilon_{\Gamma}$
and $v \in V_{\mathcal{V}}\ \cap \nabla_{\Gamma}$
and $\beta\ @\ [v]\ @\ \alpha \in Tr_{ES}$
and *alpha-C-empty*: $\alpha \restriction C_{\mathcal{V}} = []$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
moreover
with *VIsViewOnE* **have** $(v\ \#\ \alpha) \restriction C_{\mathcal{V}} = []$
unfolding *isViewOn-def V-valid-def VC-disjoint-def projection-def* **by** *auto*
ultimately
have $\beta\ @\ [c\ ,\ v]\ @\ \alpha \in Tr_{ES}$ **using** *SIA* **unfolding** *SIA-def* **by** *auto*
with *alpha-C-empty*
have $\exists \alpha'.\ \exists \delta'.$
$(set\ \delta') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma}) \wedge ((\beta\ @\ [c]\ @\ \delta'\ @\ [v]\ @\ \alpha') \in\ Tr_{ES}$
$\wedge\ \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = [])$
by (*metis append.simps*(*1*) *append.simps*(*2*) *bot-least list.set*(*1*))
}
thus *?thesis*

unfolding *SIA-def FCIA-def* **by** *auto*
qed

lemma *FCI-implies-FCIA*:
$(FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}) \Longrightarrow FCIA\ \varrho\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume *FCI*: $FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c\ v$
assume $c \in C_{\mathcal{V}}\ \cap \Upsilon_{\Gamma}$
and $v \in V_{\mathcal{V}}\ \cap \nabla_{\Gamma}$
and $\beta\ @\ [v]\ @\ \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}} = []$
with *FCI* **have** $\exists \alpha'\ \delta'.\ set\ \delta' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge$
$\beta\ @\ [c]\ @\ \delta'\ @\ [v]\ @\ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$
unfolding *FCI-def* **by** *auto*
}
thus *?thesis*
unfolding *FCI-def FCIA-def* **by** *auto*
qed

lemma *Trivially-fulfilled-SR-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow SR\ \mathcal{V}\ Tr_{ES}$
proof −
assume $C_{\mathcal{V}}=\{\}$
{
fix τ
assume $\tau \in Tr_{ES}$
hence $\tau=\tau\restriction E_{ES}$ **using** *validES*
unfolding *ES-valid-def traces-contain-events-def projection-def* **by** *auto*
with $\langle C_{\mathcal{V}}=\{\}\rangle$ **have** $\tau=\tau\restriction(V_{\mathcal{V}}\cup N_{\mathcal{V}})$
using *VIsViewOnE* **unfolding** *isViewOn-def* **by** *auto*
with $\langle \tau \in Tr_{ES}\rangle$ **have** $\tau\restriction(V_{\mathcal{V}}\cup N_{\mathcal{V}}) \in Tr_{ES}$
by *auto*
}
thus *?thesis*
unfolding *SR-def* **by** *auto*
qed

lemma *Trivially-fulfilled-R-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow R\ \mathcal{V}\ Tr_{ES}$
proof −
assume $C_{\mathcal{V}}=\{\}$
{
fix τ
assume $\tau \in Tr_{ES}$
hence $\tau=\tau\restriction E_{ES}$ **using** *validES*
unfolding *ES-valid-def traces-contain-events-def projection-def* **by** *auto*
with $\langle C_{\mathcal{V}}=\{\}\rangle$ **have** $\tau=\tau\restriction(V_{\mathcal{V}}\cup N_{\mathcal{V}})$

using *VIsViewOnE* **unfolding** *isViewOn-def* **by** *auto*
with ‹$\tau \in Tr_{ES}$› ‹$C_{\mathcal{V}}$={}› **have** $\exists \tau' \in Tr_{ES}.\ \tau{\restriction}C_{\mathcal{V}}$=[] $\wedge\ \tau'{\restriction}V_{\mathcal{V}}$=$\tau{\restriction}V_{\mathcal{V}}$
unfolding *projection-def* **by** *auto*
}
thus *?thesis*
unfolding *R-def* **by** *auto*
qed

lemma *Trivially-fulfilled-SD-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow SD\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *SD-def*)

lemma *Trivially-fulfilled-BSD-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow BSD\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *BSD-def*)

lemma *Trivially-fulfilled-D-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow D\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *D-def*)

lemma *Trivially-fulfilled-FCD-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *FCD-def*)

lemma *Trivially-fullfilled-R-V-empty*:
$V_{\mathcal{V}}$={} $\Longrightarrow R\ \mathcal{V}\ Tr_{ES}$
proof −
assume $V_{\mathcal{V}}$={}
{
fix τ
assume $\tau \in Tr_{ES}$
let *?*τ'=[]
from ‹$\tau \in Tr_{ES}$›**have** *?*$\tau' \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
with ‹$V_{\mathcal{V}}$={}›
have $\exists \tau' \in Tr_{ES}.\ \tau'{\restriction}C_{\mathcal{V}}$=[] $\wedge\ \tau'{\restriction}V_{\mathcal{V}}$=$\tau{\restriction}V_{\mathcal{V}}$
by (*metis projection-on-empty-trace projection-to-emptyset-is-empty-trace*)
}
thus *?thesis*
unfolding *R-def* **by** *auto*
qed

lemma *Trivially-fulfilled-BSD-V-empty*:
$V_{\mathcal{V}} = \{\} \Longrightarrow BSD\ \mathcal{V}\ Tr_{ES}$
proof −
assume $V_{\mathcal{V}}$={}
{
fix $\alpha\ \beta\ c$
assume β @ [c] @ $\alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}}$= []

```
    from ‹β @ [c] @ α ∈ Tr_ES› have β ∈ Tr_ES
      using validES
      unfolding ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def by auto

    let ?α'=[]
    from ‹β ∈ Tr_ES› ‹V_𝒱={}›
    have β@ ?α'∈Tr_ES ∧ ?α'↿V_𝒱 = α↿V_𝒱 ∧ ?α'↿C_𝒱 = []
      by (simp add: projection-on-empty-trace projection-to-emptyset-is-empty-trace)
    hence
    ∃α'.
      β @ α'∈Tr_ES ∧ α'↿V_𝒱 = α↿V_𝒱 ∧ α'↿C_𝒱 = [] by blast
  }
  thus ?thesis
    unfolding BSD-def by auto
qed

lemma Trivially-fulfilled-D-V-empty:
V_𝒱 = {} ⟹ D 𝒱 Tr_ES
proof −
  assume V_𝒱={}
  {
    fix α β c
    assume β @ [c] @ α ∈ Tr_ES
      and α↿C_𝒱= []

    from ‹β @ [c] @ α ∈ Tr_ES› have β ∈ Tr_ES
      using validES
      unfolding ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def by auto

    let ?β'=β and ?α'=[]
    from ‹β ∈ Tr_ES› ‹V_𝒱={}›
    have ?β'@ ?α'∈Tr_ES ∧ ?α'↿V_𝒱 = α↿V_𝒱 ∧ ?α'↿C_𝒱 = [] ∧ ?β'↿(V_𝒱 ∪ C_𝒱) = β↿(V_𝒱 ∪ C_𝒱)
      by (simp add: projection-on-empty-trace projection-to-emptyset-is-empty-trace)
    hence
    ∃α' β'.
      β'@ α'∈Tr_ES ∧ α'↿V_𝒱 = α↿V_𝒱 ∧ α'↿C_𝒱 = [] ∧ β'↿(V_𝒱 ∪ C_𝒱) = β↿(V_𝒱 ∪ C_𝒱)
      by blast
  }
  thus ?thesis
    unfolding D-def by auto
qed

lemma Trivially-fulfilled-FCD-V-empty:
V_𝒱 = {} ⟹ FCD Γ 𝒱 Tr_ES
  by (simp add: FCD-def)


lemma Trivially-fulfilled-FCD-Nabla-ϒ-empty:
⟦∇_Γ={} ∨ ϒ_Γ={}⟧⟹ FCD Γ 𝒱 Tr_ES
proof −
  assume ∇_Γ={} ∨ ϒ_Γ={}
  thus ?thesis
```

```
  proof(rule disjE)
    assume ∇Γ={} thus ?thesis
      by (simp add: FCD-def)
  next
    assume  ϒΓ={} thus ?thesis
      by (simp add: FCD-def)
  qed
qed

lemma Trivially-fulfilled-FCD-N-subseteq-Δ-and-BSD:
⟦NV ⊆ ΔΓ; BSD V TrES⟧ ⟹ FCD Γ V TrES
proof −
  assume NV ⊆ ΔΓ
     and BSD V TrES
  {
    fix α β c v
    assume c ∈ CV ∩ ϒΓ
       and v ∈ VV ∩ ∇Γ
       and β @ [c,v] @ α ∈ TrES
       and α↑CV = []
    from ‹c ∈ CV ∩ ϒΓ› have c ∈ CV
      by auto
    from ‹v ∈ VV ∩ ∇Γ› have v ∈ VV
      by auto

    let ?α=[v] @ α
    from ‹v ∈ VV› ‹α↑CV = []› have ?α↑CV=[]
      using VIsViewOnE
      unfolding isViewOn-def V-valid-def VC-disjoint-def projection-def by auto
    from ‹β @ [c,v] @ α ∈ TrES› have β @ [c] @ ?α ∈ TrES
      by auto

    from ‹BSD V TrES›
    obtain α′
      where β @ α′ ∈ TrES
        and α′↑VV = ([v] @ α)↑VV
        and α′↑CV = []
      using ‹c ∈ CV› ‹β @ [c] @ ?α ∈ TrES› ‹?α↑CV = []›
      unfolding BSD-def by auto

    from‹v ∈ VV› ‹α′↑VV = ([v] @ α)↑VV› have α′↑VV = [v] @ α↑VV
      by (simp add: projection-def)
    then obtain δ α′′
      where α′=δ @ [v] @ α′′
        and δ↑VV = []
        and α′′↑VV = α↑VV
       using projection-split-first-with-suffix by fastforce

    from ‹α′↑CV = []› ‹α′=δ @ [v] @ α′′› have δ↑CV=[]
      by (metis append-is-Nil-conv projection-concatenation-commute)
    from ‹α′↑CV = []› ‹α′=δ @ [v] @ α′′› have α′′↑CV=[]
      by (metis append-is-Nil-conv projection-concatenation-commute)
```

from ‹β @ $\alpha' \in Tr_{ES}$› **have** *set* $\alpha' \subseteq E_{ES}$ **using** *validES*
unfolding *ES-valid-def traces-contain-events-def* **by** *auto*
with ‹α'=δ @ [v] @ α''› **have** *set* $\delta \subseteq E_{ES}$
by *auto*
with ‹$\delta \restriction C_{\mathcal{V}}$=[]› ‹$\delta \restriction V_{\mathcal{V}}$ = []› ‹$N_{\mathcal{V}} \subseteq \Delta_\Gamma$›
have (*set* δ) $\subseteq$ ($N_{\mathcal{V}} \cap \Delta_\Gamma$)
using *VIsViewOnE projection-empty-implies-absence-of-events*
unfolding *isViewOn-def projection-def* **by** *blast*

let *?β*=β **and** *?δ'*=δ **and** *?α'*=α''
from ‹(*set* δ) $\subseteq$ ($N_{\mathcal{V}} \cap \Delta_\Gamma$)› ‹$\beta$ @ $\alpha' \in Tr_{ES}$› ‹α'=δ @ [v] @ α''›
‹$\alpha'' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$› ‹$\alpha'' \restriction C_{\mathcal{V}}$=[]›
have (*set ?δ'*)$\subseteq$($N_{\mathcal{V}} \cap \Delta_\Gamma$) $\wedge$ *?β* @ *?δ'* @ [v] @ *?α'* $\in Tr_{ES}$ $\wedge$ *?α'*$\restriction V_{\mathcal{V}}$=$\alpha \restriction V_{\mathcal{V}}$ $\wedge$ *?α'*$\restriction C_{\mathcal{V}}$=[]
by *auto*
hence $\exists \alpha''' \, \delta''$. (*set* δ'') $\subseteq$ ($N_{\mathcal{V}} \cap \Delta_\Gamma$) $\wedge$ (β @ δ'' @ [v] @ α''') $\in Tr_{ES}$
$\wedge$ $\alpha''' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha''' \restriction C_{\mathcal{V}} = []$
by *auto*
}
thus *?thesis*
unfolding *FCD-def* **by** *auto*
qed

lemma *Trivially-fulfilled-SI-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow SI \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *SI-def*)

lemma *Trivially-fulfilled-BSI-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow BSI \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *BSI-def*)

lemma *Trivially-fulfilled-I-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow I \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *I-def*)

lemma *Trivially-fulfilled-FCI-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow FCI \; \Gamma \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *FCI-def*)

lemma *Trivially-fulfilled-SIA-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow SIA \; \varrho \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *SIA-def*)

lemma *Trivially-fulfilled-BSIA-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow BSIA \; \varrho \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *BSIA-def*)

lemma *Trivially-fulfilled-IA-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow IA \; \varrho \; \mathcal{V} \; Tr_{ES}$
by (*simp add*: *IA-def*)

lemma *Trivially-fulfilled-FCIA-C-empty*:
$C_{\mathcal{V}} = \{\} \Longrightarrow FCIA\ \Gamma\ \varrho\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *FCIA-def*)

lemma *Trivially-fulfilled-FCI-V-empty*:
$V_{\mathcal{V}} = \{\} \Longrightarrow FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *FCI-def*)

lemma *Trivially-fulfilled-FCIA-V-empty*:
$V_{\mathcal{V}} = \{\} \Longrightarrow FCIA\ \varrho\ \Gamma\ \mathcal{V}\ Tr_{ES}$
by (*simp add*: *FCIA-def*)

lemma *Trivially-fulfilled-BSIA-V-empty-rho-subseteq-C-N*:
$[\![V_{\mathcal{V}} = \{\};\ \varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})\ ]\!] \Longrightarrow BSIA\ \varrho\ \ \mathcal{V}\ Tr_{ES}$
proof −
assume $V_{\mathcal{V}}=\{\}$
and $\varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}}=[]$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
from ‹$Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$›
obtain γ
where $\gamma\ @\ [c] \in Tr_{ES}$
and $\gamma{\restriction}(\varrho\ \mathcal{V}) = \beta{\restriction}(\varrho\ \mathcal{V})$
unfolding *Adm-def* **by** *auto*
from *this*(*1*) **have** $\gamma \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
moreover
from ‹$\beta\ @\ \alpha \in Tr_{ES}$› **have** $\beta \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
ultimately
have $\beta{\restriction}E_{ES}=\gamma{\restriction}E_{ES}$
using *validES VIsViewOnE* ‹$V_{\mathcal{V}}=\{\}$› ‹$\gamma{\restriction}(\varrho\ \mathcal{V}) = \beta{\restriction}(\varrho\ \mathcal{V})$› ‹$\varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})$›
non-empty-projection-on-subset
unfolding *ES-valid-def isViewOn-def traces-contain-events-def*
by (*metis empty-subsetI sup-absorb2 sup-commute*)
hence $\beta\ @\ [c] \in Tr_{ES}$ **using** *validES* ‹$\gamma\ @\ [c] \in Tr_{ES}$› ‹$\beta \in Tr_{ES}$› ‹$\gamma \in Tr_{ES}$›
unfolding *ES-valid-def traces-contain-events-def*
by (*metis list-subset-iff-projection-neutral subsetI*)

let $?\alpha'=[]$
from ‹$\beta\ @\ [c] \in Tr_{ES}$› ‹$V_{\mathcal{V}} = \{\}$›
have $\beta\ @\ [c]\ @\ ?\alpha' \in Tr_{ES} \wedge\ ?\alpha'{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}} \wedge\ ?\alpha'{\restriction}C_{\mathcal{V}} = []$
by (*simp add*: *projection-on-empty-trace projection-to-emptyset-is-empty-trace*)
hence $\exists\ \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge \alpha'{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}} \wedge \alpha'{\restriction}C_{\mathcal{V}} = []$
by *auto*
}

thus *?thesis*
unfolding *BSIA-def* **by** *auto*
qed

lemma *Trivially-fulfilled-IA-V-empty-rho-subseteq-C-N*:
$[\![V_{\mathcal{V}} = \{\};\ \varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})]\!] \Longrightarrow IA\ \varrho\ \ \mathcal{V}\ Tr_{ES}$
proof −
assume $V_{\mathcal{V}}=\{\}$
and $\varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}}=[]$
and $Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$
from ‹$Adm\ \mathcal{V}\ \varrho\ Tr_{ES}\ \beta\ c$›
obtain γ
where $\gamma\ @\ [c] \in Tr_{ES}$
and $\gamma{\restriction}(\varrho\ \mathcal{V}) = \beta{\restriction}(\varrho\ \mathcal{V})$
unfolding *Adm-def* **by** *auto*
from *this*(*1*) **have** $\gamma \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
moreover
from ‹$\beta\ @\ \alpha \in Tr_{ES}$› **have** $\beta \in Tr_{ES}$ **using** *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
ultimately
have $\beta{\restriction}E_{ES}=\gamma{\restriction}E_{ES}$
using *validES VIsViewOnE* ‹$V_{\mathcal{V}}=\{\}$› ‹$\gamma{\restriction}(\varrho\ \mathcal{V}) = \beta{\restriction}(\varrho\ \mathcal{V})$› ‹$\varrho\ \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}})$›
non-empty-projection-on-subset
unfolding *ES-valid-def isViewOn-def traces-contain-events-def*
by (*metis empty-subsetI sup-absorb2 sup-commute*)
hence $\beta\ @\ [c] \in Tr_{ES}$ **using** *validES* ‹$\gamma\ @\ [c] \in Tr_{ES}$› ‹$\beta \in Tr_{ES}$› ‹$\gamma \in Tr_{ES}$›
unfolding *ES-valid-def traces-contain-events-def*
by (*metis list-subset-iff-projection-neutral subsetI*)

let $?\beta'=\beta$ **and** $?\alpha'=[]$
from ‹$\beta\ @\ [c] \in Tr_{ES}$› ‹$V_{\mathcal{V}} = \{\}$›
have $?\beta'\ @\ [c]\ @\ ?\alpha' \in Tr_{ES} \wedge\ ?\alpha'{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}} \wedge\ ?\alpha'{\restriction}C_{\mathcal{V}} = []$
$\wedge\ ?\beta'{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}})$
by (*simp add*: *projection-on-empty-trace projection-to-emptyset-is-empty-trace*)
hence $\exists\ \alpha'\ \beta'.$
$\beta'\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge\ \alpha'{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}} \wedge\ \alpha'{\restriction}C_{\mathcal{V}} = []$
$\wedge\ \beta'{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}})$
by *auto*
}
thus *?thesis*
unfolding *IA-def* **by** *auto*
qed

lemma *Trivially-fulfilled-BSI-V-empty-total-ES-C*:
$[\![V_{\mathcal{V}} = \{\};\ total\ ES\ C_{\mathcal{V}}]\!] \Longrightarrow BSI\ \mathcal{V}\ Tr_{ES}$

proof −
assume $V_{\mathcal{V}} = \{\}$
and *total ES* $C_{\mathcal{V}}$
{
fix α β c
assume β @ $\alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}}{=}[]$
and $c \in C_{\mathcal{V}}$
from ‹β @ $\alpha \in Tr_{ES}$› **have** $\beta \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
with ‹*total ES* $C_{\mathcal{V}}$› **have** β @ $[c] \in Tr_{ES}$
using ‹$c \in C_{\mathcal{V}}$› **unfolding** *total-def* **by** *auto*
moreover
from ‹$V_{\mathcal{V}} = \{\}$› **have** $\alpha{\restriction}V_{\mathcal{V}}{=}[]$
unfolding *projection-def* **by** *auto*
ultimately
have $\exists \alpha'.\ \beta$ @ $[c]$ @ $\alpha' \in Tr_{ES} \wedge \alpha'{\restriction}V_{\mathcal{V}}{=}\alpha{\restriction}V_{\mathcal{V}} \wedge \alpha'{\restriction}C_{\mathcal{V}}{=}[]$
using ‹$\alpha \restriction C_{\mathcal{V}} = []$› **by** (*metis append-Nil2 projection-idempotent*)
}
thus *?thesis*
unfolding *BSI-def* **by** *auto*
qed

lemma *Trivially-fulfilled-I-V-empty-total-ES-C*:
$[\![V_{\mathcal{V}} = \{\};\ \textit{total ES}\ C_{\mathcal{V}}]\!] \Longrightarrow I\ \mathcal{V}\ Tr_{ES}$
proof −
assume $V_{\mathcal{V}} = \{\}$
and *total ES* $C_{\mathcal{V}}$
{
fix α β c
assume $c \in C_{\mathcal{V}}$
and β @ $\alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}}{=}[]$
from ‹β @ $\alpha \in Tr_{ES}$› **have** $\beta \in Tr_{ES}$
using *validES*
unfolding *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def* **by** *auto*
with ‹*total ES* $C_{\mathcal{V}}$› **have** β @ $[c] \in Tr_{ES}$
using ‹$c \in C_{\mathcal{V}}$› **unfolding** *total-def* **by** *auto*
moreover
from ‹$V_{\mathcal{V}} = \{\}$› **have** $\alpha{\restriction}V_{\mathcal{V}}{=}[]$
unfolding *projection-def* **by** *auto*
ultimately
have $\exists \beta'\ \alpha'.$
β' @ $[c]$ @ $\alpha' \in Tr_{ES} \wedge \alpha'{\restriction}V_{\mathcal{V}}{=}\alpha{\restriction}V_{\mathcal{V}} \wedge \alpha'{\restriction}C_{\mathcal{V}}{=}[] \wedge \beta'{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta{\restriction}(V_{\mathcal{V}} \cup C_{\mathcal{V}})$
using ‹$\alpha \restriction C_{\mathcal{V}} = []$› **by** (*metis append-Nil2 projection-idempotent*)
}
thus *?thesis*
unfolding *I-def* **by** *blast*
qed

lemma *Trivially-fulfilled-FCI-Nabla-Υ-empty*:
$[\![\nabla_\Gamma{=}\{\} \vee \Upsilon_\Gamma{=}\{\}]\!] \Longrightarrow FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume $\nabla_\Gamma{=}\{\} \vee \Upsilon_\Gamma{=}\{\}$
thus *?thesis*
proof(*rule disjE*)
assume $\nabla_\Gamma{=}\{\}$ **thus** *?thesis*
by (*simp add*: *FCI-def*)
next
assume $\Upsilon_\Gamma{=}\{\}$ **thus** *?thesis*
by (*simp add*: *FCI-def*)
qed
qed

lemma *Trivially-fulfilled-FCIA-Nabla-Υ-empty*:
$[\![\nabla_\Gamma{=}\{\} \vee \Upsilon_\Gamma{=}\{\}]\!] \Longrightarrow FCIA\ \varrho\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume $\nabla_\Gamma{=}\{\} \vee \Upsilon_\Gamma{=}\{\}$
thus *?thesis*
proof(*rule disjE*)
assume $\nabla_\Gamma{=}\{\}$ **thus** *?thesis*
by (*simp add*: *FCIA-def*)
next
assume $\Upsilon_\Gamma{=}\{\}$ **thus** *?thesis*
by (*simp add*: *FCIA-def*)
qed
qed

lemma *Trivially-fulfilled-FCI-N-subseteq-Δ-and-BSI*:
$[\![N_\mathcal{V} \subseteq \Delta_\Gamma;\ BSI\ \mathcal{V}\ Tr_{ES}]\!] \Longrightarrow FCI\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume $N_\mathcal{V} \subseteq \Delta_\Gamma$
and $BSI\ \mathcal{V}\ Tr_{ES}$
{
fix $\alpha\ \beta\ c\ v$
assume $c \in C_\mathcal{V} \cap \Upsilon_\Gamma$
and $v \in V_\mathcal{V} \cap \nabla_\Gamma$
and $\beta\ @\ [v]\ @\ \alpha \in Tr_{ES}$
and $\alpha{\restriction}C_\mathcal{V} = []$
from ‹$c \in C_\mathcal{V} \cap \Upsilon_\Gamma$› **have** $c \in C_\mathcal{V}$
by *auto*
from ‹$v \in V_\mathcal{V} \cap \nabla_\Gamma$› **have** $v \in V_\mathcal{V}$
by *auto*

let *?α*=$[v]\ @\ \alpha$
from ‹$v \in V_\mathcal{V}$› ‹$\alpha{\restriction}C_\mathcal{V} = []$› **have** *?α*$\restriction C_\mathcal{V}$=[]
using *VIsViewOnE*
unfolding *isViewOn-def V-valid-def VC-disjoint-def projection-def* **by** *auto*
from ‹$\beta\ @\ [v]\ @\ \alpha \in Tr_{ES}$› **have** $\beta\ @$ *?α* $\in Tr_{ES}$
by *auto*

from ‹$BSI\ \mathcal{V}\ Tr_{ES}$›

obtain α'
where β @ $[c]$ @ $\alpha' \in Tr_{ES}$
and $\alpha' \restriction V_{\mathcal{V}} = ([v] @ \alpha) \restriction V_{\mathcal{V}}$
and $\alpha' \restriction C_{\mathcal{V}} = []$
using $\langle c \in C_{\mathcal{V}} \rangle$ $\langle \beta @ ?\alpha \in Tr_{ES} \rangle$ $\langle ?\alpha \restriction C_{\mathcal{V}} = [] \rangle$
unfolding *BSI-def* **by** *blast*

from $\langle v \in V_{\mathcal{V}} \rangle$ $\langle \alpha' \restriction V_{\mathcal{V}} = ([v] @ \alpha) \restriction V_{\mathcal{V}} \rangle$ **have** $\alpha' \restriction V_{\mathcal{V}} = [v] @ \alpha \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
then
obtain δ α''
where $\alpha' = \delta @ [v] @ \alpha''$
and $\delta \restriction V_{\mathcal{V}} = []$
and $\alpha'' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
using *projection-split-first-with-suffix* **by** *fastforce*

from $\langle \alpha' \restriction C_{\mathcal{V}} = [] \rangle$ $\langle \alpha' = \delta @ [v] @ \alpha'' \rangle$ **have** $\delta \restriction C_{\mathcal{V}} = []$
by (*metis append-is-Nil-conv projection-concatenation-commute*)
from $\langle \alpha' \restriction C_{\mathcal{V}} = [] \rangle$ $\langle \alpha' = \delta @ [v] @ \alpha'' \rangle$ **have** $\alpha'' \restriction C_{\mathcal{V}} = []$
by (*metis append-is-Nil-conv projection-concatenation-commute*)

from $\langle \beta @ [c] @ \alpha' \in Tr_{ES} \rangle$ **have** *set* $\alpha' \subseteq E_{ES}$
using *validES*
unfolding *ES-valid-def traces-contain-events-def* **by** *auto*
with $\langle \alpha' = \delta @ [v] @ \alpha'' \rangle$ **have** *set* $\delta \subseteq E_{ES}$
by *auto*
with $\langle \delta \restriction C_{\mathcal{V}} = [] \rangle$ $\langle \delta \restriction V_{\mathcal{V}} = [] \rangle$ $\langle N_{\mathcal{V}} \subseteq \Delta_\Gamma \rangle$
have (*set* δ) $\subseteq$ $(N_{\mathcal{V}} \cap \Delta_\Gamma)$
using *VIsViewOnE projection-empty-implies-absence-of-events*
unfolding *isViewOn-def projection-def* **by** *blast*

let $?\beta = \beta$ **and** $?\delta' = \delta$ **and** $?\alpha' = \alpha''$
from $\langle (set\ \delta) \subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma) \rangle$ $\langle \beta @ [c] @ \alpha' \in Tr_{ES} \rangle$ $\langle \alpha' = \delta @ [v] @ \alpha'' \rangle$
$\langle \alpha'' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \rangle$ $\langle \alpha'' \restriction C_{\mathcal{V}} = [] \rangle$
have (*set* $?\delta'$) $\subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma) \wedge ?\beta @ [c] @ ?\delta' @ [v] @ ?\alpha' \in Tr_{ES} \wedge ?\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge ?\alpha' \restriction C_{\mathcal{V}} = []$
by *auto*
hence $\exists \alpha'''\ \delta''.\ (set\ \delta'') \subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma) \wedge (\beta @ [c] @ \delta'' @ [v] @ \alpha''') \in Tr_{ES}$
$\wedge\ \alpha''' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha''' \restriction C_{\mathcal{V}} = []$
by *auto*
}
thus *?thesis*
unfolding *FCI-def* **by** *auto*
qed

lemma *Trivially-fulfilled-FCIA-N-subseteq-Δ-and-BSIA*:
$[\![N_{\mathcal{V}} \subseteq \Delta_\Gamma;\ BSIA\ \varrho\ \mathcal{V}\ Tr_{ES}]\!] \Longrightarrow FCIA\ \varrho\ \Gamma\ \mathcal{V}\ Tr_{ES}$
proof −
assume $N_{\mathcal{V}} \subseteq \Delta_\Gamma$
and *BSIA* ϱ $\mathcal{V}$ Tr_{ES}
{
fix α β c v
assume $c \in C_{\mathcal{V}} \cap \Upsilon_\Gamma$

and $v \in V_{\mathcal{V}} \cap \nabla_\Gamma$
and β @ $[v]$ @ $\alpha \in Tr_{ES}$
and $\alpha{\restriction}C_{\mathcal{V}} = []$
and *Adm* $\mathcal{V}$ ϱ Tr_{ES} β c
from ‹$c \in C_{\mathcal{V}} \cap \Upsilon_\Gamma$› **have** $c \in C_{\mathcal{V}}$
by *auto*
from ‹$v \in V_{\mathcal{V}} \cap \nabla_\Gamma$› **have** $v \in V_{\mathcal{V}}$
by *auto*

let *?α*=$[v]$ @ α
from ‹$v \in V_{\mathcal{V}}$› ‹$\alpha{\restriction}C_{\mathcal{V}} = []$› **have** *?α*$\restriction C_{\mathcal{V}}$=[]
using *VIsViewOnE*
unfolding *isViewOn-def V-valid-def VC-disjoint-def projection-def* **by** *auto*
from ‹β @ $[v]$ @ $\alpha \in Tr_{ES}$› **have** β @ *?α* $\in Tr_{ES}$
by *auto*

from ‹*BSIA* ϱ $\mathcal{V}$ Tr_{ES}›
obtain α'
where β @ $[c]$ @ $\alpha' \in Tr_{ES}$
and $\alpha'{\restriction}V_{\mathcal{V}} = ([v]$ @ $\alpha){\restriction}V_{\mathcal{V}}$
and $\alpha'{\restriction}C_{\mathcal{V}} = []$
using ‹$c \in C_{\mathcal{V}}$› ‹β @ *?α* $\in Tr_{ES}$› ‹*?α*$\restriction C_{\mathcal{V}} = []$› ‹*Adm* $\mathcal{V}$ ϱ Tr_{ES} β c›
unfolding *BSIA-def* **by** *blast*

from‹$v \in V_{\mathcal{V}}$› ‹$\alpha'{\restriction}V_{\mathcal{V}} = ([v]$ @ $\alpha){\restriction}V_{\mathcal{V}}$› **have** $\alpha'{\restriction}V_{\mathcal{V}} = [v]$ @ $\alpha{\restriction}V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
then
obtain δ α''
where α'=δ @ $[v]$ @ α''
and $\delta{\restriction}V_{\mathcal{V}} = []$
and $\alpha''{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}}$
using *projection-split-first-with-suffix* **by** *fastforce*

from ‹$\alpha'{\restriction}C_{\mathcal{V}} = []$› ‹$\alpha'$=$\delta$ @ $[v]$ @ α''› **have** $\delta{\restriction}C_{\mathcal{V}}$=[]
by (*metis append-is-Nil-conv projection-concatenation-commute*)
from ‹$\alpha'{\restriction}C_{\mathcal{V}} = []$› ‹$\alpha'$=$\delta$ @ $[v]$ @ α''› **have** $\alpha''{\restriction}C_{\mathcal{V}}$=[]
by (*metis append-is-Nil-conv projection-concatenation-commute*)

from ‹β @ $[c]$ @ $\alpha' \in Tr_{ES}$› **have** *set* $\alpha' \subseteq E_{ES}$
using *validES*
unfolding *ES-valid-def traces-contain-events-def* **by** *auto*
with ‹α'=δ @ $[v]$ @ α''› **have** *set* $\delta \subseteq E_{ES}$
by *auto*
with ‹$\delta{\restriction}C_{\mathcal{V}}$=[]› ‹$\delta{\restriction}V_{\mathcal{V}} = []$› ‹$N_{\mathcal{V}} \subseteq \Delta_\Gamma$›
have (*set* δ) $\subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma)$ **using** *VIsViewOnE projection-empty-implies-absence-of-events*
unfolding *isViewOn-def projection-def* **by** *blast*

let *?β*=β **and** *?δ'*=δ **and** *?α'*=α''
from ‹(*set* δ) $\subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma)$› ‹$\beta$ @ $[c]$ @ $\alpha' \in Tr_{ES}$› ‹α'=δ @ $[v]$ @ α''›
‹$\alpha''{\restriction}V_{\mathcal{V}} = \alpha{\restriction}V_{\mathcal{V}}$› ‹$\alpha''{\restriction}C_{\mathcal{V}}$=[]›
have (*set* *?δ'*)$\subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma) \wedge$ *?β* @ $[c]$ @ *?δ'* @ $[v]$ @ *?α'* $\in Tr_{ES} \wedge$ *?α'*$\restriction V_{\mathcal{V}}$=$\alpha{\restriction}V_{\mathcal{V}} \wedge$ *?α'*$\restriction C_{\mathcal{V}}$=[]
by *auto*

hence $\exists \alpha''' \, \delta''.\ (set\ \delta'') \subseteq (N_{\mathcal{V}} \cap \Delta_\Gamma) \wedge (\beta\ @\ [c]\ @\ \delta''\ @\ [v]\ @\ \alpha''') \in Tr_{ES}$
$\wedge\ \alpha'''\restriction V_{\mathcal{V}} = \alpha\restriction V_{\mathcal{V}} \wedge \alpha'''\restriction C_{\mathcal{V}} = []$
by *auto*
}
thus *?thesis*
unfolding *FCIA-def* **by** *auto*
qed

end

context *BSPTaxonomyDifferentViewsFirstDim*
begin

lemma *R-implies-R-for-modified-view*:
$R\ \mathcal{V}_1\ Tr_{ES} \Longrightarrow R\ \mathcal{V}_2\ Tr_{ES}$
proof −
assume *R-*$\mathcal{V}_1$: $R\ \mathcal{V}_1\ Tr_{ES}$
{
fix τ
assume $\tau \in Tr_{ES}$
with *R-*$\mathcal{V}_1$ **have** $\exists\ \tau' \in Tr_{ES}.\ \tau'\restriction C_{\mathcal{V}_1} = [] \wedge \tau'\restriction V_{\mathcal{V}_1} = \tau\restriction V_{\mathcal{V}_1}$
unfolding *R-def* **by** *auto*
hence $\exists\ \tau' \in Tr_{ES}.\ \tau'\restriction C_{\mathcal{V}_2} = [] \wedge \tau'\restriction V_{\mathcal{V}_2} = \tau\restriction V_{\mathcal{V}_2}$
using *V2-subset-V1 C2-subset-C1 non-empty-projection-on-subset projection-on-subset* **by** *blast*
}
thus *?thesis*
unfolding *R-def* **by** *auto*
qed

lemma *BSD-implies-BSD-for-modified-view*:
$BSD\ \mathcal{V}_1\ Tr_{ES} \Longrightarrow BSD\ \mathcal{V}_2\ Tr_{ES}$
proof−
assume *BSD-*$\mathcal{V}_1$: $BSD\ \mathcal{V}_1\ Tr_{ES}$
{
fix $\alpha\ \beta\ c\ n$
assume *c-in-*C_2: $c \in C_{\mathcal{V}_2}$
from *C2-subset-C1 c-in-C2* **have** *c-in-*C_1: $c \in C_{\mathcal{V}_1}$
by *auto*
have $[\![\beta\ @\ [c]\ @\ \alpha \in Tr_{ES};\ \alpha\restriction C_{\mathcal{V}_2}=[];\ n= length(\alpha\restriction C_{\mathcal{V}_1})]\!]$
$\Longrightarrow \exists\ \alpha'.\ \beta\ @\ \alpha' \in Tr_{ES} \wedge \alpha'\restriction V_{\mathcal{V}_2} = \alpha\restriction V_{\mathcal{V}_2}\ \wedge \alpha'\restriction C_{\mathcal{V}_2} = []$
proof(*induct n arbitrary*: α)
case *0*
from *0.prems(3)* **have** $\alpha\restriction C_{\mathcal{V}_1} = []$ **by** *auto*
with *c-in-*C_1 *0.prems(1)*
have $\exists\ \alpha'.\ \beta\ @\ \alpha' \in Tr_{ES} \wedge \alpha'\restriction V_{\mathcal{V}_1} = \alpha\restriction V_{\mathcal{V}_1} \wedge \alpha'\restriction C_{\mathcal{V}_1} =[]$
using *BSD-*$\mathcal{V}_1$ **unfolding** *BSD-def* **by** *auto*
then
obtain α' **where** $\beta\ @\ \alpha' \in Tr_{ES}$
and $\alpha'\restriction V_{\mathcal{V}_1} = \alpha\restriction V_{\mathcal{V}_1}$
and $\alpha'\restriction C_{\mathcal{V}_1} =[]$
by *auto*
from *V2-subset-V1* ‹$\alpha'\restriction V_{\mathcal{V}_1} = \alpha\restriction V_{\mathcal{V}_1}$› **have** $\alpha'\restriction V_{\mathcal{V}_2} = \alpha\restriction V_{\mathcal{V}_2}$

using *non-empty-projection-on-subset* **by** *blast*
moreover
from ‹$\alpha' \restriction C_{\mathcal{V}_1} = []$› *C2-subset-C1* **have** $\alpha' \restriction V_{\mathcal{V}_2} = []$
using *projection-on-subset* **by** *auto*
ultimately
show *?case*
using ‹$\beta \mathbin{@} \alpha' \in Tr_{ES}$› **by** *auto*
next
case (*Suc n*)
from *Suc.prems*(*3*) *projection-split-last*[*OF Suc.prems*(*3*)]
obtain $\gamma_1\ \gamma_2\ c_1$ **where** c_1-*in*-C_1: $c_1 \in C_{\mathcal{V}_1}$
and $\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$
and $\gamma_2 \restriction C_{\mathcal{V}_1} = []$
and $n = length((\gamma_1 \mathbin{@} \gamma_2) \restriction V_{\mathcal{V}_1})$
by *auto*
from *Suc.prems*(*2*) ‹$\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$› **have** $\gamma_1 \restriction V_{\mathcal{V}_2} = []$
by (*simp add*: *projection-concatenation-commute*)
from *Suc.prems*(*1*) ‹$\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$›
obtain β' **where** $\beta' = \beta \mathbin{@} [c] \mathbin{@} \gamma_1$
and $\beta' \mathbin{@} [c_1] \mathbin{@} \gamma_2 \in Tr_{ES}$
by *auto*
from ‹$\beta' \mathbin{@} [c_1] \mathbin{@} \gamma_2 \in Tr_{ES}$› ‹$\gamma_2 \restriction C_{\mathcal{V}_1} = []$› ‹$c_1 \in C_{\mathcal{V}_1}$›
obtain γ_2' **where** $\beta' \mathbin{@} \gamma_2' \in Tr_{ES}$
and $\gamma_2' \restriction V_{\mathcal{V}_1} = \gamma_2 \restriction V_{\mathcal{V}_1}$
and $\gamma_2' \restriction C_{\mathcal{V}_1} = []$
using *BSD-*$\mathcal{V}_1$ **unfolding** *BSD-def* **by** *auto*
from ‹$\beta' = \beta \mathbin{@} [c] \mathbin{@} \gamma_1$› ‹$\beta' \mathbin{@} \gamma_2' \in Tr_{ES}$› **have** $\beta \mathbin{@} [c] \mathbin{@} \gamma_1 \mathbin{@} \gamma_2' \in Tr_{ES}$
by *auto*
moreover
from ‹$\gamma_1 \restriction C_{\mathcal{V}_2} = []$› ‹$\gamma_2' \restriction C_{\mathcal{V}_1} = []$› *C2-subset-C1* **have** $(\gamma_1 \mathbin{@} \gamma_2') \restriction C_{\mathcal{V}_2} = []$
by (*metis append-Nil projection-concatenation-commute projection-on-subset*)
moreover
from ‹$n = length((\gamma_1 \mathbin{@} \gamma_2) \restriction V_{\mathcal{V}_1})$› ‹$\gamma_2 \restriction C_{\mathcal{V}_1} = []$› ‹$\gamma_2' \restriction C_{\mathcal{V}_1} = []$›
have $n = length((\gamma_1 \mathbin{@} \gamma_2') \restriction V_{\mathcal{V}_1})$
by (*simp add*: *projection-concatenation-commute*)
ultimately
have *witness*: $\exists\ \alpha'.\ \beta \mathbin{@} \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = (\gamma_1 \mathbin{@} \gamma_2') \restriction V_{\mathcal{V}_2} \wedge \alpha' \restriction C_{\mathcal{V}_2} = []$
using *Suc.hyps* **by** *auto*

from $\mathcal{V}_1$ *IsViewOnE* $\mathcal{V}_2$ *IsViewOnE V2-subset-V1 C2-subset-C1* c_1-*in*-C_1 **have** $c_1 \notin V_{\mathcal{V}_2}$
unfolding *isViewOn-def V-valid-def VC-disjoint-def* **by** *auto*
with ‹$\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$› **have** $\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1 \mathbin{@} \gamma_2) \restriction V_{\mathcal{V}_2}$
unfolding *projection-def* **by** *auto*
hence $\alpha \restriction V_{\mathcal{V}_2} = \gamma_1 \restriction V_{\mathcal{V}_2} \mathbin{@} \gamma_2 \restriction V_{\mathcal{V}_2}$
using *projection-concatenation-commute* **by** *auto*
with *V2-subset-V1* ‹$\gamma_2' \restriction V_{\mathcal{V}_1} = \gamma_2 \restriction V_{\mathcal{V}_1}$›
have $\gamma_1 \restriction V_{\mathcal{V}_2} \mathbin{@} \gamma_2 \restriction V_{\mathcal{V}_2} = \gamma_1 \restriction V_{\mathcal{V}_2} \mathbin{@} \gamma_2' \restriction V_{\mathcal{V}_2}$
using *non-empty-projection-on-subset* **by** *metis*
with ‹$\alpha \restriction V_{\mathcal{V}_2} = \gamma_1 \restriction V_{\mathcal{V}_2} \mathbin{@} \gamma_2 \restriction V_{\mathcal{V}_2}$› **have** $\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1 \mathbin{@} \gamma_2') \restriction V_{\mathcal{V}_2}$
by (*simp add*: *projection-concatenation-commute*)

from *witness* ‹$\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1 \mathbin{@} \gamma_2') \restriction V_{\mathcal{V}_2}$›

```
      show ?case
        by auto
    qed
 }
  thus ?thesis
    unfolding BSD-def by auto
qed

lemma D-implies-D-for-modified-view:
D V₁ Tr_ES ⟹ D V₂ Tr_ES
proof−
  assume D-V₁: D V₁ Tr_ES
   from V2-subset-V1 C2-subset-C1
    have V₂-union-C₂-subset-V₁-union-C₁: V_V₂ ∪ C_V₂ ⊆ V_V₁ ∪ C_V₁ by auto
  {
    fix α β c n
    assume  c-in-C₂: c ∈ C_V₂
    from C2-subset-C1  c-in-C₂  have c-in-C₁: c ∈ C_V₁
      by auto
    have ⟦β @ [c] @ α ∈ Tr_ES; α ↾ C_V₂=[]; n= length(α ↾ C_V₁)⟧
          ⟹ ∃ α' β'.
               β' @ α' ∈ Tr_ES  ∧ α'↾ V_V₂ = α ↾V_V₂  ∧ α' ↾C_V₂ = []
                 ∧ β' ↾(V_V₂ ∪ C_V₂) = β ↾(V_V₂ ∪ C_V₂)
    proof(induct n arbitrary: α β )
      case 0
        from 0.prems(3) have α ↾ C_V₁ = [] by auto
        with c-in-C₁ 0.prems(1)
        have ∃ α' β'.
              β' @ α' ∈ Tr_ES  ∧ α' ↾ V_V₁ = α ↾ V_V₁ ∧ α' ↾C_V₁ =[]
                ∧ β' ↾(V_V₁ ∪ C_V₁) = β ↾(V_V₁ ∪ C_V₁)
          using D-V₁ unfolding D-def by fastforce
        then
        obtain β' α' where β' @ α' ∈ Tr_ES
                       and α' ↾ V_V₁ = α ↾ V_V₁
                       and α' ↾C_V₁ =[]
                       and β' ↾(V_V₁ ∪ C_V₁) = β ↾(V_V₁ ∪ C_V₁)
          by auto
        from V2-subset-V1 ‹α' ↾ V_V₁ = α ↾ V_V₁› have α'↾ V_V₂ = α ↾V_V₂
          using non-empty-projection-on-subset by blast
        moreover
        from ‹α' ↾C_V₁ =[]›  C2-subset-C1 have α' ↾ C_V₂ = []
          using projection-on-subset by auto
        moreover
        from ‹β' ↾(V_V₁ ∪ C_V₁) = β ↾(V_V₁ ∪ C_V₁)›  V₂-union-C₂-subset-V₁-union-C₁
        have β' ↾(V_V₂ ∪ C_V₂) = β ↾(V_V₂ ∪ C_V₂)
          using non-empty-projection-on-subset by blast
        ultimately
        show ?case
          using ‹β' @ α' ∈ Tr_ES› by auto
      next
      case (Suc n)
        from Suc.prems(3) projection-split-last[OF Suc.prems(3)]
```

obtain $\gamma_1\ \gamma_2\ c_1$ **where** c_1-*in*-C_1: $c_1 \in C_{\mathcal{V}_1}$
  and $\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$
  and $\gamma_2 \restriction C_{\mathcal{V}_1} = []$
  and $n = \mathit{length}((\gamma_1 \mathbin{@} \gamma_2) \restriction C_{\mathcal{V}_1})$
 by *auto*
from *Suc.prems*(2) ‹$\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$› **have** $\gamma_1 \restriction C_{\mathcal{V}_2} = []$
 by (*simp add*: *projection-concatenation-commute*)
from *Suc.prems*(1) ‹$\alpha = \gamma_1 \mathbin{@} [c_1] \mathbin{@} \gamma_2$›
obtain β' **where** β'=$\beta \mathbin{@} [c] \mathbin{@} \gamma_1$
  and $\beta' \mathbin{@} [c_1] \mathbin{@} \gamma_2 \in \mathit{Tr}_{ES}$
 by *auto*
from ‹$\beta' \mathbin{@} [c_1] \mathbin{@} \gamma_2 \in \mathit{Tr}_{ES}$› ‹$\gamma_2 \restriction C_{\mathcal{V}_1} = []$› ‹$c_1 \in C_{\mathcal{V}_1}$›
obtain $\gamma_2{}'$ β'' **where** $\beta'' \mathbin{@} \gamma_2{}' \in \mathit{Tr}_{ES}$
  and $\gamma_2{}' \restriction V_{\mathcal{V}_1} = \gamma_2 \restriction V_{\mathcal{V}_1}$
  and $\gamma_2{}' \restriction C_{\mathcal{V}_1} = []$
  and $\beta'' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
 using *D*-$\mathcal{V}_1$ **unfolding** *D-def* **by** *force*

from *c-in-C*$_1$ **have** $c \in V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}$
 by *auto*
moreover
from ‹$\beta'' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$› ‹$\beta'$=$\beta \mathbin{@} [c] \mathbin{@} \gamma_1$›
have $\beta'' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = (\beta \mathbin{@} [c] \mathbin{@} \gamma_1) \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
 by *auto*
ultimately
have $\exists\ \beta'''\ \gamma_1{}'.\ \beta''$=$\beta'''$@ $[c] \mathbin{@} \gamma_1{}'$
    $\wedge\ \beta'''\ \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
    $\wedge\ \gamma_1{}' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \gamma_1 \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
 using *projection-split-arbitrary-element* **by** *fast*
then
obtain $\beta'''\ \gamma_1{}'$ **where** β''= $\beta''' \mathbin{@} [c] \mathbin{@} \gamma_1{}'$
  and $\beta'''\ \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
  and $\gamma_1{}' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \gamma_1 \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
 using *projection-split-arbitrary-element* **by** *auto*

from ‹$\beta'' \mathbin{@} \gamma_2{}' \in \mathit{Tr}_{ES}$› *this*(1)
have $\beta''' \mathbin{@} [c] \mathbin{@} \gamma_1{}' \mathbin{@} \gamma_2{}' \in \mathit{Tr}_{ES}$
 by *simp*

from ‹$\gamma_2{}' \restriction C_{\mathcal{V}_1} = []$› **have** $\gamma_2{}' \restriction C_{\mathcal{V}_2}$=[]
 using *C2-subset-C1 projection-on-subset* **by** *auto*
moreover
from ‹$\gamma_1 \restriction C_{\mathcal{V}_2} = []$› ‹$\gamma_1{}' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \gamma_1 \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$›
have $\gamma_1{}' \restriction C_{\mathcal{V}_2} = []$ **using** *C2-subset-C1 V2-subset-V1*
  by (*metis non-empty-projection-on-subset projection-subset-eq-from-superset-eq sup-commute*)

ultimately
have $(\gamma_1{}' \mathbin{@} \gamma_2{}') \restriction C_{\mathcal{V}_2} = []$
 by (*simp add*: *projection-concatenation-commute*)

from ‹$\gamma_1{}' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \gamma_1 \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$› **have** $\gamma_1{}' \restriction C_{\mathcal{V}_1} = \gamma_1 \restriction C_{\mathcal{V}_1}$
 using *projection-subset-eq-from-superset-eq sup-commute* **by** *metis*

hence $length(\gamma_1{}' \restriction C_{\mathcal{V}_1}) = length(\gamma_1 \restriction C_{\mathcal{V}_1})$ **by** *simp*
moreover
from ‹$\gamma_2 \restriction C_{\mathcal{V}_1} = []$› ‹$\gamma_2{}' \restriction C_{\mathcal{V}_1} = []$› **have** $length(\gamma_2{}' \restriction C_{\mathcal{V}_1}) = length(\gamma_2 \restriction C_{\mathcal{V}_1})$
by *simp*
ultimately
have $n = length((\gamma_1{}' @ \gamma_2{}') \restriction C_{\mathcal{V}_1})$
by (*simp add*: ‹$n = length\ ((\gamma_1 @ \gamma_2) \restriction C_{\mathcal{V}_1})$› *projection-concatenation-commute*)

from ‹$\beta''' @ [c] @ \gamma_1{}' @ \gamma_2{}' \in Tr_{ES}$› ‹$(\gamma_1{}' @ \gamma_2{}') \restriction C_{\mathcal{V}_2} = []$› ‹$n = length((\gamma_1{}' @ \gamma_2{}') \restriction C_{\mathcal{V}_1})$›
have *witness*:
$\exists \alpha'\ \beta'.\ \beta' @ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = (\gamma_1{}' @ \gamma_2{}') \restriction V_{\mathcal{V}_2}$
$\wedge\ \alpha' \restriction C_{\mathcal{V}_2} = [] \wedge \beta' \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2}) = \beta''' \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2})$
using *Suc.hyps*[*OF* ‹$\beta''' @ [c] @ \gamma_1{}' @ \gamma_2{}' \in Tr_{ES}$›] **by** *simp*

from *V2-union-C2-subset-V1-union-C1* ‹$\beta''' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$›
have $\beta''' \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2}) = \beta \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2})$
using *non-empty-projection-on-subset* **by** *blast*

from $\mathcal{V}_1$*IsViewOnE* $\mathcal{V}_2$*IsViewOnE V2-subset-V1 C2-subset-C1* c_1*-in-*C_1 **have** $c_1 \notin V_{\mathcal{V}_2}$
unfolding *isViewOn-def V-valid-def VC-disjoint-def* **by** *auto*
with ‹$\alpha = \gamma_1 @ [c_1] @ \gamma_2$› **have** $\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1 @ \gamma_2) \restriction V_{\mathcal{V}_2}$
unfolding *projection-def* **by** *auto*
moreover
from *V2-subset-V1* ‹$\gamma_2{}' \restriction V_{\mathcal{V}_1} = \gamma_2 \restriction V_{\mathcal{V}_1}$› **have** $\gamma_2{}' \restriction V_{\mathcal{V}_2} = \gamma_2 \restriction V_{\mathcal{V}_2}$
using *V2-subset-V1* **by** (*metis projection-subset-eq-from-superset-eq subset-Un-eq*)
moreover
from ‹$\gamma_1{}' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \gamma_1 \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$› **have** $\gamma_1{}' \restriction V_{\mathcal{V}_2} = \gamma_1 \restriction V_{\mathcal{V}_2}$
using *V2-subset-V1* **by** (*metis projection-subset-eq-from-superset-eq subset-Un-eq*)
ultimately
have $\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1{}' @ \gamma_2{}') \restriction V_{\mathcal{V}_2}$ **using** ‹$\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1 @ \gamma_2) \restriction V_{\mathcal{V}_2}$›
by (*simp add*: *projection-concatenation-commute*)

from ‹$\beta''' \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2}) = \beta \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2})$› ‹$\alpha \restriction V_{\mathcal{V}_2} = (\gamma_1{}' @ \gamma_2{}') \restriction V_{\mathcal{V}_2}$›
show *?case*
using *witness* **by** *simp*
qed
}
thus *?thesis*
unfolding *D-def* **by** *auto*
qed
end

context *BSPTaxonomyDifferentViewsSecondDim*
begin

lemma *FCD-implies-FCD-for-modified-view-gamma*:
⟦*FCD* Γ_1 $\mathcal{V}_1$ Tr_{ES};
$V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$; $N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$; $C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$ ⟧
$\Longrightarrow$ *FCD* Γ_2 $\mathcal{V}_2$ Tr_{ES}
proof −

assume *FCD* Γ_1 $\mathcal{V}_1$ Tr_{ES}
  and $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$
  and $N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$
  and $C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$
{
  fix α β v c
  assume $c \in C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2}$
    and $v \in V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2}$
    and β @ [c,v] @ $\alpha \in Tr_{ES}$
    and $\alpha \restriction C_{\mathcal{V}_2} = []$

  from ‹$c \in C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2}$› ‹$C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$› **have** $c \in C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$
    by *auto*
  moreover
  from ‹$v \in V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2}$› ‹$V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$› **have** $v \in V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$
    by *auto*
  moreover
  from *C2-equals-C1* ‹$\alpha \restriction C_{\mathcal{V}_2} = []$› **have** $\alpha \restriction C_{\mathcal{V}_1} = []$
    by *auto*
  ultimately
  obtain α' δ' **where** (*set* δ') $\subseteq (N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1})$
      and β @ δ' @ [v] @ $\alpha' \in Tr_{ES}$
      and $\alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1}$
      and $\alpha' \restriction C_{\mathcal{V}_1} = []$
    using ‹β @ [c,v] @ $\alpha \in Tr_{ES}$› ‹*FCD* Γ_1 $\mathcal{V}_1$ Tr_{ES}› **unfolding** *FCD-def* **by** *blast*

  from ‹(*set* δ') $\subseteq (N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1})$› ‹$N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$›
  have (*set* δ') $\subseteq (N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2})$
    by *auto*
  moreover
  from ‹$\alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1}$› *V2-subset-V1* **have** $\alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2}$
  using *non-empty-projection-on-subset* **by** *blast*
  moreover
  from *C2-equals-C1* ‹$\alpha' \restriction C_{\mathcal{V}_1} = []$› **have** $\alpha' \restriction C_{\mathcal{V}_2} = []$
    by *auto*
  ultimately
  have $\exists\ \delta'\ \alpha'.$ (*set* δ') $\subseteq (N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2})$
      $\wedge\ \beta$ @ δ'@ [v] @ $\alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2} \wedge \alpha' \restriction C_{\mathcal{V}_2} = []$
    using ‹β @ δ' @ [v] @ $\alpha' \in Tr_{ES}$› **by** *auto*
}
thus *?thesis*
  unfolding *FCD-def* **by** *blast*
qed

lemma *SI-implies-SI-for-modified-view* :
SI $\mathcal{V}_1$ $Tr_{ES} \Longrightarrow$ *SI* $\mathcal{V}_2$ Tr_{ES}
proof −
  assume *SI*: *SI* $\mathcal{V}_1$ Tr_{ES}
  {
    fix α β c
    assume $c \in C_{\mathcal{V}_2}$

```
    and  β @ α ∈ Tr_ES
    and  alpha-C₂-empty: α ↾ C_V₂ = []
   moreover
   with  C2-equals-C1 have c ∈ C_V₁
    by auto
   moreover
   from  alpha-C₂-empty C2-equals-C1 have α ↾ C_V₁ = []
    by auto
   ultimately
   have β @ (c # α) ∈ Tr_ES
    using SI unfolding SI-def by auto
  }
  thus ?thesis
   unfolding SI-def by auto
qed



lemma BSI-implies-BSI-for-modified-view :
BSI V₁ Tr_ES ⟹ BSI V₂ Tr_ES
proof −
  assume  BSI: BSI V₁ Tr_ES
  {
   fix α β c
   assume c ∈ C_V₂
    and  β @ α ∈ Tr_ES
    and  alpha-C₂-empty: α ↾ C_V₂ = []
   moreover
   with  C2-equals-C1 have c ∈ C_V₁
    by auto
   moreover
   from  alpha-C₂-empty C2-equals-C1 have α ↾ C_V₁ = []
    by auto
   ultimately
   have ∃ α'. β @ [c] @ α' ∈ Tr_ES ∧ α' ↾ V_V₁ = α ↾ V_V₁ ∧ α' ↾ C_V₁ = []
    using BSI unfolding BSI-def by auto
   with V2-subset-V1  C2-equals-C1
   have ∃ α'. β @ [c] @ α' ∈ Tr_ES ∧ α' ↾ V_V₂ = α ↾ V_V₂ ∧ α' ↾ C_V₂ = []
    using non-empty-projection-on-subset by metis
  }
  thus ?thesis
   unfolding BSI-def by auto
qed


lemma I-implies-I-for-modified-view :
I V₁ Tr_ES ⟹ I V₂ Tr_ES
proof −
  assume  I: I V₁ Tr_ES
  from V2-subset-V1 C2-equals-C1 have V₂-union-C₂-subset-V₁-union-C₁: V_V₂ ∪ C_V₂ ⊆ V_V₁ ∪ C_V₁
   by auto
  {
```

```
    fix α β c
    assume c ∈ C_{V_2}
      and β @ α ∈ Tr_ES
      and alpha-C_2-empty: α ↾ C_{V_2} = []
    moreover
    with C2-equals-C1 have c ∈ C_{V_1}
      by auto
    moreover
    from alpha-C_2-empty C2-equals-C1 have α ↾ C_{V_1} = []
      by auto
    ultimately
    have ∃ α' β'.
          β' @ [c] @ α' ∈ Tr_ES ∧ α' ↾ V_{V_1} = α ↾ V_{V_1} ∧ α' ↾ C_{V_1} = []
            ∧ β' ↾ (V_{V_1} ∪ C_{V_1}) = β ↾ (V_{V_1} ∪ C_{V_1})
      using I unfolding I-def by auto
    with V_2-union-C_2-subset-V_1-union-C_1 V2-subset-V1 C2-equals-C1
    have ∃ α' β'.
            β' @ [c] @ α' ∈ Tr_ES ∧ α' ↾ V_{V_2} = α ↾ V_{V_2} ∧ α' ↾ C_{V_2} = []
              ∧ β' ↾ (V_{V_2} ∪ C_{V_2}) = β ↾ (V_{V_2} ∪ C_{V_2})
      using non-empty-projection-on-subset by metis
  }
  thus ?thesis
    unfolding I-def by auto
qed


lemma SIA-implies-SIA-for-modified-view :
⟦SIA ϱ_1 V_1 Tr_ES; ϱ_2(V_2) ⊇ ϱ_1(V_1) ⟧ ⟹ SIA ϱ_2 V_2 Tr_ES
proof −
  assume SIA: SIA ϱ_1 V_1 Tr_ES
    and ϱ_2-supseteq-ϱ_1: ϱ_2(V_2) ⊇ ϱ_1(V_1)
  {
    fix α β c
    assume c ∈ C_{V_2}
      and β @ α ∈ Tr_ES
      and alpha-C_2-empty: α ↾ C_{V_2} = []
      and admissible-c-ϱ_2-V_2:Adm V_2 ϱ_2 Tr_ES β c
    moreover
    with C2-equals-C1 have c ∈ C_{V_1}
      by auto
    moreover
    from alpha-C_2-empty C2-equals-C1 have α ↾ C_{V_1} = []
      by auto
    moreover
    from ϱ_2-supseteq-ϱ_1 admissible-c-ϱ_2-V_2 have Adm V_1 ϱ_1 Tr_ES β c
      by (simp add: Adm-implies-Adm-for-modified-rho)
    ultimately
    have β @ (c # α) ∈ Tr_ES
      using SIA unfolding SIA-def by auto
  }
  thus ?thesis
    unfolding SIA-def by auto
```

qed

lemma *BSIA-implies-BSIA-for-modified-view* :
$[\![BSIA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)\ ]\!] \Longrightarrow BSIA\ \varrho_2\ \mathcal{V}_2\ Tr_{ES}$
proof −
assume *BSIA*: $BSIA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES}$
and *ϱ₂-supseteq-ϱ₁*: $\varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)$
from *V2-subset-V1 C2-equals-C1*
have *V₂-union-C₂-subset-V₁-union-C₁*: $V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}$
by *auto*
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}_2}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and *alpha-C₂-empty*: $\alpha \restriction C_{\mathcal{V}_2} = []$
and *admissible-c-ϱ₂-𝒱₂:Adm* $\mathcal{V}_2\ \varrho_2\ Tr_{ES}\ \beta\ c$
moreover
with *C2-equals-C1* **have** $c \in C_{\mathcal{V}_1}$
by *auto*
moreover
from *alpha-C₂-empty C2-equals-C1* **have** $\alpha \restriction C_{\mathcal{V}_1} = []$
by *auto*
moreover
from *ϱ₂-supseteq-ϱ₁ admissible-c-ϱ₂-𝒱₂* **have** $Adm\ \mathcal{V}_1\ \varrho_1\ Tr_{ES}\ \beta\ c$
by (*simp add*: *Adm-implies-Adm-for-modified-rho*)
ultimately
have $\exists\ \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1} \wedge \alpha' \restriction C_{\mathcal{V}_1} = []$
using *BSIA* **unfolding** *BSIA-def* **by** *auto*
with *V2-subset-V1 C2-equals-C1*
have $\exists\ \alpha'.\ \beta\ @\ [c]\ @\ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2} \wedge \alpha' \restriction C_{\mathcal{V}_2} = []$
using *non-empty-projection-on-subset* **by** *metis*
}
thus *?thesis*
unfolding *BSIA-def* **by** *auto*
qed

lemma *IA-implies-IA-for-modified-view* :
$[\![IA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)\ ]\!] \Longrightarrow IA\ \varrho_2\ \mathcal{V}_2\ Tr_{ES}$
proof −
assume *IA*: $IA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES}$
and *ϱ₂-supseteq-ϱ₁*: $\varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)$
{
fix $\alpha\ \beta\ c$
assume $c \in C_{\mathcal{V}_2}$
and $\beta\ @\ \alpha \in Tr_{ES}$
and *alpha-C₂-empty*: $\alpha \restriction C_{\mathcal{V}_2} = []$
and *admissible-c-ϱ₂-𝒱₂:Adm* $\mathcal{V}_2\ \varrho_2\ Tr_{ES}\ \beta\ c$
moreover
with *C2-equals-C1* **have** $c \in C_{\mathcal{V}_1}$

by *auto*
moreover
from *alpha-C_2-empty C2-equals-C1* **have** $\alpha \restriction C_{\mathcal{V}_1} = []$
by *auto*
moreover
from *ϱ_2-supseteq-ϱ_1 admissible-c-ϱ_2-$\mathcal{V}_2$* **have** *Adm* $\mathcal{V}_1$ ϱ_1 Tr_{ES} β c
by (*simp add*: *Adm-implies-Adm-for-modified-rho*)
ultimately
have $\exists\ \alpha'\ \beta'.\ \beta' @ [c] @ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1} \wedge \alpha' \restriction C_{\mathcal{V}_1} = [] \wedge \beta' \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1}) = \beta \restriction (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
using *IA* **unfolding** *IA-def* **by** *auto*
moreover
from *V2-subset-V1 C2-equals-C1* **have** $(V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2}) \subseteq (V_{\mathcal{V}_1} \cup C_{\mathcal{V}_1})$
by *auto*
ultimately
have $\exists\ \alpha'\ \beta'.\ \beta' @ [c] @ \alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2} \wedge \alpha' \restriction C_{\mathcal{V}_2} = [] \wedge \beta' \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2}) = \beta \restriction (V_{\mathcal{V}_2} \cup C_{\mathcal{V}_2})$
using *V2-subset-V1 C2-equals-C1 non-empty-projection-on-subset* **by** *metis*
}
thus *?thesis*
unfolding *IA-def* **by** *auto*
qed

lemma *FCI-implies-FCI-for-modified-view-gamma*:
$[\![FCI\ \Gamma_1\ \mathcal{V}_1\ Tr_{ES};$
$V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1};\ N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1};\ C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}]\!]$
$\Longrightarrow FCI\ \Gamma_2\ \mathcal{V}_2\ Tr_{ES}$
proof −
assume *FCI* Γ_1 $\mathcal{V}_1$ Tr_{ES}
and $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$
and $N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$
and $C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$
{
fix $\alpha\ \beta\ v\ c$
assume $c \in C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2}$
and $v \in V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2}$
and $\beta @ [v] @ \alpha \in Tr_{ES}$
and $\alpha \restriction C_{\mathcal{V}_2} = []$

from $\langle c \in C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \rangle$ $\langle C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1} \rangle$ **have** $c \in C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$
by *auto*
moreover
from $\langle v \in V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \rangle$ $\langle V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1} \rangle$ **have** $v \in V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$
by *auto*
moreover
from *C2-equals-C1* $\langle \alpha \restriction C_{\mathcal{V}_2} = [] \rangle$ **have** $\alpha \restriction C_{\mathcal{V}_1} = []$
by *auto*
ultimately
obtain $\alpha'\ \delta'$ **where** $(set\ \delta') \subseteq (N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1})$
and $\beta @ [c] @ \delta' @ [v] @ \alpha' \in Tr_{ES}$
and $\alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1}$

```
            and α'⌉C_{V_1} = []
    using ‹β @ [v] @ α ∈ Tr_ES› ‹FCI Γ_1 V_1 Tr_ES› unfolding FCI-def by blast

  from ‹(set δ') ⊆ (N_{V_1} ∩ Δ_{Γ_1})› ‹N_{V_2}∩Δ_{Γ_2} ⊇ N_{V_1}∩Δ_{Γ_1}›
  have (set δ') ⊆ (N_{V_2} ∩ Δ_{Γ_2})
    by auto
  moreover
  from ‹α'⌉V_{V_1} = α⌉V_{V_1}› V2-subset-V1 have α'⌉V_{V_2} = α⌉V_{V_2}
    using non-empty-projection-on-subset by blast
  moreover
  from ‹C_{V_2} = C_{V_1}› ‹α'⌉C_{V_1} = []› have α'⌉C_{V_2} = []
    by auto
  ultimately have ∃ δ' α'. (set δ') ⊆ (N_{V_2} ∩ Δ_{Γ_2})
                  ∧ β @ [c] @ δ'@ [v] @ α' ∈ Tr_ES ∧ α'⌉V_{V_2} = α⌉V_{V_2} ∧ α'⌉C_{V_2} = []
          using ‹β @ [c] @ δ' @ [v] @ α' ∈ Tr_ES› by auto
 }
 thus ?thesis
   unfolding FCI-def by blast
qed



lemma FCIA-implies-FCIA-for-modified-view-rho-gamma:
⟦FCIA ϱ_1 Γ_1 V_1 Tr_ES; ϱ_2(V_2) ⊇ ϱ_1(V_1);
    V_{V_2}∩∇_{Γ_2} ⊆ V_{V_1}∩∇_{Γ_1}; N_{V_2}∩Δ_{Γ_2} ⊇ N_{V_1}∩Δ_{Γ_1}; C_{V_2}∩ϒ_{Γ_2} ⊆ C_{V_1}∩ϒ_{Γ_1} ⟧
   ⟹ FCIA ϱ_2 Γ_2 V_2 Tr_ES
proof −
 assume FCIA ϱ_1 Γ_1 V_1 Tr_ES
   and ϱ_2(V_2) ⊇ ϱ_1(V_1)
   and V_{V_2}∩∇_{Γ_2} ⊆ V_{V_1}∩∇_{Γ_1}
   and N_{V_2}∩Δ_{Γ_2} ⊇ N_{V_1}∩Δ_{Γ_1}
   and C_{V_2}∩ϒ_{Γ_2} ⊆ C_{V_1}∩ϒ_{Γ_1}
 {
   fix α β v c
   assume c ∈ C_{V_2}∩ϒ_{Γ_2}
     and v ∈ V_{V_2}∩∇_{Γ_2}
     and β @ [v] @ α ∈ Tr_ES
     and α⌉C_{V_2} = []
     and Adm V_2 ϱ_2 Tr_ES β c

   from ‹c ∈ C_{V_2}∩ϒ_{Γ_2}› ‹C_{V_2}∩ϒ_{Γ_2} ⊆ C_{V_1}∩ϒ_{Γ_1}› have c ∈ C_{V_1}∩ϒ_{Γ_1}
     by auto
   moreover
   from ‹v ∈ V_{V_2}∩∇_{Γ_2}› ‹V_{V_2}∩∇_{Γ_2} ⊆ V_{V_1}∩∇_{Γ_1}› have v ∈ V_{V_1}∩∇_{Γ_1}
     by auto
   moreover
   from C2-equals-C1 ‹α⌉C_{V_2} = []› have α⌉C_{V_1} = []
     by auto
   moreover
   from ‹Adm V_2 ϱ_2 Tr_ES β c› ‹ϱ_2(V_2) ⊇ ϱ_1(V_1)› have Adm V_1 ϱ_1 Tr_ES β c
     by (simp add: Adm-implies-Adm-for-modified-rho)
   ultimately
```

obtain α' δ' **where** $(set\ \delta') \subseteq (N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1})$
and β @ $[c]$ @ δ' @ $[v]$ @ $\alpha' \in Tr_{ES}$
and $\alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1}$
and $\alpha' \restriction C_{\mathcal{V}_1} = []$
using ‹β @ $[v]$ @ $\alpha \in Tr_{ES}$› ‹*FCIA* ϱ_1 Γ_1 $\mathcal{V}_1$ Tr_{ES}› **unfolding** *FCIA-def* **by** *blast*

from ‹$(set\ \delta') \subseteq (N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1})$› ‹$N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$›
have $(set\ \delta') \subseteq (N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2})$
by *auto*
moreover
from ‹$\alpha' \restriction V_{\mathcal{V}_1} = \alpha \restriction V_{\mathcal{V}_1}$› *V2-subset-V1* **have** $\alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2}$
using *non-empty-projection-on-subset* **by** *blast*
moreover
from ‹$C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$› ‹$\alpha' \restriction C_{\mathcal{V}_1} = []$› **have** $\alpha' \restriction C_{\mathcal{V}_2} = []$
by *auto*
ultimately
have $\exists\ \delta'\ \alpha'.\ (set\ \delta') \subseteq (N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2})$
$\wedge\ \beta$ @ $[c]$ @ δ' @ $[v]$ @ $\alpha' \in Tr_{ES} \wedge \alpha' \restriction V_{\mathcal{V}_2} = \alpha \restriction V_{\mathcal{V}_2} \wedge \alpha' \restriction C_{\mathcal{V}_2} = []$
using ‹β @ $[c]$ @ δ' @ $[v]$ @ $\alpha' \in Tr_{ES}$› **by** *auto*
}
thus *?thesis*
unfolding *FCIA-def* **by** *blast*
qed
end

end

5.3 Unwinding

We define the unwinding conditions provided in [3] and prove the unwinding theorems from [3] that use these unwinding conditions.

5.3.1 Unwinding Conditions

theory *UnwindingConditions*
imports *../Basics/BSPTaxonomy*
../../SystemSpecification/StateEventSystems
begin

locale *Unwinding* =
fixes *SES* :: $('s, 'e)$ *SES-rec*
and $\mathcal{V}$ :: $'e$ *V-rec*

assumes *validSES*: *SES-valid SES*
and *validVU*: *isViewOn* $\mathcal{V}$ E_{SES}

sublocale *Unwinding* ⊆ *BSPTaxonomyDifferentCorrections induceES SES* $\mathcal{V}$
by (*unfold-locales*, *simp add*: *induceES-yields-ES validSES*,
simp add: *induceES-def validVU*)

context *Unwinding*
begin

definition *osc* :: *'s rel* ⇒ *bool*
where
osc ur ≡
 ∀ *s1* ∈ S_{SES}. ∀ *s1'* ∈ S_{SES}. ∀ *s2'* ∈ S_{SES}. ∀ *e* ∈ (E_{SES} − $C_{\mathcal{V}}$).
  (*reachable SES s1* ∧ *reachable SES s1'*
   ∧ *s1' e*⟶$_{SES}$ *s2'* ∧ (*s1'*, *s1*) ∈ *ur*)
  ⟶ (∃ *s2* ∈ S_{SES}. ∃ *δ*. *δ* ↾ $C_{\mathcal{V}}$ = [] ∧ *δ* ↾ $V_{\mathcal{V}}$ = [*e*] ↾ $V_{\mathcal{V}}$
   ∧ *s1* *δ*⟹$_{SES}$ *s2* ∧ (*s2'*, *s2*) ∈ *ur*)

definition *lrf* :: *'s rel* ⇒ *bool*
where
lrf ur ≡
 ∀ *s* ∈ S_{SES}. ∀ *s'* ∈ S_{SES}. ∀ *c* ∈ $C_{\mathcal{V}}$.
 ((*reachable SES s* ∧ *s c*⟶$_{SES}$ *s'*) ⟶ (*s'*, *s*) ∈ *ur*)

definition *lrb* :: *'s rel* ⇒ *bool*
where
lrb ur ≡ ∀ *s* ∈ S_{SES}. ∀ *c* ∈ $C_{\mathcal{V}}$.
 (*reachable SES s* ⟶ (∃ *s'* ∈ S_{SES}. (*s c*⟶$_{SES}$ *s'* ∧ ((*s*, *s'*) ∈ *ur*))))

definition *fcrf* :: *'e Gamma* ⇒ *'s rel* ⇒ *bool*
where
fcrf Γ *ur* ≡
 ∀ *c* ∈ ($C_{\mathcal{V}}$ ∩ Υ_Γ). ∀ *v* ∈ ($V_{\mathcal{V}}$ ∩ ∇_Γ). ∀ *s* ∈ S_{SES}. ∀ *s'* ∈ S_{SES}.
  ((*reachable SES s* ∧ *s* ([*c*] @ [*v*])⟹$_{SES}$ *s'*)
   ⟶ (∃ *s''* ∈ S_{SES}. ∃ *δ*. (∀ *d* ∈ (*set δ*). *d* ∈ ($N_{\mathcal{V}}$ ∩ Δ_Γ)) ∧
    s (*δ* @ [*v*])⟹$_{SES}$ *s''* ∧ (*s'*, *s''*) ∈ *ur*))

definition *fcrb* :: *'e Gamma* ⇒ *'s rel* ⇒ *bool*
where
fcrb Γ *ur* ≡
 ∀ *c* ∈ ($C_{\mathcal{V}}$ ∩ Υ_Γ). ∀ *v* ∈ ($V_{\mathcal{V}}$ ∩ ∇_Γ). ∀ *s* ∈ S_{SES}. ∀ *s''* ∈ S_{SES}.
 ((*reachable SES s* ∧ *s v*⟶$_{SES}$ *s''*)
  ⟶ (∃ *s'* ∈ S_{SES}. ∃ *δ*. (∀ *d* ∈ (*set δ*). *d* ∈ ($N_{\mathcal{V}}$ ∩ Δ_Γ)) ∧
   s ([*c*] @ *δ* @ [*v*])⟹$_{SES}$ *s'* ∧ (*s''*, *s'*) ∈ *ur*))

definition *En* :: *'e Rho* ⇒ *'s* ⇒ *'e* ⇒ *bool*
where
En ϱ s e ≡
 ∃ *β* *γ*. ∃ *s'* ∈ S_{SES}. ∃ *s''* ∈ S_{SES}.
  $s0_{SES}$ *β*⟹$_{SES}$ *s* ∧ (*γ* ↾ (*ϱ* $\mathcal{V}$) = *β* ↾ (*ϱ* $\mathcal{V}$))

$\wedge$ $s0_{SES}$ $\gamma \Longrightarrow_{SES}$ $s' \wedge s'$ $e \longrightarrow_{SES}$ s''

definition *lrbe* :: *'e Rho* $\Rightarrow$ *'s rel* $\Rightarrow$ *bool*
where
lrbe ϱ *ur* $\equiv$
$\forall s \in S_{SES}. \forall c \in C_{\mathcal{V}}$.
((*reachable SES s* $\wedge$ (*En* ϱ *s c*))
$\longrightarrow$ $(\exists s' \in S_{SES}. (s\ c \longrightarrow_{SES} s' \wedge (s, s') \in ur)))$

definition *fcrbe* :: *'e Gamma* $\Rightarrow$ *'e Rho* $\Rightarrow$ *'s rel* $\Rightarrow$ *bool*
where
fcrbe Γ ϱ *ur* $\equiv$
$\forall c \in (C_{\mathcal{V}} \cap \Upsilon_\Gamma). \forall v \in (V_{\mathcal{V}} \cap \nabla_\Gamma). \forall s \in S_{SES}. \forall s'' \in S_{SES}.$
((*reachable SES s* $\wedge$ *s* $v \longrightarrow_{SES}$ $s'' \wedge$ (*En* ϱ *s c*))
$\longrightarrow$ $(\exists s' \in S_{SES}. \exists \delta. (\forall d \in (set\ \delta).\ d \in (N_{\mathcal{V}} \cap \Delta_\Gamma)) \wedge$
s $([c] @ \delta @ [v]) \Longrightarrow_{SES}$ $s' \wedge (s'', s') \in ur))$

end

end

5.3.2 Auxiliary Results

theory *AuxiliaryLemmas*
imports *UnwindingConditions*
begin

context *Unwinding*
begin

lemma *osc-property*:
$\bigwedge s1\ s1'.$ ⟦ *osc ur*; *s1* $\in S_{SES}$; *s1'* $\in S_{SES}$; $\alpha \restriction C_{\mathcal{V}} = []$;
reachable SES s1; *reachable SES s1'*; *enabled SES s1'* α; $(s1', s1) \in ur$ ⟧
$\Longrightarrow (\exists \alpha'.\ \alpha' \restriction C_{\mathcal{V}} = [] \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge$ *enabled SES s1* $\alpha')$
proof (*induct* α)
case *Nil*
have $[] \restriction C_{\mathcal{V}} = [] \wedge$
$[] \restriction V_{\mathcal{V}} = [] \restriction V_{\mathcal{V}} \wedge$ *enabled SES s1* []
by (*simp add*: *enabled-def projection-def*)
thus *?case* **by** (*rule exI*)
next
case (*Cons e1* $\alpha 1$)
assume *osc-true*: *osc ur*
assume *s1-in-S*: *s1* $\in S_{SES}$
assume *s1'-in-S*: *s1'* $\in S_{SES}$
assume *e1α1-C-empty*: $(e1\ \#\ \alpha 1) \restriction C_{\mathcal{V}} = []$
assume *reachable-s1*: *reachable SES s1*
assume *reachable-s1'*: *reachable SES s1'*

assume *enabled-s1'-e1α1*: *enabled SES s1' (e1 # α1)*
assume *unwindingrel-s1'-s1*: *(s1', s1) ∈ ur*

have *e1α1-no-c*: ∀ *a* ∈ (*set* (*e1 # α1*)). *a* ∈ (E_{SES} − $C_{\mathcal{V}}$)
proof −
  from *reachable-s1'* **obtain** *β*
    where *s0*$_{SES}$ *β*⟹$_{SES}$ *s1'*
    by(*simp add*: *reachable-def*, *auto*)
  moreover
  from *enabled-s1'-e1α1* **obtain** *s1337*
    where *s1' (e1 # α1)*⟹$_{SES}$ *s1337*
    by(*simp add*: *enabled-def*, *auto*)
  ultimately have *s0*$_{SES}$ (*β* @ (*e1 # α1*))⟹$_{SES}$ *s1337*
    by(*rule path-trans*)
  hence *β* @ (*e1 # α1*) ∈ $Tr_{(induceES\ SES)}$
    by (*simp add*: *induceES-def possible-traces-def enabled-def*)
  with *validSES induceES-yields-ES*[*of SES*] **have** ∀ *a* ∈ (*set* (*β* @ (*e1 # α1*))). *a* ∈ E_{SES}
    by (*simp add*: *induceES-def ES-valid-def traces-contain-events-def*)
  hence ∀ *a* ∈ (*set* (*e1 # α1*)). *a* ∈ E_{SES}
    by *auto*
  with *e1α1-C-empty* **show** *?thesis*
    by (*simp only*: *projection-def filter-empty-conv*, *auto*)
qed

from *enabled-s1'-e1α1* **obtain** *s2'* **where**
  s1'-e1-s2': *s1' e1*⟶$_{SES}$ *s2'*
  by (*simp add*: *enabled-def*, *split if-split-asm*, *auto*)
with *validSES* **have** *s2'-in-S*: *s2'* ∈ S_{SES}
  by (*simp add*: *SES-valid-def correct-transition-relation-def*)
have *reachable-s2'*: *reachable SES s2'*
proof −
  from *reachable-s1'* **obtain** *t* **where**
    path-to-s1': *s0*$_{SES}$ *t*⟹$_{SES}$ *s1'*
    by (*simp add*: *reachable-def*, *auto*)
  from *s1'-e1-s2'* **have** *s1'* [*e1*]⟹$_{SES}$ *s2'*
    by *simp*
  with *path-to-s1'* **have** *s0*$_{SES}$ (*t* @ [*e1*])⟹$_{SES}$ *s2'*
    by (*simp add*: *path-trans*)
  thus *?thesis* **by** (*simp add*: *reachable-def*, *rule exI*)
qed
from *s1'-e1-s2' enabled-s1'-e1α1* **obtain** *sn'* **where**
  s2' α1⟹$_{SES}$ *sn'*
  by (*simp add*: *enabled-def*, *auto*)
hence *enabled-s2'-α1*: *enabled SES s2' α1*
  by (*simp add*: *enabled-def*)
from *e1α1-no-c* **have** *e1-no-c*: *e1* ∈ (E_{SES} − $C_{\mathcal{V}}$)
  by *simp*
from *e1α1-no-c* **have** *α1-no-c*: ∀ *a*∈(*set α1*). (*a* ∈ (E_{SES} − $C_{\mathcal{V}}$))
  by *simp*
hence *α1-proj-C-empty*: *α1* ↾ $C_{\mathcal{V}}$ = []
  by (*simp add*: *projection-def*)
from *osc-true* **have**

⟦ $s1 \in S_{SES}$; $s1' \in S_{SES}$; $s2' \in S_{SES}$;
$e1 \in (E_{SES} - C_{\mathcal{V}})$; *reachable SES s1*; *reachable SES s1'*;
$s1'\ e1 \longrightarrow_{SES} s2'$; $(s1', s1) \in ur$ ⟧
$\Longrightarrow (\exists\, s2 \in S_{SES}.\ \exists \delta.\ \delta \restriction C_{\mathcal{V}} = []$
$\wedge\ (\delta \restriction V_{\mathcal{V}}) = ([e1] \restriction V_{\mathcal{V}}) \wedge (s1\ \delta \Longrightarrow_{SES} s2 \wedge$
$((s2', s2) \in ur)))$
by (*simp add*: *osc-def*)
with *s1-in-S s1'-in-S e1-no-c reachable-s1 reachable-s1' s2'-in-S s1'-e1-s2' unwindingrel-s1'-s1*
obtain *s2* δ **where**
osc-conclusion:
$s2 \in S_{SES} \wedge \delta \restriction C_{\mathcal{V}} = [] \wedge$
$(\delta \restriction V_{\mathcal{V}}) = ([e1] \restriction V_{\mathcal{V}}) \wedge s1\ \delta \Longrightarrow_{SES} s2 \wedge$
$((s2', s2) \in ur)$
by *auto*
hence *δ-proj-C-empty*: $\delta \restriction C_{\mathcal{V}} = []$
by (*simp add*: *projection-def*)
from *osc-conclusion* **have** *s2-in-S*: $s2 \in S_{SES}$
by *auto*
from *osc-conclusion* **have** *unwindingrel-s2'-s2*: $(s2', s2) \in ur$
by *auto*
have *reachable-s2*: *reachable SES s2*
proof −
from *reachable-s1* **obtain** *t* **where**
path-to-s1: $s0_{SES}\ t \Longrightarrow_{SES} s1$
by (*simp add*: *reachable-def*, *auto*)
from *osc-conclusion* **have** $s1\ \delta \Longrightarrow_{SES} s2$
by *auto*
with *path-to-s1* **have** $s0_{SES}\ (t\ @\ \delta) \Longrightarrow_{SES} s2$
by (*simp add*: *path-trans*)
thus *?thesis* **by** (*simp add*: *reachable-def*, *rule exI*)
qed

from *Cons osc-true s2-in-S s2'-in-S α1-proj-C-empty reachable-s2 reachable-s2' enabled-s2'-α1 unwindingrel-s2'-s2*
obtain α'' **where** *α''-props*:
$\alpha'' \restriction C_{\mathcal{V}} = [] \wedge \alpha'' \restriction V_{\mathcal{V}} = \alpha 1 \restriction V_{\mathcal{V}} \wedge$ *enabled SES s2* α''
by *auto*
with *osc-conclusion* **have** *δα''-props*:
$(\delta\ @\ \alpha'') \restriction C_{\mathcal{V}} = [] \wedge$
$(\delta\ @\ \alpha'') \restriction V_{\mathcal{V}} = (e1 \# \alpha 1) \restriction V_{\mathcal{V}} \wedge$ *enabled SES s1* $(\delta\ @\ \alpha'')$
by (*simp add*: *projection-def enabled-def*, *auto*, *simp add*: *path-trans*)
hence $(\delta\ @\ \alpha'') \restriction C_{\mathcal{V}} = []$
by (*simp add*: *projection-def*)
thus *?case* **using** *δα''-props* **by** *auto*
qed

lemma *path-state-closure*: ⟦ $s\ \tau \Longrightarrow_{SES} s'$; $s \in S_{SES}$ ⟧ $\Longrightarrow s' \in S_{SES}$
(**is** ⟦ *?P s τ s'*; *?S s SES* ⟧ $\Longrightarrow$ *?S s' SES*)
proof (*induct τ arbitrary*: *s s'*)
case *Nil* **with** *validSES* **show** *?case*

by (*auto simp add*: *SES-valid-def correct-transition-relation-def*)
next
case (*Cons e τ*) **thus** *?case*
proof −
assume *path-eτ*: *?P s* (*e* # *τ*) *s′*
assume *induct-hypo*: ⋀ *s s′*. ⟦ *?P s τ s′*; *?S s SES* ⟧ ⟹ *?S s′ SES*

from *path-eτ* **obtain** *s″* **where** *s-e-s″*: *s e*⟶$_{SES}$ *s″*
by(*simp add*: *path-def*, *split if-split-asm*, *auto*)
with *validSES* **have** *s″-in-S*: *?S s″ SES*
by (*simp add*: *SES-valid-def correct-transition-relation-def*)

from *s-e-s″ path-eτ* **have** *path-τ*: *?P s″ τ s′* **by** *auto*

from *path-τ s″-in-S* **show** *?case* **by** (*rule induct-hypo*)
qed
qed

theorem *En-to-Adm*:
⟦ *reachable SES s*; *En ϱ s e*⟧
⟹ ∃ *β*. (*s0*$_{SES}$ *β*⟹$_{SES}$ *s* ∧ *Adm V ϱ Tr*$_{(induceES\ SES)}$ *β e*)
proof −
assume *En ϱ s e*
then obtain *β γ s′ s″*
where *s0*$_{SES}$ *β*⟹$_{SES}$ *s*
and *γ* ↾ (*ϱ V*) = *β* ↾ (*ϱ V*)
and *s0-γ-s′*: *s0*$_{SES}$ *γ*⟹$_{SES}$ *s′*
and *s′-e-s″*: *s′ e*⟶$_{SES}$ *s″*
by (*simp add*: *En-def*, *auto*)
moreover
from *s0-γ-s′ s′-e-s″* **have** *s0*$_{SES}$ (*γ* @ [*e*])⟹$_{SES}$ *s″*
by (*rule path-trans-single*)
hence (*γ* @ [*e*]) ∈ *Tr*$_{(induceES\ SES)}$
by(*simp add*: *induceES-def possible-traces-def enabled-def*)
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed

theorem *Adm-to-En*:
⟦ *β* ∈ *Tr*$_{(induceES\ SES)}$; *Adm V ϱ Tr*$_{(induceES\ SES)}$ *β e* ⟧
⟹ ∃ *s* ∈ *S*$_{SES}$. (*s0*$_{SES}$ *β*⟹$_{SES}$ *s* ∧ *En ϱ s e*)
proof −
from *validSES* **have** *s0-in-S*: *s0*$_{SES}$ ∈ *S*$_{SES}$
by (*simp add*: *SES-valid-def s0-is-state-def*)

assume *β* ∈ *Tr*$_{(induceES\ SES)}$
then obtain *s*

where *s0-β-s*: $s0_{SES}\ \beta \Longrightarrow_{SES} s$
by (*simp add*: *induceES-def possible-traces-def enabled-def*, *auto*)
from *this* **have** *s-in-S*: $s \in S_{SES}$ **using** *s0-in-S*
by (*rule path-state-closure*)

assume *Adm* $\mathcal{V}\ \varrho\ Tr_{(induceES\ SES)}\ \beta\ e$
then obtain γ
where *ϱγ-is-ϱβ*: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and $\exists s''.\ s0_{SES}\ (\gamma\ @\ [e]) \Longrightarrow_{SES} s''$
by(*simp add*: *Adm-def induceES-def possible-traces-def enabled-def*, *auto*)
then obtain s''
where *s0-γe-s''*: $s0_{SES}\ (\gamma\ @\ [e]) \Longrightarrow_{SES} s''$
by *auto*
from *this* **have** *s''-in-S*: $s'' \in S_{SES}$ **using** *s0-in-S*
by (*rule path-state-closure*)

from *path-split-single*[*OF s0-γe-s''*] **obtain** s'
where *s0-γ-s'*: $s0_{SES}\ \gamma \Longrightarrow_{SES} s'$
and *s'-e-s''*: $s'\ e \longrightarrow_{SES} s''$
by *auto*

from *path-state-closure*[*OF s0-γ-s' s0-in-S*] **have** *s'-in-S*: $s' \in S_{SES}$**.**

from *s'-in-S s''-in-S s0-β-s ϱγ-is-ϱβ s0-γ-s' s'-e-s'' s-in-S* **show** *?thesis*
by (*simp add*: *En-def*, *auto*)
qed

lemma *state-from-induceES-trace*:
$[\![\ (\beta\ @\ \alpha) \in Tr_{(induceES\ SES)}\ ]\!]$
$\Longrightarrow \exists s \in S_{SES}.\ s0_{SES}\ \beta \Longrightarrow_{SES} s \wedge enabled\ SES\ s\ \alpha \wedge reachable\ SES\ s$
proof −

assume *βα-in-Tr*: $(\beta\ @\ \alpha) \in Tr_{(induceES\ SES)}$
then obtain s' **where** *s0-βα-s'*:$s0_{SES}\ (\beta\ @\ \alpha) \Longrightarrow_{SES} s'$
by (*simp add*: *induceES-def possible-traces-def enabled-def*, *auto*)

from *path-split*[*OF s0-βα-s'*] **obtain** s
where *s0-β-s*: $s0_{SES}\ \beta \Longrightarrow_{SES} s$
and $s\ \alpha \Longrightarrow_{SES} s'$
by *auto*
hence *enabled-s-α*: *enabled SES s α*
by (*simp add*: *enabled-def*)

from *s0-β-s* **have** *reachable-s*: *reachable SES s*
by(*simp add*: *reachable-def*, *auto*)

from *validSES* **have** $s0_{SES} \in S_{SES}$
by (*simp add*: *SES-valid-def s0-is-state-def*)
with *s0-β-s* **have** *s-in-S*: $s \in S_{SES}$
by (*rule path-state-closure*)

with *s0-β-s enabled-s-α reachable-s* **show** *?thesis*
by *auto*
qed

lemma *path-split2*:$s0_{SES}\ (\beta\ @\ \alpha) \Longrightarrow_{SES} s$
$\Longrightarrow \exists\, s' \in S_{SES}.\ (\ s0_{SES}\ \beta \Longrightarrow_{SES} s' \wedge s'\ \alpha \Longrightarrow_{SES} s \wedge reachable\ SES\ s'\ )$
proof −
assume *s0-βα-s*: $s0_{SES}\ (\beta\ @\ \alpha) \Longrightarrow_{SES} s$

from *path-split*[*OF s0-βα-s*] **obtain** s'
where *s0-β-s'*: $s0_{SES}\ \beta \Longrightarrow_{SES} s'$
and *s'-α-s*: $s'\ \alpha \Longrightarrow_{SES} s$
by *auto*
hence *reachable SES s'*
by(*simp add*: *reachable-def*, *auto*)
moreover
have $s' \in S_{SES}$
proof −
from *s0-β-s' validSES path-state-closure* **show** *?thesis*
by (*auto simp add*: *SES-valid-def s0-is-state-def*)
qed

ultimately show *?thesis* **using** *s'-α-s s0-β-s'*
by(*auto*)
qed

lemma *path-split-single2*:
$s0_{SES}\ (\beta\ @\ [x]) \Longrightarrow_{SES} s$
$\Longrightarrow \exists\, s' \in S_{SES}.\ (\ s0_{SES}\ \beta \Longrightarrow_{SES} s' \wedge s'\ x \longrightarrow_{SES} s \wedge reachable\ SES\ s'\ )$
proof −
assume *s0-βx-s*: $s0_{SES}\ (\beta\ @\ [x]) \Longrightarrow_{SES} s$

from *path-split2*[*OF s0-βx-s*] **show** *?thesis*
by (*auto*, *split if-split-asm*, *auto*)
qed

lemma *modified-view-valid*: *isViewOn* $(\!|\, V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}),\ N = \{\},\ C = C_{\mathcal{V}} \,|\!)\ E_{SES}$
using *validVU*
unfolding *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def* **by** *auto*

end

end

5.3.3 Unwinding Theorems

theory *UnwindingResults*
imports *AuxiliaryLemmas*

begin

context *Unwinding*
begin
theorem *unwinding-theorem-BSD*:
$[\![$ *lrf ur*; *osc ur* $]\!] \Longrightarrow$ *BSD* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
proof −
assume *lrf-true*: *lrf ur*
assume *osc-true*: *osc ur*

{
fix α β *c*
assume *c-in-C*: $c \in C_{\mathcal{V}}$
assume $\beta c\alpha$*-in-Tr*: $((\beta\ @\ [c])\ @\ \alpha) \in Tr_{(induceES\ SES)}$
assume α*-contains-no-c*: $\alpha \restriction C_{\mathcal{V}} = []$

from *state-from-induceES-trace*[*OF* $\beta c\alpha$*-in-Tr*] **obtain** *s1′*
where *s1′-in-S*: $s1' \in S_{SES}$
and *enabled-s1′-*α: *enabled SES s1′* α
and *s0-*β*c-s1′*: $s0_{SES}\ (\beta\ @\ [c]) \Longrightarrow_{SES} s1'$
and *reachable-s1′*: *reachable SES s1′*
by *auto*

from *path-split-single2*[*OF s0-*β*c-s1′*] **obtain** *s1*
where *s1-in-S*: $s1 \in S_{SES}$
and *s0-*β*-s1*: $s0_{SES}\ \beta \Longrightarrow_{SES} s1$
and *s1-c-s1′*: $s1\ c \longrightarrow_{SES} s1'$
and *reachable-s1*: *reachable SES s1*
by *auto*

from *s1-in-S s1′-in-S c-in-C reachable-s1 s1-c-s1′ lrf-true*
have *s1′-ur-s1*: $((s1', s1) \in ur)$
by (*simp add*: *lrf-def*, *auto*)

from *osc-property*[*OF osc-true s1-in-S s1′-in-S* α*-contains-no-c reachable-s1*
reachable-s1′ enabled-s1′-α *s1′-ur-s1*]
obtain α'
where α'*-contains-no-c*: $\alpha' \restriction C_{\mathcal{V}} = []$
and α'*-V-is-*α*-V*: $\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
and *enabled-s1-*α': *enabled SES s1* α'
by *auto*

have $\beta\alpha'$*-in-Tr*: $\beta\ @\ \alpha' \in Tr_{(induceES\ SES)}$
proof −
note *s0-*β*-s1*
moreover
from *enabled-s1-*α' **obtain** *s2*
where *s1* $\alpha' \Longrightarrow_{SES} s2$
by (*simp add*: *enabled-def*, *auto*)
ultimately have $s0_{SES}\ (\beta\ @\ \alpha') \Longrightarrow_{SES} s2$
by (*rule path-trans*)
thus *?thesis*

```
      by (simp add: induceES-def possible-traces-def enabled-def)
    qed

    from βα'-in-Tr α'-V-is-α-V α'-contains-no-c have
      ∃α'. ((β @ α') ∈ (Tr_(induceES SES)) ∧ (α' ↑ (V_V)) = (α ↑ V_V) ∧ α' ↑ C_V = [])
      by auto
  }
  thus ?thesis
    by (simp add: BSD-def)
qed

theorem unwinding-theorem-BSI:
⟦ lrb ur; osc ur ⟧ ⟹ BSI V Tr_(induceES SES)
proof −
  assume lrb-true: lrb ur
  assume osc-true: osc ur

  {
    fix α β c
    assume c-in-C: c ∈ C_V
    assume βα-in-ind-Tr: (β @ α) ∈ Tr_(induceES SES)
    assume α-contains-no-c: α ↑ C_V = []

    from state-from-induceES-trace[OF βα-in-ind-Tr] obtain s1
      where s1-in-S : s1 ∈ S_SES
      and path-β-yields-s1: s0_SES β⟹_SES s1
      and enabled-s1-α: enabled SES s1 α
      and reachable-s1: reachable SES s1
      by auto

    from reachable-s1 s1-in-S c-in-C lrb-true
    have ∃s1'∈ S_SES. s1 c⟶_SES s1' ∧ (s1, s1') ∈ ur
      by(simp add: lrb-def)
    then obtain s1'
      where s1'-in-S: s1' ∈ S_SES
      and s1-trans-c-s1': s1 c⟶_SES s1'
      and s1-s1'-in-ur: (s1, s1') ∈ ur
      by auto

    have reachable-s1': reachable SES s1'
    proof −
      from path-β-yields-s1 s1-trans-c-s1' have s0_SES (β @ [c])⟹_SES s1'
        by (rule path-trans-single)
      thus ?thesis by (simp add: reachable-def, auto)
    qed

    from osc-property[OF osc-true s1'-in-S s1-in-S α-contains-no-c
      reachable-s1' reachable-s1 enabled-s1-α s1-s1'-in-ur]
    obtain α'
      where α'-contains-no-c: α' ↑ C_V = []
      and α'-V-is-α-V: α' ↑ V_V = α ↑ V_V
      and enabled-s1'-α': enabled SES s1' α'
```

by *auto*

have $\beta c\alpha'$*-in-ind-Tr*: β @ $[c]$ @ $\alpha' \in Tr_{(induceES\ SES)}$
proof −
from *path-*β*-yields-s1 s1-trans-c-s1'* **have** $s0_{SES}\ (\beta$ @ $[c]) \Longrightarrow_{SES} s1'$
by (*rule path-trans-single*)
moreover
from *enabled-s1'-*α' **obtain** *s2*
where $s1'\ \alpha' \Longrightarrow_{SES} s2$
by (*simp add*: *enabled-def*, *auto*)
ultimately have $s0_{SES}\ ((\beta$ @ $[c])$ @ $\alpha') \Longrightarrow_{SES} s2$
by (*rule path-trans*)
thus *?thesis*
by (*simp add*: *induceES-def possible-traces-def enabled-def*)
qed

from $\beta c\alpha'$*-in-ind-Tr* α'*-V-is-*α*-V* α'*-contains-no-c*
have $\exists \alpha'.\ \beta$ @ $c\ \#\ \alpha' \in Tr_{(induceES\ SES)} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$
by *auto*
}
thus *?thesis*
by(*simp add*: *BSI-def*)
qed

theorem *unwinding-theorem-BSIA*:
$[\![$ *lrbe* ϱ *ur*; *osc ur* $]\!] \Longrightarrow$ *BSIA* $\varrho\ \mathcal{V}\ Tr_{(induceES\ SES)}$
proof −
assume *lrbe-true*: *lrbe* ϱ *ur*
assume *osc-true*: *osc ur*

{
fix $\alpha\ \beta\ c$
assume *c-in-C*: $c \in C_{\mathcal{V}}$
assume $\beta\alpha$*-in-ind-Tr*: $(\beta$ @ $\alpha) \in Tr_{(induceES\ SES)}$
assume α*-contains-no-c*: $\alpha \restriction C_{\mathcal{V}} = []$

assume *adm*: *Adm* $\mathcal{V}\ \varrho\ Tr_{(induceES\ SES)}\ \beta\ c$

from *state-from-induceES-trace*[*OF* $\beta\alpha$*-in-ind-Tr*]
obtain *s1*
where *s1-in-S* : $s1 \in S_{SES}$
and *s0-*β*-s1*: $s0_{SES}\ \beta \Longrightarrow_{SES} s1$
and *enabled-s1-*α: *enabled SES s1* α
and *reachable-s1*: *reachable SES s1*
by *auto*

have $\exists \alpha'.\ \beta$ @ $[c]$ @ $\alpha' \in Tr_{(induceES\ SES)} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$
proof *cases*
assume *en*: *En* ϱ *s1 c*

from *reachable-s1 s1-in-S c-in-C en lrbe-true*
have $\exists\, s1' \in S_{SES}.\ s1\ c \longrightarrow_{SES} s1' \wedge (s1,\ s1') \in ur$
by(*simp add*: *lrbe-def*)
then obtain *s1'*
where *s1'-in-S*: $s1' \in S_{SES}$
and *s1-trans-c-s1'*: $s1\ c \longrightarrow_{SES} s1'$
and *s1-s1'-in-ur*: $(s1,\ s1') \in ur$
by *auto*

have *reachable-s1'*: *reachable SES s1'*
proof −
from *s0-β-s1 s1-trans-c-s1'* **have** $s0_{SES}\ (\beta\ @\ [c]) \Longrightarrow_{SES} s1'$
by (*rule path-trans-single*)
thus *?thesis* **by** (*simp add*: *reachable-def*, *auto*)
qed

from *osc-property*[*OF osc-true s1'-in-S s1-in-S α-contains-no-c*
reachable-s1' reachable-s1 enabled-s1-α s1-s1'-in-ur]
obtain α'
where α'*-contains-no-c*: $\alpha' \restriction C_{\mathcal{V}} = []$
and α'*-V-is-α-V*: $\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
and *enabled-s1'-α'*: *enabled SES s1'* α'
by *auto*

have *βcα'-in-ind-Tr*: $\beta\ @\ [c]\ @\ \alpha' \in Tr_{(induceES\ SES)}$
proof −
from *s0-β-s1 s1-trans-c-s1'* **have** $s0_{SES}\ (\beta\ @\ [c]) \Longrightarrow_{SES} s1'$
by (*rule path-trans-single*)
moreover
from *enabled-s1'-α'* **obtain** *s2*
where *s1'* $\alpha' \Longrightarrow_{SES} s2$
by (*simp add*: *enabled-def*, *auto*)
ultimately have $s0_{SES}\ ((\beta\ @\ [c])\ @\ \alpha') \Longrightarrow_{SES} s2$
by (*rule path-trans*)
thus *?thesis*
by (*simp add*: *induceES-def possible-traces-def enabled-def*)
qed

from *βcα'-in-ind-Tr α'-V-is-α-V α'-contains-no-c* **show** *?thesis*
by *auto*
next
assume *not-en*: $\neg$ *En ϱ s1 c*

let *?A* = (*Adm* $\mathcal{V}$ *ϱ* ($Tr_{(induceES\ SES)}$) *β c*)
let *?E* = $\exists\, s \in S_{SES}.\ (s0_{SES}\ \beta \Longrightarrow_{SES} s \wedge$ *En ϱ s c*)

{
assume *adm*: *?A*

from *s0-β-s1* **have** *β-in-Tr*: $\beta \in Tr_{(induceES\ SES)}$
by (*simp add*: *induceES-def possible-traces-def enabled-def*)

```
      from β-in-Tr adm have ?E
        by (rule Adm-to-En)
    }
    hence Adm-to-En-contr: ¬ ?E ⟹ ¬ ?A
      by blast
    with s1-in-S s0-β-s1 not-en have not-adm: ¬ ?A
      by auto
    with adm show ?thesis
      by auto
  qed
 }
 thus ?thesis
  by (simp add: BSIA-def)
qed

theorem unwinding-theorem-FCD:
⟦ fcrf Γ ur; osc ur ⟧ ⟹ FCD Γ 𝒱 Tr(induceES SES)
proof −
 assume fcrf: fcrf Γ ur
 assume osc: osc ur

 {
  fix α β c v

  assume c-in-C-inter-Y: c ∈ (C𝒱 ∩ ϒΓ)
  assume v-in-V-inter-Nabla: v ∈ (V𝒱 ∩ ∇Γ)
  assume βcvα-in-Tr: ((β @ [c] @ [v]) @ α) ∈ Tr(induceES SES)
  assume α-contains-no-c: α ↾ C𝒱 = []

  from state-from-induceES-trace[OF βcvα-in-Tr] obtain s1'
    where s1'-in-S: s1' ∈ SSES
    and s0-βcv-s1': s0SES (β @ ([c] @ [v]))⟹SES s1'
    and enabled-s1'-α: enabled SES s1' α
    and reachable-s1': reachable SES s1'
    by auto

  from path-split2[OF s0-βcv-s1'] obtain s1
    where s1-in-S: s1 ∈ SSES
    and s0-β-s1: s0SES β⟹SES s1
    and s1-cv-s1': s1 ([c] @ [v])⟹SES s1'
    and reachable-s1: reachable SES s1
    by (auto)

  from c-in-C-inter-Y v-in-V-inter-Nabla s1-in-S s1'-in-S reachable-s1 s1-cv-s1' fcrf
  have ∃ s1'' ∈ SSES. ∃ δ. (∀ d ∈ (set δ). d ∈ (N𝒱 ∩ ΔΓ)) ∧
    s1 (δ @ [v])⟹SES s1'' ∧ (s1', s1'') ∈ ur
    by (simp add: fcrf-def)
  then obtain s1'' δ
    where s1''-in-S: s1'' ∈ SSES
    and δ-in-N-inter-Delta-star: (∀ d ∈ (set δ). d ∈ (N𝒱 ∩ ΔΓ))
    and s1-δv-s1'': s1 (δ @ [v])⟹SES s1''
    and s1'-ur-s1'': (s1', s1'') ∈ ur
```

```
    by auto

  have reachable-s1'': reachable SES s1''
  proof −
    from s0-β-s1 s1-δv-s1'' have s0 ⟹_SES (β @ (δ @ [v]))⟹_SES s1''
      by (rule path-trans)
    thus ?thesis
      by (simp add: reachable-def, auto)
  qed

  from osc-property[OF  osc s1''-in-S s1'-in-S α-contains-no-c
    reachable-s1'' reachable-s1' enabled-s1'-α s1'-ur-s1'']
  obtain α'
    where  α'-contains-no-c: α' ↾ C_V = []
    and α'-V-is-α-V: α' ↾ V_V = α ↾ V_V
    and enabled-s1''-α': enabled SES s1'' α'
    by auto

  have βδvα'-in-Tr: β @ δ @ [v] @ α' ∈ Tr_(induceES SES)
    proof −
      from s0-β-s1 s1-δv-s1'' have s0 ⟹_SES (β @ δ @ [v])⟹_SES s1''
        by (rule path-trans)
      moreover
      from enabled-s1''-α' obtain s2
        where s1'' α'⟹_SES s2
        by (simp add: enabled-def, auto)
      ultimately have s0 ⟹_SES ((β @ δ @ [v]) @ α')⟹_SES s2
        by (rule path-trans)
      thus ?thesis
        by (simp add: induceES-def possible-traces-def enabled-def)
    qed

    from δ-in-N-inter-Delta-star βδvα'-in-Tr α'-V-is-α-V α'-contains-no-c
    have ∃α'. ∃δ'. set δ' ⊆ (N_V ∩ Δ_Γ) ∧ β @ δ' @ [v] @ α' ∈ Tr_(induceES SES)
      ∧ α' ↾ V_V = α ↾ V_V ∧ α' ↾ C_V = []
      by auto
  }
  thus ?thesis
    by (simp add: FCD-def)
qed

theorem unwinding-theorem-FCI:
⟦ fcrb Γ ur; osc ur ⟧ ⟹ FCI Γ V Tr_(induceES SES)
proof −
  assume fcrb: fcrb Γ ur
  assume osc: osc ur

  {
    fix α β c v

    assume c-in-C-inter-Y: c ∈ (C_V ∩ ϒ_Γ)
    assume v-in-V-inter-Nabla: v ∈ (V_V ∩ ∇_Γ)
```

assume *βvα-in-Tr*: $((\beta\ @\ [v])\ @\ \alpha) \in Tr_{(induceES\ SES)}$
assume *α-contains-no-c*: $\alpha \upharpoonright C_{\mathcal{V}} = []$

from *state-from-induceES-trace*[*OF βvα-in-Tr*] **obtain** *s1''*
  where *s1''-in-S*: $s1'' \in S_{SES}$
  and *s0-βv-s1''*: $s0_{SES}\ (\beta\ @\ [v]) \Longrightarrow_{SES} s1''$
  and *enabled-s1''-α*: *enabled SES s1'' α*
  and *reachable-s1''*: *reachable SES s1''*
  by *auto*

from *path-split-single2*[*OF s0-βv-s1''*] **obtain** *s1*
  where *s1-in-S*: $s1 \in S_{SES}$
  and *s0-β-s1*: $s0_{SES}\ \beta \Longrightarrow_{SES} s1$
  and *s1-v-s1''*: $s1\ v \longrightarrow_{SES} s1''$
  and *reachable-s1*: *reachable SES s1*
  by (*auto*)

from *c-in-C-inter-Y v-in-V-inter-Nabla s1-in-S*
  s1''-in-S reachable-s1 s1-v-s1'' fcrb
have $\exists\, s1' \in S_{SES}.\ \exists\, \delta.\ (\forall\, d \in (set\ \delta).\ d \in (N_{\mathcal{V}} \cap \Delta_\Gamma))$
  $\wedge$ $s1\ ([c]\ @\ \delta\ @\ [v]) \Longrightarrow_{SES} s1'$
  $\wedge$ $(s1'', s1') \in ur$
  by (*simp add*: *fcrb-def*)
then obtain *s1' δ*
  where *s1'-in-S*: $s1' \in S_{SES}$
  and *δ-in-N-inter-Delta-star*: $(\forall\, d \in (set\ \delta).\ d \in (N_{\mathcal{V}} \cap \Delta_\Gamma))$
  and *s1-cδv-s1'*: $s1\ ([c]\ @\ \delta\ @\ [v]) \Longrightarrow_{SES} s1'$
  and *s1''-ur-s1'*: $(s1'', s1') \in ur$
  by *auto*

have *reachable-s1'*: *reachable SES s1'*
proof −
  from *s0-β-s1 s1-cδv-s1'* **have** $s0_{SES}\ (\beta\ @\ ([c]\ @\ \delta\ @\ [v])) \Longrightarrow_{SES} s1'$
    by (*rule path-trans*)
  thus *?thesis*
    by (*simp add*: *reachable-def*, *auto*)
qed

from *osc-property*[*OF osc s1'-in-S s1''-in-S α-contains-no-c*
  reachable-s1' reachable-s1'' enabled-s1''-α s1''-ur-s1']
obtain *α'*
  where *α'-contains-no-c*: $\alpha' \upharpoonright C_{\mathcal{V}} = []$
  and *α'-V-is-α-V*: $\alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}}$
  and *enabled-s1'-α'*: *enabled SES s1' α'*
  by *auto*

have *βcδvα'-in-Tr*: $\beta\ @\ [c]\ @\ \delta\ @\ [v]\ @\ \alpha' \in Tr_{(induceES\ SES)}$
proof −
  let *?l1* = $\beta\ @\ [c]\ @\ \delta\ @\ [v]$
  let *?l2* = α'
  from *s0-β-s1 s1-cδv-s1'* **have** $s0_{SES}\ (?l1) \Longrightarrow_{SES} s1'$
    by (*rule path-trans*)

```
      moreover
      from enabled-s1'-α' obtain s1337 where s1' ?l2 ⟹_SES s1337
        by (simp add: enabled-def, auto)
      ultimately have s0_SES (?l1 @ ?l2)⟹_SES s1337
        by (rule path-trans)
      thus ?thesis
        by (simp add: induceES-def possible-traces-def enabled-def)
    qed

from δ-in-N-inter-Delta-star βcδvα'-in-Tr α'-V-is-α-V α'-contains-no-c
    have ∃α' δ'.
      set δ' ⊆ (N_𝒱 ∩ Δ_Γ) ∧ β @ [c] @ δ' @ [v] @ α' ∈ Tr_(induceES SES)
      ∧ α' ↑ V_𝒱 = α ↑ V_𝒱 ∧ α' ↑ C_𝒱 = []
      by auto
  }
  thus ?thesis
    by(simp add: FCI-def)
qed

theorem unwinding-theorem-FCIA:
⟦ fcrbe Γ ϱ ur; osc ur ⟧ ⟹ FCIA ϱ Γ 𝒱 Tr_(induceES SES)
proof −
  assume fcrbe: fcrbe Γ ϱ ur
  assume osc: osc ur

  {
    fix α β c v

    assume c-in-C-inter-Y: c ∈ (C_𝒱 ∩ ϒ_Γ)
    assume v-in-V-inter-Nabla: v ∈ (V_𝒱 ∩ ∇_Γ)
    assume βvα-in-Tr: ((β @ [v]) @ α) ∈ Tr_(induceES SES)
    assume α-contains-no-c: α ↑ C_𝒱 = []
    assume adm: Adm 𝒱 ϱ Tr_(induceES SES) β c

    from state-from-induceES-trace[OF βvα-in-Tr] obtain s1''
      where s1''-in-S: s1'' ∈ S_SES
      and s0-βv-s1'': s0_SES (β @ [v])⟹_SES s1''
      and enabled-s1''-α: enabled SES s1'' α
      and reachable-s1'': reachable SES s1''
      by auto

    from path-split-single2[OF s0-βv-s1''] obtain s1
      where s1-in-S: s1 ∈ S_SES
      and s0-β-s1: s0_SES β⟹_SES s1
      and s1-v-s1'': s1 v⟶_SES s1''
      and reachable-s1: reachable SES s1
      by (auto)

    have ∃α' δ'.(set δ' ⊆ (N_𝒱 ∩ Δ_Γ) ∧ β @ [c] @ δ' @ [v] @ α' ∈ Tr_(induceES SES)
      ∧ α' ↑ V_𝒱 = α ↑ V_𝒱 ∧ α' ↑ C_𝒱 = [])
    proof (cases)
      assume en: En ϱ s1 c
```

from *c-in-C-inter-Y v-in-V-inter-Nabla s1-in-S s1''-in-S reachable-s1 s1-v-s1'' en fcrbe*
have $\exists\, s1' \in S_{SES}.\ \exists\, \delta.\ (\forall\, d \in (set\ \delta).\ d \in (N_{\mathcal{V}} \cap \Delta_{\Gamma}))$
$\wedge$ *s1* $([c] \mathbin{@} \delta \mathbin{@} [v]) \Longrightarrow_{SES} s1'$
$\wedge$ $(s1'', s1') \in ur$
by (*simp add*: *fcrbe-def*)
then obtain *s1' δ*
where *s1'-in-S*: $s1' \in S_{SES}$
and *δ-in-N-inter-Delta-star*: $(\forall\, d \in (set\ \delta).\ d \in (N_{\mathcal{V}} \cap \Delta_{\Gamma}))$
and *s1-cδv-s1'*: *s1* $([c] \mathbin{@} \delta \mathbin{@} [v]) \Longrightarrow_{SES} s1'$
and *s1''-ur-s1'*: $(s1'', s1') \in ur$
by (*auto*)

have *reachable-s1'*: *reachable SES s1'*
proof −
from *s0-β-s1 s1-cδv-s1'* **have** $s0_{SES}\ (\beta \mathbin{@} ([c] \mathbin{@} \delta \mathbin{@} [v])) \Longrightarrow_{SES} s1'$
by (*rule path-trans*)
thus *?thesis*
by (*simp add*: *reachable-def*, *auto*)
qed

from *osc-property*[*OF osc s1'-in-S s1''-in-S α-contains-no-c reachable-s1'*
reachable-s1'' *enabled-s1''-α s1''-ur-s1'*]
obtain *α'*
where *α'-contains-no-c*: $\alpha' \restriction C_{\mathcal{V}} = []$
and *α'-V-is-α-V*: $\alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}}$
and *enabled-s1'-α'*: *enabled SES s1' α'*
by *auto*

have *βcδvα'-in-Tr*: $\beta \mathbin{@} [c] \mathbin{@} \delta \mathbin{@} [v] \mathbin{@} \alpha' \in Tr_{(induceES\ SES)}$
proof −
let *?l1* $= \beta \mathbin{@} [c] \mathbin{@} \delta \mathbin{@} [v]$
let *?l2* $= \alpha'$
from *s0-β-s1 s1-cδv-s1'* **have** $s0_{SES}\ (?l1) \Longrightarrow_{SES} s1'$
by (*rule path-trans*)
moreover
from *enabled-s1'-α'* **obtain** *s1337* **where** $s1'\ ?l2 \Longrightarrow_{SES} s1337$
by (*simp add*: *enabled-def*, *auto*)
ultimately have $s0_{SES}\ (?l1 \mathbin{@} ?l2) \Longrightarrow_{SES} s1337$
by (*rule path-trans*)
thus *?thesis*
by (*simp add*: *induceES-def possible-traces-def enabled-def*)
qed

from *δ-in-N-inter-Delta-star βcδvα'-in-Tr α'-V-is-α-V α'-contains-no-c*
show *?thesis*
by *auto*
next
assume *not-en*: $\neg$ *En ϱ s1 c*

let *?A* $= (Adm\ \mathcal{V}\ \varrho\ Tr_{(induceES\ SES)}\ \beta\ c)$
let *?E* $= \exists\, s \in S_{SES}.\ (s0_{SES}\ \beta \Longrightarrow_{SES} s \wedge En\ \varrho\ s\ c)$

```
    {
      assume adm: ?A

      from s0-β-s1 have β-in-Tr: β ∈ Tr(induceES SES)
        by (simp add: induceES-def possible-traces-def enabled-def)

      from  β-in-Tr adm have ?E
        by (rule Adm-to-En)
    }
    hence Adm-to-En-contr: ¬ ?E ⟹ ¬ ?A
      by blast
    with s1-in-S s0-β-s1 not-en have not-adm: ¬ ?A
      by auto
    with adm show ?thesis
      by auto
  qed
 }
 thus ?thesis
  by (simp add: FCIA-def)
qed

theorem unwinding-theorem-SD:
⟦ 𝒱' = (| V = (V_𝒱 ∪ N_𝒱), N = {}, C = C_𝒱 |);
  Unwinding.lrf SES 𝒱' ur; Unwinding.osc SES 𝒱' ur ⟧
  ⟹ SD 𝒱 Tr(induceES SES)
proof −
 assume view'-def : 𝒱' = (|V = (V_𝒱 ∪ N_𝒱), N = {}, C = C_𝒱|)
 assume lrf-view' : Unwinding.lrf SES 𝒱' ur
 assume osc-view' : Unwinding.osc SES 𝒱' ur

 interpret modified-view: Unwinding SES 𝒱'
  by (unfold-locales, rule validSES, simp add: view'-def modified-view-valid)

 from lrf-view' osc-view' have BSD-view' : BSD 𝒱' Tr(induceES SES)
   by (rule-tac ur=ur in modified-view.unwinding-theorem-BSD)
 with view'-def BSD-implies-SD-for-modified-view show ?thesis
  by auto
qed

theorem unwinding-theorem-SI:
⟦ 𝒱' = (| V = (V_𝒱 ∪ N_𝒱), N = {}, C = C_𝒱 |);
  Unwinding.lrb SES 𝒱' ur; Unwinding.osc SES 𝒱' ur ⟧
  ⟹ SI 𝒱 Tr(induceES SES)
proof −
 assume view'-def : 𝒱' = (|V = V_𝒱 ∪ N_𝒱, N = {}, C = C_𝒱|)
 assume lrb-view' : Unwinding.lrb SES 𝒱' ur
 assume osc-view' : Unwinding.osc SES 𝒱' ur

 interpret modified-view: Unwinding SES 𝒱'
  by (unfold-locales, rule validSES, simp add: view'-def modified-view-valid)
```

from *lrb-view′ osc-view′* **have** *BSI-view′* : *BSI* $\mathcal{V}'$ $Tr_{(induceES\ SES)}$
by (*rule-tac ur=ur* **in** *modified-view.unwinding-theorem-BSI*)
with *view′-def BSI-implies-SI-for-modified-view* **show** *?thesis*
by *auto*
qed

theorem *unwinding-theorem-SIA*:
$[\![\ \mathcal{V}' = (\!|\ V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}),\ N = \{\},\ C = C_{\mathcal{V}}\ |\!);\ \varrho\ \mathcal{V} = \varrho\ \mathcal{V}';$
Unwinding.lrbe SES $\mathcal{V}'$ ϱ *ur*; *Unwinding.osc SES* $\mathcal{V}'$ *ur* $]\!]$
$\Longrightarrow$ *SIA* ϱ $\mathcal{V}$ $Tr_{(induceES\ SES)}$
proof −
assume *view′-def* : $\mathcal{V}' = (\!|V = V_{\mathcal{V}} \cup N_{\mathcal{V}},\ N = \{\},\ C = C_{\mathcal{V}}|\!)$
assume *ϱ-eq* : $\varrho\ \mathcal{V} = \varrho\ \mathcal{V}'$
assume *lrbe-view′* : *Unwinding.lrbe SES* $\mathcal{V}'$ ϱ *ur*
assume *osc-view′* : *Unwinding.osc SES* $\mathcal{V}'$ *ur*

interpret *modified-view*: *Unwinding SES* $\mathcal{V}'$
by (*unfold-locales*, *rule validSES*, *simp add*: *view′-def modified-view-valid*)

from *lrbe-view′ osc-view′* **have** *BSIA-view′* : *BSIA* ϱ $\mathcal{V}'$ $Tr_{(induceES\ SES)}$
by (*rule-tac ur=ur* **in** *modified-view.unwinding-theorem-BSIA*)
with *view′-def BSIA-implies-SIA-for-modified-view ϱ-eq* **show** *?thesis*
by *auto*
qed

theorem *unwinding-theorem-SR*:
$[\![\ \mathcal{V}' = (\!|\ V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}),\ N = \{\},\ C = C_{\mathcal{V}}\ |\!);$
Unwinding.lrf SES $\mathcal{V}'$ *ur*; *Unwinding.osc SES* $\mathcal{V}'$ *ur* $]\!]$
$\Longrightarrow$ *SR* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
proof −
assume *view′-def* : $\mathcal{V}' = (\!|V = V_{\mathcal{V}} \cup N_{\mathcal{V}},\ N = \{\},\ C = C_{\mathcal{V}}|\!)$
assume *lrf-view′* : *Unwinding.lrf SES* $\mathcal{V}'$ *ur*
assume *osc-view′* : *Unwinding.osc SES* $\mathcal{V}'$ *ur*

from *lrf-view′ osc-view′ view′-def* **have** *S-view* : *SD* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule-tac ur=ur* **in** *unwinding-theorem-SD*, *auto*)
with *SD-implies-SR* **show** *?thesis*
by *auto*
qed

theorem *unwinding-theorem-D*:
$[\![$ *lrf ur*; *osc ur* $]\!] \Longrightarrow$ *D* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
proof −
assume *lrf ur*
and *osc ur*
hence *BSD* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule unwinding-theorem-BSD*)
thus *?thesis*
by (*rule BSD-implies-D*)
qed

theorem *unwinding-theorem-I*:
$[\![$ *lrb ur*; *osc ur* $]\!] \Longrightarrow I\ \mathcal{V}\ Tr_{(induceES\ SES)}$
proof −
assume *lrb ur*
and *osc ur*
hence *BSI* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule unwinding-theorem-BSI*)
thus *?thesis*
by (*rule BSI-implies-I*)
qed

theorem *unwinding-theorem-IA*:
$[\![$ *lrbe* ϱ *ur*; *osc ur* $]\!] \Longrightarrow IA\ \varrho\ \mathcal{V}\ Tr_{(induceES\ SES)}$
proof −
assume *lrbe* ϱ *ur*
and *osc ur*
hence *BSIA* ϱ $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule unwinding-theorem-BSIA*)
thus *?thesis*
by (*rule BSIA-implies-IA*)
qed

theorem *unwinding-theorem-R*:
$[\![$ *lrf ur*; *osc ur* $]\!] \Longrightarrow R\ \mathcal{V}\ (Tr_{(induceES\ SES)})$
proof −
assume *lrf ur*
and *osc ur*
hence *BSD* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule unwinding-theorem-BSD*)
hence *D* $\mathcal{V}$ $Tr_{(induceES\ SES)}$
by (*rule BSD-implies-D*)
thus *?thesis*
by (*rule D-implies-R*)
qed

end

end

5.4 Compositionality

We prove the compositionality results from [3].

5.4.1 Auxiliary Definitions & Results

theory *CompositionBase*
imports *../Basics/BSPTaxonomy*
begin

definition
properSeparationOfViews ::

'e ES-rec ⇒ 'e ES-rec ⇒ 'e V-rec ⇒ 'e V-rec ⇒ 'e V-rec ⇒ bool
where
properSeparationOfViews ES1 ES2 $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ ≡
$V_{\mathcal{V}} \cap E_{ES1} = V_{\mathcal{V}1}$
$\wedge\ V_{\mathcal{V}} \cap E_{ES2} = V_{\mathcal{V}2}$
$\wedge\ C_{\mathcal{V}} \cap E_{ES1} \subseteq C_{\mathcal{V}1}$
$\wedge\ C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2}$
$\wedge\ N_{\mathcal{V}1} \cap N_{\mathcal{V}2} = \{\}$

definition
wellBehavedComposition ::
'e ES-rec ⇒ 'e ES-rec ⇒ 'e V-rec ⇒ 'e V-rec ⇒ 'e V-rec ⇒ bool
where
wellBehavedComposition ES1 ES2 $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ ≡
($N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap E_{ES1} = \{\}$)
$\vee$ ($\exists\, \varrho 1.$ ($N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2})$
$\wedge$ *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}))
$\vee$ ($\exists\, \varrho 2.$ ($N_{\mathcal{V}2} \cap E_{ES1} = \{\} \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
$\wedge$ *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2}))
$\vee$ ($\exists\, \varrho 1\ \varrho 2\ \Gamma 1\ \Gamma 2.$ (
$\nabla_{\Gamma 1} \subseteq E_{ES1} \wedge \Delta_{\Gamma 1} \subseteq E_{ES1} \wedge \Upsilon_{\Gamma 1} \subseteq E_{ES1}$
$\wedge\ \nabla_{\Gamma 2} \subseteq E_{ES2} \wedge \Delta_{\Gamma 2} \subseteq E_{ES2} \wedge \Upsilon_{\Gamma 2} \subseteq E_{ES2}$
$\wedge$ *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1} $\wedge$ *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2}
$\wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
$\wedge$ *FCIA* $\varrho 1$ $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1} $\wedge$ *FCIA* $\varrho 2$ $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2}
$\wedge\ V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma 1} \cup \nabla_{\Gamma 2}$
$\wedge\ C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma 1} \wedge C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma 2}$
$\wedge\ N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$))

locale *Compositionality* =
fixes *ES1* :: *'e ES-rec*
and *ES2* :: *'e ES-rec*
and $\mathcal{V}$ :: *'e V-rec*
and $\mathcal{V}1$ :: *'e V-rec*
and $\mathcal{V}2$ :: *'e V-rec*

assumes *validES1*: *ES-valid ES1*
and *validES2*: *ES-valid ES2*
and *composableES1ES2*: *composable ES1 ES2*

and *validVC*: *isViewOn* $\mathcal{V}$ $(E_{(ES1 \parallel ES2)})$
and *validV1*: *isViewOn* $\mathcal{V}1$ E_{ES1}
and *validV2*: *isViewOn* $\mathcal{V}2$ E_{ES2}

and *propSepViews*: *properSeparationOfViews ES1 ES2* $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$

and *well-behaved-composition*: *wellBehavedComposition ES1 ES2* $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$

sublocale *Compositionality* $\subseteq$ *BSPTaxonomyDifferentCorrections ES1* $\parallel$ *ES2* $\mathcal{V}$
by (*unfold-locales*, *rule composeES-yields-ES*, *rule validES1*,
rule validES2, *rule validVC*)

context *Compositionality*
begin

lemma *Vv-is-Vv1-union-Vv2*: $V_{\mathcal{V}} = V_{\mathcal{V}1} \cup V_{\mathcal{V}2}$
proof −
from *propSepViews* **have** $V_{\mathcal{V}} \cap E_{ES1} \cup V_{\mathcal{V}} \cap E_{ES2} = V_{\mathcal{V}1} \cup V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence $V_{\mathcal{V}} \cap (E_{ES1} \cup E_{ES2}) = V_{\mathcal{V}1} \cup V_{\mathcal{V}2}$
by *auto*
hence $V_{\mathcal{V}} \cap E_{(ES1 \parallel ES2)} = V_{\mathcal{V}1} \cup V_{\mathcal{V}2}$
by (*simp add*: *composeES-def*)
with *validVC* **show** *?thesis*
by (*simp add*: *isViewOn-def*, *auto*)
qed

lemma *disjoint-Nv1-Vv2*: $N_{\mathcal{V}1} \cap V_{\mathcal{V}2} = \{\}$
proof −
from *validV1* **have** $N_{\mathcal{V}1} \subseteq E_{ES1}$
by (*simp add*: *isViewOn-def*, *auto*)
with *propSepViews* **have** $N_{\mathcal{V}1} \cap V_{\mathcal{V}2} = (N_{\mathcal{V}1} \cap E_{ES1} \cap V_{\mathcal{V}}) \cap E_{ES2}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence $N_{\mathcal{V}1} \cap V_{\mathcal{V}2} = (N_{\mathcal{V}1} \cap V_{\mathcal{V}} \cap E_{ES1}) \cap E_{ES2}$
by *auto*
moreover
from *validV1* **have** $N_{\mathcal{V}1} \cap V_{\mathcal{V}} \cap E_{ES1} = \{\}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis VN-disjoint-def V-valid-def inf-assoc inf-commute isViewOn-def*)
ultimately show *?thesis*
by *auto*
qed

lemma *disjoint-Nv2-Vv1*: $N_{\mathcal{V}2} \cap V_{\mathcal{V}1} = \{\}$
proof −
from *validV2* **have** $N_{\mathcal{V}2} \subseteq E_{ES2}$
by (*simp add*:*isViewOn-def*, *auto*)
with *propSepViews* **have** $N_{\mathcal{V}2} \cap V_{\mathcal{V}1} = (N_{\mathcal{V}2} \cap E_{ES2} \cap V_{\mathcal{V}}) \cap E_{ES1}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence $N_{\mathcal{V}2} \cap V_{\mathcal{V}1} = (N_{\mathcal{V}2} \cap V_{\mathcal{V}} \cap E_{ES2}) \cap E_{ES1}$
by *auto*
moreover

from *validV2* **have** $N_{\mathcal{V}2} \cap V_{\mathcal{V}} \cap E_{ES2} = \{\}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis VN-disjoint-def V-valid-def inf-assoc inf-commute isViewOn-def*)
ultimately show *?thesis*
by *auto*
qed

lemma *merge-property'*: $[\![$ *set t1* $\subseteq E_{ES1}$; *set t2* $\subseteq E_{ES2}$;
t1 $\restriction E_{ES2} = t2 \restriction E_{ES1}$; *t1* $\restriction V_{\mathcal{V}} = []$; *t2* $\restriction V_{\mathcal{V}} = []$;
t1 $\restriction C_{\mathcal{V}} = []$; *t2* $\restriction C_{\mathcal{V}} = []]\!]$
$\Longrightarrow \exists\ t.\ (t \restriction E_{ES1} = t1 \wedge t \restriction E_{ES2} = t2 \wedge t \restriction V_{\mathcal{V}} = [] \wedge t \restriction C_{\mathcal{V}} = [] \wedge$ *set t* $\subseteq (E_{ES1} \cup E_{ES2}))$
proof −
assume *t1-in-E1star*: *set t1* $\subseteq E_{ES1}$
and *t2-in-E2star*: *set t2* $\subseteq E_{ES2}$
and *t1-t2-synchronized*: *t1* $\restriction E_{ES2} = t2 \restriction E_{ES1}$
and *t1Vv-empty*: *t1* $\restriction V_{\mathcal{V}} = []$
and *t2Vv-empty*: *t2* $\restriction V_{\mathcal{V}} = []$
and *t1Cv-empty*: *t1* $\restriction C_{\mathcal{V}} = []$
and *t2Cv-empty*: *t2* $\restriction C_{\mathcal{V}} = []$

from *merge-property*[*OF t1-in-E1star t2-in-E2star t1-t2-synchronized*] **obtain** *t*
where *t-is-interleaving*: *t* $\restriction E_{ES1} = t1 \wedge t \restriction E_{ES2} = t2$
and *t-contains-only-events-from-t1-t2*: *set t* $\subseteq$ *set t1* $\cup$ *set t2*
unfolding *Let-def*
by *auto*
moreover
from *t1Vv-empty t2Vv-empty t-contains-only-events-from-t1-t2*
have *t* $\restriction V_{\mathcal{V}} = []$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute Vv-is-Vv1-union-Vv2 projection-on-union projection-sequence t-is-interleaving*)
moreover
have *t* $\restriction C_{\mathcal{V}} = []$
proof −
from *t1Cv-empty* **have** $\forall\, c \in C_{\mathcal{V}}.\ c \notin$ *set t1*
by (*simp add*: *projection-def filter-empty-conv*, *fast*)
moreover
from *t2Cv-empty* **have** $\forall\, c \in C_{\mathcal{V}}.\ c \notin$ *set t2*
by (*simp add*: *projection-def filter-empty-conv*, *fast*)
ultimately have
$\forall\, c \in C_{\mathcal{V}}.\ c \notin$ (*set t1* $\cup$ *set t2*)
by *auto*
with *t-contains-only-events-from-t1-t2* **have** $\forall\, c \in C_{\mathcal{V}}.\ c \notin$ *set t*
by *auto*
thus *?thesis*
by (*simp add*: *projection-def*, *metis filter-empty-conv*)
qed
moreover
from *t1-in-E1star t2-in-E2star t-contains-only-events-from-t1-t2*
have *set t* $\subseteq (E_{ES1} \cup E_{ES2})$
by *auto*
ultimately show *?thesis*

```
    by blast
qed

lemma Nv1-union-Nv2-subsetof-Nv: N_V1 ∪ N_V2 ⊆ N_V
proof −
  {
    fix e
    assume e-in-N1: e ∈ N_V1
    with validV1 have
      e-in-E1: e ∈ E_ES1
      and e-notin-V1: e ∉ V_V1
      and e-notin-C1: e ∉ C_V1
      by (simp only: isViewOn-def V-valid-def VC-disjoint-def NC-disjoint-def
        VN-disjoint-def, auto)+

    from e-in-E1 e-notin-V1 propSepViews have e ∉ V_V
     unfolding properSeparationOfViews-def by auto
    moreover
    from e-in-E1 e-notin-C1 propSepViews have e ∉ C_V
     unfolding properSeparationOfViews-def by auto
    moreover
    note e-in-E1 validVC
    ultimately have e ∈ N_V
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def NC-disjoint-def VN-disjoint-def
        composeES-def, auto)
  }
  moreover {
    fix e
    assume e-in-N2: e ∈ N_V2
    with validV2 have
      e-in-E2: e ∈ E-ES ES2
      and e-notin-V2: e ∉ V_V2
      and e-notin-C2: e ∉ C_V2
      by (simp only: isViewOn-def V-valid-def VC-disjoint-def NC-disjoint-def VN-disjoint-def
        , auto)+

    from e-in-E2 e-notin-V2 propSepViews have e ∉ V_V
     unfolding properSeparationOfViews-def by auto
    moreover
    from e-in-E2 e-notin-C2 propSepViews have e ∉ C_V
     unfolding properSeparationOfViews-def by auto
    moreover
    note e-in-E2 validVC
    ultimately have e ∈ N_V
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def
        composeES-def, auto)
  }
  ultimately show ?thesis
    by auto
qed

end
```

end
theory *CompositionSupport*
imports *CompositionBase*
begin

locale *CompositionSupport* =
fixes *ESi* :: *'e ES-rec*
and $\mathcal{V}$ :: *'e V-rec*
and $\mathcal{V}i$ :: *'e V-rec*

assumes *validESi*: *ES-valid ESi*

and *validVi*: *isViewOn* $\mathcal{V}i$ E_{ESi}
and *Vv-inter-Ei-is-Vvi*: $V_{\mathcal{V}} \cap E_{ESi} = V_{\mathcal{V}i}$
and *Cv-inter-Ei-subsetof-Cvi*: $C_{\mathcal{V}} \cap E_{ESi} \subseteq C_{\mathcal{V}i}$

context *CompositionSupport*
begin

lemma *BSD-in-subsystem*:
$[\![\ c \in C_{\mathcal{V}};\ ((\beta\ @\ [c]\ @\ \alpha) \restriction E_{ESi}) \in Tr_{ESi}\ ;\ BSD\ \mathcal{V}i\ Tr_{ESi}\ ]\!]$
$\Longrightarrow \exists\, \alpha\text{-}i'.\ (\ ((\beta \restriction E_{ESi})\ @\ \alpha\text{-}i') \in Tr_{ESi}$
$\wedge\ (\alpha\text{-}i' \restriction V_{\mathcal{V}i}) = (\alpha \restriction V_{\mathcal{V}i}) \wedge \alpha\text{-}i' \restriction C_{\mathcal{V}i} = [])$
proof (*induct length* $(([c]\ @\ \alpha) \restriction C_{\mathcal{V}i})$ *arbitrary*: $\beta\ c\ \alpha$)
case *0*

let *?L* = $([c]\ @\ \alpha) \restriction E_{ESi}$

from *0*(*3*) **have** *β-E1-cα-E1-in-Tr1*: $((\beta \restriction E_{ESi})\ @\ (([c]\ @\ \alpha) \restriction E_{ESi})) \in Tr_{ESi}$
by (*simp only*: *projection-concatenation-commute*)
moreover
have $(?L \restriction V_{\mathcal{V}i}) = (\alpha \restriction V_{\mathcal{V}i})$
proof −
have $(?L \restriction V_{\mathcal{V}i}) = ([c]\ @\ \alpha) \restriction V_{\mathcal{V}i}$
proof −
from *validVi* **have** $E_{ESi} \cap V_{\mathcal{V}i} = V_{\mathcal{V}i}$
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def VC-disjoint-def NC-disjoint-def*
, *auto*)
moreover
have $(?L \restriction V_{\mathcal{V}i}) = ([c]\ @\ \alpha) \restriction (E_{ESi} \cap V_{\mathcal{V}i})$
by (*simp add*: *projection-def*)
ultimately show *?thesis*
by *auto*
qed
moreover

```
    have ([c] @ α) ↾ V_{V i} = α ↾ V_{V i}
    proof −
      have ([c] @ α) ↾ V_{V i} = ([c] ↾ V_{V i}) @ (α ↾ V_{V i})
        by (rule projection-concatenation-commute)
      moreover
      have ([c] ↾ V_{V i}) = []
      proof −
        from 0(2) have [c] ↾ C_V = [c]
          by (simp add: projection-def)
        moreover
        have [c] ↾ C_V ↾ V_{V i} = []
        proof −
          from validVi Cv-inter-Ei-subsetof-Cvi have C_V ∩ V_{V i} ⊆ C_{V i}
            by (simp add: isViewOn-def  V-valid-def VC-disjoint-def, auto)
          moreover
          from 0(1) have [c] ↾ C_{V i} = []
            by (simp only: projection-concatenation-commute, auto)
          ultimately have [c] ↾ (C_V ∩ V_{V i}) = []
            by (rule projection-on-subset)
          thus ?thesis
            by (simp only: projection-def, auto)
        qed
        ultimately show ?thesis
          by auto
      qed
      ultimately show ?thesis
        by auto
    qed
    ultimately show ?thesis
      by auto
  qed
  moreover
  have ?L ↾ C_{V i} = []
  proof −
    from 0(1) have ([c] @ α) ↾ C_{V i} = []
      by auto
    hence ([c] @ α) ↾ (C_{V i} ∩ E_{ESi}) = []
      by (rule projection-on-intersection)
    hence ([c] @ α) ↾ (E_{ESi} ∩ C_{V i}) = []
      by (simp only: Int-commute)
    thus ?thesis
      by (simp only: projection-def, auto)
  qed
  ultimately show ?case
    by auto

next
  case (Suc n)

  from projection-split-last[OF Suc(2)] obtain γ c-i δ
    where c-i-in-CVi: c-i ∈ C_{V i}
    and   cα-is-γc-iδ: [c] @ α = γ @ [c-i] @ δ
```

and δ-no-$C\mathcal{V}i$: $\delta \restriction C_{\mathcal{V}i} = []$
and n-is-len-$\gamma\delta$-$C\mathcal{V}i$: $n = length\ ((\gamma\ @\ \delta) \restriction C_{\mathcal{V}i})$
by *auto*

let *?L1* $= ((\beta\ @\ \gamma) \restriction E_{ESi})$
let *?L2* $= (\delta \restriction E_{ESi})$

note c-i-in-$C\mathcal{V}i$
moreover
have list-with-c-i-in-Tr1: $(?L1\ @\ [c\text{-}i]\ @\ ?L2) \in Tr_{ESi}$
proof −
from c-i-in-$C\mathcal{V}i$ validVi **have** $[c\text{-}i] \restriction E_{ESi} = [c\text{-}i]$
by (*simp only*: isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def projection-def, *auto*)
moreover
from Suc(4) cα-is-γc-iδ **have** $((\beta\ @\ \gamma\ @\ [c\text{-}i]\ @\ \delta) \restriction E_{ESi}) \in Tr_{ESi}$
by *auto*
hence $(?L1\ @\ ([c\text{-}i] \restriction E_{ESi})\ @\ ?L2) \in Tr_{ESi}$
by (*simp only*: projection-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
moreover
have $?L2 \restriction C_{\mathcal{V}i} = []$
proof −
from validVi **have** $\bigwedge x.\ (x \in E_{ESi} \wedge x \in C_{\mathcal{V}i}) = (x \in C_{\mathcal{V}i})$
by (*simp add*: isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
with δ-no-$C\mathcal{V}i$ **show** *?thesis*
by (*simp add*: projection-def)
qed
moreover note Suc(5)
ultimately obtain δ'
where δ'-1: $(?L1\ @\ \delta') \in Tr_{ESi}$
and δ'-2: $\delta' \restriction V_{\mathcal{V}i} = ?L2 \restriction V_{\mathcal{V}i}$
and δ'-3: $\delta' \restriction C_{\mathcal{V}i} = []$
unfolding BSD-def
by *blast*
hence δ'-2′: $\delta' \restriction V_{\mathcal{V}i} = \delta \restriction V_{\mathcal{V}i}$
proof −
have $?L2 \restriction V_{\mathcal{V}i} = \delta \restriction V_{\mathcal{V}i}$
proof −
from validVi **have** $\bigwedge x.\ (x \in E_{ESi} \wedge x \in V_{\mathcal{V}i}) = (x \in V_{\mathcal{V}i})$
by (*simp add*: isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*simp add*: projection-def)
qed
with δ'-2 **show** *?thesis*
by *auto*
qed

show *?case*
proof (*cases* γ)
case *Nil*
with *cα-is-γc-iδ* **have** $[c] \mathbin{@} \alpha = [c\text{-}i] \mathbin{@} \delta$
by *auto*
hence *δ-is-α*: $\delta = \alpha$
by *auto*

from *δ′-1* **have** *δ′-1′*: $((\beta \restriction E_{ESi}) \mathbin{@} \delta') \in Tr_{ESi}$
by (*simp only*: *Nil*, *auto*)
moreover
note *δ′-2′*
moreover note *δ′-3*
ultimately show *?thesis*
by (*simp only*: *δ-is-α*, *auto*)
next
case (*Cons x γ′*)
with *cα-is-γc-iδ* **have** *γ-is-cγ′*: $\gamma = [c] \mathbin{@} \gamma'$
by *simp*
with *n-is-len-γδ-CVi* **have** $n = length\ (([c] \mathbin{@} \gamma' \mathbin{@} \delta) \restriction C_{\mathcal{V}i})$
by *auto*
with *δ-no-CVi δ′-3* **have** $n = length\ (([c] \mathbin{@} \gamma' \mathbin{@} \delta') \restriction C_{\mathcal{V}i})$
by (*simp only*: *projection-concatenation-commute*)
moreover
note *Suc*(*3*)
moreover
have $((\beta \mathbin{@} [c] \mathbin{@} (\gamma' \mathbin{@} \delta')) \restriction E_{ESi}) \in Tr_{ESi}$
proof −
from *δ′-1 validESi* **have** $\delta' = \delta' \restriction E_{ESi}$
proof −
let *?L* = $(\beta \mathbin{@} \gamma) \restriction E_{ESi} \mathbin{@} \delta'$

from *δ′-1 validESi* **have** $\forall e \in set\ ?L.\ e \in E_{ESi}$
by (*simp add*: *ES-valid-def traces-contain-events-def*)
hence *set* $\delta' \subseteq E_{ESi}$
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
with *δ′-1* **have** *?L1* $\mathbin{@} \delta' = (\beta \mathbin{@} \gamma \mathbin{@} \delta') \restriction E_{ESi}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
with *γ-is-cγ′ δ′-1* **show** *?thesis*
by *auto*
qed
moreover
note *Suc*(*5*)
moreover note *Suc*(*1*)[*of c γ′* @ *δ′ β*]
ultimately obtain *α-i′*
where *α-i′-1*: $\beta \restriction E_{ESi} \mathbin{@} \alpha\text{-}i' \in Tr_{ESi}$
and *α-i′-2*: $\alpha\text{-}i' \restriction V_{\mathcal{V}i} = (\gamma' \mathbin{@} \delta') \restriction V_{\mathcal{V}i}$
and *α-i′-3*: $\alpha\text{-}i' \restriction C_{\mathcal{V}i} = []$
by *auto*

```
    moreover
    have α-i' ↾ V_{V i} = α ↾ V_{V i}
    proof −
      have α ↾ V_{V i} = (γ' @ δ) ↾ V_{V i}
      proof −
        from cα-is-γc-iδ γ-is-cγ' have α ↾ V_{V i} = (γ' @ [c-i] @ δ) ↾ V_{V i}
          by simp
        with validVi c-i-in-CVi show ?thesis
          by (simp only: isViewOn-def V-valid-def  VC-disjoint-def
            VN-disjoint-def NC-disjoint-def projection-concatenation-commute
            projection-def, auto)
      qed
      moreover
      from α-i'-2 δ'-2' have α-i' ↾ V_{V i} = (γ' @ δ) ↾ V_{V i}
        by (simp only: projection-concatenation-commute)
      ultimately show ?thesis
        by auto
    qed
    ultimately show ?thesis
      by auto
  qed
qed


lemma BSD-in-subsystem2:
⟦ ((β @ α) ↾ E_{ESi}) ∈ Tr_{ESi} ; BSD V i Tr_{ESi} ⟧
  ⟹ ∃ α-i'. ( ((β ↾ E_{ESi}) @ α-i') ∈ Tr_{ESi} ∧ (α-i' ↾ V_{V i}) = (α ↾ V_{V i}) ∧ α-i' ↾ C_{V i} = [] )
proof (induct length (α ↾ C_{V i}) arbitrary: β α)
  case 0

  let ?L = α ↾ E_{ESi}

  from 0(2) have β-E1-α-E1-in-Tr1: ((β ↾ E_{ESi}) @ ?L) ∈ Tr_{ESi}
    by (simp only: projection-concatenation-commute)
  moreover
  have (?L ↾ V_{V i}) = (α ↾ V_{V i})
    proof −
      from validVi have E_{ESi} ∩ V_{V i} = V_{V i}
        by (simp add: isViewOn-def V-valid-def  VC-disjoint-def
        VN-disjoint-def NC-disjoint-def, auto)
      moreover
      have (?L ↾ V_{V i}) = α ↾ (E_{ESi} ∩ V_{V i})
        by (simp add: projection-def)
      ultimately show ?thesis
        by auto
    qed
  moreover
  have ?L ↾ C_{V i} = []
  proof −
    from 0(1) have α ↾ C_{V i} = []
      by auto
    hence α ↾ (C_{V i} ∩ E_{ESi}) = []
```

```
    by (rule projection-on-intersection)
  hence α ↾ (E_ESi ∩ C_Vi) = []
    by (simp only: Int-commute)
  thus ?thesis
    by (simp only: projection-def, auto)
 qed
 ultimately show ?case
  by auto

next
 case (Suc n)

 from projection-split-last[OF Suc(2)] obtain γ c-i δ
  where c-i-in-CVi: c-i ∈ C_Vi
  and   α-is-γc-iδ: α = γ @ [c-i] @ δ
  and   δ-no-CVi:  δ ↾ C_Vi = []
  and   n-is-len-γδ-CVi: n = length ((γ @ δ) ↾ C_Vi)
  by auto

 let ?L1 = ((β @ γ) ↾ E_ESi)
 let ?L2 = (δ ↾ E_ESi)



 note c-i-in-CVi
 moreover
 have list-with-c-i-in-Tr1: (?L1 @ [c-i] @ ?L2) ∈ Tr_ESi
 proof −
  from c-i-in-CVi validVi have [c-i] ↾ E_ESi = [c-i]
    by (simp only: isViewOn-def V-valid-def  VC-disjoint-def
      VN-disjoint-def NC-disjoint-def projection-def, auto)
  moreover
  from Suc(3) α-is-γc-iδ have ((β @ γ @ [c-i] @ δ) ↾ E_ESi) ∈ Tr_ESi
    by auto
  hence (?L1 @ ([c-i] ↾ E_ESi) @ ?L2) ∈ Tr_ESi
    by (simp only: projection-def, auto)
  ultimately show ?thesis
    by auto
 qed
 moreover
 have ?L2 ↾ C_Vi = []
 proof −
  from validVi have ⋀x. (x ∈ E_ESi ∧ x ∈ C_Vi) = (x ∈ C_Vi)
    by (simp add: isViewOn-def V-valid-def  VC-disjoint-def
      VN-disjoint-def NC-disjoint-def, auto)
  with δ-no-CVi show ?thesis
    by (simp add: projection-def)
 qed
 moreover note Suc(4)
 ultimately obtain δ'
  where δ'-1: (?L1 @ δ') ∈ Tr_ESi
  and δ'-2: δ' ↾ V_Vi = ?L2 ↾ V_Vi
```

```
  and δ'-3: δ' ↑ C_Vi = []
  unfolding BSD-def
  by blast
hence δ'-2': δ' ↑ V_Vi = δ ↑ V_Vi
proof −
  have ?L2 ↑ V_Vi = δ ↑ V_Vi
  proof −
    from validVi have ⋀x. (x ∈ E_ESi ∧ x ∈ V_Vi) = (x ∈ V_Vi)
      by (simp add: isViewOn-def V-valid-def  VC-disjoint-def
      VN-disjoint-def NC-disjoint-def, auto)
    thus ?thesis
      by (simp add: projection-def)
  qed
  with δ'-2 show ?thesis
    by auto
qed


from n-is-len-γδ-CVi δ-no-CVi δ'-3 have n = length ((γ @ δ') ↑ C_Vi)
  by (simp add: projection-concatenation-commute)
moreover
have (β @ (γ @ δ')) ↑ E_ESi ∈ Tr_ESi
  proof −
    have δ' = δ' ↑ E_ESi
      proof −
        let ?L = (β @ γ) ↑ E_ESi @ δ'

        from δ'-1 validESi have ∀ e ∈ set ?L. e ∈ E_ESi
          by (simp add: ES-valid-def traces-contain-events-def)
        hence set δ' ⊆ E_ESi
          by auto
        thus ?thesis
          by (simp add: list-subset-iff-projection-neutral)
      qed
    with δ'-1  have ?L1 @ δ' = (β @ γ @ δ') ↑ E_ESi
      by (simp only: projection-concatenation-commute, auto)
    with δ'-1  show ?thesis
      by auto
  qed
moreover
note Suc(4) Suc(1)[of γ @ δ' β]
ultimately obtain α-i'
  where res1: β ↑ E_ESi @ α-i' ∈ Tr_ESi
  and res2: α-i' ↑ V_Vi = (γ @ δ') ↑ V_Vi
  and res3: α-i' ↑ C_Vi = []
  by auto


have α-i' ↑ V_Vi = α ↑ V_Vi
  proof −
    from c-i-in-CVi validVi have [c-i] ↑ V_Vi = []
      by (simp add: isViewOn-def V-valid-def  VC-disjoint-def
```

VN-disjoint-def NC-disjoint-def projection-def, *auto*)
with α-*is-*γ*c-i*δ δ'*-2'* **have** $\alpha \restriction V_{\mathcal{V}i} = (\gamma \mathbin{@} \delta') \restriction V_{\mathcal{V}i}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
with *res2* **show** *?thesis*
by *auto*
qed
with *res1 res3* **show** *?case*
by *auto*
qed

end

end

5.4.2 Generalized Zipping Lemma

theory *GeneralizedZippingLemma*
imports *CompositionBase*
begin

context *Compositionality*
begin

lemma *generalized-zipping-lemma1*: $[\![N_{\mathcal{V}1} \cap E_{ES2} = \{\}; N_{\mathcal{V}2} \cap E_{ES1} = \{\}]\!] \Longrightarrow$
$\forall\ \tau\ lambda\ t1\ t2.\ (\ (\ set\ \tau \subseteq E_{(ES1 \parallel ES2)} \wedge set\ lambda \subseteq V_{\mathcal{V}} \wedge set\ t1 \subseteq E_{ES1} \wedge set\ t2 \subseteq E_{ES2}$
$\wedge\ ((\tau \restriction E_{ES1}) \mathbin{@} t1) \in Tr_{ES1} \wedge ((\tau \restriction E_{ES2}) \mathbin{@} t2) \in Tr_{ES2} \wedge (lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$
$\wedge\ (lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}}) \wedge (t1 \restriction C_{\mathcal{V}1}) = [] \wedge (t2 \restriction C_{\mathcal{V}2}) = [])$
$\longrightarrow (\exists\ t.\ ((\tau \mathbin{@} t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = [])))$)
proof −
assume *Nv1-inter-E2-empty*: $N_{\mathcal{V}1} \cap E_{ES2} = \{\}$
and *Nv2-inter-E1-empty*: $N_{\mathcal{V}2} \cap E_{ES1} = \{\}$

{
fix τ *lambda t1 t2*
assume τ*-in-Estar*: *set* $\tau \subseteq E_{(ES1 \parallel ES2)}$
and *lambda-in-Vvstar*: *set lambda* $\subseteq V_{\mathcal{V}}$
and *t1-in-E1star*: *set t1* $\subseteq E_{ES1}$
and *t2-in-E2star*: *set t2* $\subseteq E_{ES2}$
and τ*-E1-t1-in-Tr1*: $((\tau \restriction E_{ES1}) \mathbin{@} t1) \in Tr_{ES1}$
and τ*-E2-t2-in-Tr2*: $((\tau \restriction E_{ES2}) \mathbin{@} t2) \in Tr_{ES2}$
and *lambda-E1-is-t1-Vv*: $(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$
and *lambda-E2-is-t2-Vv*: $(lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$
and *t1-no-Cv1*: $(t1 \restriction C_{\mathcal{V}1}) = []$
and *t2-no-Cv2*: $(t2 \restriction C_{\mathcal{V}2}) = []$

have $[\![$ *set* $\tau \subseteq E_{(ES1 \parallel ES2)}$;
set lambda $\subseteq V_{\mathcal{V}}$;
set t1 $\subseteq E_{ES1}$;
set t2 $\subseteq E_{ES2}$;

$((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1}$;
$((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$;
$(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$;
$(lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$;
$(t1 \restriction C_{\mathcal{V}1}) = []$;
$(t2 \restriction C_{\mathcal{V}2}) = []$ ⟧
$\Longrightarrow (\exists\ t.\ ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = []))$
proof (*induct lambda arbitrary*: τ *t1 t2*)
case (*Nil* τ *t1 t2*)

have $(\tau @ []) \in Tr_{(ES1 \parallel ES2)}$
proof −
have $\tau \in Tr_{(ES1 \parallel ES2)}$
proof −
from *Nil*(*5*) *validES1* **have** $\tau \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
moreover
from *Nil*(*6*) *validES2* **have** $\tau \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
moreover
note *Nil*(*1*)
ultimately show *?thesis*
by (*simp add*: *composeES-def*)
qed
thus *?thesis*
by *auto*
qed
moreover
have $([] \restriction V_{\mathcal{V}}) = []$
by (*simp add*: *projection-def*)
moreover
have $([] \restriction C_{\mathcal{V}}) = []$
by (*simp add*: *projection-def*)
ultimately show *?case*
by *blast*
next
case (*Cons* $\mathcal{V}'$ *lambda'* τ *t1 t2*)
thus *?case*
proof −
from *Cons*(*3*) **have** *v'-in-Vv*: $\mathcal{V}' \in V_{\mathcal{V}}$
by *auto*

have $\mathcal{V}' \in V_{\mathcal{V}1} \cap V_{\mathcal{V}2}$
$\vee\ \mathcal{V}' \in V_{\mathcal{V}1} - E_{ES2}$
$\vee\ \mathcal{V}' \in V_{\mathcal{V}2} - E_{ES1}$
using *Vv-is-Vv1-union-Vv2 v'-in-Vv propSepViews*
unfolding *properSeparationOfViews-def*
by *fastforce*
moreover {
assume *v'-in-Vv1-inter-Vv2*: $\mathcal{V}' \in V_{\mathcal{V}1} \cap V_{\mathcal{V}2}$

hence *v′-in-Vv1*: $\mathcal{V}' \in V_{\mathcal{V}1}$ **and** *v′-in-Vv2*: $\mathcal{V}' \in V_{\mathcal{V}2}$
by *auto*
with *v′-in-Vv propSepViews*
have *v′-in-E1*: $\mathcal{V}' \in E_{ES1}$ **and** *v′-in-E2*: $\mathcal{V}' \in E_{ES2}$
unfolding *properSeparationOfViews-def* **by** *auto*

from *Cons*(*2*,*4*,*8*) *v′-in-E1* **have** *t1* ↾ $V_{\mathcal{V}} = \mathcal{V}'$ # (*lambda′* ↾ E_{ES1})
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r1 s1*
where *t1-is-r1-v′-s1*: *t1* = *r1* @ [$\mathcal{V}'$] @ *s1*
and *r1-Vv-empty*: *r1* ↾ $V_{\mathcal{V}}$ = []
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}1}$ $V_{\mathcal{V}}$ *r1*]
have *r1-Vv1-empty*: *r1* ↾ $V_{\mathcal{V}1}$ = []
by *auto*

from *Cons*(*3*,*5*,*9*) *v′-in-E2* **have** *t2* ↾ $V_{\mathcal{V}}$ = $\mathcal{V}'$ # (*lambda′* ↾ E_{ES2})
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r2 s2*
where *t2-is-r2-v′-s2*: *t2* = *r2* @ [$\mathcal{V}'$] @ *s2*
and *r2-Vv-empty*: *r2* ↾ $V_{\mathcal{V}}$ = []
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: *r2* ↾ $V_{\mathcal{V}2}$ = []
by *auto*

from *t1-is-r1-v′-s1 Cons*(*10*) **have** *r1-Cv1-empty*: *r1* ↾ $C_{\mathcal{V}1}$ = []
by (*simp add*: *projection-concatenation-commute*)

from *t1-is-r1-v′-s1 Cons*(*10*) **have** *s1-Cv1-empty*: *s1* ↾ $C_{\mathcal{V}1}$ = []
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*4*) *t1-is-r1-v′-s1* **have** *r1-in-E1star*: *set r1* ⊆ E_{ES1}
and *s1-in-E1star*: *set s1* ⊆ E_{ES1}
by *auto*

from *Cons*(*6*) *t1-is-r1-v′-s1*
have *τE1-r1-v′-s1-in-Tr1*: *τ* ↾ E_{ES1} @ *r1* @ [$\mathcal{V}'$] @ *s1* ∈ Tr_{ES1}
by *simp*

have *r1-in-Nv1star*: *set r1* ⊆ $N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** *set r1* ∩ $V_{\mathcal{V}1}$ = {}
by (*metis Compl-Diff-eq Diff-cancel Diff-eq Int-commute*
Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
from *r1-Cv1-empty* **have** *set r1* ∩ $C_{\mathcal{V}1}$ = {}

by (*metis Compl-Diff-eq Diff-cancel Diff-eq Int-commute*
Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv1-inter-E2-empty* **have** *r1E2-empty*: $r1 \upharpoonright E_{ES2} = []$
by (*metis Int-commute empty-subsetI projection-on-subset2 r1-Vv-empty*)

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *r2-Cv2-empty*: $r2 \upharpoonright C_{\mathcal{V}2} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *s2-Cv2-empty*: $s2 \upharpoonright C_{\mathcal{V}2} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*5*) *t2-is-r2-v'-s2* **have** *r2-in-E2star*: *set r2* $\subseteq E_{ES2}$
and *s2-in-E2star*: *set s2* $\subseteq E_{ES2}$
by *auto*

from *Cons*(*7*) *t2-is-r2-v'-s2*
have *τE2-r2-v'-s2-in-Tr2*: $\tau \upharpoonright E_{ES2} @ r2 @ [\mathcal{V}'] @ s2 \in Tr_{ES2}$
by *simp*

have *r2-in-Nv2star*: *set r2* $\subseteq N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** *set r2* $\cap V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r2-Cv2-empty* **have** *set r2* $\cap C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv2-inter-E1-empty* **have** *r2E1-empty*: $r2 \upharpoonright E_{ES1} = []$
by (*metis Int-commute empty-subsetI projection-on-subset2 r2-Vv-empty*)

let *?tau* = $\tau @ r1 @ r2 @ [\mathcal{V}']$

from *Cons*(*2*) *r1-in-E1star r2-in-E2star v'-in-E2*
have *set ?tau* $\subseteq (E_{(ES1 \parallel ES2)})$

by (*simp add*: *composeES-def*, *auto*)
moreover
from *Cons*(*3*) **have** *set lambda'* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
note *s1-in-E1star s2-in-E2star*
moreover
from *Cons*(*6*) *r1-in-E1star r2E1-empty v'-in-E1 t1-is-r1-v'-s1*
have ((*?tau* $\restriction$ E_{ES1}) @ *s1*) $\in$ Tr_{ES1}
by (*simp only*: *projection-concatenation-commute*
list-subset-iff-projection-neutral projection-def, *auto*)
moreover
from *Cons*(*7*) *r2-in-E2star r1E2-empty v'-in-E2 t2-is-r2-v'-s2*
have ((*?tau* $\restriction$ E_{ES2}) @ *s2*) $\in$ Tr_{ES2}
by (*simp only*: *projection-concatenation-commute*
list-subset-iff-projection-neutral projection-def, *auto*)
moreover
have *lambda'* $\restriction$ E_{ES1} = *s1* $\restriction$ $V_{\mathcal{V}}$
proof −
from *Cons*(*2*,*4*,*8*) *v'-in-E1* **have** *t1* $\restriction$ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*lambda'* $\restriction$ E_{ES1})
by (*simp add*: *projection-def*)
moreover
from *t1-is-r1-v'-s1 r1-Vv-empty v'-in-Vv1 Vv-is-Vv1-union-Vv2*
have *t1* $\restriction$ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*s1* $\restriction$ $V_{\mathcal{V}}$)
by (*simp only*: *t1-is-r1-v'-s1 projection-concatenation-commute*
projection-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
moreover
have *lambda'* $\restriction$ E_{ES2} = *s2* $\restriction$ $V_{\mathcal{V}}$
proof −
from *Cons*(*3*,*5*,*9*) *v'-in-E2* **have** *t2* $\restriction$ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*lambda'* $\restriction$ E_{ES2})
by (*simp add*: *projection-def*)
moreover
from *t2-is-r2-v'-s2 r2-Vv-empty v'-in-Vv2 Vv-is-Vv1-union-Vv2*
have *t2* $\restriction$ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*s2* $\restriction$ $V_{\mathcal{V}}$)
by (*simp only*: *t2-is-r2-v'-s2 projection-concatenation-commute*
projection-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
moreover
note *s1-Cv1-empty s2-Cv2-empty Cons.hyps*(*1*)[*of ?tau s1 s2*]
ultimately obtain *t'*
where *tau-t'-in-Tr*: *?tau* @ *t'* $\in$ $Tr_{(ES1 \parallel ES2)}$
and *t'Vv-is-lambda'*: *t'* $\restriction$ $V_{\mathcal{V}}$ = *lambda'*
and *t'Cv-empty*: *t'* $\restriction$ $C_{\mathcal{V}}$ = []
by *auto*

let *?t* = *r1* @ *r2* @ [$\mathcal{V}'$] @ *t'*

note *tau-t′-in-Tr*
moreover
from *r1-Vv-empty r2-Vv-empty t′Vv-is-lambda′ v′-in-Vv*
have *?t* ↾ $V_{\mathcal{V}} = \mathcal{V}' \# lambda'$
by (*simp add*: *projection-def*)
moreover
have *?t* ↾ $C_{\mathcal{V}} = []$
proof −
from *propSepViews* **have** $C_{\mathcal{V}} \cap E_{ES1} \subseteq C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence *r1* ↾ $C_{\mathcal{V}} = []$
by (*metis projection-on-subset2 r1-Cv1-empty r1-in-E1star*)
moreover
from *propSepViews* **have** $C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence *r2* ↾ $C_{\mathcal{V}} = []$
by (*metis projection-on-subset2 r2-Cv2-empty r2-in-E2star*)
moreover
note *v′-in-Vv VIsViewOnE t′Cv-empty*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def projection-def*, *auto*)
qed
ultimately have *?thesis*
by *auto*
}
moreover {
assume *v′-in-Vv1-minus-E2*: $\mathcal{V}' \in V_{\mathcal{V}1} - E_{ES2}$
hence *v′-in-Vv1*: $\mathcal{V}' \in V_{\mathcal{V}1}$
by *auto*
with *v′-in-Vv propSepViews* **have** *v′-in-E1*: $\mathcal{V}' \in E_{ES1}$
unfolding *properSeparationOfViews-def*
by *auto*

from *v′-in-Vv1-minus-E2* **have** *v′-notin-E2*: $\mathcal{V}' \notin E_{ES2}$
by (*auto*)
with *validV2* **have** *v′-notin-Vv2*: $\mathcal{V}' \notin V_{\mathcal{V}2}$
by (*simp add*: *isViewOn-def V-valid-def*, *auto*)

from *Cons(3) Cons(4) Cons(8) v′-in-E1* **have** *t1* ↾ $V_{\mathcal{V}} = \mathcal{V}' \# (lambda' \restriction E_{ES1})$
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r1 s1*
where *t1-is-r1-v′-s1*: *t1* = *r1* @ [$\mathcal{V}'$] @ *s1*
and *r1-Vv-empty*: *r1* ↾ $V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}1}$ $V_{\mathcal{V}}$ *r1*]
have *r1-Vv1-empty*: *r1* ↾ $V_{\mathcal{V}1} = []$
by *auto*

from *t1-is-r1-v′-s1 Cons(10)* **have** *r1-Cv1-empty*: *r1* ↾ $C_{\mathcal{V}1} = []$

by (*simp add*: *projection-concatenation-commute*)

from *t1-is-r1-v′-s1 Cons*(*10*) **have** *s1-Cv1-empty*: $s1 \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*4*) *t1-is-r1-v′-s1* **have** *r1-in-E1star*: $set\ r1 \subseteq E_{ES1}$
by *auto*

have *r1-in-Nv1star*: $set\ r1 \subseteq N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** $set\ r1 \cap V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Diff-eq Int-commute*
Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
from *r1-Cv1-empty* **have** $set\ r1 \cap C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Diff-eq Int-commute*
Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv1-inter-E2-empty* **have** *r1E2-empty*: $r1 \restriction E_{ES2} = []$
by (*metis Int-commute empty-subsetI*
projection-on-subset2 r1-Vv1-empty)

let *?tau* = τ @ *r1* @ $[\mathcal{V}']$

from *v′-in-E1 Cons*(*2*) *r1-in-Nv1star validV1*
have $set\ ?tau \subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *isViewOn-def composeES-def V-valid-def*, *auto*)
moreover
from *Cons*(*3*) **have** $set\ lambda' \subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *Cons*(*4*) *t1-is-r1-v′-s1* **have** $set\ s1 \subseteq E_{ES1}$
by *auto*
moreover
note *Cons*(*5*)
moreover
have $?tau \restriction E_{ES1}$ @ $s1 \in Tr_{ES1}$
by (*metis Cons-eq-appendI append-eq-appendI calculation*(*3*) *eq-Nil-appendI*
list-subset-iff-projection-neutral Cons.prems(*3*) *Cons.prems*(*5*)
projection-concatenation-commute t1-is-r1-v′-s1)
moreover
have $?tau \restriction E_{ES2}$ @ $t2 \in Tr_{ES2}$
proof −

```
    from v'-notin-E2 have [V'] ↾ E_ES2 = []
      by (simp add: projection-def)
    with Cons(7) Cons(4) t1-is-r1-v'-s1 v'-notin-E2
      r1-in-Nv1star Nv1-inter-E2-empty r1E2-empty
      show ?thesis
        by (simp only: t1-is-r1-v'-s1 list-subset-iff-projection-neutral
          projection-concatenation-commute, auto)
  qed
  moreover
  from Cons(8) t1-is-r1-v'-s1 r1-Vv-empty v'-in-E1 v'-in-Vv have lambda' ↾ E_ES1 = s1 ↾ V_V
    by (simp add: projection-def)
  moreover
  from Cons(9) v'-notin-E2 have lambda' ↾ E_ES2 = t2 ↾ V_V
    by (simp add: projection-def)
  moreover
  note s1-Cv1-empty Cons(11)
  moreover
  note Cons.hyps(1)[of ?tau s1 t2]
  ultimately obtain t'
    where tau-t'-in-Tr: ?tau @ t' ∈ Tr_(ES1 ∥ ES2)
    and t'-Vv-is-lambda': t' ↾ V_V = lambda'
    and t'-Cv-empty: t' ↾ C_V = []
    by auto

  let ?t = r1 @ [V'] @ t'


  note tau-t'-in-Tr
  moreover
  from r1-Vv-empty t'-Vv-is-lambda' v'-in-Vv
  have ?t ↾ V_V = V' # lambda'
    by (simp add: projection-def)
  moreover
  have ?t ↾ C_V = []
  proof −
    from propSepViews have C_V ∩ E_ES1 ⊆ C_V1
      unfolding properSeparationOfViews-def by auto
    hencer1 ↾ C_V = []
      by (metis projection-on-subset2 r1-Cv1-empty r1-in-E1star)
    with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def projection-def, auto)
  qed
  ultimately have ?thesis
    by auto
}
moreover {
  assume v'-in-Vv2-minus-E1: V' ∈ V_V2 − E_ES1
  hence v'-in-Vv2: V' ∈ V_V2
    by auto
  with v'-in-Vv propSepViews
  have v'-in-E2: V' ∈ E_ES2
    unfolding properSeparationOfViews-def by auto
```

from *v′-in-Vv2-minus-E1*
have *v′-notin-E1*: $\mathcal{V}' \notin E_{ES1}$
by (*auto*)
with *validV1*
have *v′-notin-Vv1*: $\mathcal{V}' \notin V_{\mathcal{V}1}$
by (*simp add*:*isViewOn-def V-valid-def*, *auto*)

from *Cons*(*4*) *Cons*(*5*) *Cons*(*9*) *v′-in-E2*
have *t2* ↾ $V_{\mathcal{V}} = \mathcal{V}' \# (lambda' \restriction E_{ES2})$
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r2 s2*
where *t2-is-r2-v′-s2*: *t2* = *r2* @ [$\mathcal{V}'$] @ *s2*
and *r2-Vv-empty*: *r2* ↾ $V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: *r2* ↾ $V_{\mathcal{V}2} = []$
by *auto*

from *t2-is-r2-v′-s2 Cons*(*11*) **have** *r2-Cv2-empty*: *r2* ↾ $C_{\mathcal{V}2} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v′-s2 Cons*(*11*) **have** *s2-Cv2-empty*: *s2* ↾ $C_{\mathcal{V}2} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*5*) *t2-is-r2-v′-s2* **have** *r2-in-E2star*: *set r2* ⊆ E_{ES2}
by *auto*

have *r2-in-Nv2star*: *set r2* ⊆ $N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** *set r2* ∩ $V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
from *r2-Cv2-empty* **have** *set r2* ∩ $C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv2-inter-E1-empty* **have** *r2E1-empty*: *r2* ↾ $E_{ES1} = []$
by (*metis Int-commute empty-subsetI*
projection-on-subset2 r2-Vv2-empty)

let *?tau* = τ @ *r2* @ $[\mathcal{V}']$

from *v′-in-E2 Cons(2) r2-in-Nv2star validV2*
have *set ?tau* $\subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *composeES-def isViewOn-def V-valid-def*, *auto*)
moreover
from *Cons(3)* **have** *set lambda′* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
note *Cons(4)*
moreover
from *Cons(5) t2-is-r2-v′-s2* **have** *set s2* $\subseteq E_{ES2}$
by *auto*
moreover
have *?tau* $\restriction E_{ES1}$ @ *t1* $\in Tr_{ES1}$
proof −
from *v′-notin-E1* **have** $[\mathcal{V}'] \restriction E_{ES1} = []$
by (*simp add*: *projection-def*)
with *Cons(6) Cons(3) t2-is-r2-v′-s2 v′-notin-E1 r2-in-Nv2star Nv2-inter-E1-empty r2E1-empty*
show *?thesis*
by (*simp only*: *t2-is-r2-v′-s2 list-subset-iff-projection-neutral projection-concatenation-commute*, *auto*)
qed
moreover
have *?tau* $\restriction E_{ES2}$ @ *s2* $\in Tr_{ES2}$
by (*metis Cons-eq-appendI append-eq-appendI calculation(4) eq-Nil-appendI list-subset-iff-projection-neutral Cons.prems(4) Cons.prems(6) projection-concatenation-commute t2-is-r2-v′-s2*)
moreover
from *Cons(8) v′-notin-E1* **have** *lambda′* $\restriction E_{ES1} = t1 \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover
from *Cons(9) t2-is-r2-v′-s2 r2-Vv-empty v′-in-E2 v′-in-Vv*
have *lambda′* $\restriction E_{ES2} = s2 \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover
note *Cons(10) s2-Cv2-empty*
moreover
note *Cons.hyps(1)*[*of ?tau t1 s2*]
ultimately obtain t'
where *tau-t′-in-Tr*: *?tau* @ $t' \in Tr_{(ES1 \parallel ES2)}$
and *t′-Vv-is-lambda′*: $t' \restriction V_{\mathcal{V}} =$ *lambda′*
and *t′-Cv-empty*: $t' \restriction C_{\mathcal{V}} = []$
by *auto*

let *?t* = *r2* @ $[\mathcal{V}']$ @ t'

note *tau-t′-in-Tr*
moreover

```
          from r2-Vv-empty t'-Vv-is-lambda' v'-in-Vv
          have ?t ↾ V_𝒱 = 𝒱' # lambda'
            by (simp add: projection-def)
          moreover
          have ?t ↾ C_𝒱 = []
          proof −
            from propSepViews have C_𝒱 ∩ E_ES2 ⊆ C_𝒱2
              unfolding properSeparationOfViews-def by auto
            hence r2 ↾ C_𝒱 = []
              by (metis projection-on-subset2 r2-Cv2-empty r2-in-E2star)
            with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
              by (simp add: isViewOn-def V-valid-def VC-disjoint-def projection-def, auto)
          qed
          ultimately have ?thesis
            by auto
        }
        ultimately show ?thesis
          by blast
      qed
    qed
  }
  thus ?thesis
    by auto
qed


lemma generalized-zipping-lemma2: ⟦ N_𝒱1 ∩ E_ES2 = {}; total ES1 (C_𝒱1 ∩ N_𝒱2); BSIA ϱ1 𝒱1 Tr_ES1 ⟧
⟹
  ∀ τ lambda t1 t2. ( ( set τ ⊆ (E_(ES1 ∥ ES2)) ∧ set lambda ⊆ V_𝒱 ∧ set t1 ⊆ E_ES1 ∧ set t2 ⊆ E_ES2
  ∧ ((τ ↾ E_ES1) @ t1) ∈ Tr_ES1 ∧ ((τ ↾ E_ES2) @ t2) ∈ Tr_ES2
  ∧ (lambda ↾ E_ES1) = (t1 ↾ V_𝒱) ∧ (lambda ↾ E_ES2) = (t2 ↾ V_𝒱)
  ∧ (t1 ↾ C_𝒱1) = [] ∧ (t2 ↾ C_𝒱2) = [])
  ⟶ (∃ t. ((τ @ t) ∈ (Tr_(ES1 ∥ ES2)) ∧ (t ↾ V_𝒱) = lambda ∧ (t ↾ C_𝒱) = [])) )
proof −
  assume Nv1-inter-E2-empty: N_𝒱1 ∩ E_ES2 = {}
  assume total-ES1-Cv1-inter-Nv2: total ES1 (C_𝒱1 ∩ N_𝒱2)
  assume BSIA: BSIA ϱ1 𝒱1 Tr_ES1

  {
    fix τ lambda t1 t2
    assume τ-in-Estar: set τ ⊆ E_(ES1 ∥ ES2)
      and lambda-in-Vvstar: set lambda ⊆ V_𝒱
      and t1-in-E1star: set t1 ⊆ E_ES1
      and t2-in-E2star: set t2 ⊆ E_ES2
      and τ-E1-t1-in-Tr1: ((τ ↾ E_ES1) @ t1) ∈ Tr_ES1
      and τ-E2-t2-in-Tr2: ((τ ↾ E_ES2) @ t2) ∈ Tr_ES2
      and lambda-E1-is-t1-Vv: (lambda ↾ E_ES1) = (t1 ↾ V_𝒱)
      and lambda-E2-is-t2-Vv: (lambda ↾ E_ES2) = (t2 ↾ V_𝒱)
      and t1-no-Cv1: (t1 ↾ C_𝒱1) = []
      and t2-no-Cv2: (t2 ↾ C_𝒱2) = []

    have ⟦ set τ ⊆ E_(ES1 ∥ ES2); set lambda ⊆ V_𝒱;
```

```
  set t1 ⊆ E_ES1; set t2 ⊆ E_ES2;
((τ ↾ E_ES1) @ t1) ∈ Tr_ES1; ((τ ↾ E_ES2) @ t2) ∈ Tr_ES2;
(lambda ↾ E_ES1) = (t1 ↾ V_V); (lambda ↾ E_ES2) = (t2 ↾ V_V);
(t1 ↾ C_V1) = []; (t2 ↾ C_V2) = [] ]]
⟹ (∃ t. ((τ @ t) ∈ Tr_(ES1 ∥ ES2) ∧ (t ↾ V_V) = lambda ∧ (t ↾ C_V) = []))
proof (induct lambda arbitrary: τ t1 t2)
  case (Nil τ t1 t2)

  have (τ @ []) ∈ Tr_(ES1 ∥ ES2)
    proof −
      have τ ∈ Tr_(ES1 ∥ ES2)
        proof −
          from Nil(5) validES1 have τ ↾ E_ES1 ∈ Tr_ES1
            by (simp add: ES-valid-def traces-prefixclosed-def
              prefixclosed-def prefix-def)
          moreover
          from Nil(6) validES2 have τ ↾ E_ES2 ∈ Tr_ES2
            by (simp add: ES-valid-def traces-prefixclosed-def
              prefixclosed-def prefix-def)
          moreover
          note Nil(1)
          ultimately show ?thesis
            by (simp add: composeES-def)
        qed
      thus ?thesis
        by auto
    qed
  moreover
  have ([] ↾ V_V) = []
    by (simp add: projection-def)
  moreover
  have ([] ↾ C_V) = []
    by (simp add: projection-def)
  ultimately show ?case
    by blast
next
  case (Cons V' lambda' τ t1 t2)
  thus ?case
    proof −
      from Cons(3) have v'-in-Vv: V' ∈ V_V
        by auto

      have V' ∈ V_V1 ∩ V_V2 ∨ V' ∈ V_V1 − E_ES2 ∨ V' ∈ V_V2 − E_ES1
        using propSepViews unfolding properSeparationOfViews-def
        using Vv-is-Vv1-union-Vv2 v'-in-Vv by fastforce
      moreover {
        assume v'-in-Vv1-inter-Vv2: V' ∈ V_V1 ∩ V_V2
        hence v'-in-Vv1: V' ∈ V_V1 and v'-in-Vv2: V' ∈ V_V2
          by auto
        with v'-in-Vv propSepViews
        have v'-in-E1: V' ∈ E_ES1 and v'-in-E2: V' ∈ E_ES2
          unfolding properSeparationOfViews-def by auto
```

from *Cons*(3,5,9) *v'-in-E2*
have *t2* ↾ $V_{\mathcal{V}} = \mathcal{V}'$ # (*lambda'* ↾ E_{ES2})
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r2 s2*
where *t2-is-r2-v'-s2*: *t2* = *r2* @ [$\mathcal{V}'$] @ *s2*
and *r2-Vv-empty*: *r2* ↾ $V_{\mathcal{V}}$ = []
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: *r2* ↾ $V_{\mathcal{V}2}$ = []
by *auto*

from *t2-is-r2-v'-s2 Cons*(11) **have** *r2-Cv2-empty*: *r2* ↾ $C_{\mathcal{V}2}$ = []
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v'-s2 Cons*(11) **have** *s2-Cv2-empty*: *s2* ↾ $C_{\mathcal{V}2}$ = []
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(5) *t2-is-r2-v'-s2* **have** *r2-in-E2star*: *set r2* ⊆ E_{ES2}
and *s2-in-E2star*: *set s2* ⊆ E_{ES2}
by *auto*

from *Cons*(7) *t2-is-r2-v'-s2*
have *τE2-r2-v'-s2-in-Tr2*: *τ* ↾ E_{ES2} @ *r2* @ [$\mathcal{V}'$] @ *s2* ∈ Tr_{ES2}
by *simp*

have *r2-in-Nv2star*: *set r2* ⊆ $N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** *set r2* ∩ $V_{\mathcal{V}2}$ = {}
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r2-Cv2-empty* **have** *set r2* ∩ $C_{\mathcal{V}2}$ = {}
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed

have *r2E1-in-Nv2-inter-C1-star*: *set* (*r2* ↾ E_{ES1}) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
proof −
have *set* (*r2* ↾ E_{ES1}) = *set r2* ∩ E_{ES1}
by (*simp add*: *projection-def*, *auto*)
with *r2-in-Nv2star* **have** *set* (*r2* ↾ E_{ES1}) ⊆ (E_{ES1} ∩ $N_{\mathcal{V}2}$)

by *auto*
moreover
from *validV1 propSepViews*
have $E_{ES1} \cap N_{\mathcal{V}2} = N_{\mathcal{V}2} \cap C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def isViewOn-def V-valid-def*
using *disjoint-Nv2-Vv1* **by** *blast*
ultimately show *?thesis*
by *auto*
qed

note *outerCons-prems* = *Cons.prems*

have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1}) \Longrightarrow$
$\exists\ t1'.\ (\ set\ t1' \subseteq E_{ES1}$
$\wedge\ ((\tau\ @\ r2) \restriction E_{ES1})\ @\ t1' \in Tr_{ES1}$
$\wedge\ t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
$\wedge\ t1' \restriction C_{\mathcal{V}1} = [])$
proof (*induct r2* $\restriction E_{ES1}$ *arbitrary*: *r2 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*9*)
outerCons-prems(*3*) *outerCons-prems*(*5*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE1*: $xs = xs \restriction E_{ES1}$
proof −
from *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
hence *set* $xs \subseteq E_{ES1}$
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* $(xs \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by (*metis Int-commute snoc.prems*)
with *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
hence *set* $xs \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *auto*
with *xs-is-xsE1* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain $t1''$
where $t1''$*-in-E1star*: *set* $t1'' \subseteq E_{ES1}$
and τ*-xs-E1-t1*$''$*-in-Tr1*: $((\tau\ @\ xs) \restriction E_{ES1})\ @\ t1'' \in Tr_{ES1}$
and $t1''$*Vv1-is-t1Vv1*: $t1'' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$

and *t1''Cv1-empty*: $t1'' \restriction C_{\mathcal{V}1} = []$
by *auto*

have *x-in-Cv1-inter-Nv2*: $x \in C_{\mathcal{V}1} \cap N_{\mathcal{V}2}$
proof −
from *snoc(2−3)* **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
thus *?thesis*
by *auto*
qed
hence *x-in-Cv1*: $x \in C_{\mathcal{V}1}$
by *auto*
moreover
note *τ-xs-E1-t1''-in-Tr1 t1''Cv1-empty*
moreover
have *Adm*: $(Adm\ \mathcal{V}1\ \varrho 1\ Tr_{ES1}\ ((\tau \mathbin{@} xs) \restriction E_{ES1})\ x)$
proof −
from *τ-xs-E1-t1''-in-Tr1 validES1*
have *τ-xsE1-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
with *x-in-Cv1-inter-Nv2 total-ES1-Cv1-inter-Nv2*
have *τ-xsE1-x-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \in Tr_{ES1}$
by (*simp only*: *total-def*)
moreover
have $((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1) = ((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA*
ultimately obtain *t1'*
where *res1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1' \in Tr_{ES1}$
and *res2*: $t1' \restriction V_{\mathcal{V}1} = t1'' \restriction V_{\mathcal{V}1}$
and *res3*: $t1' \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t1'* $\subseteq E_{ES1}$
proof −
from *res1 validES1*
have *set* $(((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1') \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau \mathbin{@} r2) \restriction E_{ES1}) \mathbin{@} t1' \in Tr_{ES1}$
proof −
from *res1 xs-is-xsE1* **have** $((\tau \restriction E_{ES1}) \mathbin{@} (xs \mathbin{@} [x])) \mathbin{@} t1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc(2) projection-concatenation-commute*)
qed

```
  moreover
  from t1''Vv1-is-t1Vv1 res2 have t1' ↾ V_V1 = t1 ↾ V_V1
    by auto
  moreover
  note res3
  ultimately show ?case
    by auto
qed
from this[OF r2E1-in-Nv2-inter-C1-star] obtain t1'
  where t1'-in-E1star: set t1' ⊆ E_ES1
  and τr2E1-t1'-in-Tr1: ((τ @ r2) ↾ E_ES1) @ t1' ∈ Tr_ES1
  and t1'-Vv1-is-t1-Vv1: t1' ↾ V_V1 = t1 ↾ V_V1
  and t1'-Cv1-empty: t1' ↾ C_V1 = []
  by auto


have t1' ↾ V_V1 = V' # (lambda' ↾ E_ES1)
  proof −
    from projection-intersection-neutral[OF Cons(4), of V_V]
    propSepViews
    have t1 ↾ V_V = t1 ↾ V_V1
      unfolding properSeparationOfViews-def
      by (simp only: Int-commute)
    with Cons(8) t1'-Vv1-is-t1-Vv1 v'-in-E1 show ?thesis
      by (simp add: projection-def)
  qed
from projection-split-first[OF this] obtain r1' s1'
  where t1'-is-r1'-v'-s1': t1' = r1' @ [V'] @ s1'
  and r1'-Vv1-empty: r1' ↾ V_V1 = []
  by auto


from t1'-is-r1'-v'-s1' t1'-Cv1-empty
have r1'-Cv1-empty: r1' ↾ C_V1 = []
  by (simp add: projection-concatenation-commute)

from t1'-is-r1'-v'-s1' t1'-Cv1-empty
have s1'-Cv1-empty: s1' ↾ C_V1 = []
  by (simp only: projection-concatenation-commute, auto)

from t1'-in-E1star t1'-is-r1'-v'-s1'
have r1'-in-E1star: set r1' ⊆ E_ES1
  by auto
with propSepViews r1'-Vv1-empty
have r1'-Vv-empty: r1' ↾ V_V = []
  unfolding properSeparationOfViews-def
  by (metis projection-on-subset2 subset-iff-psubset-eq)

have r1'-in-Nv1star: set r1' ⊆ N_V1
  proof −
    note r1'-in-E1star
    moreover
```

from *r1′-Vv1-empty* **have** *set r1′* $\cap$ $V_{\mathcal{V}1}$ = {}
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
from *r1′-Cv1-empty* **have** *set r1′* $\cap$ $C_{\mathcal{V}1}$ = {}
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv1-inter-E2-empty* **have** *r1′E2-empty*: *r1′* $\restriction$ E_{ES2} = []
by (*metis Int-commute empty-subsetI projection-on-subset2 r1′-Vv1-empty*)

let *?tau* = τ @ *r2* @ *r1′* @ [$\mathcal{V}'$]

from *Cons*(*2*) *r2-in-E2star r1′-in-E1star v′-in-E2*
have *set ?tau* $\subseteq$ ($E_{(ES1 \parallel ES2)}$)
by (*simp add*: *composeES-def*, *auto*)
moreover
from *Cons*(*3*) **have** *set lambda′* $\subseteq$ $V_{\mathcal{V}}$
by *auto*
moreover
from *t1′-in-E1star t1′-is-r1′-v′-s1′*
have *set s1′* $\subseteq$ E_{ES1}
by *simp*
moreover
note *s2-in-E2star*
moreover
from τ*r2E1-t1′-in-Tr1 t1′-is-r1′-v′-s1′ v′-in-E1*
have *?tau* $\restriction$ E_{ES1} @ *s1′* $\in$ Tr_{ES1}
proof −
from *v′-in-E1 r1′-in-E1star*
have (*τ* @ *r2* @ *r1′* @ [$\mathcal{V}'$]) $\restriction$ E_{ES1} = (*τ* @ *r2*) $\restriction$ E_{ES1} @ *r1′* @ [$\mathcal{V}'$]
by (*simp only*: *projection-concatenation-commute list-subset-iff-projection-neutral projection-def*, *auto*)
with τ*r2E1-t1′-in-Tr1 t1′-is-r1′-v′-s1′ v′-in-E1* **show** *?thesis*
by *simp*
qed
moreover
from *r2-in-E2star v′-in-E2 r1′E2-empty* τ*E2-r2-v′-s2-in-Tr2*
have *?tau* $\restriction$ E_{ES2} @ *s2* $\in$ Tr_{ES2}
by (*simp only*: *list-subset-iff-projection-neutral projection-concatenation-commute projection-def*, *auto*)
moreover
have *lambda′* $\restriction$ E_{ES1} = *s1′* $\restriction$ $V_{\mathcal{V}}$
proof −

from *Cons*(2,4,8) *v′-in-E1* **have** *t1* ↾ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*lambda′* ↾ E_{ES1})
by (*simp add*: *projection-def*)
moreover
from *t1′-is-r1′-v′-s1′ r1′-Vv1-empty r1′-in-E1star v′-in-Vv1 propSepViews*
have *t1′* ↾ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*s1′* ↾ $V_{\mathcal{V}}$)
proof −
have *r1′* ↾ $V_{\mathcal{V}}$ =[]
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2*
r1′-Vv1-empty r1′-in-E1star subset-iff-psubset-eq)
with *t1′-is-r1′-v′-s1′ v′-in-Vv1 Vv-is-Vv1-union-Vv2* **show** *?thesis*
by (*simp only*: *t1′-is-r1′-v′-s1′ projection-concatenation-commute*
projection-def, *auto*)
qed
moreover
have *t1* ↾ $V_{\mathcal{V}}$ = *t1′* ↾ $V_{\mathcal{V}}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute outerCons-prems*(3)
projection-intersection-neutral
t1′-Vv1-is-t1-Vv1 t1′-in-E1star)
ultimately show *?thesis*
by *auto*
qed
moreover
have *lambda′* ↾ E_{ES2} = *s2* ↾ $V_{\mathcal{V}}$
proof −
from *Cons*(3,5,9) *v′-in-E2* **have** *t2* ↾ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*lambda′* ↾ E_{ES2})
by (*simp add*: *projection-def*)
moreover
from *t2-is-r2-v′-s2 r2-Vv-empty v′-in-Vv2 Vv-is-Vv1-union-Vv2*
have *t2* ↾ $V_{\mathcal{V}}$ = [$\mathcal{V}'$] @ (*s2* ↾ $V_{\mathcal{V}}$)
by (*simp only*: *t2-is-r2-v′-s2 projection-concatenation-commute projection-def*, *auto*)
ultimately show *?thesis*
by *auto*
qed
moreover
note *s1′-Cv1-empty s2-Cv2-empty Cons.hyps*[*of ?tau s1′ s2*]
ultimately obtain *t′*
where *tau-t′-in-Tr*: *?tau* @ *t′* ∈ $Tr_{(ES1 \parallel ES2)}$
and *t′Vv-is-lambda′*: *t′* ↾ $V_{\mathcal{V}}$ = *lambda′*
and *t′Cv-empty*: *t′* ↾ $C_{\mathcal{V}}$ = []
by *auto*

let *?t* = *r2* @ *r1′* @ [$\mathcal{V}'$] @ *t′*

note *tau-t′-in-Tr*
moreover
from *r2-Vv-empty r1′-Vv-empty t′Vv-is-lambda′ v′-in-Vv* **have** *?t* ↾ $V_{\mathcal{V}}$ = $\mathcal{V}'$ # *lambda′*
by(*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from *VIsViewOnE r2-Cv2-empty t′Cv-empty r1′-Cv1-empty v′-in-Vv*

```
  have ?t ↾ C_V = []
  proof −
    from VIsViewOnE v'-in-Vv have [V'] ↾ C_V = []
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def projection-def, auto)
    moreover
    from r2-in-E2star r2-Cv2-empty propSepViews
    have r2 ↾ C_V = []
      unfolding properSeparationOfViews-def
      using projection-on-subset2 by auto
    moreover
    from r1'-in-E1star r1'-Cv1-empty propSepViews
    have r1' ↾ C_V = []
      unfolding properSeparationOfViews-def
      using projection-on-subset2 by auto
    moreover
    note t'Cv-empty
    ultimately show ?thesis
      by (simp only: projection-concatenation-commute, auto)
  qed
  ultimately have ?thesis
    by auto
}
moreover {
  assume v'-in-Vv1-minus-E2: V' ∈ V_V1 − E_ES2
  hence v'-in-Vv1: V' ∈ V_V1
    by auto
  with v'-in-Vv propSepViews have v'-in-E1: V' ∈ E_ES1
    unfolding properSeparationOfViews-def by auto

  from v'-in-Vv1-minus-E2 have v'-notin-E2: V' ∉ E_ES2
    by (auto)
  with validV2 have v'-notin-Vv2: V' ∉ V_V2
    by (simp add: isViewOn-def V-valid-def, auto)


  from Cons(3) Cons(4) Cons(8) v'-in-E1
  have t1 ↾ V_V = V' # (lambda' ↾ E_ES1)
    by (simp add: projection-def)
  from projection-split-first[OF this] obtain r1 s1
    where t1-is-r1-v'-s1: t1 = r1 @ [V'] @ s1
    and r1-Vv-empty: r1 ↾ V_V = []
    by auto
  with Vv-is-Vv1-union-Vv2 projection-on-subset[of V_V1 V_V r1]
  have r1-Vv1-empty: r1 ↾ V_V1 = []
    by auto


  from t1-is-r1-v'-s1 Cons(10)
  have r1-Cv1-empty: r1 ↾ C_V1 = []
    by (simp add: projection-concatenation-commute)

  from t1-is-r1-v'-s1 Cons(10)
```

have *s1-Cv1-empty*: $s1 \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*4*) *t1-is-r1-v'-s1*
have *r1-in-E1star*: $set\ r1 \subseteq E_{ES1}$
by *auto*

have *r1-in-Nv1star*: $set\ r1 \subseteq N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** $set\ r1 \cap V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Diff-eq*
Int-commute Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
from *r1-Cv1-empty* **have** $set\ r1 \cap C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Diff-eq*
Int-commute Int-empty-right disjoint-eq-subset-Compl
list-subset-iff-projection-neutral projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv1-inter-E2-empty* **have** *r1E2-empty*: $r1 \restriction E_{ES2} = []$
by (*metis Int-commute empty-subsetI projection-on-subset2 r1-Vv1-empty*)

let *?tau* = τ @ *r1* @ $[\mathcal{V}']$

from *v'-in-E1 Cons*(*2*) *r1-in-Nv1star validV1*
have *set ?tau* $\subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *composeES-def isViewOn-def V-valid-def*, *auto*)
moreover
from *Cons*(*3*) **have** $set\ lambda' \subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *Cons*(*4*) *t1-is-r1-v'-s1* **have** $set\ s1 \subseteq E_{ES1}$
by *auto*
moreover
note *Cons*(*5*)
moreover
have *?tau* $\restriction E_{ES1}$ @ *s1* $\in Tr_{ES1}$
by (*metis Cons-eq-appendI append-eq-appendI calculation*(*3*) *eq-Nil-appendI*
list-subset-iff-projection-neutral Cons.prems(*3*) *Cons.prems*(*5*)
projection-concatenation-commute t1-is-r1-v'-s1)
moreover
have *?tau* $\restriction E_{ES2}$ @ *t2* $\in Tr_{ES2}$
proof −
from *v'-notin-E2* **have** $[\mathcal{V}'] \restriction E_{ES2} = []$

```
      by (simp add: projection-def)
    with Cons(7) Cons(4) t1-is-r1-v'-s1 v'-notin-E2 r1-in-Nv1star
      Nv1-inter-E2-empty r1E2-empty
      show ?thesis
        by (simp only: t1-is-r1-v'-s1 list-subset-iff-projection-neutral
          projection-concatenation-commute, auto)
  qed
  moreover
  from Cons(8) t1-is-r1-v'-s1 r1-Vv-empty v'-in-E1 v'-in-Vv
  have lambda' ↾ E_ES1 = s1 ↾ V_V
    by (simp add: projection-def)
  moreover
  from Cons(9) v'-notin-E2 have lambda' ↾ E_ES2 = t2 ↾ V_V
    by (simp add: projection-def)
  moreover
  note s1-Cv1-empty Cons(11)
  moreover
  note Cons.hyps(1)[of ?tau s1 t2]
  ultimately obtain t'
    where τr1v't'-in-Tr: ?tau @ t' ∈ Tr_(ES1 ∥ ES2)
    and t'-Vv-is-lambda': t' ↾ V_V = lambda'
    and t'-Cv-empty: t' ↾ C_V = []
    by auto

  let ?t = r1 @ [V'] @ t'


  note τr1v't'-in-Tr
  moreover
  from r1-Vv-empty t'-Vv-is-lambda' v'-in-Vv have ?t ↾ V_V = V' # lambda'
    by (simp add: projection-def)
  moreover
  have ?t ↾ C_V = []
  proof −
    have r1 ↾ C_V = []
      using propSepViews unfolding properSeparationOfViews-def
      by (metis  projection-on-subset2 r1-Cv1-empty r1-in-E1star)
    with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def projection-def, auto)
  qed
  ultimately have ?thesis
    by auto
}
moreover {
  assume v'-in-Vv2-minus-E1: V' ∈ V_V2 − E_ES1
  hence v'-in-Vv2: V' ∈ V_V2
    by auto
  with v'-in-Vv propSepViews
  have v'-in-E2: V' ∈ E_ES2
    unfolding properSeparationOfViews-def by auto

  from v'-in-Vv2-minus-E1
```

have *v′-notin-E1*: $\mathcal{V}' \notin E_{ES1}$
by (*auto*)
with *validV1*
have *v′-notin-Vv1*: $\mathcal{V}' \notin V_{\mathcal{V}1}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)

from *Cons*(3) *Cons*(5) *Cons*(9) *v′-in-E2* **have** *t2* ↾ $V_{\mathcal{V}} = \mathcal{V}'$ # (*lambda′* ↾ E_{ES2})
by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r2 s2*
where *t2-is-r2-v′-s2*: *t2* = *r2* @ [$\mathcal{V}'$] @ *s2*
and *r2-Vv-empty*: *r2* ↾ $V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: *r2* ↾ $V_{\mathcal{V}2} = []$
by *auto*

from *t2-is-r2-v′-s2 Cons*(11) **have** *r2-Cv2-empty*: *r2* ↾ $C_{\mathcal{V}2} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v′-s2 Cons*(11) **have** *s2-Cv2-empty*: *s2* ↾ $C_{\mathcal{V}2} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(5) *t2-is-r2-v′-s2* **have** *r2-in-E2star*: *set r2* ⊆ E_{ES2}
by *auto*

have *r2-in-Nv2star*: *set r2* ⊆ $N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** *set r2* ∩ $V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
from *r2-Cv2-empty* **have** *set r2* ∩ $C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed

have *r2E1-in-Nv2-inter-C1-star*: *set* (*r2* ↾ E_{ES1}) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
proof −
have *set* (*r2* ↾ E_{ES1}) = *set r2* ∩ E_{ES1}
by (*simp add*: *projection-def*, *auto*)
with *r2-in-Nv2star* **have** *set* (*r2* ↾ E_{ES1}) ⊆ (E_{ES1} ∩ $N_{\mathcal{V}2}$)
by *auto*

moreover
from *validV1 propSepViews disjoint-Nv2-Vv1* **have** $E_{ES1} \cap N_{\mathcal{V}2} = N_{\mathcal{V}2} \cap C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
ultimately show *?thesis*
by *auto*
qed

note *outerCons-prems* = *Cons.prems*

have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1}) \Longrightarrow$
$\exists\ t1'.\ (\ set\ t1' \subseteq E_{ES1}$
$\wedge\ ((\tau\ @\ r2) \restriction E_{ES1})\ @\ t1' \in Tr_{ES1}$
$\wedge\ t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
$\wedge\ t1' \restriction C_{\mathcal{V}1} = [])$
proof (*induct r2* $\restriction E_{ES1}$ *arbitrary*: *r2 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*9*) *outerCons-prems*(*3*) *outerCons-prems*(*5*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE1*: $xs = xs \restriction E_{ES1}$
proof −
from *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
hence *set* $xs \subseteq E_{ES1}$
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* $(xs \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by (*metis Int-commute snoc.prems*)
with *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
hence *set* $xs \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *auto*
with *xs-is-xsE1* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *t1''*
where *t1''-in-E1star*: *set t1''* $\subseteq E_{ES1}$
and τ*-xs-E1-t1''-in-Tr1*: $((\tau\ @\ xs) \restriction E_{ES1})\ @\ t1'' \in Tr_{ES1}$
and *t1''Vv1-is-t1Vv1*: $t1'' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
and *t1''Cv1-empty*: $t1'' \restriction C_{\mathcal{V}1} = []$

by *auto*

have *x-in-Cv1-inter-Nv2*: $x \in C_{\mathcal{V}1} \cap N_{\mathcal{V}2}$
proof −
from *snoc*(2−3) **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
thus *?thesis*
by *auto*
qed
hence *x-in-Cv1*: $x \in C_{\mathcal{V}1}$
by *auto*
moreover
note *τ-xs-E1-t1''-in-Tr1 t1''Cv1-empty*
moreover
have *Adm*: (*Adm* $\mathcal{V}1$ $\varrho 1$ Tr_{ES1} $((\tau \mathbin{@} xs) \restriction E_{ES1})$ x)
proof −
from *τ-xs-E1-t1''-in-Tr1 validES1*
have *τ-xsE1-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def)
with *x-in-Cv1-inter-Nv2 total-ES1-Cv1-inter-Nv2*
have *τ-xsE1-x-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \in Tr_{ES1}$
by (*simp only*: *total-def*)
moreover
have $((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1) = ((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA*
ultimately obtain *t1'*
where *res1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1' \in Tr_{ES1}$
and *res2*: $t1' \restriction V_{\mathcal{V}1} = t1'' \restriction V_{\mathcal{V}1}$
and *res3*: $t1' \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t1'* $\subseteq E_{ES1}$
proof −
from *res1 validES1* **have** *set* $(((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1') \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau \mathbin{@} r2) \restriction E_{ES1}) \mathbin{@} t1' \in Tr_{ES1}$
proof −
from *res1 xs-is-xsE1* **have** $((\tau \restriction E_{ES1}) \mathbin{@} (xs \mathbin{@} [x])) \mathbin{@} t1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(2) *projection-concatenation-commute*)
qed
moreover
from *t1''Vv1-is-t1Vv1 res2* **have** $t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$

```
    by auto
  moreover
  note res3
  ultimately show ?case
    by auto
qed
from this[OF r2E1-in-Nv2-inter-C1-star] obtain t1'
  where t1'-in-E1star: set t1' ⊆ E_ES1
  and τr2E1-t1'-in-Tr1: ((τ @ r2) ↾ E_ES1) @ t1' ∈ Tr_ES1
  and t1'-Vv1-is-t1-Vv1: t1' ↾ V_V1 = t1 ↾ V_V1
  and t1'-Cv1-empty: t1' ↾ C_V1 = []
  by auto


let ?tau = τ @ r2 @ [V']

from v'-in-E2 Cons(2) r2-in-Nv2star validV2 have set ?tau ⊆ E_(ES1 ∥ ES2)
  by (simp only: composeES-def isViewOn-def V-valid-def
    VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
moreover
from Cons(3) have set lambda' ⊆ V_V
  by auto
moreover
from Cons(5) t2-is-r2-v'-s2 have set s2 ⊆ E_ES2
  by auto
moreover
note t1'-in-E1star
moreover
have ?tau ↾ E_ES2 @ s2 ∈ Tr_ES2
  by (metis Cons-eq-appendI append-eq-appendI calculation(3) eq-Nil-appendI
    list-subset-iff-projection-neutral Cons.prems(4) Cons.prems(6)
    projection-concatenation-commute t2-is-r2-v'-s2)
moreover
from τr2E1-t1'-in-Tr1 v'-notin-E1 have ?tau ↾ E_ES1 @ t1' ∈ Tr_ES1
  by (simp add: projection-def)
moreover
from Cons(9) t2-is-r2-v'-s2 r2-Vv-empty v'-in-E2 v'-in-Vv
have lambda' ↾ E_ES2 = s2 ↾ V_V
  by (simp add: projection-def)
moreover
from Cons(10) v'-notin-E1 t1'-Vv1-is-t1-Vv1 have lambda' ↾ E_ES1 = t1' ↾ V_V
proof −
  have t1' ↾ V_V = t1' ↾ V_V1
    using propSepViews unfolding properSeparationOfViews-def
    by (simp add: projection-def, metis Int-commute
      projection-def projection-intersection-neutral
      t1'-in-E1star)
  moreover
  have t1 ↾ V_V = t1 ↾ V_V1
    using propSepViews unfolding properSeparationOfViews-def
    by (simp add: projection-def, metis Int-commute
      projection-def
```

```
            projection-intersection-neutral Cons(4))
          moreover
          note Cons(8) v'-notin-E1 t1'-Vv1-is-t1-Vv1
          ultimately show ?thesis
            by (simp add: projection-def)
        qed
        moreover
        note s2-Cv2-empty t1'-Cv1-empty
        moreover
        note Cons.hyps(1)[of ?tau t1' s2]
        ultimately obtain t'
          where τr2v't'-in-Tr: ?tau @ t' ∈ Tr(ES1 ∥ ES2)
          and t'-Vv-is-lambda': t' ↾ V_V = lambda'
          and t'-Cv-empty: t' ↾ C_V = []
          by auto

        let ?t = r2 @ [V'] @ t'


        note τr2v't'-in-Tr
        moreover
        from r2-Vv-empty t'-Vv-is-lambda' v'-in-Vv
        have ?t ↾ V_V = V' # lambda'
          by (simp add: projection-def)
        moreover
        have ?t ↾ C_V = []
        proof −
          have r2 ↾ C_V = []
          proof −
            from propSepViews have C_V ∩ E_ES2 ⊆ C_V2
              unfolding properSeparationOfViews-def by auto
            from projection-on-subset[OF ‹C_V ∩ E_ES2 ⊆ C_V2› r2-Cv2-empty]
            have r2 ↾ (E_ES2 ∩ C_V) = []
              by (simp only: Int-commute)
            with projection-intersection-neutral[OF r2-in-E2star, of C_V] show ?thesis
              by simp
          qed
          with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
            by (simp add: isViewOn-def V-valid-def VC-disjoint-def
              VN-disjoint-def NC-disjoint-def projection-def, auto)
        qed
        ultimately have ?thesis
          by auto
      }
      ultimately show ?thesis
        by blast
    qed
  qed
}
thus ?thesis
  by auto
qed
```

lemma *generalized-zipping-lemma3*: $[\![N_{\mathcal{V}2} \cap E_{ES1} = \{\}$; *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$; *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}]\!]$
$\Longrightarrow$
$\forall\ \tau$ *lambda t1 t2*. ((*set* $\tau \subseteq E_{(ES1 \parallel ES2)} \wedge$ *set lambda* $\subseteq V_{\mathcal{V}} \wedge$ *set t1* $\subseteq E_{ES1} \wedge$ *set t2* $\subseteq E_{ES2}$
$\wedge ((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$
$\wedge (lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}}) \wedge (lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$
$\wedge (t1 \restriction C_{\mathcal{V}1}) = [] \wedge (t2 \restriction C_{\mathcal{V}2}) = []$)
$\longrightarrow (\exists\ t.\ ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = [])$)
proof −
assume *Nv2-inter-E1-empty*: $N_{\mathcal{V}2} \cap E_{ES1} = \{\}$
assume *total-ES2-Cv2-inter-Nv1*: *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
assume *BSIA*: *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$

{
fix τ *lambda t1 t2*
assume τ*-in-Estar*: *set* $\tau \subseteq E_{(ES1 \parallel ES2)}$
and *lambda-in-Vvstar*: *set lambda* $\subseteq V_{\mathcal{V}}$
and *t1-in-E1star*: *set t1* $\subseteq E_{ES1}$
and *t2-in-E2star*: *set t2* $\subseteq E_{ES2}$
and τ*-E1-t1-in-Tr1*: $((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1}$
and τ*-E2-t2-in-Tr2*: $((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$
and *lambda-E1-is-t1-Vv*: $(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$
and *lambda-E2-is-t2-Vv*: $(lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$
and *t1-no-Cv1*: $(t1 \restriction C_{\mathcal{V}1}) = []$
and *t2-no-Cv2*: $(t2 \restriction C_{\mathcal{V}2}) = []$

have $[\![$ *set* $\tau \subseteq E_{(ES1 \parallel ES2)}$;
set lambda $\subseteq V_{\mathcal{V}}$;
set t1 $\subseteq E_{ES1}$;
set t2 $\subseteq E_{ES2}$;
$((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1}$;
$((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$;
$(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$;
$(lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$;
$(t1 \restriction C_{\mathcal{V}1}) = []$;
$(t2 \restriction C_{\mathcal{V}2}) = []]\!]$
$\Longrightarrow (\exists\ t.\ ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = []))$
proof (*induct lambda arbitrary*: τ *t1 t2*)
case (*Nil* τ *t1 t2*)

have $(\tau @ []) \in Tr_{(ES1 \parallel ES2)}$
proof −
have $\tau \in Tr_{(ES1 \parallel ES2)}$
proof −
from *Nil*(*5*) *validES1* **have** $\tau \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def)
moreover
from *Nil*(*6*) *validES2* **have** $\tau \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def*

```
            prefixclosed-def prefix-def)
        moreover
        note Nil(1)
        ultimately show ?thesis
          by (simp add: composeES-def)
      qed
    thus ?thesis
      by auto
  qed
 moreover
 have ([] ↾ V_𝒱) = []
  by (simp add: projection-def)
 moreover
 have ([] ↾ C_𝒱) = []
  by (simp add: projection-def)
 ultimately show ?case
  by blast
next
 case (Cons 𝒱' lambda' τ t1 t2)
 thus ?case
  proof −
   from Cons(3) have v'-in-Vv: 𝒱' ∈ V_𝒱
    by auto

   have 𝒱' ∈ V_𝒱1 ∩ V_𝒱2
    ∨ 𝒱' ∈ V_𝒱1 − E_ES2
    ∨ 𝒱' ∈ V_𝒱2 − E_ES1
    using propSepViews unfolding properSeparationOfViews-def
    by (metis Diff-iff Int-commute Int-iff Un-iff
      Vv-is-Vv1-union-Vv2 v'-in-Vv)
   moreover {
    assume v'-in-Vv1-inter-Vv2: 𝒱' ∈ V_𝒱1 ∩ V_𝒱2
    hence v'-in-Vv2: 𝒱' ∈ V_𝒱2 and v'-in-Vv1: 𝒱' ∈ V_𝒱1
     by auto
    with v'-in-Vv
    have v'-in-E2: 𝒱' ∈ E_ES2 and v'-in-E1: 𝒱' ∈ E_ES1
     using propSepViews unfolding properSeparationOfViews-def by auto


    from Cons(2,4,8) v'-in-E1 have t1 ↾ V_𝒱 = 𝒱' # (lambda' ↾ E_ES1)
     by (simp add: projection-def)
    from projection-split-first[OF this] obtain r1 s1
     where t1-is-r1-v'-s1: t1 = r1 @ [𝒱'] @ s1
     and r1-Vv-empty: r1 ↾ V_𝒱 = []
     by auto
    with Vv-is-Vv1-union-Vv2 projection-on-subset[of V_𝒱1 V_𝒱 r1]
    have r1-Vv1-empty: r1 ↾ V_𝒱1 = []
     by auto


    from t1-is-r1-v'-s1 Cons(10) have r1-Cv1-empty: r1 ↾ C_𝒱1 = []
     by (simp add: projection-concatenation-commute)
```

from *t1-is-r1-v′-s1 Cons(10)* **have** *s1-Cv1-empty*: $s1 \upharpoonright C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons(4) t1-is-r1-v′-s1*
have *r1-in-E1star*: *set r1* $\subseteq E_{ES1}$ **and** *s1-in-E1star*: *set s1* $\subseteq E_{ES1}$
by *auto*

from *Cons(6) t1-is-r1-v′-s1*
have τ*E1-r1-v′-s1-in-Tr1*: $\tau \upharpoonright E_{ES1}$ @ *r1* @ $[\mathcal{V}']$ @ *s1* $\in Tr_{ES1}$
by *simp*

have *r1-in-Nv1star*: *set r1* $\subseteq N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** *set r1* $\cap$ $V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r1-Cv1-empty* **have** *set r1* $\cap$ $C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r1E2-in-Nv1-inter-C2-star*: *set* $(r1 \upharpoonright E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
have *set* $(r1 \upharpoonright E_{ES2})$ = *set r1* $\cap$ E_{ES2}
by (*simp add*: *projection-def*, *auto*)
with *r1-in-Nv1star* **have** *set* $(r1 \upharpoonright E_{ES2}) \subseteq (E_{ES2} \cap N_{\mathcal{V}1})$
by *auto*
moreover
from *validV2 disjoint-Nv1-Vv2*
have $E_{ES2} \cap N_{\mathcal{V}1} = N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*:*isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*
qed

note *outerCons-prems* = *Cons.prems*

have *set* $(r1 \upharpoonright E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2}) \Longrightarrow$

```
  ∃ t2′. ( set t2′ ⊆ E_ES2
  ∧ ((τ @ r1) ↾ E_ES2) @ t2′ ∈ Tr_ES2
  ∧ t2′ ↾ V_V2 = t2 ↾ V_V2
  ∧ t2′ ↾ C_V2 = [] )
proof (induct r1 ↾ E_ES2 arbitrary: r1 rule: rev-induct)
  case Nil thus ?case
    by (metis append-self-conv outerCons-prems(10) outerCons-prems(4)
      outerCons-prems(6) projection-concatenation-commute)
next
  case (snoc x xs)

  have xs-is-xsE2: xs = xs ↾ E_ES2
    proof −
      from snoc(2) have set (xs @ [x]) ⊆ E_ES2
        by (simp add: projection-def, auto)
      hence set xs ⊆ E_ES2
        by auto
      thus ?thesis
        by (simp add: list-subset-iff-projection-neutral)
    qed
  moreover
  have set (xs ↾ E_ES2) ⊆ (N_V1 ∩ C_V2)
    proof −
      have set (r1 ↾ E_ES2) ⊆ (N_V1 ∩ C_V2)
        by (metis Int-commute snoc.prems)
      with snoc(2) have set (xs @ [x]) ⊆ (N_V1 ∩ C_V2)
        by simp
      hence set xs ⊆ (N_V1 ∩ C_V2)
        by auto
      with xs-is-xsE2 show ?thesis
        by auto
    qed
  moreover
  note snoc.hyps(1)[of xs]
  ultimately obtain t2′′
    where t2′′-in-E2star: set t2′′ ⊆ E_ES2
    and τ-xs-E2-t2′′-in-Tr2: ((τ @ xs) ↾ E_ES2) @ t2′′ ∈ Tr_ES2
    and t2′′Vv2-is-t2Vv2: t2′′ ↾ V_V2 = t2 ↾ V_V2
    and t2′′Cv2-empty: t2′′ ↾ C_V2 = []
    by auto

  have x-in-Cv2-inter-Nv1: x ∈ C_V2 ∩ N_V1
    proof −
      from snoc(2−3) have set (xs @ [x]) ⊆ (N_V1 ∩ C_V2)
        by simp
      thus ?thesis
        by auto
    qed
  hence x-in-Cv2: x ∈ C_V2
    by auto
  moreover
  note τ-xs-E2-t2′′-in-Tr2 t2′′Cv2-empty
```

moreover
have *Adm*: (*Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $((\tau$ @ $xs) \restriction E_{ES2})$ $x)$
proof −
from τ-*xs-E2-t2''-in-Tr2 validES2*
have τ-*xsE2-in-Tr2*: $((\tau$ @ $xs) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
with *x-in-Cv2-inter-Nv1 total-ES2-Cv2-inter-Nv1*
have τ-*xsE2-x-in-Tr2*: $((\tau$ @ $xs) \restriction E_{ES2})$ @ $[x] \in Tr_{ES2}$
by (*simp only*: *total-def*)
moreover
have $((\tau$ @ $xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2) = ((\tau$ @ $xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA*
ultimately obtain *t2'*
where *res1*: $((\tau$ @ $xs) \restriction E_{ES2})$ @ $[x]$ @ $t2' \in Tr_{ES2}$
and *res2*: $t2' \restriction V_{\mathcal{V}2} = t2'' \restriction V_{\mathcal{V}2}$
and *res3*: $t2' \restriction C_{\mathcal{V}2} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t2'* $\subseteq E_{ES2}$
proof −
from *res1 validES2*
have *set* $(((\tau$ @ $xs) \restriction E_{ES2})$ @ $[x]$ @ $t2') \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau$ @ $r1) \restriction E_{ES2})$ @ $t2' \in Tr_{ES2}$
proof −
from *res1 xs-is-xsE2* **have** $((\tau \restriction E_{ES2})$ @ $(xs$ @ $[x]))$ @ $t2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(2) *projection-concatenation-commute*)
qed
moreover
from *t2''Vv2-is-t2Vv2 res2* **have** $t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
by *auto*
moreover
note *res3*
ultimately show *?case*
by *auto*
qed
from *this*[*OF r1E2-in-Nv1-inter-C2-star*] **obtain** *t2'*
where *t2'-in-E2star*: *set t2'* $\subseteq E_{ES2}$
and τ*r1E2-t2'-in-Tr2*: $((\tau$ @ $r1) \restriction E_{ES2})$ @ $t2' \in Tr_{ES2}$
and *t2'-Vv2-is-t2-Vv2*: $t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
and *t2'-Cv2-empty*: $t2' \restriction C_{\mathcal{V}2} = []$
by *auto*

have *t2′ ↾ V$_{\mathcal{V}2}$ = $\mathcal{V}'$ # (lambda′ ↾ E$_{ES2}$)*
proof −
from *projection-intersection-neutral*[*OF Cons*(5), *of V$_{\mathcal{V}}$*]
have *t2 ↾ V$_{\mathcal{V}}$ = t2 ↾ V$_{\mathcal{V}2}$*
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp only*: *Int-commute*)
with *Cons*(9) *t2′-Vv2-is-t2-Vv2 v′-in-E2* **show** *?thesis*
by (*simp add*: *projection-def*)
qed
from *projection-split-first*[*OF this*] **obtain** *r2′ s2′*
where *t2′-is-r2′-v′-s2′*: *t2′ = r2′ @ [$\mathcal{V}'$] @ s2′*
and *r2′-Vv2-empty*: *r2′ ↾ V$_{\mathcal{V}2}$ = []*
by *auto*

from *t2′-is-r2′-v′-s2′ t2′-Cv2-empty*
have *r2′-Cv2-empty*: *r2′ ↾ C$_{\mathcal{V}2}$ = []*
by (*simp add*: *projection-concatenation-commute*)

from *t2′-is-r2′-v′-s2′ t2′-Cv2-empty*
have *s2′-Cv2-empty*: *s2′ ↾ C$_{\mathcal{V}2}$ = []*
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *t2′-in-E2star t2′-is-r2′-v′-s2′*
have *r2′-in-E2star*: *set r2′ ⊆ E$_{ES2}$*
by *auto*
with *r2′-Vv2-empty*
have *r2′-Vv-empty*: *r2′ ↾ V$_{\mathcal{V}}$ = []*
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2 subset-iff-psubset-eq*)

have *r2′-in-Nv2star*: *set r2′ ⊆ N$_{\mathcal{V}2}$*
proof −
note *r2′-in-E2star*
moreover
from *r2′-Vv2-empty* **have** *set r2′ ∩ V$_{\mathcal{V}2}$ = {}*
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r2′-Cv2-empty* **have** *set r2′ ∩ C$_{\mathcal{V}2}$ = {}*
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
qed

with *Nv2-inter-E1-empty* **have** *r2'E1-empty*: $r2' \restriction E_{ES1} = []$
by (*metis Int-commute empty-subsetI projection-on-subset2 r2'-Vv2-empty*)

let *?tau* $= \tau$ @ *r1* @ *r2'* @ $[\mathcal{V}']$

from *Cons(2) r1-in-E1star r2'-in-E2star v'-in-E1*
have *set ?tau* $\subseteq (E_{(ES1 \parallel ES2)})$
by (*simp add*: *composeES-def*, *auto*)
moreover
from *Cons(3)* **have** *set lambda'* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
note *s1-in-E1star*
moreover
from *t2'-in-E2star t2'-is-r2'-v'-s2'* **have** *set s2'* $\subseteq E_{ES2}$
by *simp*
moreover
from *r1-in-E1star v'-in-E1 r2'E1-empty* τ*E1-r1-v'-s1-in-Tr1*
have *?tau* $\restriction E_{ES1}$ @ *s1* $\in Tr_{ES1}$
by (*simp only*: *list-subset-iff-projection-neutral*
projection-concatenation-commute projection-def, *auto*)
moreover
from τ*r1E2-t2'-in-Tr2 t2'-is-r2'-v'-s2' v'-in-E2*
have *?tau* $\restriction E_{ES2}$ @ *s2'* $\in Tr_{ES2}$
proof −
from *v'-in-E2 r2'-in-E2star*
have $(\tau$ @ *r1* @ *r2'* @ $[\mathcal{V}']) \restriction E_{ES2} = (\tau$ @ *r1*$) \restriction E_{ES2}$ @ *r2'* @ $[\mathcal{V}']$
by (*simp only*: *projection-concatenation-commute*
list-subset-iff-projection-neutral projection-def, *auto*)
with τ*r1E2-t2'-in-Tr2 t2'-is-r2'-v'-s2' v'-in-E2* **show** *?thesis*
by *simp*
qed
moreover
have *lambda'* $\restriction E_{ES1} =$ *s1* $\restriction V_{\mathcal{V}}$
proof −
from *Cons(3,4,8) v'-in-E1* **have** *t1* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*lambda'* $\restriction E_{ES1}$)
by (*simp add*: *projection-def*)
moreover
from *t1-is-r1-v'-s1 r1-Vv-empty v'-in-Vv1 Vv-is-Vv1-union-Vv2*
have *t1* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*s1* $\restriction V_{\mathcal{V}}$)
by (*simp only*: *t1-is-r1-v'-s1 projection-concatenation-commute projection-def*, *auto*)
ultimately show *?thesis*
by *auto*
qed
moreover
have *lambda'* $\restriction E_{ES2} =$ *s2'* $\restriction V_{\mathcal{V}}$
proof −
from *Cons(4,5,9) v'-in-E2* **have** *t2* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*lambda'* $\restriction E_{ES2}$)
by (*simp add*: *projection-def*)
moreover
from *t2'-is-r2'-v'-s2' r2'-Vv2-empty r2'-in-E2star v'-in-Vv2 propSepViews*

```
  have t2' ↓ V_𝒱 = [𝒱'] @ (s2' ↓ V_𝒱)
  proof −
    have r2' ↓ V_𝒱 =[]
      using propSepViews unfolding properSeparationOfViews-def
      by (metis projection-on-subset2
        r2'-Vv2-empty r2'-in-E2star subset-iff-psubset-eq)
    with t2'-is-r2'-v'-s2' v'-in-Vv2 Vv-is-Vv1-union-Vv2 show ?thesis
      by (simp only: t2'-is-r2'-v'-s2' projection-concatenation-commute
        projection-def, auto)
  qed
  moreover
  have t2 ↓ V_𝒱 = t2' ↓ V_𝒱
    using propSepViews unfolding properSeparationOfViews-def
    by (metis Int-commute  outerCons-prems(4)
      projection-intersection-neutral
      t2'-Vv2-is-t2-Vv2 t2'-in-E2star)
  ultimately show ?thesis
    by auto
qed
moreover
note s1-Cv1-empty s2'-Cv2-empty Cons.hyps[of ?tau s1 s2']
ultimately obtain t'
  where tau-t'-in-Tr: ?tau @ t' ∈ Tr(ES1 ∥ ES2)
  and t'Vv-is-lambda': t' ↓ V_𝒱 = lambda'
  and t'Cv-empty: t' ↓ C_𝒱 = []
  by auto

let ?t = r1 @ r2' @ [𝒱'] @ t'


note tau-t'-in-Tr
moreover
from r1-Vv-empty r2'-Vv-empty t'Vv-is-lambda' v'-in-Vv
have ?t ↓ V_𝒱 = 𝒱' # lambda'
  by(simp only: projection-concatenation-commute projection-def, auto)
moreover
from VIsViewOnE r1-Cv1-empty t'Cv-empty r2'-Cv2-empty v'-in-Vv
have ?t ↓ C_𝒱 = []
proof −
  from VIsViewOnE v'-in-Vv have [𝒱'] ↓ C_𝒱 = []
    by (simp add:isViewOn-def V-valid-def VC-disjoint-def
      VN-disjoint-def NC-disjoint-def projection-def, auto)
  moreover
  from r1-in-E1star r1-Cv1-empty
  have r1 ↓ C_𝒱 = []
    using propSepViews projection-on-subset2 unfolding properSeparationOfViews-def
    by auto
  moreover
  from r2'-in-E2star r2'-Cv2-empty
  have r2' ↓ C_𝒱 = []
    using propSepViews projection-on-subset2 unfolding properSeparationOfViews-def
    by auto
```

```
    moreover
    note t'Cv-empty
    ultimately show ?thesis
      by (simp only: projection-concatenation-commute, auto)
  qed
  ultimately have ?thesis
    by auto
}
moreover {
  assume v'-in-Vv1-minus-E2: v' ∈ V_V1 − E_ES2
  hence v'-in-Vv1: v' ∈ V_V1
    by auto
  with v'-in-Vv  have v'-in-E1: v' ∈ E_ES1
    using propSepViews unfolding properSeparationOfViews-def
    by auto

  from v'-in-Vv1-minus-E2 have v'-notin-E2: v' ∉ E_ES2
    by (auto)
  with validV2 have v'-notin-Vv2: v' ∉ V_V2
    by (simp add: isViewOn-def V-valid-def VC-disjoint-def
      VN-disjoint-def NC-disjoint-def, auto)


  from Cons(3) Cons(4) Cons(8) v'-in-E1
  have t1 ↾ V_V = v' # (lambda' ↾ E_ES1)
    by (simp add: projection-def)
  from projection-split-first[OF this] obtain r1 s1
    where t1-is-r1-v'-s1: t1 = r1 @ [v'] @ s1
    and r1-Vv-empty: r1 ↾ V_V = []
    by auto
  with Vv-is-Vv1-union-Vv2 projection-on-subset[of V_V1 V_V r1]
  have r1-Vv1-empty: r1 ↾ V_V1 = []
    by auto


  from t1-is-r1-v'-s1 Cons(10) have r1-Cv1-empty: r1 ↾ C_V1 = []
    by (simp add: projection-concatenation-commute)

  from t1-is-r1-v'-s1 Cons(10) have s1-Cv1-empty: s1 ↾ C_V1 = []
    by (simp only: projection-concatenation-commute, auto)

  from Cons(4) t1-is-r1-v'-s1 have r1-in-E1star: set r1 ⊆ E_ES1
    by auto

  have r1-in-Nv1star: set r1 ⊆ N_V1
  proof −
    note r1-in-E1star
    moreover
    from r1-Vv1-empty have set r1 ∩ V_V1 = {}
      by (metis Compl-Diff-eq Diff-cancel Diff-eq
        Int-commute Int-empty-right disjoint-eq-subset-Compl
        list-subset-iff-projection-neutral projection-on-union)
```

moreover
from *r1-Cv1-empty* **have** *set r1* $\cap$ $C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Diff-eq Int-commute Int-empty-right disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*:*isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed

have *r1E2-in-Nv1-inter-C2-star*: *set* (*r1* $\restriction$ E_{ES2}) $\subseteq$ ($N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$)
proof −
have *set* (*r1* $\restriction$ E_{ES2}) = *set r1* $\cap$ E_{ES2}
by (*simp add*: *projection-def*, *auto*)
with *r1-in-Nv1star* **have** *set* (*r1* $\restriction$ E_{ES2}) $\subseteq$ ($E_{ES2} \cap N_{\mathcal{V}1}$)
by *auto*
moreover
from *validV2 disjoint-Nv1-Vv2*
have $E_{ES2} \cap N_{\mathcal{V}1} = N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
ultimately show *?thesis*
by *auto*
qed

note *outerCons-prems* = *Cons.prems*

have *set* (*r1* $\restriction$ E_{ES2}) $\subseteq$ ($N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$) $\Longrightarrow$
$\exists$ *t2'*. (*set t2'* $\subseteq$ E_{ES2}
$\wedge$ ((*τ* @ *r1*) $\restriction$ E_{ES2}) @ *t2'* $\in$ Tr_{ES2}
$\wedge$ *t2'* $\restriction$ $V_{\mathcal{V}2}$ = *t2* $\restriction$ $V_{\mathcal{V}2}$
$\wedge$ *t2'* $\restriction$ $C_{\mathcal{V}2}$ = [])
proof (*induct r1* $\restriction$ E_{ES2} *arbitrary*: *r1 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*10*) *outerCons-prems*(*4*) *outerCons-prems*(*6*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE2*: *xs* = *xs* $\restriction$ E_{ES2}
proof −
from *snoc*(*2*) **have** *set* (*xs* @ [*x*]) $\subseteq$ E_{ES2}
by (*simp add*: *projection-def*, *auto*)
hence *set xs* $\subseteq$ E_{ES2}
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed

moreover
have *set* $(xs \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
  have *set* $(r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
    by (*metis Int-commute snoc.prems*)
  with *snoc*(*2*) **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
    by *simp*
  hence *set xs* $\subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
    by *auto*
  with *xs-is-xsE2* **show** *?thesis*
    by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *t2''*
  where *t2''-in-E2star*: *set t2''* $\subseteq E_{ES2}$
  and *τ-xs-E2-t2''-in-Tr2*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} t2'' \in Tr_{ES2}$
  and *t2''Vv2-is-t2Vv2*: $t2'' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
  and *t2''Cv2-empty*: $t2'' \restriction C_{\mathcal{V}2} = []$
  by *auto*

have *x-in-Cv2-inter-Nv1*: $x \in C_{\mathcal{V}2} \cap N_{\mathcal{V}1}$
proof −
  from *snoc*(*2*−*3*) **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
    by *simp*
  thus *?thesis*
    by *auto*
qed
hence *x-in-Cv2*: $x \in C_{\mathcal{V}2}$
  by *auto*
moreover
note *τ-xs-E2-t2''-in-Tr2 t2''Cv2-empty*
moreover
have *Adm*: (*Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $((\tau \mathbin{@} xs) \restriction E_{ES2})$ x)
proof −
  from *τ-xs-E2-t2''-in-Tr2 validES2*
  have *τ-xsE2-in-Tr2*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \in Tr_{ES2}$
    by (*simp add*: *ES-valid-def traces-prefixclosed-def*
      prefixclosed-def prefix-def)
  with *x-in-Cv2-inter-Nv1 total-ES2-Cv2-inter-Nv1*
  have *τ-xsE2-x-in-Tr2*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} [x] \in Tr_{ES2}$
    by (*simp only*: *total-def*)
  moreover
  have $((\tau \mathbin{@} xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2) = ((\tau \mathbin{@} xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2)$ **..**
  ultimately show *?thesis*
    by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA*
ultimately obtain *t2'*
  where *res1*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} [x] \mathbin{@} t2' \in Tr_{ES2}$
  and *res2*: $t2' \restriction V_{\mathcal{V}2} = t2'' \restriction V_{\mathcal{V}2}$
  and *res3*: $t2' \restriction C_{\mathcal{V}2} = []$

by (*simp only*: *BSIA-def*, *blast*)

have *set t2′* $\subseteq E_{ES2}$
proof −
from *res1 validES2* **have** *set* (((*τ* @ *xs*) ↑ E_{ES2}) @ [*x*] @ *t2′*) $\subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have ((*τ* @ *r1*) ↑ E_{ES2}) @ *t2′* ∈ Tr_{ES2}
proof −
from *res1 xs-is-xsE2* **have** ((*τ* ↑ E_{ES2}) @ (*xs* @ [*x*])) @ *t2′* ∈ Tr_{ES2}
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(2) *projection-concatenation-commute*)
qed
moreover
from *t2′′Vv2-is-t2Vv2 res2* **have** *t2′* ↑ $V_{\mathcal{V}2}$ = *t2* ↑ $V_{\mathcal{V}2}$
by *auto*
moreover
note *res3*
ultimately show *?case*
by *auto*
qed
from *this*[*OF r1E2-in-Nv1-inter-C2-star*] **obtain** *t2′*
where *t2′-in-E2star*: *set t2′* $\subseteq E_{ES2}$
and *τr1E2-t2′-in-Tr2*: ((*τ* @ *r1*) ↑ E_{ES2}) @ *t2′* ∈ Tr_{ES2}
and *t2′-Vv2-is-t2-Vv2*: *t2′* ↑ $V_{\mathcal{V}2}$ = *t2* ↑ $V_{\mathcal{V}2}$
and *t2′-Cv2-empty*: *t2′* ↑ $C_{\mathcal{V}2}$ = []
by *auto*

let *?tau* = *τ* @ *r1* @ [$\mathcal{V}'$]

from *v′-in-E1 Cons*(2) *r1-in-Nv1star validV1* **have** *set ?tau* $\subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *composeES-def isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
moreover
from *Cons*(3) **have** *set lambda′* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *Cons*(4) *t1-is-r1-v′-s1* **have** *set s1* $\subseteq E_{ES1}$
by *auto*
moreover
note *t2′-in-E2star*
moreover
have *?tau* ↑ E_{ES1} @ *s1* ∈ Tr_{ES1}
by (*metis Cons-eq-appendI append-eq-appendI calculation*(3) *eq-Nil-appendI*
list-subset-iff-projection-neutral Cons.prems(3) *Cons.prems*(5)
projection-concatenation-commute t1-is-r1-v′-s1)
moreover

from $\tau r1E2\text{-}t2'\text{-}in\text{-}Tr2$ $v'\text{-}notin\text{-}E2$
have $?tau \restriction E_{ES2} @ t2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(8) *t1-is-r1-v'-s1 r1-Vv-empty v'-in-E1 v'-in-Vv*
have $lambda' \restriction E_{ES1} = s1 \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(11) *v'-notin-E2 t2'-Vv2-is-t2-Vv2*
have $lambda' \restriction E_{ES2} = t2' \restriction V_{\mathcal{V}}$
proof −
have $t2' \restriction V_{\mathcal{V}} = t2' \restriction V_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def projection-intersection-neutral
t2'-in-E2star)
moreover
have $t2 \restriction V_{\mathcal{V}} = t2 \restriction V_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def
projection-intersection-neutral Cons(5))
moreover
note *Cons*(9) *v'-notin-E2 t2'-Vv2-is-t2-Vv2*
ultimately show *?thesis*
by (*simp add*: *projection-def*)
qed
moreover
note *s1-Cv1-empty t2'-Cv2-empty*
moreover
note *Cons.hyps*(1)[*of ?tau s1 t2'*]
ultimately obtain t'
where *tau-t'-in-Tr*: $?tau @ t' \in Tr_{(ES1 \parallel ES2)}$
and *t'-Vv-is-lambda'*: $t' \restriction V_{\mathcal{V}} = lambda'$
and *t'-Cv-empty*: $t' \restriction C_{\mathcal{V}} = []$
by *auto*

let $?t = r1 @ [\mathcal{V}'] @ t'$

note *tau-t'-in-Tr*
moreover
from *r1-Vv-empty t'-Vv-is-lambda' v'-in-Vv*
have $?t \restriction V_{\mathcal{V}} = \mathcal{V}' \# lambda'$
by (*simp add*: *projection-def*)
moreover
have $?t \restriction C_{\mathcal{V}} = []$
proof −
have $r1 \restriction C_{\mathcal{V}} = []$
proof −
from *propSepViews* **have** $E_{ES1} \cap C_{\mathcal{V}} \subseteq C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** *auto*

```
    from projection-on-subset[OF ‹E_ES1 ∩ C_V ⊆ C_V1› r1-Cv1-empty]
    have r1 ↾ (E_ES1 ∩ C_V) = []
      by (simp only: Int-commute)
    with projection-intersection-neutral[OF r1-in-E1star, of C_V] show ?thesis
      by simp
  qed
  with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
    by (simp add:isViewOn-def V-valid-def  VC-disjoint-def
      VN-disjoint-def NC-disjoint-def projection-def, auto)
 qed
 ultimately have ?thesis
  by auto
}
moreover {
 assume v'-in-Vv2-minus-E1: V' ∈ V_V2 − E_ES1
 hence v'-in-Vv2: V' ∈ V_V2
  by auto
 with v'-in-Vv  have v'-in-E2: V' ∈ E_ES2
  using propSepViews unfolding properSeparationOfViews-def
  by auto

 from v'-in-Vv2-minus-E1 have v'-notin-E1: V' ∉ E_ES1
  by (auto)
 with validV1 have v'-notin-Vv1: V' ∉ V_V1
  by (simp add: isViewOn-def V-valid-def
    VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)


 from Cons(4) Cons(5) Cons(9) v'-in-E2 have t2 ↾ V_V = V' # (lambda' ↾ E_ES2)
  by (simp add: projection-def)
 from projection-split-first[OF this] obtain r2 s2
  where t2-is-r2-v'-s2: t2 = r2 @ [V'] @ s2
  and r2-Vv-empty: r2 ↾ V_V = []
  by auto
 with Vv-is-Vv1-union-Vv2 projection-on-subset[of V_V2 V_V r2]
 have r2-Vv2-empty: r2 ↾ V_V2 = []
  by auto


 from t2-is-r2-v'-s2 Cons(11) have r2-Cv2-empty: r2 ↾ C_V2 = []
  by (simp add: projection-concatenation-commute)

 from t2-is-r2-v'-s2 Cons(11) have s2-Cv2-empty: s2 ↾ C_V2 = []
  by (simp only: projection-concatenation-commute, auto)

 from Cons(5) t2-is-r2-v'-s2 have r2-in-E2star: set r2 ⊆ E_ES2
  by auto

 have r2-in-Nv2star: set r2 ⊆ N_V2
 proof −
  note r2-in-E2star
  moreover
```

from *r2-Vv2-empty* **have** *set r2* $\cap$ $V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
from *r2-Cv2-empty* **have** *set r2* $\cap$ $C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2 disjoint-eq-subset-Compl list-subset-iff-projection-neutral projection-on-union*)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
qed
with *Nv2-inter-E1-empty* **have** *r2E1-empty*: *r2* $\restriction$ $E_{ES1} = []$
by (*metis Int-commute empty-subsetI projection-on-subset2 r2-Vv2-empty*)

let *?tau* = τ @ *r2* @ $[\mathcal{V}']$

from *v'-in-E2 Cons*(*2*) *r2-in-Nv2star validV2* **have** *set ?tau* $\subseteq$ $E_{(ES1 \parallel ES2)}$
by (*simp only*: *composeES-def isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)
moreover
from *Cons*(*3*) **have** *set lambda'* $\subseteq$ $V_{\mathcal{V}}$
by *auto*
moreover
note *Cons*(*4*)
moreover
from *Cons*(*5*) *t2-is-r2-v'-s2* **have** *set s2* $\subseteq$ E_{ES2}
by *auto*
moreover
have *?tau* $\restriction$ E_{ES1} @ *t1* $\in$ Tr_{ES1}
proof −
from *v'-notin-E1* **have** $[\mathcal{V}'] \restriction E_{ES1} = []$
by (*simp add*: *projection-def*)
with *Cons*(*6*) *Cons*(*3*) *t2-is-r2-v'-s2 v'-notin-E1 r2-in-Nv2star Nv2-inter-E1-empty r2E1-empty*
show *?thesis*
by (*simp only*: *t2-is-r2-v'-s2 list-subset-iff-projection-neutral projection-concatenation-commute*, *auto*)
qed
moreover
have *?tau* $\restriction$ E_{ES2} @ *s2* $\in$ Tr_{ES2}
by (*metis Cons-eq-appendI append-eq-appendI calculation*(*4*) *eq-Nil-appendI list-subset-iff-projection-neutral Cons.prems*(*4*) *Cons.prems*(*6*) *projection-concatenation-commute t2-is-r2-v'-s2*)
moreover
from *Cons*(*8*) *v'-notin-E1* **have** *lambda'* $\restriction$ E_{ES1} = *t1* $\restriction$ $V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover

```
        from Cons(9) t2-is-r2-v'-s2 r2-Vv-empty v'-in-E2 v'-in-Vv
        have lambda' ↾ E_ES2 = s2 ↾ V_𝒱
          by (simp add: projection-def)
        moreover
        note Cons(10) s2-Cv2-empty
        moreover
        note Cons.hyps(1)[of ?tau t1 s2]
        ultimately obtain t'
          where tau-t'-in-Tr: ?tau @ t' ∈ Tr_(ES1 ∥ ES2)
          and t'-Vv-is-lambda': t' ↾ V_𝒱 = lambda'
          and t'-Cv-empty: t' ↾ C_𝒱 = []
          by auto

        let ?t = r2 @ [𝒱'] @ t'


        note tau-t'-in-Tr
        moreover
        from r2-Vv-empty t'-Vv-is-lambda' v'-in-Vv
          have ?t ↾ V_𝒱 = 𝒱' # lambda'
          by (simp add: projection-def)
        moreover
        have ?t ↾ C_𝒱 = []
        proof −
          have r2 ↾ C_𝒱 = []
            using propSepViews unfolding properSeparationOfViews-def
            by (metis  projection-on-subset2
              r2-Cv2-empty r2-in-E2star)
          with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
            by (simp add: isViewOn-def V-valid-def
              VC-disjoint-def VN-disjoint-def NC-disjoint-def projection-def, auto)
        qed
        ultimately have ?thesis
          by auto
      }
      ultimately show ?thesis
        by blast
    qed
  qed
 }
 thus ?thesis
  by auto
qed


lemma generalized-zipping-lemma4:
⟦ ∇_Γ1 ⊆ E_ES1; Δ_Γ1 ⊆ E_ES1; ϒ_Γ1 ⊆ E_ES1; ∇_Γ2 ⊆ E_ES2; Δ_Γ2 ⊆ E_ES2; ϒ_Γ2 ⊆ E_ES2;
  BSIA ϱ1 𝒱1 Tr_ES1; BSIA ϱ2 𝒱2 Tr_ES2; total ES1 (C_𝒱1 ∩ N_𝒱2); total ES2 (C_𝒱2 ∩ N_𝒱1);
  FCIA ϱ1 Γ1 𝒱1 Tr_ES1; FCIA ϱ2 Γ2 𝒱2 Tr_ES2; V_𝒱1 ∩ V_𝒱2 ⊆ ∇_Γ1 ∪ ∇_Γ2;
  C_𝒱1 ∩ N_𝒱2 ⊆ ϒ_Γ1; C_𝒱2 ∩ N_𝒱1 ⊆ ϒ_Γ2;
  N_𝒱1 ∩ Δ_Γ1 ∩ E_ES2 = {}; N_𝒱2 ∩ Δ_Γ2 ∩ E_ES1 = {} ⟧ ⟹
 ∀ τ lambda t1 t2. ( ( set τ ⊆ (E_(ES1 ∥ ES2)) ∧ set lambda ⊆ V_𝒱  ∧ set t1 ⊆ E_ES1
```

$\wedge$ *set t2* $\subseteq E_{ES2} \wedge ((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$
$\wedge$ $(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}}) \wedge (lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$
$\wedge$ $(t1 \restriction C_{\mathcal{V}1}) = [] \wedge (t2 \restriction C_{\mathcal{V}2}) = []$)
$\longrightarrow$ $(\exists\, t.\ ((\tau @ t) \in (Tr_{(ES1 \parallel ES2)}) \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = [])))$
proof −
assume *Nabla1-subsetof-E1*: $\nabla_{\Gamma 1} \subseteq E_{ES1}$
and *Delta1-subsetof-E1*: $\Delta_{\Gamma 1} \subseteq E_{ES1}$
and *Upsilon1-subsetof-E1*: $\Upsilon_{\Gamma 1} \subseteq E_{ES1}$
and *Nabla2-subsetof-E2*: $\nabla_{\Gamma 2} \subseteq E_{ES2}$
and *Delta2-subsetof-E2*: $\Delta_{\Gamma 2} \subseteq E_{ES2}$
and *Upsilon2-subsetof-E2*: $\Upsilon_{\Gamma 2} \subseteq E_{ES2}$
and *BSIA1*: *BSIA* $\varrho 1\ \mathcal{V}1\ Tr_{ES1}$
and *BSIA2*: *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
and *ES1-total-Cv1-inter-Nv2*: *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2})$
and *ES2-total-Cv2-inter-Nv1*: *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
and *FCIA1*: *FCIA* $\varrho 1\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1}$
and *FCIA2*: *FCIA* $\varrho 2\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}$
and *Vv1-inter-Vv2-subsetof-Nabla1-union-Nabla2*: $V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma 1} \cup \nabla_{\Gamma 2}$
and *Cv1-inter-Nv2-subsetof-Upsilon1*: $C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma 1}$
and *Cv2-inter-Nv1-subsetof-Upsilon2*: $C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma 2}$
and *disjoint-Nv1-inter-Delta1-inter-E2*: $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}$
and *disjoint-Nv2-inter-Delta2-inter-E1*: $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$

{
fix τ *lambda t1 t2*

have ⟦ *set* $\tau \subseteq (E_{(ES1 \parallel ES2)})$;
set lambda $\subseteq V_{\mathcal{V}}$;
set t1 $\subseteq E_{ES1}$;
set t2 $\subseteq E_{ES2}$;
$((\tau \restriction E_{ES1}) @ t1) \in Tr_{ES1}$;
$((\tau \restriction E_{ES2}) @ t2) \in Tr_{ES2}$;
$(lambda \restriction E_{ES1}) = (t1 \restriction V_{\mathcal{V}})$;
$(lambda \restriction E_{ES2}) = (t2 \restriction V_{\mathcal{V}})$;
$(t1 \restriction C_{\mathcal{V}1}) = []$;
$(t2 \restriction C_{\mathcal{V}2}) = []$ ⟧
$\Longrightarrow$ $(\exists\ t.\ ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \restriction V_{\mathcal{V}}) = lambda \wedge (t \restriction C_{\mathcal{V}}) = []))$
proof (*induct lambda arbitrary*: τ *t1 t2*)
case (*Nil* τ *t1 t2*)

have $(\tau @ []) \in Tr_{(ES1 \parallel ES2)}$
proof −
have $\tau \in Tr_{(ES1 \parallel ES2)}$
proof −
from *Nil*(*5*) *validES1* **have** $\tau \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
moreover
from *Nil*(*6*) *validES2* **have** $\tau \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
moreover

```
        note Nil(1)
        ultimately show ?thesis
          by (simp add: composeES-def)
      qed
    thus ?thesis
      by auto
  qed
 moreover
 have ([] ↾ V_𝒱) = []
  by (simp add: projection-def)
 moreover
 have ([] ↾ C_𝒱) = []
  by (simp add: projection-def)
 ultimately show ?case
  by blast
next
 case (Cons 𝒱' lambda' τ t1 t2)
 thus ?case
  proof −

   from Cons(3) have v'-in-Vv: 𝒱' ∈ V_𝒱
    by auto

   have 𝒱' ∈ V_𝒱1 ∩ V_𝒱2 ∩ ∇_Γ1
    ∨ 𝒱' ∈ V_𝒱1 ∩ V_𝒱2 ∩ ∇_Γ2
    ∨ 𝒱' ∈ V_𝒱1 − E_ES2
    ∨ 𝒱' ∈ V_𝒱2 − E_ES1
    proof −
     let ?S = V_𝒱1 ∩ V_𝒱2 ∪ ( V_𝒱1 − V_𝒱2 ) ∪ ( V_𝒱2 − V_𝒱1 )
     have V_𝒱1 ∪ V_𝒱2 = ?S
      by auto
     moreover
     have V_𝒱1 − V_𝒱2 = V_𝒱1 − E_ES2
      and V_𝒱2 − V_𝒱1 = V_𝒱2 − E_ES1
      using propSepViews unfolding properSeparationOfViews-def by auto
     moreover
     note Vv1-inter-Vv2-subsetof-Nabla1-union-Nabla2
       Vv-is-Vv1-union-Vv2 v'-in-Vv
     ultimately show ?thesis
      by auto
    qed
   moreover
   {
    assume v'-in-Vv1-inter-Vv2-inter-Nabla1: 𝒱' ∈ V_𝒱1 ∩ V_𝒱2 ∩ ∇_Γ1
    hence v'-in-Vv1: 𝒱' ∈ V_𝒱1 and v'-in-Vv2: 𝒱' ∈ V_𝒱2
     and v'-in-Nabla2: 𝒱' ∈ ∇_Γ1
     by auto
    with v'-in-Vv
    have v'-in-E1: 𝒱' ∈ E_ES1 and v'-in-E2: 𝒱' ∈ E_ES2
     using propSepViews unfolding properSeparationOfViews-def by auto

   from Cons(3−4) Cons(8) v'-in-E1 have t1 ↾ V_𝒱 = 𝒱' # (lambda' ↾ E_ES1)
```

by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r1 s1*
where *t1-is-r1-v'-s1*: $t1 = r1 \mathbin{@} [\mathcal{V}'] \mathbin{@} s1$
and *r1-Vv-empty*: $r1 \restriction V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}1}$ $V_{\mathcal{V}}$ *r1*]
have *r1-Vv1-empty*: $r1 \restriction V_{\mathcal{V}1} = []$
by *auto*

from *t1-is-r1-v'-s1 Cons*(*10*) **have** *r1-Cv1-empty*: $r1 \restriction C_{\mathcal{V}1} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t1-is-r1-v'-s1 Cons*(*10*) **have** *s1-Cv1-empty*: $s1 \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*4*) *t1-is-r1-v'-s1*
have *r1-in-E1star*: *set r1* $\subseteq E_{ES1}$ **and** *s1-in-E1star*: *set s1* $\subseteq E_{ES1}$
by *auto*

have *r1-in-Nv1star*: *set r1* $\subseteq N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** *set r1* $\cap$ $V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r1-Cv1-empty* **have** *set r1* $\cap$ $C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r1E2-in-Nv1-inter-C2-star*: *set* $(r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
have *set* $(r1 \restriction E_{ES2})$ = *set r1* $\cap$ E_{ES2}
by (*simp add*: *projection-def*, *auto*)
with *r1-in-Nv1star* **have** *set* $(r1 \restriction E_{ES2}) \subseteq (E_{ES2} \cap N_{\mathcal{V}1})$
by *auto*
moreover
from *validV2 disjoint-Nv1-Vv2*
have $E_{ES2} \cap N_{\mathcal{V}1} = N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*

```
    by auto
  qed
with Cv2-inter-Nv1-subsetof-Upsilon2
have r1E2-in-Nv1-inter-C2-Upsilon2-star: set (r1 ↾ E_ES2) ⊆ (N_V1 ∩ C_V2 ∩ ϒ_Γ2)
  by auto

note outerCons-prems = Cons.prems

have set (r1 ↾ E_ES2) ⊆ (N_V1 ∩ C_V2) ⟹
  ∃ t2'. ( set t2' ⊆ E_ES2
  ∧ ((τ @ r1) ↾ E_ES2) @ t2' ∈ Tr_ES2
  ∧ t2' ↾ V_V2 = t2 ↾ V_V2
  ∧ t2' ↾ C_V2 = [] )
proof (induct r1 ↾ E_ES2 arbitrary: r1 rule: rev-induct)
  case Nil thus ?case
    by (metis append-self-conv outerCons-prems(10)
      outerCons-prems(4) outerCons-prems(6) projection-concatenation-commute)
next
  case (snoc x xs)

  have xs-is-xsE2: xs = xs ↾ E_ES2
    proof −
      from snoc(2) have set (xs @ [x]) ⊆ E_ES2
        by (simp add: projection-def, auto)
      hence set xs ⊆ E_ES2
        by auto
      thus ?thesis
        by (simp add: list-subset-iff-projection-neutral)
    qed
  moreover
  have set (xs ↾ E_ES2) ⊆ (N_V1 ∩ C_V2)
    proof −
      have set (r1 ↾ E_ES2) ⊆ (N_V1 ∩ C_V2)
        by (metis Int-commute snoc.prems)
      with snoc(2) have set (xs @ [x]) ⊆ (N_V1 ∩ C_V2)
        by simp
      hence set xs ⊆ (N_V1 ∩ C_V2)
        by auto
      with xs-is-xsE2 show ?thesis
        by auto
    qed
  moreover
  note snoc.hyps(1)[of xs]
  ultimately obtain t2''
    where t2''-in-E2star: set t2'' ⊆ E_ES2
    and τ-xs-E2-t2''-in-Tr2: ((τ @ xs) ↾ E_ES2) @ t2'' ∈ Tr_ES2
    and t2''Vv2-is-t2Vv2: t2'' ↾ V_V2 = t2 ↾ V_V2
    and t2''Cv2-empty: t2'' ↾ C_V2 = []
    by auto

  have x-in-Cv2-inter-Nv1: x ∈ C_V2 ∩ N_V1
    proof −
```

from *snoc*(2−3) **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
by *simp*
thus *?thesis*
by *auto*
qed
hence *x-in-Cv2*: $x \in C_{\mathcal{V}2}$
by *auto*
moreover
note *τ-xs-E2-t2''-in-Tr2 t2''Cv2-empty*
moreover
have *Adm*: (*Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $((\tau \mathbin{@} xs) \restriction E_{ES2})$ x)
proof −
from *τ-xs-E2-t2''-in-Tr2 validES2*
have *τ-xsE2-in-Tr2*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def)
with *x-in-Cv2-inter-Nv1 ES2-total-Cv2-inter-Nv1*
have *τ-xsE2-x-in-Tr2*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} [x] \in Tr_{ES2}$
by (*simp only*: *total-def*)
moreover
have $((\tau \mathbin{@} xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2) = ((\tau \mathbin{@} xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA2*
ultimately obtain *t2'*
where *res1*: $((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} [x] \mathbin{@} t2' \in Tr_{ES2}$
and *res2*: $t2' \restriction V_{\mathcal{V}2} = t2'' \restriction V_{\mathcal{V}2}$
and *res3*: $t2' \restriction C_{\mathcal{V}2} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t2'* $\subseteq E_{ES2}$
proof −
from *res1 validES2* **have** *set* $(((\tau \mathbin{@} xs) \restriction E_{ES2}) \mathbin{@} [x] \mathbin{@} t2') \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau \mathbin{@} r1) \restriction E_{ES2}) \mathbin{@} t2' \in Tr_{ES2}$
proof −
from *res1 xs-is-xsE2* **have** $((\tau \restriction E_{ES2}) \mathbin{@} (xs \mathbin{@} [x])) \mathbin{@} t2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(2) *projection-concatenation-commute*)
qed
moreover
from *t2''Vv2-is-t2Vv2 res2* **have** $t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
by *auto*
moreover
note *res3*
ultimately show *?case*

```
    by auto
qed
from this[OF r1E2-in-Nv1-inter-C2-star] obtain t2'
  where t2'-in-E2star: set t2' ⊆ E_ES2
  and τr1E2-t2'-in-Tr2: ((τ @ r1) ↾ E_ES2) @ t2' ∈ Tr_ES2
  and t2'-Vv2-is-t2-Vv2: t2' ↾ V_V2 = t2 ↾ V_V2
  and t2'-Cv2-empty: t2' ↾ C_V2 = []
  by auto

have t2' ↾ V_V2 = V' # (lambda' ↾ E_ES2)
  proof −
    from projection-intersection-neutral[OF Cons(5), of V_V]
    have t2 ↾ V_V = t2 ↾ V_V2
      using propSepViews unfolding properSeparationOfViews-def
      by (simp only: Int-commute)
    with Cons(9) t2'-Vv2-is-t2-Vv2 v'-in-E2 show ?thesis
      by (simp add: projection-def)
  qed
from projection-split-first[OF this] obtain r2' s2'
  where t2'-is-r2'-v'-s2': t2' = r2' @ [V'] @ s2'
  and r2'-Vv2-empty: r2' ↾ V_V2 = []
  by auto

from t2'-is-r2'-v'-s2' t2'-Cv2-empty have r2'-Cv2-empty: r2' ↾ C_V2 = []
  by (simp add: projection-concatenation-commute)

from t2'-is-r2'-v'-s2' t2'-Cv2-empty have s2'-Cv2-empty: s2' ↾ C_V2 = []
  by (simp only: projection-concatenation-commute, auto)

from t2'-in-E2star t2'-is-r2'-v'-s2' have r2'-in-E2star: set r2' ⊆ E_ES2
  by auto

have r2'-in-Nv2star: set r2' ⊆ N_V2
  proof −
    note r2'-in-E2star
    moreover
    from r2'-Vv2-empty have set r2' ∩ V_V2 = {}
      by (metis Compl-Diff-eq Diff-cancel Un-upper2
        disjoint-eq-subset-Compl list-subset-iff-projection-neutral
        projection-on-union)
    moreover
    from r2'-Cv2-empty have set r2' ∩ C_V2 = {}
      by (metis Compl-Diff-eq Diff-cancel Un-upper2
        disjoint-eq-subset-Compl list-subset-iff-projection-neutral
        projection-on-union)
    moreover
    note validV2
    ultimately show ?thesis
      by (simp add: isViewOn-def V-valid-def
        VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
  qed
```

have *r2′E1-in-Nv2-inter-C1-star*: *set* $(r2' \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have *set* $(r2' \restriction E_{ES1}) =$ *set* $r2' \cap E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
with *r2′-in-Nv2star* **have** *set* $(r2' \restriction E_{ES1}) \subseteq (E_{ES1} \cap N_{\mathcal{V}2})$
by *auto*
moreover
from *validV1* *disjoint-Nv2-Vv1*
have $E_{ES1} \cap N_{\mathcal{V}2} = N_{\mathcal{V}2} \cap C_{\mathcal{V}1}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
with *Cv1-inter-Nv2-subsetof-Upsilon1*
have *r2′E1-in-Nv2-inter-Cv1-Upsilon1-star*:
set $(r2' \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
by *auto*

have *set* $(r2' \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \Longrightarrow$
$\exists$ *s1′ q1′*. (
set s1′ $\subseteq E_{ES1} \wedge$ *set q1′* $\subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge (\tau \restriction E_{ES1})$ @ *r1* @ *q1′* @ $[\mathcal{V}']$ @ *s1′* $\in Tr_{ES1}$
$\wedge$ *q1′* $\restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = r2' \restriction E_{ES1}$
$\wedge$ *s1′* $\restriction V_{\mathcal{V}1} =$ *s1* $\restriction V_{\mathcal{V}1}$
$\wedge$ *s1′* $\restriction C_{\mathcal{V}1} = []$)
proof (*induct r2′* $\restriction E_{ES1}$ *arbitrary*: *r2′ rule*: *rev-induct*)
case *Nil*

note *s1-in-E1star*
moreover
have *set* $[] \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
from *outerCons-prems*(5) *t1-is-r1-v′-s1*
have $\tau \restriction E_{ES1}$ @ *r1* @ [] @ $[\mathcal{V}']$ @ *s1* $\in Tr_{ES1}$
by *auto*
moreover
from *Nil* **have** [] $\restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = r2' \restriction E_{ES1}$
by (*simp add*: *projection-def*)
moreover
have *s1* $\restriction V_{\mathcal{V}1} =$ *s1* $\restriction V_{\mathcal{V}1}$..
moreover
note *s1-Cv1-empty*
ultimately show *?case*
by *blast*

next
case (*snoc x xs*)

have *xs-is-xsE1*: $xs = xs \restriction E_{ES1}$
proof −
from *snoc(2)* **have** *set* $(xs \mathbin{@} [x]) \subseteq E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* $(xs \restriction E_{ES1}) \subseteq N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
proof −
from *snoc(2−3)* **have** *set* $(xs \mathbin{@} [x]) \subseteq N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by *simp*
with *xs-is-xsE1* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps(1)*[*of xs*]
ultimately obtain *s1'' q1''*
where *s1''-in-E1star*: *set s1''* $\subseteq E_{ES1}$
and *q1''-in-C1-inter-Upsilon1-inter-Delta1*: *set q1''* $\subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *τE1-r1-q1''-v'-s1''-in-Tr1*: $(\tau \restriction E_{ES1} \mathbin{@} r1 \mathbin{@} q1'') \mathbin{@} [\mathcal{V}'] \mathbin{@} s1'' \in Tr_{ES1}$
and *q1''C1-Upsilon1-is-xsE1*: $q1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = xs \restriction E_{ES1}$
and *s1''V1-is-s1V1*: $s1'' \restriction V_{\mathcal{V}1} = s1 \restriction V_{\mathcal{V}1}$
and *s1''C1-empty*: $s1'' \restriction C_{\mathcal{V}1} = []$
by *auto*

have *x-in-Cv1-inter-Upsilon1*: $x \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
and *x-in-Cv1-inter-Nv2*: $x \in C_{\mathcal{V}1} \cap N_{\mathcal{V}2}$
proof −
from *snoc(2−3)* **have** *set* $(xs \mathbin{@} [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
by *simp*
thus $x \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
and $x \in C_{\mathcal{V}1} \cap N_{\mathcal{V}2}$
by *auto*
qed
with *validV1* **have** *x-in-E1*: $x \in E_{ES1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

note *x-in-Cv1-inter-Upsilon1*
moreover
from *v'-in-Vv1-inter-Vv2-inter-Nabla1* **have** $\mathcal{V}' \in V_{\mathcal{V}1} \cap \nabla_{\Gamma 1}$
by *auto*
moreover
note *τE1-r1-q1''-v'-s1''-in-Tr1 s1''C1-empty*
moreover
have *Adm*: $(Adm\ \mathcal{V}1\ \varrho 1\ Tr_{ES1}\ (\tau \restriction E_{ES1} \mathbin{@} r1 \mathbin{@} q1'')\ x)$
proof −
from *τE1-r1-q1''-v'-s1''-in-Tr1 validES1*
have $(\tau \restriction E_{ES1} \mathbin{@} r1 \mathbin{@} q1'') \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def)

```
    with x-in-Cv1-inter-Nv2 ES1-total-Cv1-inter-Nv2
    have (τ ↑ E_ES1 @ r1 @ q1'') @ [x] ∈ Tr_ES1
      by (simp only: total-def)
    moreover
    have (τ ↑ E_ES1 @ r1 @ q1'') ↑ (ϱ1 V1) = (τ ↑ E_ES1 @ r1 @ q1'') ↑ (ϱ1 V1) ..
    ultimately show ?thesis
      by (simp only: Adm-def, blast)
  qed
moreover
note FCIA1
ultimately
obtain s1' γ'
  where res1: (set γ') ⊆ (N_V1 ∩ Δ_Γ1)
  and res2: ((τ ↑ E_ES1 @ r1 @ q1'') @ [x] @ γ' @ [V'] @ s1') ∈ Tr_ES1
  and res3: (s1' ↑ V_V1) = (s1'' ↑ V_V1)
  and res4: s1' ↑ C_V1 = []
  unfolding FCIA-def
  by blast

let ?q1' = q1'' @ [x] @ γ'

from res2 validES1 have set s1' ⊆ E_ES1
  by (simp add: ES-valid-def traces-contain-events-def, auto)
moreover
from res1 x-in-Cv1-inter-Upsilon1 q1''-in-C1-inter-Upsilon1-inter-Delta1
have set ?q1' ⊆ C_V1 ∩ Υ_Γ1 ∪ N_V1 ∩ Δ_Γ1
  by auto
moreover
from res2 have τ ↑ E_ES1 @ r1 @ ?q1' @ [V'] @ s1' ∈ Tr_ES1
  by auto
moreover
have ?q1' ↑ (C_V1 ∩ Υ_Γ1) = r2' ↑ E_ES1
  proof −
    from validV1 res1 have γ' ↑ (C_V1 ∩ Υ_Γ1) = []
      proof −
        from res1 have γ' = γ' ↑ (N_V1 ∩ Δ_Γ1)
          by (simp only: list-subset-iff-projection-neutral)
        hence γ' ↑ (C_V1 ∩ Υ_Γ1) = γ' ↑ (N_V1 ∩ Δ_Γ1) ↑ (C_V1 ∩ Υ_Γ1)
          by simp
        hence γ' ↑ (C_V1 ∩ Υ_Γ1) = γ' ↑ (N_V1 ∩ Δ_Γ1 ∩ C_V1 ∩ Υ_Γ1)
          by (simp only: projection-def, auto)
        moreover
        from validV1 have N_V1 ∩ Δ_Γ1 ∩ C_V1 ∩ Υ_Γ1 = {}
          by (simp add: isViewOn-def V-valid-def
            VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
        ultimately show ?thesis
          by (simp add: projection-def)
      qed
    hence ?q1' ↑ (C_V1 ∩ Υ_Γ1) = (q1'' @ [x]) ↑ (C_V1 ∩ Υ_Γ1)
      by (simp only: projection-concatenation-commute, auto)
    with q1''C1-Upsilon1-is-xsE1 x-in-Cv1-inter-Upsilon1
    have ?q1' ↑ (C_V1 ∩ Υ_Γ1) = (xs ↑ E_ES1) @ [x]
```

by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
with *xs-is-xsE1 snoc*(*2*) **show** *?thesis*
by *simp*
qed
moreover
from *res3 s1''V1-is-s1V1* **have** $s1' \restriction V_{\mathcal{V}1} = s1 \restriction V_{\mathcal{V}1}$
by *simp*
moreover
note *res4*
ultimately show *?case*
by *blast*
qed
from *this*[*OF r2'E1-in-Nv2-inter-Cv1-Upsilon1-star*] **obtain** *s1' q1'*
where *s1'-in-E1star*: *set* $s1' \subseteq E_{ES1}$
and *q1'-in-Cv1-inter-Upsilon1-union-Nv1-inter-Delta1*:
set $q1' \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and τ*E1-r1-q1'-v'-s1'-in-Tr1*: $(\tau \restriction E_{ES1})$ @ *r1* @ *q1'* @ $[\mathcal{V}']$ @ $s1' \in Tr_{ES1}$
and *q1'Cv1-inter-Upsilon1-is-r2'E1*: $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = r2' \restriction E_{ES1}$
and *s1'Vv1-is-s1-Vv1*: $s1' \restriction V_{\mathcal{V}1} = s1 \restriction V_{\mathcal{V}1}$
and *s1'Cv1-empty*: $s1' \restriction C_{\mathcal{V}1} = []$
by *auto*

from *q1'-in-Cv1-inter-Upsilon1-union-Nv1-inter-Delta1 validV1*
have *q1'-in-E1star*: *set* $q1' \subseteq E_{ES1}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)

have *r2'Cv-empty*: $r2' \restriction C_{\mathcal{V}} = []$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2*
r2'-Cv2-empty r2'-in-E2star)

from *validES1* τ*E1-r1-q1'-v'-s1'-in-Tr1*
have *q1'-in-E1star*: *set* $q1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
note *r2'-in-E2star*
moreover
have *q1'E2-is-r2'E1*: $q1' \restriction E_{ES2} = r2' \restriction E_{ES1}$
proof −
from *q1'-in-Cv1-inter-Upsilon1-union-Nv1-inter-Delta1*
have $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) = q1'$
by (*simp add*: *list-subset-iff-projection-neutral*)
hence $(q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})) \restriction E_{ES2} = q1' \restriction E_{ES2}$
by *simp*
hence $q1' \restriction ((C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \cap E_{ES2}) = q1' \restriction E_{ES2}$
by (*simp add*: *projection-def*)
hence $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap E_{ES2}) = q1' \restriction E_{ES2}$
by (*simp only*: *Int-Un-distrib2 disjoint-Nv1-inter-Delta1-inter-E2*, *auto*)
moreover
from *q1'Cv1-inter-Upsilon1-is-r2'E1*

```
    have (q1' ⌉ (C_V1 ∩ ϒ_Γ1)) ⌉ E_ES2 = (r2' ⌉ E_ES1) ⌉ E_ES2
      by simp
    hence q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∩ E_ES2) = (r2' ⌉ E_ES2) ⌉ E_ES1
      by (simp add: projection-def conj-commute)
    with r2'-in-E2star have q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∩ E_ES2) = r2' ⌉ E_ES1
      by (simp only: list-subset-iff-projection-neutral)
    ultimately show ?thesis
      by auto
  qed
moreover
have q1' ⌉ V_V = []
  proof −
    from q1'-in-Cv1-inter-Upsilon1-union-Nv1-inter-Delta1
    have q1' = q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∪ N_V1 ∩ Δ_Γ1)
      by (simp add: list-subset-iff-projection-neutral)
    moreover
    from q1'-in-E1star have q1' = q1' ⌉ E_ES1
      by (simp add: list-subset-iff-projection-neutral)
    ultimately have q1' = q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∪ N_V1 ∩ Δ_Γ1) ⌉ E_ES1
      by simp
    hence q1' ⌉ V_V = q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∪ N_V1 ∩ Δ_Γ1) ⌉ E_ES1 ⌉ V_V
      by simp
    hence q1' ⌉ V_V = q1' ⌉ (C_V1 ∩ ϒ_Γ1 ∪ N_V1 ∩ Δ_Γ1) ⌉ (V_V ∩ E_ES1)
      by (simp add: Int-commute projection-def)
    hence q1' ⌉ V_V = q1' ⌉ ((C_V1 ∩ ϒ_Γ1 ∪ N_V1 ∩ Δ_Γ1) ∩ V_V1)
      using propSepViews unfolding properSeparationOfViews-def
      by (simp add: projection-def)
    hence q1' ⌉ V_V = q1' ⌉ (V_V1 ∩ C_V1 ∩ ϒ_Γ1 ∪ V_V1 ∩ N_V1 ∩ Δ_Γ1)
      by (simp add: Int-Un-distrib2, metis Int-assoc Int-commute Int-left-commute Un-commute)
    with validV1 show ?thesis
      by (simp add: isViewOn-def V-valid-def  VC-disjoint-def
        VN-disjoint-def NC-disjoint-def, auto, simp add: projection-def)
  qed
moreover
have r2' ⌉ V_V = []
  using propSepViews unfolding properSeparationOfViews-def
  by (metis Int-commute  projection-intersection-neutral
    r2'-Vv2-empty r2'-in-E2star)
moreover
have q1'Cv-empty: q1' ⌉ C_V = []
  proof −
    from q1'-in-E1star have foo: q1' = q1' ⌉ E_ES1
      by (simp add: list-subset-iff-projection-neutral)
    hence q1' ⌉ C_V = q1' ⌉ (C_V ∩ E_ES1)
      by (metis Int-commute list-subset-iff-projection-neutral projection-intersection-neutral)
    moreover
    from propSepViews have C_V ∩ E_ES1⊆C_V1
      unfolding properSeparationOfViews-def by auto
    from projection-subset-elim[OF ‹C_V ∩ E_ES1⊆C_V1›, of q1']
    have q1' ⌉ C_V1 ⌉ C_V ⌉ E_ES1 = q1' ⌉ (C_V ∩ E_ES1)
      using propSepViews unfolding properSeparationOfViews-def
      by (simp add: projection-def)
```

hence $q1' \restriction E_{ES1} \restriction C_{\mathcal{V}1} \restriction C_{\mathcal{V}} = q1' \restriction (C_{\mathcal{V}} \cap E_{ES1})$
by (*simp add*: *projection-commute*)
with *foo* **have** $q1' \restriction (C_{\mathcal{V}1} \cap C_{\mathcal{V}}) = q1' \restriction (C_{\mathcal{V}} \cap E_{ES1})$
by (*simp add*: *projection-def*)
moreover
from *q1'-in-Cv1-inter-Upsilon1-union-Nv1-inter-Delta1*
have $q1' \restriction (C_{\mathcal{V}1} \cap C_{\mathcal{V}}) = q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction (C_{\mathcal{V}1} \cap C_{\mathcal{V}})$
by (*simp add*: *list-subset-iff-projection-neutral*)
moreover
have $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \cap (C_{\mathcal{V}1} \cap C_{\mathcal{V}})$
$= (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup C_{\mathcal{V}1} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \cap C_{\mathcal{V}}$
by *fast*
hence $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction (C_{\mathcal{V}1} \cap C_{\mathcal{V}})$
$= q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup C_{\mathcal{V}1} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction C_{\mathcal{V}}$
by (*simp add*: *projection-sequence*)
moreover
from *validV1*
have $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cup C_{\mathcal{V}1} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction C_{\mathcal{V}}$
$= q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \restriction C_{\mathcal{V}}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def Int-commute)
moreover
from *q1'Cv1-inter-Upsilon1-is-r2'E1*
have $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \restriction C_{\mathcal{V}} = r2' \restriction E_{ES1} \restriction C_{\mathcal{V}}$
by *simp*
with *projection-on-intersection*[*OF r2'Cv-empty*]
have $q1' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \restriction C_{\mathcal{V}} = []$
by (*simp add*: *Int-commute projection-def*)
ultimately show *?thesis*
by *auto*
qed
moreover
note *r2'Cv-empty merge-property'*[*of q1' r2'*]
ultimately obtain q'
where *q'E1-is-q1'*: $q' \restriction E_{ES1} = q1'$
and *q'E2-is-r2'*: $q' \restriction E_{ES2} = r2'$
and *q'V-empty*: $q' \restriction V_{\mathcal{V}} = []$
and *q'C-empty*: $q' \restriction C_{\mathcal{V}} = []$
and *q'-in-E1-union-E2-star*: *set* $q' \subseteq (E_{ES1} \cup E_{ES2})$
unfolding *Let-def*
by *auto*

let *?tau* $= \tau$ @ *r1* @ q' @ $[\mathcal{V}']$

from *Cons*(*2*) *r1-in-E1star q'-in-E1-union-E2-star v'-in-E1*
have *set ?tau* $\subseteq (E_{(ES1 \parallel ES2)})$
by (*simp add*: *composeES-def*, *auto*)
moreover
from *Cons*(*3*) **have** *set lambda'* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
note *s1'-in-E1star*

moreover
from *t2′-in-E2star t2′-is-r2′-v′-s2′* **have** *set s2′* $\subseteq E_{ES2}$
by *simp*
moreover
from *q′E1-is-q1′ r1-in-E1star v′-in-E1 q1′-in-E1star τE1-r1-q1′-v′-s1′-in-Tr1*
have *?tau* $\restriction E_{ES1}$ @ *s1′* $\in Tr_{ES1}$
by (*simp only*: *list-subset-iff-projection-neutral*
projection-concatenation-commute projection-def, *auto*)
moreover
from *τr1E2-t2′-in-Tr2 t2′-is-r2′-v′-s2′ v′-in-E2 q′E2-is-r2′*
have *?tau* $\restriction E_{ES2}$ @ *s2′* $\in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
have *lambda′* $\restriction E_{ES1} =$ *s1′* $\restriction V_{\mathcal{V}}$
proof −
from *Cons*(*3*−*4*) *Cons*(*8*) *v′-in-E1* **have** *t1* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*lambda′* $\restriction E_{ES1}$)
by (*simp add*: *projection-def*)
moreover
from *t1-is-r1-v′-s1 r1-Vv-empty v′-in-Vv1 Vv-is-Vv1-union-Vv2*
have *t1* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*s1* $\restriction V_{\mathcal{V}}$)
by (*simp only*: *t1-is-r1-v′-s1 projection-concatenation-commute*
projection-def, *auto*)
moreover
have *s1* $\restriction V_{\mathcal{V}} =$ *s1′* $\restriction V_{\mathcal{V}}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*
s1′Vv1-is-s1-Vv1 s1′-in-E1star s1-in-E1star)
ultimately show *?thesis*
by *auto*
qed
moreover
have *lambda′* $\restriction E_{ES2} =$ *s2′* $\restriction V_{\mathcal{V}}$
proof −
from *Cons*(*3*,*5*,*9*) *v′-in-E2* **have** *t2* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*lambda′* $\restriction E_{ES2}$)
by (*simp add*: *projection-def*)
moreover
from *t2′-is-r2′-v′-s2′ r2′-Vv2-empty r2′-in-E2star v′-in-Vv2 propSepViews*
have *t2′* $\restriction V_{\mathcal{V}} = [\mathcal{V}']$ @ (*s2′* $\restriction V_{\mathcal{V}}$)
proof −
have *r2′* $\restriction V_{\mathcal{V}} =$[]
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2 r2′-Vv2-empty*
r2′-in-E2star subset-iff-psubset-eq)
with *t2′-is-r2′-v′-s2′ v′-in-Vv2 Vv-is-Vv1-union-Vv2* **show** *?thesis*
by (*simp only*: *t2′-is-r2′-v′-s2′*
projection-concatenation-commute projection-def, *auto*)
qed
moreover
have *t2* $\restriction V_{\mathcal{V}} =$ *t2′* $\restriction V_{\mathcal{V}}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute outerCons-prems*(*4*)
projection-intersection-neutral t2′-Vv2-is-t2-Vv2 t2′-in-E2star)

```
      ultimately show ?thesis
        by auto
    qed
  moreover
  note s1'Cv1-empty s2'-Cv2-empty Cons.hyps[of ?tau s1' s2']
  ultimately obtain t'
    where τ-r1-q'-v'-t'-in-Tr: ?tau @ t' ∈ Tr(ES1 ∥ ES2)
    and t'Vv-is-lambda': t' ↓ V_V = lambda'
    and t'Cv-empty: t' ↓ C_V = []
    by auto

  let ?t = r1 @ q' @ [V'] @ t'

  note τ-r1-q'-v'-t'-in-Tr
  moreover
  from r1-Vv-empty q'V-empty t'Vv-is-lambda' v'-in-Vv
  have ?t ↓ V_V = V' # lambda'
    by(simp only: projection-concatenation-commute projection-def, auto)
  moreover
  from VIsViewOnE r1-Cv1-empty t'Cv-empty q'C-empty v'-in-Vv
  have ?t ↓ C_V = []
    proof −
      from VIsViewOnE v'-in-Vv have [V'] ↓ C_V = []
        by (simp add: isViewOn-def V-valid-def VC-disjoint-def
          VN-disjoint-def NC-disjoint-def projection-def, auto)
      moreover
      from r1-in-E1star r1-Cv1-empty
      have r1 ↓ C_V = []
        using propSepViews projection-on-subset2
        unfolding properSeparationOfViews-def by auto
      moreover
      note t'Cv-empty q'C-empty
      ultimately show ?thesis
        by (simp only: projection-concatenation-commute, auto)
    qed
  ultimately have ?thesis
    by auto
}
  moreover
  {
    assume v'-in-Vv1-inter-Vv2-inter-Nabla2: V' ∈ V_V1 ∩ V_V2 ∩ ∇_Γ2
    hence v'-in-Vv1: V' ∈ V_V1 and v'-in-Vv2: V' ∈ V_V2
      and v'-in-Nabla2: V' ∈ ∇_Γ2
      by auto
    with v'-in-Vv  propSepViews
    have v'-in-E1: V' ∈ E_ES1 and v'-in-E2: V' ∈ E_ES2
      unfolding properSeparationOfViews-def by auto

    from Cons(3,5,9) v'-in-E2 have t2 ↓ V_V = V' # (lambda' ↓ E_ES2)
      by (simp add: projection-def)
    from projection-split-first[OF this] obtain r2 s2
      where t2-is-r2-v'-s2: t2 = r2 @ [V'] @ s2
```

and *r2-Vv-empty*: $r2 \restriction V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: $r2 \restriction V_{\mathcal{V}2} = []$
by *auto*

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *r2-Cv2-empty*: $r2 \restriction C_{\mathcal{V}2} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *s2-Cv2-empty*: $s2 \restriction C_{\mathcal{V}2} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*5*) *t2-is-r2-v'-s2* **have** *r2-in-E2star*: *set r2* $\subseteq E_{ES2}$
and *s2-in-E2star*: *set s2* $\subseteq E_{ES2}$
by *auto*

have *r2-in-Nv2star*: *set r2* $\subseteq N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** *set r2* $\cap$ $V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r2-Cv2-empty* **have** *set r2* $\cap$ $C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r2E1-in-Nv2-inter-C1-star*: *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have *set* $(r2 \restriction E_{ES1})$ = *set r2* $\cap$ E_{ES1}
by (*simp add*: *projection-def*, *auto*)
with *r2-in-Nv2star* **have** *set* $(r2 \restriction E_{ES1}) \subseteq (E_{ES1} \cap N_{\mathcal{V}2})$
by *auto*
moreover
from *validV1 disjoint-Nv2-Vv1 propSepViews*
have $E_{ES1} \cap N_{\mathcal{V}2} = N_{\mathcal{V}2} \cap C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
with *Cv1-inter-Nv2-subsetof-Upsilon1*

have *r2E1-in-Nv2-inter-C1-Upsilon1-star*: *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
by *auto*

note *outerCons-prems* = *Cons.prems*

have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1}) \Longrightarrow$
$\exists\ t1'.\ (\ set\ t1' \subseteq E_{ES1}$
$\wedge\ ((\tau\ @\ r2) \restriction E_{ES1})\ @\ t1' \in Tr_{ES1}$
$\wedge\ t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
$\wedge\ t1' \restriction C_{\mathcal{V}1} = [])$
proof (*induct r2* $\restriction E_{ES1}$ *arbitrary*: *r2 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*9*) *outerCons-prems*(*3*)
outerCons-prems(*5*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE1*: $xs = xs \restriction E_{ES1}$
proof −
from *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
hence *set* $xs \subseteq E_{ES1}$
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* $(xs \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have *set* $(r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by (*metis Int-commute snoc.prems*)
with *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
hence *set* $xs \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *auto*
with *xs-is-xsE1* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *t1''*
where *t1''-in-E1star*: *set t1''* $\subseteq E_{ES1}$
and τ*-xs-E1-t1''-in-Tr1*: $((\tau\ @\ xs) \restriction E_{ES1})\ @\ t1'' \in Tr_{ES1}$
and *t1''Vv1-is-t1Vv1*: $t1'' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
and *t1''Cv1-empty*: $t1'' \restriction C_{\mathcal{V}1} = []$
by *auto*

have *x-in-Cv1-inter-Nv2*: $x \in C_{\mathcal{V}1} \cap N_{\mathcal{V}2}$
proof −
from *snoc*(*2*−*3*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
by *simp*
thus *?thesis*

```
      by auto
    qed
  hence x-in-Cv1: x ∈ C_V1
    by auto
  moreover
  note τ-xs-E1-t1''-in-Tr1 t1''Cv1-empty
  moreover
  have Adm: (Adm V1 ϱ1 Tr_ES1 ((τ @ xs) ↾ E_ES1) x)
    proof −
      from τ-xs-E1-t1''-in-Tr1 validES1
      have τ-xsE1-in-Tr1: ((τ @ xs) ↾ E_ES1) ∈ Tr_ES1
        by (simp add: ES-valid-def traces-prefixclosed-def
          prefixclosed-def prefix-def)
      with x-in-Cv1-inter-Nv2 ES1-total-Cv1-inter-Nv2
      have τ-xsE1-x-in-Tr1: ((τ @ xs) ↾ E_ES1) @ [x] ∈ Tr_ES1
        by (simp only: total-def)
      moreover
      have ((τ @ xs) ↾ E_ES1) ↾ (ϱ1 V1) = ((τ @ xs) ↾ E_ES1) ↾ (ϱ1 V1) ..
      ultimately show ?thesis
        by (simp add: Adm-def, auto)
    qed
  moreover note BSIA1
  ultimately obtain t1'
    where res1: ((τ @ xs) ↾ E_ES1) @ [x] @ t1' ∈ Tr_ES1
    and res2: t1' ↾ V_V1 = t1'' ↾ V_V1
    and res3: t1' ↾ C_V1 = []
    by (simp only: BSIA-def, blast)

  have set t1' ⊆ E_ES1
    proof −
      from res1 validES1 have set (((τ @ xs) ↾ E_ES1) @ [x] @ t1') ⊆ E_ES1
        by (simp add: ES-valid-def traces-contain-events-def, auto)
      thus ?thesis
        by auto
    qed
  moreover
  have ((τ @ r2) ↾ E_ES1) @ t1' ∈ Tr_ES1
    proof −
      from res1 xs-is-xsE1 have ((τ ↾ E_ES1) @ (xs @ [x])) @ t1' ∈ Tr_ES1
        by (simp only: projection-concatenation-commute, auto)
      thus ?thesis
        by (simp only: snoc(2) projection-concatenation-commute)
    qed
  moreover
  from t1''Vv1-is-t1Vv1 res2 have t1' ↾ V_V1 = t1 ↾ V_V1
    by auto
  moreover
  note res3
  ultimately show ?case
    by auto
qed
from this[OF r2E1-in-Nv2-inter-C1-star] obtain t1'
```

where *t1′-in-E1star*: *set t1′* $\subseteq E_{ES1}$
and *τr2E1-t1′-in-Tr1*: $((\tau\ @\ r2) \restriction E_{ES1})\ @\ t1' \in Tr_{ES1}$
and *t1′-Vv1-is-t1-Vv1*: $t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
and *t1′-Cv1-empty*: $t1' \restriction C_{\mathcal{V}1} = []$
by *auto*

have $t1' \restriction V_{\mathcal{V}1} = \mathcal{V}' \# (lambda' \restriction E_{ES1})$
proof −
from *projection-intersection-neutral*[*OF Cons*(*4*), *of* $V_{\mathcal{V}}$] *propSepViews*
have $t1 \restriction V_{\mathcal{V}} = t1 \restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp only*: *Int-commute*)
with *Cons*(*8*) *t1′-Vv1-is-t1-Vv1 v′-in-E1* **show** *?thesis*
by (*simp add*: *projection-def*)
qed
from *projection-split-first*[*OF this*] **obtain** *r1′ s1′*
where *t1′-is-r1′-v′-s1′*: $t1' = r1'\ @\ [\mathcal{V}']\ @\ s1'$
and *r1′-Vv1-empty*: $r1' \restriction V_{\mathcal{V}1} = []$
by *auto*

from *t1′-is-r1′-v′-s1′ t1′-Cv1-empty* **have** *r1′-Cv1-empty*: $r1' \restriction C_{\mathcal{V}1} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t1′-is-r1′-v′-s1′ t1′-Cv1-empty* **have** *s1′-Cv1-empty*: $s1' \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *t1′-in-E1star t1′-is-r1′-v′-s1′* **have** *r1′-in-E1star*: *set r1′* $\subseteq E_{ES1}$
by *auto*

have *r1′-in-Nv1star*: *set r1′* $\subseteq N_{\mathcal{V}1}$
proof −
note *r1′-in-E1star*
moreover
from *r1′-Vv1-empty* **have** *set r1′* $\cap\ V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r1′-Cv1-empty* **have** *set r1′* $\cap\ C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r1′E2-in-Nv1-inter-C2-star*: *set* $(r1' \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
have *set* $(r1' \restriction E_{ES2})$ = *set r1′* $\cap\ E_{ES2}$

by (*simp add*: *projection-def*, *auto*)
with *r1′-in-Nv1star* **have** *set* $(r1' \restriction E_{ES2}) \subseteq (E_{ES2} \cap N_{\mathcal{V}1})$
by *auto*
moreover
from *validV2 propSepViews disjoint-Nv1-Vv2*
have $E_{ES2} \cap N_{\mathcal{V}1} = N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
with *Cv2-inter-Nv1-subsetof-Upsilon2*
have *r1′E2-in-Nv1-inter-Cv2-Upsilon2-star*:
set $(r1' \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$
by *auto*

have *set* $(r1' \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) \Longrightarrow$
$\exists$ *s2′ q2′*. (
set s2′ $\subseteq E_{ES2} \wedge$ *set q2′* $\subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge (\tau \restriction E_{ES2})$ @ *r2* @ *q2′* @ $[\mathcal{V}']$ @ *s2′* $\in Tr_{ES2}$
$\wedge$ *q2′* $\restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = r1' \restriction E_{ES2}$
$\wedge$ *s2′* $\restriction V_{\mathcal{V}2} =$ *s2* $\restriction V_{\mathcal{V}2}$
$\wedge$ *s2′* $\restriction C_{\mathcal{V}2} = []$)
proof (*induct r1′* $\restriction E_{ES2}$ *arbitrary*: *r1′ rule*: *rev-induct*)
case *Nil*

note *s2-in-E2star*
moreover
have *set* $[] \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
from *outerCons-prems*(6) *t2-is-r2-v′-s2*
have $\tau \restriction E_{ES2}$ @ *r2* @ [] @ $[\mathcal{V}']$ @ *s2* $\in Tr_{ES2}$
by *auto*
moreover
from *Nil* **have** $[] \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = r1' \restriction E_{ES2}$
by (*simp add*: *projection-def*)
moreover
have *s2* $\restriction V_{\mathcal{V}2} =$ *s2* $\restriction V_{\mathcal{V}2}$..
moreover
note *s2-Cv2-empty*
ultimately show *?case*
by *blast*

next
case (*snoc x xs*)

have *xs-is-xsE2*: *xs* = *xs* $\restriction E_{ES2}$
proof −
from *snoc*(2) **have** *set* (*xs* @ [*x*]) $\subseteq E_{ES2}$
by (*simp add*: *projection-def*, *auto*)

thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* (*xs* ↾ E_{ES2}) ⊆ $N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
proof −
from *snoc*(*2*−*3*) **have** *set* (*xs* @ [*x*]) ⊆ $N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by *simp*
with *xs-is-xsE2* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *s2''* *q2''*
where *s2''-in-E2star*: *set s2''* ⊆ E_{ES2}
and *q2''-in-C2-inter-Upsilon2-inter-Delta2*: *set q2''* ⊆ $C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *τE2-r2-q2''-v'-s2''-in-Tr2*: (*τ* ↾ E_{ES2} @ *r2* @ *q2''*) @ [$\mathcal{V}'$] @ *s2''* ∈ Tr_{ES2}
and *q2''C2-Upsilon2-is-xsE2*: *q2''* ↾ ($C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$) = *xs* ↾ E_{ES2}
and *s2''V2-is-s2V2*: *s2''* ↾ $V_{\mathcal{V}2}$ = *s2* ↾ $V_{\mathcal{V}2}$
and *s2''C2-empty*: *s2''* ↾ $C_{\mathcal{V}2}$ = []
by *auto*

have *x-in-Cv2-inter-Upsilon2*: *x* ∈ $C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
and *x-in-Cv2-inter-Nv1*: *x* ∈ $C_{\mathcal{V}2} \cap N_{\mathcal{V}1}$
proof −
from *snoc*(*2*−*3*) **have** *set* (*xs* @ [*x*]) ⊆ ($N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$)
by *simp*
thus *x* ∈ $C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
and *x* ∈ $C_{\mathcal{V}2} \cap N_{\mathcal{V}1}$
by *auto*
qed
with *validV2* **have** *x-in-E2*: *x* ∈ E_{ES2}
by (*simp add*:*isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

note *x-in-Cv2-inter-Upsilon2*
moreover
from *v'-in-Vv1-inter-Vv2-inter-Nabla2* **have** $\mathcal{V}'$ ∈ $V_{\mathcal{V}2} \cap \nabla_{\Gamma 2}$
by *auto*
moreover
note *τE2-r2-q2''-v'-s2''-in-Tr2 s2''C2-empty*
moreover
have *Adm*: (*Adm* $\mathcal{V}2$ *ϱ2* Tr_{ES2} (*τ* ↾ E_{ES2} @ *r2* @ *q2''*) *x*)
proof −
from *τE2-r2-q2''-v'-s2''-in-Tr2 validES2*
have (*τ* ↾ E_{ES2} @ *r2* @ *q2''*) ∈ Tr_{ES2}
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def)
with *x-in-Cv2-inter-Nv1 ES2-total-Cv2-inter-Nv1*
have (*τ* ↾ E_{ES2} @ *r2* @ *q2''*) @ [*x*] ∈ Tr_{ES2}
by (*simp only*: *total-def*)
moreover

```
    have (τ ↓ E_ES2 @ r2 @ q2'') ↓ (ϱ2 V2) = (τ ↓ E_ES2 @ r2 @ q2'') ↓ (ϱ2 V2) ..
    ultimately show ?thesis
      by (simp only: Adm-def, blast)
  qed
moreover
note FCIA2
ultimately
obtain s2' γ'
  where res1: (set γ') ⊆ (N_V2 ∩ Δ_Γ2)
  and res2: ((τ ↓ E_ES2 @ r2 @ q2'') @ [x] @ γ' @ [V'] @ s2') ∈ Tr_ES2
  and res3: (s2' ↓ V_V2) = (s2'' ↓ V_V2)
  and res4: s2' ↓ C_V2 = []
  unfolding FCIA-def
  by blast

let ?q2' = q2'' @ [x] @ γ'

from res2 validES2 have set s2' ⊆ E_ES2
  by (simp add: ES-valid-def traces-contain-events-def, auto)
moreover
from res1 x-in-Cv2-inter-Upsilon2 q2''-in-C2-inter-Upsilon2-inter-Delta2
have set ?q2' ⊆ C_V2 ∩ ϒ_Γ2 ∪ N_V2 ∩ Δ_Γ2
  by auto
moreover
from res2 have τ ↓ E_ES2 @ r2 @ ?q2' @ [V'] @ s2' ∈ Tr_ES2
  by auto
moreover
have ?q2' ↓ (C_V2 ∩ ϒ_Γ2) = r1' ↓ E_ES2
  proof −
    from validV2 res1 have γ' ↓ (C_V2 ∩ ϒ_Γ2) = []
      proof −
        from res1 have γ' = γ' ↓ (N_V2 ∩ Δ_Γ2)
          by (simp only: list-subset-iff-projection-neutral)
        hence γ' ↓ (C_V2 ∩ ϒ_Γ2) = γ' ↓ (N_V2 ∩ Δ_Γ2) ↓ (C_V2 ∩ ϒ_Γ2)
          by simp
        hence γ' ↓ (C_V2 ∩ ϒ_Γ2) = γ' ↓ (N_V2 ∩ Δ_Γ2 ∩ C_V2 ∩ ϒ_Γ2)
          by (simp only: projection-def, auto)
        moreover
        from validV2 have N_V2 ∩ Δ_Γ2 ∩ C_V2 ∩ ϒ_Γ2 = {}
          by (simp add:isViewOn-def V-valid-def
            VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
        ultimately show ?thesis
          by (simp add: projection-def)
      qed
    hence ?q2' ↓ (C_V2 ∩ ϒ_Γ2) = (q2'' @ [x]) ↓ (C_V2 ∩ ϒ_Γ2)
      by (simp only: projection-concatenation-commute, auto)
    with q2''C2-Upsilon2-is-xsE2 x-in-Cv2-inter-Upsilon2
    have ?q2' ↓ (C_V2 ∩ ϒ_Γ2) = (xs ↓ E_ES2) @ [x]
      by (simp only: projection-concatenation-commute projection-def, auto)
    with xs-is-xsE2 snoc(2) show ?thesis
      by simp
  qed
```

moreover
from *res3 s2''V2-is-s2V2* **have** $s2' \restriction V_{\mathcal{V}2} = s2 \restriction V_{\mathcal{V}2}$
by *simp*
moreover
note *res4*
ultimately show *?case*
by *blast*
qed
from *this*[*OF r1'E2-in-Nv1-inter-Cv2-Upsilon2-star*] **obtain** *s2' q2'*
where *s2'-in-E2star*: *set* $s2' \subseteq E_{ES2}$
and *q2'-in-Cv2-inter-Upsilon2-union-Nv2-inter-Delta2*:
set $q2' \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and τ*E2-r2-q2'-v'-s2'-in-Tr2*: $(\tau \restriction E_{ES2})$ @ *r2* @ *q2'* @ $[\mathcal{V}']$ @ $s2' \in Tr_{ES2}$
and *q2'Cv2-inter-Upsilon2-is-r1'E2*: $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = r1' \restriction E_{ES2}$
and *s2'Vv2-is-s2-Vv2*: $s2' \restriction V_{\mathcal{V}2} = s2 \restriction V_{\mathcal{V}2}$
and *s2'Cv2-empty*: $s2' \restriction C_{\mathcal{V}2} = []$
by *auto*

from *q2'-in-Cv2-inter-Upsilon2-union-Nv2-inter-Delta2 validV2*
have *q2'-in-E2star*: *set* $q2' \subseteq E_{ES2}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)

have *r1'Cv-empty*: $r1' \restriction C_{\mathcal{V}} = []$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2*
r1'-Cv1-empty r1'-in-E1star)

from *validES2* τ*E2-r2-q2'-v'-s2'-in-Tr2*
have *q2'-in-E2star*: *set* $q2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
note *r1'-in-E1star*
moreover
have *q2'E1-is-r1'E2*: $q2' \restriction E_{ES1} = r1' \restriction E_{ES2}$
proof −
from *q2'-in-Cv2-inter-Upsilon2-union-Nv2-inter-Delta2*
have $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) = q2'$
by (*simp add*: *list-subset-iff-projection-neutral*)
hence $(q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})) \restriction E_{ES1} = q2' \restriction E_{ES1}$
by *simp*
hence $q2' \restriction ((C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \cap E_{ES1}) = q2' \restriction E_{ES1}$
by (*simp add*: *projection-def*)
hence $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap E_{ES1}) = q2' \restriction E_{ES1}$
by (*simp only*: *Int-Un-distrib2 disjoint-Nv2-inter-Delta2-inter-E1*, *auto*)
moreover
from *q2'Cv2-inter-Upsilon2-is-r1'E2*
have $(q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})) \restriction E_{ES1} = (r1' \restriction E_{ES2}) \restriction E_{ES1}$
by *simp*
hence $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap E_{ES1}) = (r1' \restriction E_{ES1}) \restriction E_{ES2}$
by (*simp add*: *projection-def conj-commute*)

with *r1′-in-E1star* **have** $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap E_{ES1}) = r1' \restriction E_{ES2}$
by (*simp only*: *list-subset-iff-projection-neutral*)
ultimately show *?thesis*
by *auto*
qed
moreover
have $q2' \restriction V_{\mathcal{V}} = []$
proof −
from *q2′-in-Cv2-inter-Upsilon2-union-Nv2-inter-Delta2*
have $q2' = q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by (*simp add*: *list-subset-iff-projection-neutral*)
moreover
from *q2′-in-E2star* **have** $q2' = q2' \restriction E_{ES2}$
by (*simp add*: *list-subset-iff-projection-neutral*)
ultimately have $q2' = q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES2}$
by *simp*
hence $q2' \restriction V_{\mathcal{V}} = q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES2} \restriction V_{\mathcal{V}}$
by *simp*
hence $q2' \restriction V_{\mathcal{V}} = q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction (V_{\mathcal{V}} \cap E_{ES2})$
by (*simp add*: *Int-commute projection-def*)
with *propSepViews*
have $q2' \restriction V_{\mathcal{V}} = q2' \restriction ((C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \cap V_{\mathcal{V}2})$
unfolding *properSeparationOfViews-def*
by (*simp add*: *projection-def*)
hence $q2' \restriction V_{\mathcal{V}} = q2' \restriction (V_{\mathcal{V}2} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup V_{\mathcal{V}2} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by (*simp add*: *Int-Un-distrib2*, *metis Int-assoc*
Int-commute Int-left-commute Un-commute)
with *validV2* **show** *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*, *simp add*: *projection-def*)
qed
moreover
have $r1' \restriction V_{\mathcal{V}} = []$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*
r1′-Vv1-empty r1′-in-E1star)
moreover
have *q2′Cv-empty*: $q2' \restriction C_{\mathcal{V}} = []$
proof −
from *q2′-in-E2star* **have** *foo*: $q2' = q2' \restriction E_{ES2}$
by (*simp add*: *list-subset-iff-projection-neutral*)
hence $q2' \restriction C_{\mathcal{V}} = q2' \restriction (C_{\mathcal{V}} \cap E_{ES2})$
by (*metis Int-commute list-subset-iff-projection-neutral*
projection-intersection-neutral)
moreover
from *propSepViews* **have** $C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *projection-subset-elim*[*OF* ‹$C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2}$›, *of q2′*]
have $q2' \restriction C_{\mathcal{V}2} \restriction C_{\mathcal{V}} \restriction E_{ES2} = q2' \restriction (C_{\mathcal{V}} \cap E_{ES2})$
by (*simp add*: *projection-def*)
hence $q2' \restriction E_{ES2} \restriction C_{\mathcal{V}2} \restriction C_{\mathcal{V}} = q2' \restriction (C_{\mathcal{V}} \cap E_{ES2})$
by (*simp add*: *projection-commute*)

with *foo* **have** $q2' \restriction (C_{\mathcal{V}2} \cap C_{\mathcal{V}}) = q2' \restriction (C_{\mathcal{V}} \cap E_{ES2})$
by (*simp add*: *projection-def*)
moreover
from *q2'-in-Cv2-inter-Upsilon2-union-Nv2-inter-Delta2*
have $q2' \restriction (C_{\mathcal{V}2} \cap C_{\mathcal{V}}) = q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction (C_{\mathcal{V}2} \cap C_{\mathcal{V}})$
by (*simp add*: *list-subset-iff-projection-neutral*)
moreover
have $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \cap (C_{\mathcal{V}2} \cap C_{\mathcal{V}})$
$= (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup C_{\mathcal{V}2} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \cap C_{\mathcal{V}}$
by *fast*
hence $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction (C_{\mathcal{V}2} \cap C_{\mathcal{V}})$
$= q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup C_{\mathcal{V}2} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction C_{\mathcal{V}}$
by (*simp add*: *projection-sequence*)
moreover
from *validV2*
have $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cup C_{\mathcal{V}2} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction C_{\mathcal{V}}$
$= q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) \restriction C_{\mathcal{V}}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def Int-commute)
moreover
from *q2'Cv2-inter-Upsilon2-is-r1'E2*
have $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) \restriction C_{\mathcal{V}} = r1' \restriction E_{ES2} \restriction C_{\mathcal{V}}$
by *simp*
with *projection-on-intersection*[*OF r1'Cv-empty*] **have** $q2' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) \restriction C_{\mathcal{V}} = []$
by (*simp add*: *Int-commute projection-def*)
ultimately show *?thesis*
by *auto*
qed
moreover
note *r1'Cv-empty merge-property'*[*of r1' q2'*]
ultimately obtain q'
where *q'E2-is-q2'*: $q' \restriction E_{ES2} = q2'$
and *q'E1-is-r1'*: $q' \restriction E_{ES1} = r1'$
and *q'V-empty*: $q' \restriction V_{\mathcal{V}} = []$
and *q'C-empty*: $q' \restriction C_{\mathcal{V}} = []$
and *q'-in-E1-union-E2-star*: *set* $q' \subseteq (E_{ES1} \cup E_{ES2})$
unfolding *Let-def*
by *auto*

let *?tau* $= \tau$ @ *r2* @ q' @ $[\mathcal{V}']$

from *Cons*(*2*) *r2-in-E2star q'-in-E1-union-E2-star v'-in-E2*
have *set ?tau* $\subseteq (E_{(ES1 \parallel ES2)})$
by (*simp add*: *composeES-def*, *auto*)
moreover
from *Cons*(*3*) **have** *set lambda'* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *t1'-in-E1star t1'-is-r1'-v'-s1'* **have** *set s1'* $\subseteq E_{ES1}$
by *simp*
moreover
note *s2'-in-E2star*

moreover
from *τr2E1-t1′-in-Tr1 t1′-is-r1′-v′-s1′ v′-in-E1 q′E1-is-r1′*
have *?tau ↑ E_ES1 @ s1′ ∈ Tr_ES1*
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from *q′E2-is-q2′ r2-in-E2star v′-in-E2 q2′-in-E2star τE2-r2-q2′-v′-s2′-in-Tr2*
have *?tau ↑ E_ES2 @ s2′ ∈ Tr_ES2*
by (*simp only*: *list-subset-iff-projection-neutral*
projection-concatenation-commute projection-def, *auto*)
moreover
have *lambda′ ↑ E_ES1 = s1′ ↑ V_𝒱*
proof −
from *Cons*(*2*,*4*,*8*) *v′-in-E1* **have** *t1 ↑ V_𝒱 = [𝒱′] @ (lambda′ ↑ E_ES1)*
by (*simp add*: *projection-def*)
moreover
from *t1′-is-r1′-v′-s1′ r1′-Vv1-empty r1′-in-E1star*
v′-in-Vv1 propSepViews
have *t1′ ↑ V_𝒱 = [𝒱′] @ (s1′ ↑ V_𝒱)*
proof −
have *r1′ ↑ V_𝒱 =[]*
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis projection-on-subset2 r1′-Vv1-empty*
r1′-in-E1star subset-iff-psubset-eq)
with *t1′-is-r1′-v′-s1′ v′-in-Vv1 Vv-is-Vv1-union-Vv2* **show** *?thesis*
by (*simp only*: *t1′-is-r1′-v′-s1′ projection-concatenation-commute*
projection-def, *auto*)
qed
moreover
have *t1 ↑ V_𝒱 = t1′ ↑ V_𝒱*
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute outerCons-prems*(*3*)
projection-intersection-neutral t1′-Vv1-is-t1-Vv1 t1′-in-E1star)
ultimately show *?thesis*
by *auto*
qed
moreover
have *lambda′ ↑ E_ES2 = s2′ ↑ V_𝒱*
proof −
from *Cons*(*3*,*5*,*9*) *v′-in-E2* **have** *t2 ↑ V_𝒱 = [𝒱′] @ (lambda′ ↑ E_ES2)*
by (*simp add*: *projection-def*)
moreover
from *t2-is-r2-v′-s2 r2-Vv-empty v′-in-Vv2 Vv-is-Vv1-union-Vv2*
have *t2 ↑ V_𝒱 = [𝒱′] @ (s2 ↑ V_𝒱)*
by (*simp only*: *t2-is-r2-v′-s2 projection-concatenation-commute*
projection-def, *auto*)
moreover
have *s2 ↑ V_𝒱 = s2′ ↑ V_𝒱*
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*
s2′Vv2-is-s2-Vv2 s2′-in-E2star s2-in-E2star)
ultimately show *?thesis*
by *auto*

```
  qed
moreover
note s1'-Cv1-empty s2'Cv2-empty Cons.hyps[of ?tau s1' s2']
ultimately obtain t'
  where τ-r2-q'-v'-t'-in-Tr: ?tau @ t' ∈ Tr(ES1 ∥ ES2)
  and t'Vv-is-lambda': t' ↿ V_V = lambda'
  and t'Cv-empty: t' ↿ C_V = []
  by auto

let ?t = r2 @ q' @ [V'] @ t'

note τ-r2-q'-v'-t'-in-Tr
moreover
from r2-Vv-empty q'V-empty t'Vv-is-lambda' v'-in-Vv
have ?t ↿ V_V = V' # lambda'
  by(simp only: projection-concatenation-commute projection-def, auto)
moreover
from VIsViewOnE r2-Cv2-empty t'Cv-empty q'C-empty v'-in-Vv
have ?t ↿ C_V = []
  proof −
    from VIsViewOnE v'-in-Vv have [V'] ↿ C_V = []
      by (simp add: isViewOn-def V-valid-def
        VC-disjoint-def VN-disjoint-def NC-disjoint-def projection-def, auto)
    moreover
    from r2-in-E2star r2-Cv2-empty
    have r2 ↿ C_V = []
      using propSepViews projection-on-subset2 unfolding properSeparationOfViews-def
      by auto
    moreover
    note t'Cv-empty q'C-empty
    ultimately show ?thesis
      by (simp only: projection-concatenation-commute, auto)
  qed
ultimately have ?thesis
  by auto
}
moreover
{
  assume v'-in-Vv1-minus-E2: V' ∈ V_V1 − E_ES2
  hence v'-in-Vv1: V' ∈ V_V1
    by auto
  with v'-in-Vv have v'-in-E1: V' ∈ E_ES1
    using propSepViews unfolding properSeparationOfViews-def
    by auto

  from v'-in-Vv1-minus-E2 have v'-notin-E2: V' ∉ E_ES2
    by auto
  with validV2 have v'-notin-Vv2: V' ∉ V_V2
    by (simp add: isViewOn-def V-valid-def
      VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)

  from Cons(3−4) Cons(8) v'-in-E1 have t1 ↿ V_V = V' # (lambda' ↿ E_ES1)
```

by (*simp add*: *projection-def*)
from *projection-split-first*[*OF this*] **obtain** *r1 s1*
where *t1-is-r1-v'-s1*: $t1 = r1 \mathbin{@} [\mathcal{V}'] \mathbin{@} s1$
and *r1-Vv-empty*: $r1 \restriction V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}1}$ $V_{\mathcal{V}}$ *r1*]
have *r1-Vv1-empty*: $r1 \restriction V_{\mathcal{V}1} = []$
by *auto*

from *t1-is-r1-v'-s1 Cons*(*10*) **have** *r1-Cv1-empty*: $r1 \restriction C_{\mathcal{V}1} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t1-is-r1-v'-s1 Cons*(*10*) **have** *s1-Cv1-empty*: $s1 \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*4*) *t1-is-r1-v'-s1* **have** *r1-in-E1star*: $set\ r1 \subseteq E_{ES1}$
by *auto*

have *r1-in-Nv1star*: $set\ r1 \subseteq N_{\mathcal{V}1}$
proof −
note *r1-in-E1star*
moreover
from *r1-Vv1-empty* **have** $set\ r1 \cap V_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r1-Cv1-empty* **have** $set\ r1 \cap C_{\mathcal{V}1} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV1*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r1E2-in-Nv1-inter-C2-star*: $set\ (r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
have $set\ (r1 \restriction E_{ES2}) = set\ r1 \cap E_{ES2}$
by (*simp add*: *projection-def*, *auto*)
with *r1-in-Nv1star* **have** $set\ (r1 \restriction E_{ES2}) \subseteq (E_{ES2} \cap N_{\mathcal{V}1})$
by *auto*
moreover
from *validV2 disjoint-Nv1-Vv2*
have $E_{ES2} \cap N_{\mathcal{V}1} = N_{\mathcal{V}1} \cap C_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*

qed
with *Cv2-inter-Nv1-subsetof-Upsilon2*
have *r1E2-in-Nv1-inter-C2-Upsilon2-star*: *set* $(r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$
by *auto*

note *outerCons-prems* = *Cons.prems*

have *set* $(r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2}) \Longrightarrow$
$\exists\ t2'.$ (*set* $t2' \subseteq E_{ES2}$
$\wedge\ ((\tau\ @\ r1) \restriction E_{ES2})\ @\ t2' \in Tr_{ES2}$
$\wedge\ t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
$\wedge\ t2' \restriction C_{\mathcal{V}2} = []$)
proof (*induct r1* $\restriction E_{ES2}$ *arbitrary*: *r1 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*10*) *outerCons-prems*(*4*)
outerCons-prems(*6*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE2*: *xs* = *xs* $\restriction E_{ES2}$
proof −
from *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq E_{ES2}$
by (*simp add*: *projection-def*, *auto*)
hence *set xs* $\subseteq (E_{ES2})$
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* $(xs \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
proof −
have *set* $(r1 \restriction E_{ES2}) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
by (*metis Int-commute snoc.prems*)
with *snoc*(*2*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
by *simp*
hence *set xs* $\subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$
by *auto*
with *xs-is-xsE2* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *t2''*
where *t2''-in-E2star*: *set* $t2'' \subseteq E_{ES2}$
and *τ-xs-E2-t2''-in-Tr2*: $((\tau\ @\ xs) \restriction E_{ES2})\ @\ t2'' \in Tr_{ES2}$
and *t2''Vv2-is-t2Vv2*: $t2'' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
and *t2''Cv2-empty*: $t2'' \restriction C_{\mathcal{V}2} = []$
by *auto*

have *x-in-Cv2-inter-Nv1*: $x \in C_{\mathcal{V}2} \cap N_{\mathcal{V}1}$
proof −
from *snoc*(*2*−*3*) **have** *set* $(xs\ @\ [x]) \subseteq (N_{\mathcal{V}1} \cap C_{\mathcal{V}2})$

by *simp*
thus *?thesis*
by *auto*
qed
hence *x-in-Cv2*: $x \in C_{\mathcal{V}2}$
by *auto*
moreover
note *τ-xs-E2-t2''-in-Tr2 t2''Cv2-empty*
moreover
have *Adm*: $(Adm\ \mathcal{V}2\ \varrho 2\ Tr_{ES2}\ ((\tau\ @\ xs) \restriction E_{ES2})\ x)$
proof −
from *τ-xs-E2-t2''-in-Tr2 validES2*
have *τ-xsE2-in-Tr2*: $((\tau\ @\ xs) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
with *x-in-Cv2-inter-Nv1 ES2-total-Cv2-inter-Nv1*
have *τ-xsE2-x-in-Tr2*: $((\tau\ @\ xs) \restriction E_{ES2})\ @\ [x] \in Tr_{ES2}$
by (*simp only*: *total-def*)
moreover
have $((\tau\ @\ xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2) = ((\tau\ @\ xs) \restriction E_{ES2}) \restriction (\varrho 2\ \mathcal{V}2)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA2*
ultimately obtain *t2'*
where *res1*: $((\tau\ @\ xs) \restriction E_{ES2})\ @\ [x]\ @\ t2' \in Tr_{ES2}$
and *res2*: $t2' \restriction V_{\mathcal{V}2} = t2'' \restriction V_{\mathcal{V}2}$
and *res3*: $t2' \restriction C_{\mathcal{V}2} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t2'* $\subseteq E_{ES2}$
proof −
from *res1 validES2* **have** *set* $(((\tau\ @\ xs) \restriction E_{ES2})\ @\ [x]\ @\ t2') \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau\ @\ r1) \restriction E_{ES2})\ @\ t2' \in Tr_{ES2}$
proof −
from *res1 xs-is-xsE2* **have** $((\tau \restriction E_{ES2})\ @\ (xs\ @\ [x]))\ @\ t2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(2) *projection-concatenation-commute*)
qed
moreover
from *t2''Vv2-is-t2Vv2 res2* **have** $t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
by *auto*
moreover
note *res3*
ultimately show *?case*
by *auto*

qed
from *this*[*OF r1E2-in-Nv1-inter-C2-star*] **obtain** $t2'$
where *t2′-in-E2star*: *set* $t2' \subseteq E_{ES2}$
and τ*r1E2-t2′-in-Tr2*: $((\tau\ @\ r1) \restriction E_{ES2})\ @\ t2' \in Tr_{ES2}$
and *t2′-Vv2-is-t2-Vv2*: $t2' \restriction V_{\mathcal{V}2} = t2 \restriction V_{\mathcal{V}2}$
and *t2′-Cv2-empty*: $t2' \restriction C_{\mathcal{V}2} = []$
by *auto*

let *?tau* = $\tau\ @\ r1\ @\ [\mathcal{V}']$

from *v′-in-E1 Cons*(2) *r1-in-Nv1star validV1* **have** *set ?tau* $\subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *isViewOn-def composeES-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
moreover
from *Cons*(3) **have** *set lambda′* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *Cons*(4) *t1-is-r1-v′-s1* **have** *set s1* $\subseteq E_{ES1}$
by *auto*
moreover
note *t2′-in-E2star*
moreover
have *?tau* $\restriction E_{ES1}\ @\ s1 \in Tr_{ES1}$
by (*metis Cons-eq-appendI append-eq-appendI calculation*(3) *eq-Nil-appendI*
list-subset-iff-projection-neutral Cons.prems(3) *Cons.prems*(5)
projection-concatenation-commute t1-is-r1-v′-s1)
moreover
from τ*r1E2-t2′-in-Tr2 v′-notin-E2* **have** *?tau* $\restriction E_{ES2}\ @\ t2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(8) *t1-is-r1-v′-s1 r1-Vv-empty v′-in-E1 v′-in-Vv* **have** *lambda′* $\restriction E_{ES1} = s1 \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(9) *v′-notin-E2 t2′-Vv2-is-t2-Vv2* **have** *lambda′* $\restriction E_{ES2} = t2' \restriction V_{\mathcal{V}}$
proof −
have $t2' \restriction V_{\mathcal{V}} = t2' \restriction V_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def projection-intersection-neutral t2′-in-E2star)
moreover
have $t2 \restriction V_{\mathcal{V}} = t2 \restriction V_{\mathcal{V}2}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def projection-intersection-neutral Cons(5))
moreover
note *Cons*(9) *v′-notin-E2 t2′-Vv2-is-t2-Vv2*
ultimately show *?thesis*
by (*simp add*: *projection-def*)
qed
moreover
note *s1-Cv1-empty t2′-Cv2-empty*
moreover

note *Cons.hyps*(*1*)[*of ?tau s1 t2'*]
ultimately obtain t'
  where $\tau r1v't'$*-in-Tr*: *?tau* @ $t' \in Tr_{(ES1 \parallel ES2)}$
  and t'*-Vv-is-lambda'*: $t' \restriction V_{\mathcal{V}} = lambda'$
  and t'*-Cv-empty*: $t' \restriction C_{\mathcal{V}} = []$
  by *auto*

let *?t* = *r1* @ [$\mathcal{V}'$] @ t'

note $\tau r1v't'$*-in-Tr*
moreover
from *r1-Vv-empty* t'*-Vv-is-lambda'* v'*-in-Vv* **have** *?t* $\restriction V_{\mathcal{V}} = \mathcal{V}' \# lambda'$
  by (*simp add*: *projection-def*)
moreover
have *?t* $\restriction C_{\mathcal{V}} = []$
  proof −
    have *r1* $\restriction C_{\mathcal{V}} = []$
    proof −
      from *propSepViews* **have** $E_{ES1} \cap C_{\mathcal{V}} \subseteq C_{\mathcal{V}1}$
        unfolding *properSeparationOfViews-def* **by** *auto*
        from *projection-on-subset*[*OF* ‹$E_{ES1} \cap C_{\mathcal{V}} \subseteq C_{\mathcal{V}1}$› *r1-Cv1-empty*]
        have *r1* $\restriction (E_{ES1} \cap C_{\mathcal{V}}) = []$
          by (*simp only*: *Int-commute*)
        with *projection-intersection-neutral*[*OF r1-in-E1star*, *of* $C_{\mathcal{V}}$] **show** *?thesis*
          by *simp*
      qed
    with v'*-in-Vv VIsViewOnE* t'*-Cv-empty* **show** *?thesis*
      by (*simp add*: *isViewOn-def V-valid-def*
        VC-disjoint-def VN-disjoint-def NC-disjoint-def projection-def, *auto*)
  qed
ultimately have *?thesis*
  by *auto*
}
moreover
{
  assume v'*-in-Vv2-minus-E1*: $\mathcal{V}' \in V_{\mathcal{V}2} - E_{ES1}$
  hence v'*-in-Vv2*: $\mathcal{V}' \in V_{\mathcal{V}2}$
    by *auto*
  with v'*-in-Vv propSepViews* **have** v'*-in-E2*: $\mathcal{V}' \in E_{ES2}$
    unfolding *properSeparationOfViews-def*
    by *auto*

  from v'*-in-Vv2-minus-E1* **have** v'*-notin-E1*: $\mathcal{V}' \notin E_{ES1}$
    by *auto*
  with *validV1* **have** v'*-notin-Vv1*: $\mathcal{V}' \notin V_{\mathcal{V}1}$
    by (*simp add*: *isViewOn-def V-valid-def*
      VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

  from *Cons*(*3*) *Cons*(*5*) *Cons*(*9*) v'*-in-E2* **have** *t2* $\restriction V_{\mathcal{V}} = \mathcal{V}' \# (lambda' \restriction E_{ES2})$
    by (*simp add*: *projection-def*)
  from *projection-split-first*[*OF this*] **obtain** *r2 s2*
    where *t2-is-r2-v'-s2*: *t2* = *r2* @ [$\mathcal{V}'$] @ *s2*

and *r2-Vv-empty*: $r2 \restriction V_{\mathcal{V}} = []$
by *auto*
with *Vv-is-Vv1-union-Vv2 projection-on-subset*[*of* $V_{\mathcal{V}2}$ $V_{\mathcal{V}}$ *r2*]
have *r2-Vv2-empty*: $r2 \restriction V_{\mathcal{V}2} = []$
by *auto*

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *r2-Cv2-empty*: $r2 \restriction C_{\mathcal{V}2} = []$
by (*simp add*: *projection-concatenation-commute*)

from *t2-is-r2-v'-s2 Cons*(*11*) **have** *s2-Cv2-empty*: $s2 \restriction C_{\mathcal{V}2} = []$
by (*simp only*: *projection-concatenation-commute*, *auto*)

from *Cons*(*5*) *t2-is-r2-v'-s2* **have** *r2-in-E2star*: $set\ r2 \subseteq E_{ES2}$
by *auto*

have *r2-in-Nv2star*: $set\ r2 \subseteq N_{\mathcal{V}2}$
proof −
note *r2-in-E2star*
moreover
from *r2-Vv2-empty* **have** $set\ r2 \cap V_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
from *r2-Cv2-empty* **have** $set\ r2 \cap C_{\mathcal{V}2} = \{\}$
by (*metis Compl-Diff-eq Diff-cancel Un-upper2*
disjoint-eq-subset-Compl list-subset-iff-projection-neutral
projection-on-union)
moreover
note *validV2*
ultimately show *?thesis*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
qed

have *r2E1-in-Nv2-inter-C1-star*: $set\ (r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1})$
proof −
have $set\ (r2 \restriction E_{ES1}) = set\ r2 \cap E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
with *r2-in-Nv2star* **have** $set\ (r2 \restriction E_{ES1}) \subseteq (E_{ES1} \cap N_{\mathcal{V}2})$
by *auto*
moreover
from *validV1 propSepViews disjoint-Nv2-Vv1*
have $E_{ES1} \cap N_{\mathcal{V}2} = N_{\mathcal{V}2} \cap C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by *auto*
qed
with *Cv1-inter-Nv2-subsetof-Upsilon1*
have *r2E1-in-Nv2-inter-C1-Upsilon1-star*: $set\ (r2 \restriction E_{ES1}) \subseteq (N_{\mathcal{V}2} \cap C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$

by *auto*

note *outerCons-prems* = *Cons.prems*

have *set* (*r2* ↾ E_{ES1}) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$) ⟹
∃ *t1'*. (*set t1'* ⊆ E_{ES1}
∧ ((τ @ *r2*) ↾ E_{ES1}) @ *t1'* ∈ Tr_{ES1}
∧ *t1'* ↾ $V_{\mathcal{V}1}$ = *t1* ↾ $V_{\mathcal{V}1}$
∧ *t1'* ↾ $C_{\mathcal{V}1}$ = [])
proof (*induct r2* ↾ E_{ES1} *arbitrary*: *r2 rule*: *rev-induct*)
case *Nil* **thus** *?case*
by (*metis append-self-conv outerCons-prems*(*9*) *outerCons-prems*(*3*)
outerCons-prems(*5*) *projection-concatenation-commute*)
next
case (*snoc x xs*)

have *xs-is-xsE1*: *xs* = *xs* ↾ E_{ES1}
proof −
from *snoc*(*2*) **have** *set* (*xs* @ [*x*]) ⊆ E_{ES1}
by (*simp add*: *projection-def*, *auto*)
hence *set xs* ⊆ E_{ES1}
by *auto*
thus *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have *set* (*xs* ↾ E_{ES1}) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
proof −
have *set* (*r2* ↾ E_{ES1}) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
by (*metis Int-commute snoc.prems*)
with *snoc*(*2*) **have** *set* (*xs* @ [*x*]) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
by *simp*
hence *set xs* ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
by *auto*
with *xs-is-xsE1* **show** *?thesis*
by *auto*
qed
moreover
note *snoc.hyps*(*1*)[*of xs*]
ultimately obtain *t1''*
where *t1''-in-E1star*: *set t1''* ⊆ E_{ES1}
and *τ-xs-E1-t1''-in-Tr1*: ((τ @ *xs*) ↾ E_{ES1}) @ *t1''* ∈ Tr_{ES1}
and *t1''Vv1-is-t1Vv1*: *t1''* ↾ $V_{\mathcal{V}1}$ = *t1* ↾ $V_{\mathcal{V}1}$
and *t1''Cv1-empty*: *t1''* ↾ $C_{\mathcal{V}1}$ = []
by *auto*

have *x-in-Cv1-inter-Nv2*: *x* ∈ $C_{\mathcal{V}1}$ ∩ $N_{\mathcal{V}2}$
proof −
from *snoc*(*2*−*3*) **have** *set* (*xs* @ [*x*]) ⊆ ($N_{\mathcal{V}2}$ ∩ $C_{\mathcal{V}1}$)
by *simp*
thus *?thesis*
by *auto*

qed
hence *x-in-Cv1*: $x \in C_{\mathcal{V}1}$
by *auto*
moreover
note *τ-xs-E1-t1''-in-Tr1 t1''Cv1-empty*
moreover
have *Adm*: (*Adm* $\mathcal{V}1$ $\varrho 1$ Tr_{ES1} $((\tau \mathbin{@} xs) \restriction E_{ES1})$ x)
proof −
from *τ-xs-E1-t1''-in-Tr1 validES1*
have *τ-xsE1-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
with *x-in-Cv1-inter-Nv2 ES1-total-Cv1-inter-Nv2*
have *τ-xsE1-x-in-Tr1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \in Tr_{ES1}$
by (*simp only*: *total-def*)
moreover
have $((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1) = ((\tau \mathbin{@} xs) \restriction E_{ES1}) \restriction (\varrho 1\ \mathcal{V}1)$ **..**
ultimately show *?thesis*
by (*simp add*: *Adm-def*, *auto*)
qed
moreover note *BSIA1*
ultimately obtain *t1'*
where *res1*: $((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1' \in Tr_{ES1}$
and *res2*: $t1' \restriction V_{\mathcal{V}1} = t1'' \restriction V_{\mathcal{V}1}$
and *res3*: $t1' \restriction C_{\mathcal{V}1} = []$
by (*simp only*: *BSIA-def*, *blast*)

have *set t1'* $\subseteq E_{ES1}$
proof −
from *res1 validES1* **have** *set* $(((\tau \mathbin{@} xs) \restriction E_{ES1}) \mathbin{@} [x] \mathbin{@} t1') \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
thus *?thesis*
by *auto*
qed
moreover
have $((\tau \mathbin{@} r2) \restriction E_{ES1}) \mathbin{@} t1' \in Tr_{ES1}$
proof −
from *res1 xs-is-xsE1* **have** $((\tau \restriction E_{ES1}) \mathbin{@} (xs \mathbin{@} [x])) \mathbin{@} t1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by (*simp only*: *snoc*(*2*) *projection-concatenation-commute*)
qed
moreover
from *t1''Vv1-is-t1Vv1 res2* **have** $t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
by *auto*
moreover
note *res3*
ultimately show *?case*
by *auto*
qed
from *this*[*OF r2E1-in-Nv2-inter-C1-star*] **obtain** *t1'*
where *t1'-in-E1star*: *set t1'* $\subseteq E_{ES1}$

and *τr2E1-t1′-in-Tr1*: $((\tau \mathbin{@} r2) \restriction E_{ES1}) \mathbin{@} t1' \in Tr_{ES1}$
and *t1′-Vv1-is-t1-Vv1*: $t1' \restriction V_{\mathcal{V}1} = t1 \restriction V_{\mathcal{V}1}$
and *t1′-Cv1-empty*: $t1' \restriction C_{\mathcal{V}1} = []$
by *auto*

let *?tau* $= \tau \mathbin{@} r2 \mathbin{@} [\mathcal{V}']$

from *v′-in-E2 Cons*(2) *r2-in-Nv2star validV2* **have** *set ?tau* $\subseteq E_{(ES1 \parallel ES2)}$
by (*simp only*: *composeES-def isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
moreover
from *Cons*(3) **have** *set lambda′* $\subseteq V_{\mathcal{V}}$
by *auto*
moreover
from *Cons*(5) *t2-is-r2-v′-s2* **have** *set s2* $\subseteq E_{ES2}$
by *auto*
moreover
note *t1′-in-E1star*
moreover
have *?tau* $\restriction E_{ES2} \mathbin{@} s2 \in Tr_{ES2}$
by (*metis Cons-eq-appendI append-eq-appendI calculation*(3) *eq-Nil-appendI*
list-subset-iff-projection-neutral Cons.prems(4) *Cons.prems*(6)
projection-concatenation-commute t2-is-r2-v′-s2)
moreover
from *τr2E1-t1′-in-Tr1 v′-notin-E1* **have** *?tau* $\restriction E_{ES1} \mathbin{@} t1' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(9) *t2-is-r2-v′-s2 r2-Vv-empty v′-in-E2 v′-in-Vv*
have *lambda′* $\restriction E_{ES2} = s2 \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
moreover
from *Cons*(10) *v′-notin-E1 t1′-Vv1-is-t1-Vv1*
have *lambda′* $\restriction E_{ES1} = t1' \restriction V_{\mathcal{V}}$
proof −
have $t1' \restriction V_{\mathcal{V}} = t1' \restriction V_{\mathcal{V}1}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def projection-intersection-neutral t1′-in-E1star)
moreover
have $t1 \restriction V_{\mathcal{V}} = t1 \restriction V_{\mathcal{V}1}$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *projection-def*, *metis Int-commute*
projection-def projection-intersection-neutral Cons(4))
moreover
note *Cons*(8) *v′-notin-E1 t1′-Vv1-is-t1-Vv1*
ultimately show *?thesis*
by (*simp add*: *projection-def*)
qed
moreover
note *s2-Cv2-empty t1′-Cv1-empty*
moreover
note *Cons.hyps*(1)[*of ?tau t1′ s2*]

```
          ultimately obtain t'
            where τr2v't'-in-Tr: ?tau @ t' ∈ Tr(ES1 ∥ ES2)
            and t'-Vv-is-lambda': t' ↿ V_V = lambda'
            and t'-Cv-empty: t' ↿ C_V = []
            by auto

          let ?t = r2 @ [V'] @ t'

          note τr2v't'-in-Tr
          moreover
          from r2-Vv-empty t'-Vv-is-lambda' v'-in-Vv have ?t ↿ V_V = V' # lambda'
            by (simp add: projection-def)
          moreover
          have ?t ↿ C_V = []
          proof −
            have r2 ↿ C_V = []
            proof −
              from propSepViews have E_ES2 ∩ C_V ⊆ C_V2
                unfolding properSeparationOfViews-def by auto
              from projection-on-subset[OF ‹E_ES2 ∩ C_V ⊆ C_V2› r2-Cv2-empty]
              have r2 ↿ (E_ES2 ∩ C_V) = []
                by (simp only: Int-commute)
              with projection-intersection-neutral[OF r2-in-E2star, of C_V] show ?thesis
                by simp
            qed
            with v'-in-Vv VIsViewOnE t'-Cv-empty show ?thesis
              by (simp add: isViewOn-def V-valid-def
                VC-disjoint-def VN-disjoint-def NC-disjoint-def projection-def, auto)
          qed
          ultimately have ?thesis
            by auto
        }
        ultimately show ?thesis
          by blast
      qed

    qed
  }
  thus ?thesis
    by auto
qed


lemma generalized-zipping-lemma:
 ∀ τ lambda t1 t2. ( ( set τ ⊆ E(ES1 ∥ ES2)
  ∧ set lambda ⊆ V_V ∧ set t1 ⊆ E_ES1 ∧ set t2 ⊆ E_ES2
  ∧ ((τ ↿ E_ES1) @ t1) ∈ Tr_ES1 ∧ ((τ ↿ E_ES2) @ t2) ∈ Tr_ES2
  ∧ (lambda ↿ E_ES1) = (t1 ↿ V_V) ∧ (lambda ↿ E_ES2) = (t2 ↿ V_V)
  ∧ (t1 ↿ C_V1) = [] ∧ (t2 ↿ C_V2) = [])
  ⟶ (∃ t. ((τ @ t) ∈ Tr(ES1 ∥ ES2) ∧ (t ↿ V_V) = lambda ∧ (t ↿ C_V) = [])) )
proof −
  note well-behaved-composition
```

moreover {
assume $N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap E_{ES1} = \{\}$
with *generalized-zipping-lemma1* **have** *?thesis*
by *auto*
}
moreover {
assume $\exists\ \varrho 1.\ N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge$ *BSIA* $\varrho 1\ \mathcal{V}1\ Tr_{ES1}$
then obtain $\varrho 1$ **where** $N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge$ *BSIA* $\varrho 1\ \mathcal{V}1\ Tr_{ES1}$
by *auto*
with *generalized-zipping-lemma2*[*of* $\varrho 1$] **have** *?thesis*
by *auto*
}
moreover {
assume $\exists\ \varrho 2.\ N_{\mathcal{V}2} \cap E_{ES1} = \{\} \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1}) \wedge$ *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
then obtain $\varrho 2$ **where** $N_{\mathcal{V}2} \cap E_{ES1} = \{\} \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1}) \wedge$ *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
by *auto*
with *generalized-zipping-lemma3*[*of* $\varrho 2$] **have** *?thesis*
by *auto*
}
moreover {
assume $\exists\ \varrho 1\ \varrho 2\ \Gamma 1\ \Gamma 2.\ (\ \nabla_{\Gamma 1} \subseteq E_{ES1} \wedge \Delta_{\Gamma 1} \subseteq E_{ES1} \wedge \Upsilon_{\Gamma 1} \subseteq E_{ES1}$
$\wedge\ \nabla_{\Gamma 2} \subseteq E_{ES2} \wedge \Delta_{\Gamma 2} \subseteq E_{ES2} \wedge \Upsilon_{\Gamma 2} \subseteq E_{ES2}$
$\wedge$ *BSIA* $\varrho 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
$\wedge$ *FCIA* $\varrho 1\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ *FCIA* $\varrho 2\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge\ V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma 1} \cup \nabla_{\Gamma 2}$
$\wedge\ C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma 1} \wedge C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma 2}$
$\wedge\ N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}\ )$
then obtain $\varrho 1\ \varrho 2\ \Gamma 1\ \Gamma 2$ **where** $\nabla_{\Gamma 1} \subseteq E_{ES1} \wedge \Delta_{\Gamma 1} \subseteq E_{ES1} \wedge \Upsilon_{\Gamma 1} \subseteq E_{ES1}$
$\wedge\ \nabla_{\Gamma 2} \subseteq E_{ES2} \wedge \Delta_{\Gamma 2} \subseteq E_{ES2} \wedge \Upsilon_{\Gamma 2} \subseteq E_{ES2}$
$\wedge$ *BSIA* $\varrho 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ *BSIA* $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge$ *total ES1* $(C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge$ *total ES2* $(C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
$\wedge$ *FCIA* $\varrho 1\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ *FCIA* $\varrho 2\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge\ V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma 1} \cup \nabla_{\Gamma 2}$
$\wedge\ C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma 1} \wedge C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma 2}$
$\wedge\ N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$
by *auto*
with *generalized-zipping-lemma4*[*of* $\Gamma 1\ \Gamma 2\ \varrho 1\ \varrho 2$] **have** *?thesis*
by *auto*
}
ultimately show *?thesis* **unfolding** *wellBehavedComposition-def*
by *blast*
qed

end

end

5.4.3 Compositionality Results

theory *CompositionalityResults*
imports *GeneralizedZippingLemma CompositionSupport*

begin

context *Compositionality*
begin

theorem *compositionality-BSD*:
$[\![\mathit{BSD}\ \mathcal{V}1\ \mathit{Tr}_{ES1};\ \mathit{BSD}\ \mathcal{V}2\ \mathit{Tr}_{ES2}]\!] \Longrightarrow \mathit{BSD}\ \mathcal{V}\ \mathit{Tr}_{(ES1\ \|\ ES2)}$
proof −
assume *BSD-Tr1-v1*: $\mathit{BSD}\ \mathcal{V}1\ \mathit{Tr}_{ES1}$
assume *BSD-Tr2-v2*: $\mathit{BSD}\ \mathcal{V}2\ \mathit{Tr}_{ES2}$
{
fix $\alpha\ \beta\ c$
assume *c-in-Cv*: $c \in C_{\mathcal{V}}$
assume $\beta c\alpha$*-in-Tr*: $(\beta\ @\ [c]\ @\ \alpha) \in \mathit{Tr}_{(ES1\ \|\ ES2)}$
assume α*-contains-no-c*: $\alpha \restriction C_{\mathcal{V}} = []$

interpret *CSES1*: *CompositionSupport ES1* $\mathcal{V}$ $\mathcal{V}1$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES1 validV1*)

interpret *CSES2*: *CompositionSupport ES2* $\mathcal{V}$ $\mathcal{V}2$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES2 validV2*)

from $\beta c\alpha$*-in-Tr*
have $\beta c\alpha$*-E1-in-Tr1*: $((\beta\ @\ [c]\ @\ \alpha) \restriction E_{ES1}) \in \mathit{Tr}_{ES1}$
and $\beta c\alpha$*-E2-in-Tr2*: $((\beta\ @\ [c]\ @\ \alpha) \restriction E_{ES2}) \in \mathit{Tr}_{ES2}$
by (*auto*, *simp add*: *composeES-def*)+

from *composeES-yields-ES validES1 validES2* **have** *ES-valid* $(ES1\ \|\ ES2)$
by *auto*

with $\beta c\alpha$*-in-Tr* **have** *set* $\beta \subseteq E_{(ES1\ \|\ ES2)}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $(\alpha \restriction V_{\mathcal{V}}) \subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
have $(\alpha \restriction V_{\mathcal{V}}) \restriction V_{\mathcal{V}} = (\alpha \restriction V_{\mathcal{V}})$
by (*simp add*: *projection-def*)
moreover
from *CSES1.BSD-in-subsystem*[*OF c-in-Cv* $\beta c\alpha$*-E1-in-Tr1 BSD-Tr1-v1*]
obtain $\alpha 1'$
where $\alpha 1'$*-1*: $((\beta \restriction E_{ES1})\ @\ \alpha 1') \in \mathit{Tr}_{ES1}$
and $\alpha 1'$*-2*: $(\alpha 1' \restriction V_{\mathcal{V}1}) = (\alpha \restriction V_{\mathcal{V}1})$
and $\alpha 1' \restriction C_{\mathcal{V}1} = []$
by *auto*
moreover
from $\alpha 1'$*-1 validES1* **have** $\alpha 1'$*-in-E1*: *set* $\alpha 1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)

moreover
from *α1′-2 propSepViews* **have** $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES1}) = (\alpha 1' \restriction V_{\mathcal{V}})$
proof −
have $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES1}) = \alpha \restriction (V_{\mathcal{V}} \cap E_{ES1})$
by (*simp only*: *projection-def*, *auto*)
with *propSepViews* **have** $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES1}) = (\alpha \restriction V_{\mathcal{V}1})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *α1′-2* **have** $(\alpha 1' \restriction V_{\mathcal{V}1}) = (\alpha 1' \restriction V_{\mathcal{V}})$
proof −
from *α1′-in-E1* **have** $\alpha 1' \restriction E_{ES1} = \alpha 1'$
by (*simp add*: *list-subset-iff-projection-neutral*)
hence $(\alpha 1' \restriction E_{ES1}) \restriction V_{\mathcal{V}} = \alpha 1' \restriction V_{\mathcal{V}}$
by *simp*
with *Vv-is-Vv1-union-Vv2* **have** $(\alpha 1' \restriction E_{ES1}) \restriction (V_{\mathcal{V}1} \cup V_{\mathcal{V}2}) = \alpha 1' \restriction V_{\mathcal{V}}$
by *simp*
hence $\alpha 1' \restriction (E_{ES1} \cap (V_{\mathcal{V}1} \cup V_{\mathcal{V}2})) = \alpha 1' \restriction V_{\mathcal{V}}$
by (*simp only*: *projection-def*, *auto*)
hence $\alpha 1' \restriction (E_{ES1} \cap V_{\mathcal{V}1} \cup E_{ES1} \cap V_{\mathcal{V}2}) = \alpha 1' \restriction V_{\mathcal{V}}$
by (*simp add*: *Int-Un-distrib*)
moreover
from *validV1* **have** $E_{ES1} \cap V_{\mathcal{V}1} = V_{\mathcal{V}1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\alpha 1' \restriction (V_{\mathcal{V}1} \cup E_{ES1} \cap V_{\mathcal{V}2}) = \alpha 1' \restriction V_{\mathcal{V}}$
by *simp*
moreover
have $E_{ES1} \cap V_{\mathcal{V}2} \subseteq V_{\mathcal{V}1}$
proof −
from *propSepViews Vv-is-Vv1-union-Vv2* **have** $(V_{\mathcal{V}1} \cup V_{\mathcal{V}2}) \cap E_{ES1} = V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** *simp*
hence $(V_{\mathcal{V}1} \cap E_{ES1} \cup V_{\mathcal{V}2} \cap E_{ES1}) = V_{\mathcal{V}1}$
by *auto*
with *validV1* **have** $(V_{\mathcal{V}1} \cup V_{\mathcal{V}2} \cap E_{ES1}) = V_{\mathcal{V}1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by *auto*
qed
ultimately show *?thesis*
by (*simp add*: *Un-absorb2*)
qed
moreover note *α1′-2*
ultimately show *?thesis*
by *auto*
qed
moreover
from *CSES2.BSD-in-subsystem*[*OF c-in-Cv βcα-E2-in-Tr2 BSD-Tr2-v2*]
obtain *α2′*
where *α2′-1*: $((\beta \restriction E_{ES2}) \mathbin{@} \alpha 2') \in Tr_{ES2}$
and *α2′-2*: $(\alpha 2' \restriction V_{\mathcal{V}2}) = (\alpha \restriction V_{\mathcal{V}2})$
and $\alpha 2' \restriction C_{\mathcal{V}2} = []$

by *auto*
moreover
from *α2′-1 validES2* **have** *α2′-in-E2*: *set* $\alpha 2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *α2′-2 propSepViews* **have** $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES2}) = (\alpha 2' \restriction V_{\mathcal{V}})$
proof −
have $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES2}) = \alpha \restriction (V_{\mathcal{V}} \cap E_{ES2})$
by (*simp only*: *projection-def*, *auto*)
with *propSepViews* **have** $((\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES2}) = (\alpha \restriction V_{\mathcal{V}2})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *α2′-2* **have** $(\alpha 2' \restriction V_{\mathcal{V}2}) = (\alpha 2' \restriction V_{\mathcal{V}})$
proof −
from *α2′-in-E2* **have** $\alpha 2' \restriction E_{ES2} = \alpha 2'$
by (*simp add*: *list-subset-iff-projection-neutral*)
hence $(\alpha 2' \restriction E_{ES2}) \restriction V_{\mathcal{V}} = \alpha 2' \restriction V_{\mathcal{V}}$
by *simp*
with *Vv-is-Vv1-union-Vv2* **have** $(\alpha 2' \restriction E_{ES2}) \restriction (V_{\mathcal{V}2} \cup V_{\mathcal{V}1}) = \alpha 2' \restriction V_{\mathcal{V}}$
by (*simp add*: *Un-commute*)
hence $\alpha 2' \restriction (E_{ES2} \cap (V_{\mathcal{V}2} \cup V_{\mathcal{V}1})) = \alpha 2' \restriction V_{\mathcal{V}}$
by (*simp only*: *projection-def*, *auto*)
hence $\alpha 2' \restriction (E_{ES2} \cap V_{\mathcal{V}2} \cup E_{ES2} \cap V_{\mathcal{V}1}) = \alpha 2' \restriction V_{\mathcal{V}}$
by (*simp add*: *Int-Un-distrib*)
moreover
from *validV2* **have** $E_{ES2} \cap V_{\mathcal{V}2} = V_{\mathcal{V}2}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\alpha 2' \restriction (V_{\mathcal{V}2} \cup E_{ES2} \cap V_{\mathcal{V}1}) = \alpha 2' \restriction V_{\mathcal{V}}$
by *simp*
moreover
have $E_{ES2} \cap V_{\mathcal{V}1} \subseteq V_{\mathcal{V}2}$
proof −
from *propSepViews Vv-is-Vv1-union-Vv2* **have** $(V_{\mathcal{V}2} \cup V_{\mathcal{V}1}) \cap E_{ES2} = V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *Un-commute*)
hence $(V_{\mathcal{V}2} \cap E_{ES2} \cup V_{\mathcal{V}1} \cap E_{ES2}) = V_{\mathcal{V}2}$
by *auto*
with *validV2* **have** $(V_{\mathcal{V}2} \cup V_{\mathcal{V}1} \cap E_{ES2}) = V_{\mathcal{V}2}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by *auto*
qed
ultimately show *?thesis*
by (*simp add*: *Un-absorb2*)
qed
moreover note *α2′-2*
ultimately show *?thesis*
by *auto*
qed
moreover note *generalized-zipping-lemma*
ultimately have $\exists \alpha'.\ ((\beta \mathbin{@} \alpha') \in (Tr_{(ES1 \parallel ES2)}) \wedge (\alpha' \restriction V_{\mathcal{V}} = (\alpha \restriction V_{\mathcal{V}})) \wedge \alpha' \restriction C_{\mathcal{V}} = [])$

```
    by blast
  }
  thus ?thesis
    unfolding BSD-def
    by auto
qed


theorem compositionality-BSI:
⟦ BSD 𝒱1 Tr_ES1; BSD 𝒱2 Tr_ES2; BSI 𝒱1 Tr_ES1; BSI 𝒱2 Tr_ES2 ⟧
   ⟹ BSI 𝒱 Tr_(ES1 ∥ ES2)
proof −
  assume BSD1: BSD 𝒱1 Tr_ES1
    and BSD2: BSD 𝒱2 Tr_ES2
    and BSI1: BSI 𝒱1 Tr_ES1
    and BSI2: BSI 𝒱2 Tr_ES2

  {
    fix α β c
    assume c-in-Cv: c ∈ C_𝒱
    assume βα-in-Tr: (β @ α) ∈ Tr_(ES1 ∥ ES2)
    assume α-no-Cv: α ↾ C_𝒱 = []

    from βα-in-Tr
    have βα-E1-in-Tr1: ((β @ α) ↾ E_ES1) ∈ Tr_ES1
      and βα-E2-in-Tr2: ((β @ α) ↾ E_ES2) ∈ Tr_ES2
      by (simp add: composeES-def)+

    interpret CSES1: CompositionSupport ES1 𝒱 𝒱1
      using propSepViews unfolding properSeparationOfViews-def
      by (simp add: CompositionSupport-def validES1 validV1)

    interpret CSES2: CompositionSupport ES2 𝒱 𝒱2
      using propSepViews unfolding properSeparationOfViews-def
      by (simp add: CompositionSupport-def validES2 validV2)

    from CSES1.BSD-in-subsystem2[OF βα-E1-in-Tr1 BSD1] obtain α1'
      where βE1α1'-in-Tr1: β ↾ E_ES1 @ α1' ∈ Tr_ES1
      and α1'Vv1-is-αVv1: α1' ↾ V_𝒱1 = α ↾ V_𝒱1
      and α1'Cv1-empty: α1' ↾ C_𝒱1 = []
      by auto

    from CSES2.BSD-in-subsystem2[OF βα-E2-in-Tr2 BSD2] obtain α2'
      where βE2α2'-in-Tr2: β ↾ E_ES2 @ α2' ∈ Tr_ES2
      and α2'Vv2-is-αVv2: α2' ↾ V_𝒱2 = α ↾ V_𝒱2
      and α2'Cv2-empty: α2' ↾ C_𝒱2 = []
      by auto

    have ∃ α1''. (set α1'' ⊆ E_ES1 ∧ ((β @ [c]) ↾ E_ES1) @ α1'' ∈ Tr_ES1
      ∧ α1'' ↾ V_𝒱1 = α ↾ V_𝒱1 ∧ α1'' ↾ C_𝒱1 = [])
      proof cases
        assume cE1-empty: [c] ↾ E_ES1 = []
```

from $\beta E1\alpha1'$*-in-Tr1 validES1* **have** *set* $\alpha1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *cE1-empty* $\beta E1\alpha1'$*-in-Tr1* **have** $((\beta \mathbin{@} [c]) \restriction E_{ES1}) \mathbin{@} \alpha1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
note $\alpha1'$*Vv1-is-*α*Vv1* $\alpha1'$*Cv1-empty*
ultimately show *?thesis*
by *auto*
next
assume *cE1-not-empty*: $[c] \restriction E_{ES1} \neq []$
hence *c-in-E1*: $c \in E_{ES1}$
by (*simp only*: *projection-def*, *auto*, *split if-split-asm*, *auto*)

from *c-in-Cv c-in-E1 propSepViews* **have** $c \in C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
note $\beta E1\alpha1'$*-in-Tr1* $\alpha1'$*Cv1-empty BSI1*
ultimately obtain $\alpha1''$
where $\beta E1c\alpha1''$*-in-Tr1*: $(\beta \restriction E_{ES1}) \mathbin{@} [c] \mathbin{@} \alpha1'' \in Tr_{ES1}$
and $\alpha1''$*Vv1-is-*$\alpha1'$*Vv1*: $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha1' \restriction V_{\mathcal{V}1}$
and $\alpha1''$*Cv1-empty*: $\alpha1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSI-def*
by *blast*

from *validES1* $\beta E1c\alpha1''$*-in-Tr1* **have** *set* $\alpha1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from $\beta E1c\alpha1''$*-in-Tr1 c-in-E1* **have** $((\beta \mathbin{@} [c]) \restriction E_{ES1}) \mathbin{@} \alpha1'' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from $\alpha1''$*Vv1-is-*$\alpha1'$*Vv1* $\alpha1'$*Vv1-is-*α*Vv1* **have** $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
by *auto*
moreover
note $\alpha1''$*Cv1-empty*
ultimately show *?thesis*
by *auto*
qed
then obtain $\alpha1''$
where $\alpha1''$*-in-E1star*: *set* $\alpha1'' \subseteq E_{ES1}$
and $\beta cE1\alpha1''$*-in-Tr1*: $((\beta \mathbin{@} [c]) \restriction E_{ES1}) \mathbin{@} \alpha1'' \in Tr_{ES1}$
and $\alpha1''$*Vv1-is-*α*Vv1*: $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and $\alpha1''$*Cv1-empty*: $\alpha1'' \restriction C_{\mathcal{V}1} = []$
by *auto*

have $\exists\ \alpha2''.$ (*set* $\alpha2'' \subseteq E_{ES2}$
$\wedge\ ((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha2'' \in Tr_{ES2}$
$\wedge\ \alpha2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
$\wedge\ \alpha2'' \restriction C_{\mathcal{V}2} = []$)
proof *cases*
assume *cE2-empty*: $[c] \restriction E_{ES2} = []$

from $\beta E2\alpha2'$*-in-Tr2 validES2* **have** *set* $\alpha2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *cE2-empty* $\beta E2\alpha2'$*-in-Tr2* **have** $((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
note $\alpha2'$*Vv2-is-*α*Vv2* $\alpha2'$*Cv2-empty*
ultimately show *?thesis*
by *auto*
next
assume *cE2-not-empty*: $[c] \restriction E_{ES2} \neq []$
hence *c-in-E2*: $c \in E_{ES2}$
by (*simp only*: *projection-def*, *auto*, *split if-split-asm*, *auto*)

from *c-in-Cv c-in-E2 propSepViews* **have** $c \in C_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
note $\beta E2\alpha2'$*-in-Tr2* $\alpha2'$*Cv2-empty BSI2*
ultimately obtain $\alpha2''$
where $\beta E2c\alpha2''$*-in-Tr2*: $(\beta \restriction E_{ES2}) \mathbin{@} [c] \mathbin{@} \alpha2'' \in Tr_{ES2}$
and $\alpha2''$*Vv2-is-*$\alpha2'$*Vv2*: $\alpha2'' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2}$
and $\alpha2''$*Cv2-empty*: $\alpha2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSI-def*
by *blast*

from *validES2* $\beta E2c\alpha2''$*-in-Tr2* **have** *set* $\alpha2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from $\beta E2c\alpha2''$*-in-Tr2 c-in-E2* **have** $((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha2'' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from $\alpha2''$*Vv2-is-*$\alpha2'$*Vv2* $\alpha2'$*Vv2-is-*α*Vv2* **have** $\alpha2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
by *auto*
moreover
note $\alpha2''$*Cv2-empty*
ultimately show *?thesis*
by *auto*
qed
then obtain $\alpha2''$
where $\alpha2''$*-in-E2star*: *set* $\alpha2'' \subseteq E_{ES2}$
and $\beta cE2\alpha2''$*-in-Tr2*: $((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha2'' \in Tr_{ES2}$
and $\alpha2''$*Vv2-is-*α*Vv2*: $\alpha2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and $\alpha2''$*Cv2-empty*: $\alpha2'' \restriction C_{\mathcal{V}2} = []$
by *auto*

from *VIsViewOnE c-in-Cv* $\beta\alpha$*-in-Tr* **have** *set* $(\beta \mathbin{@} [c]) \subseteq E_{(ES1 \parallel ES2)}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def composeES-def, *auto*)
moreover
have *set* $(\alpha \restriction V_{\mathcal{V}}) \subseteq V_{\mathcal{V}}$

```
    by (simp add: projection-def, auto)
  moreover
  note α1''-in-E1star α2''-in-E2star βcE1α1''-in-Tr1 βcE2α2''-in-Tr2
  moreover
  have (α ↾ V_𝒱) ↾ E_ES1 = α1'' ↾ V_𝒱
    proof −
      from α1''Vv1-is-αVv1 propSepViews have α ↾ (V_𝒱 ∩ E_ES1) = α1'' ↾ (E_ES1 ∩ V_𝒱)
        unfolding properSeparationOfViews-def by (simp add: Int-commute)
      hence α ↾ V_𝒱 ↾ E_ES1 = α1'' ↾ E_ES1 ↾ V_𝒱
        by (simp add: projection-def)
      with α1''-in-E1star show ?thesis
        by (simp add: list-subset-iff-projection-neutral)
    qed
  moreover
  have (α ↾ V_𝒱) ↾ E_ES2 = α2'' ↾ V_𝒱
    proof −
      from α2''Vv2-is-αVv2 propSepViews have α ↾ (V_𝒱 ∩ E_ES2) = α2'' ↾ (E_ES2 ∩ V_𝒱)
        unfolding properSeparationOfViews-def by (simp add: Int-commute)
      hence α ↾ V_𝒱 ↾ E_ES2 = α2'' ↾ E_ES2 ↾ V_𝒱
        by (simp add: projection-def)
      with α2''-in-E2star show ?thesis
        by (simp add: list-subset-iff-projection-neutral)
    qed
  moreover
  note α1''Cv1-empty α2''Cv2-empty generalized-zipping-lemma
  ultimately have ∃α'. (β @ [c]) @ α' ∈ Tr_(ES1 ∥ ES2) ∧ α' ↾ V_𝒱 = α ↾ V_𝒱 ∧ α' ↾ C_𝒱 = []
    by blast
}
thus ?thesis
  unfolding BSI-def
  by auto
qed


theorem compositionality-BSIA:
⟦ BSD 𝒱1 Tr_ES1; BSD 𝒱2 Tr_ES2; BSIA ϱ1 𝒱1 Tr_ES1; BSIA ϱ2 𝒱2 Tr_ES2;
  (ϱ1 𝒱1) ⊆ (ϱ 𝒱) ∩ E_ES1; (ϱ2 𝒱2) ⊆ (ϱ 𝒱) ∩ E_ES2 ⟧
  ⟹ BSIA ϱ 𝒱 (Tr_(ES1 ∥ ES2))
proof −
  assume BSD1: BSD 𝒱1 Tr_ES1
  and BSD2: BSD 𝒱2 Tr_ES2
  and BSIA1: BSIA ϱ1 𝒱1 Tr_ES1
  and BSIA2: BSIA ϱ2 𝒱2 Tr_ES2
  and ϱ1v1-subset-ϱv-inter-E1: (ϱ1 𝒱1) ⊆ (ϱ 𝒱) ∩ E_ES1
  and ϱ2v2-subset-ϱv-inter-E2:(ϱ2 𝒱2) ⊆ (ϱ 𝒱) ∩ E_ES2

  {
    fix α β c
    assume c-in-Cv: c ∈ C_𝒱
    assume βα-in-Tr: (β @ α) ∈ Tr_(ES1 ∥ ES2)
    assume α-no-Cv: α ↾ C_𝒱 = []
    assume Adm: (Adm 𝒱 ϱ Tr_(ES1 ∥ ES2) β c)
```

then obtain γ
where $\gamma\varrho v\text{-}is\text{-}\beta\varrho v$: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and $\gamma c\text{-}in\text{-}Tr$: $(\gamma\ @\ [c]) \in Tr_{(ES1\ \|\ ES2)}$
unfolding *Adm-def*
by *auto*

from $\beta\alpha\text{-}in\text{-}Tr$
have $\beta\alpha\text{-}E1\text{-}in\text{-}Tr1$: $((\beta\ @\ \alpha) \restriction E_{ES1}) \in Tr_{ES1}$
and $\beta\alpha\text{-}E2\text{-}in\text{-}Tr2$: $((\beta\ @\ \alpha) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *composeES-def*)+

interpret *CSES1*: *CompositionSupport ES1* $\mathcal{V}$ $\mathcal{V}1$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES1 validV1*)

interpret *CSES2*: *CompositionSupport ES2* $\mathcal{V}$ $\mathcal{V}2$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES2 validV2*)

from *CSES1.BSD-in-subsystem2*[*OF* $\beta\alpha\text{-}E1\text{-}in\text{-}Tr1$ *BSD1*] **obtain** $\alpha1'$
where $\beta E1\alpha1'\text{-}in\text{-}Tr1$: $\beta \restriction E_{ES1}\ @\ \alpha1' \in Tr_{ES1}$
and $\alpha1'Vv1\text{-}is\text{-}\alpha Vv1$: $\alpha1' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and $\alpha1'Cv1\text{-}empty$: $\alpha1' \restriction C_{\mathcal{V}1} = []$
by *auto*

from *CSES2.BSD-in-subsystem2*[*OF* $\beta\alpha\text{-}E2\text{-}in\text{-}Tr2$ *BSD2*] **obtain** $\alpha2'$
where $\beta E2\alpha2'\text{-}in\text{-}Tr2$: $\beta \restriction E_{ES2}\ @\ \alpha2' \in Tr_{ES2}$
and $\alpha2'Vv2\text{-}is\text{-}\alpha Vv2$: $\alpha2' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and $\alpha2'Cv2\text{-}empty$: $\alpha2' \restriction C_{\mathcal{V}2} = []$
by *auto*

have $\exists\ \alpha1''.\ (set\ \alpha1'' \subseteq E_{ES1}$
$\wedge\ ((\beta\ @\ [c]) \restriction E_{ES1})\ @\ \alpha1'' \in Tr_{ES1}$
$\wedge\ \alpha1'' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
$\wedge\ \alpha1'' \restriction C_{\mathcal{V}1} = [])$
proof *cases*
assume $cE1\text{-}empty$: $[c] \restriction E_{ES1} = []$

from $\beta E1\alpha1'\text{-}in\text{-}Tr1$ *validES1* **have** $set\ \alpha1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from $cE1\text{-}empty$ $\beta E1\alpha1'\text{-}in\text{-}Tr1$ **have** $((\beta\ @\ [c]) \restriction E_{ES1})\ @\ \alpha1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
note $\alpha1'Vv1\text{-}is\text{-}\alpha Vv1$ $\alpha1'Cv1\text{-}empty$
ultimately show *?thesis*
by *auto*
next
assume $cE1\text{-}not\text{-}empty$: $[c] \restriction E_{ES1} \neq []$
hence $c\text{-}in\text{-}E1$: $c \in E_{ES1}$
by (*simp only*: *projection-def*, *auto*, *split if-split-asm*, *auto*)

from *c-in-Cv c-in-E1 propSepViews* **have** $c \in C_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
note $\beta E1\alpha1'$*-in-Tr1* $\alpha1'$*Cv1-empty*
moreover
have (*Adm* $\mathcal{V}1$ $\varrho1$ Tr_{ES1} ($\beta \restriction E_{ES1}$) c)
proof −
from *c-in-E1* γ*c-in-Tr* **have** ($\gamma \restriction E_{ES1}$) @ $[c] \in Tr_{ES1}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES1} \restriction (\varrho1\ \mathcal{V}1) = \beta \restriction E_{ES1} \restriction (\varrho1\ \mathcal{V}1)$
proof −
from $\gamma\varrho v$*-is-*$\beta\varrho v$ **have** $\gamma \restriction E_{ES1} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES1} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with $\varrho1v1$*-subset-*ϱv*-inter-E1* **have** $\gamma \restriction (\varrho1\ \mathcal{V}1) = \beta \restriction (\varrho1\ \mathcal{V}1)$
by (*metis Int-subset-iff* $\gamma\varrho v$*-is-*$\beta\varrho v$ *projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *BSIA1*
ultimately obtain $\alpha1''$
where $\beta E1c\alpha1''$*-in-Tr1*: ($\beta \restriction E_{ES1}$) @ $[c]$ @ $\alpha1'' \in Tr_{ES1}$
and $\alpha1''Vv1$*-is-*$\alpha1'Vv1$: $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha1' \restriction V_{\mathcal{V}1}$
and $\alpha1''Cv1$*-empty*: $\alpha1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSIA-def*
by *blast*

from *validES1* $\beta E1c\alpha1''$*-in-Tr1* **have** *set* $\alpha1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from $\beta E1c\alpha1''$*-in-Tr1 c-in-E1* **have** ((β @ $[c]$) $\restriction E_{ES1}$) @ $\alpha1'' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from $\alpha1''Vv1$*-is-*$\alpha1'Vv1$ $\alpha1'Vv1$*-is-*$\alpha Vv1$ **have** $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
by *auto*
moreover
note $\alpha1''Cv1$*-empty*
ultimately show *?thesis*
by *auto*
qed
then obtain $\alpha1''$
where $\alpha1''$*-in-E1star*: *set* $\alpha1'' \subseteq E_{ES1}$
and $\beta cE1\alpha1''$*-in-Tr1*: ((β @ $[c]$) $\restriction E_{ES1}$) @ $\alpha1'' \in Tr_{ES1}$
and $\alpha1''Vv1$*-is-*$\alpha Vv1$: $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and $\alpha1''Cv1$*-empty*: $\alpha1'' \restriction C_{\mathcal{V}1} = []$
by *auto*

have $\exists\ \alpha 2''.\ (set\ \alpha 2'' \subseteq E_{ES2}$
$\wedge\ ((\beta\ @\ [c]) \restriction E_{ES2})\ @\ \alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
$\wedge\ \alpha 2'' \restriction C_{\mathcal{V}2} = [])$
proof *cases*
assume *cE2-empty*: $[c] \restriction E_{ES2} = []$

from *βE2α2′-in-Tr2 validES2* **have** *set* $\alpha 2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *cE2-empty βE2α2′-in-Tr2* **have** $((\beta\ @\ [c]) \restriction E_{ES2})\ @\ \alpha 2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
note *α2′Vv2-is-αVv2 α2′Cv2-empty*
ultimately show *?thesis*
by *auto*
next
assume *cE2-not-empty*: $[c] \restriction E_{ES2} \neq []$
hence *c-in-E2*: $c \in E_{ES2}$
by (*simp only*: *projection-def*, *auto*, *split if-split-asm*, *auto*)

from *c-in-Cv c-in-E2 propSepViews* **have** $c \in C_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
note *βE2α2′-in-Tr2 α2′Cv2-empty*
moreover
have $(Adm\ \mathcal{V}2\ \varrho 2\ Tr_{ES2}\ (\beta \restriction E_{ES2})\ c)$
proof −
from *c-in-E2 γc-in-Tr* **have** $(\gamma \restriction E_{ES2})\ @\ [c] \in Tr_{ES2}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2)$
proof −
from *γϱv-is-βϱv* **have** $\gamma \restriction E_{ES2} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES2} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with *ϱ2v2-subset-ϱv-inter-E2* **have** $\gamma \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction (\varrho 2\ \mathcal{V}2)$
by (*metis Int-subset-iff γϱv-is-βϱv projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *βE2cα2″-in-Tr2*: $(\beta \restriction E_{ES2})\ @\ [c]\ @\ \alpha 2'' \in Tr_{ES2}$
and *α2″Vv2-is-α2′Vv2*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *α2″Cv2-empty*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSIA-def*
by *blast*

from *validES2* $\beta E2c\alpha 2''$*-in-Tr2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from $\beta E2c\alpha 2''$*-in-Tr2 c-in-E2* **have** $((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha 2'' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
from $\alpha 2''$*Vv2-is-*$\alpha 2'$*Vv2* $\alpha 2'$*Vv2-is-*α*Vv2* **have** $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
by *auto*
moreover
note $\alpha 2''$*Cv2-empty*
ultimately show *?thesis*
by *auto*
qed
then obtain $\alpha 2''$
where $\alpha 2''$*-in-E2star*: *set* $\alpha 2'' \subseteq E_{ES2}$
and $\beta cE2\alpha 2''$*-in-Tr2*: $((\beta \mathbin{@} [c]) \restriction E_{ES2}) \mathbin{@} \alpha 2'' \in Tr_{ES2}$
and $\alpha 2''$*Vv2-is-*α*Vv2*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and $\alpha 2''$*Cv2-empty*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
by *auto*

from *VIsViewOnE c-in-Cv* $\beta\alpha$*-in-Tr* **have** *set* $(\beta \mathbin{@} [c]) \subseteq E_{(ES1 \parallel ES2)}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def composeES-def, *auto*)
moreover
have *set* $(\alpha \restriction V_{\mathcal{V}}) \subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
note $\alpha 1''$*-in-E1star* $\alpha 2''$*-in-E2star* $\beta cE1\alpha 1''$*-in-Tr1* $\beta cE2\alpha 2''$*-in-Tr2*
moreover
have $(\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES1} = \alpha 1'' \restriction V_{\mathcal{V}}$
proof −
from $\alpha 1''$*Vv1-is-*α*Vv1 propSepViews*
have $\alpha \restriction (V_{\mathcal{V}} \cap E_{ES1}) = \alpha 1'' \restriction (E_{ES1} \cap V_{\mathcal{V}})$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *Int-commute*)
hence $\alpha \restriction V_{\mathcal{V}} \restriction E_{ES1} = \alpha 1'' \restriction E_{ES1} \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
with $\alpha 1''$*-in-E1star* **show** *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover
have $(\alpha \restriction V_{\mathcal{V}}) \restriction E_{ES2} = \alpha 2'' \restriction V_{\mathcal{V}}$
proof −
from $\alpha 2''$*Vv2-is-*α*Vv2 propSepViews*
have $\alpha \restriction (V_{\mathcal{V}} \cap E_{ES2}) = \alpha 2'' \restriction (E_{ES2} \cap V_{\mathcal{V}})$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *Int-commute*)
hence $\alpha \restriction V_{\mathcal{V}} \restriction E_{ES2} = \alpha 2'' \restriction E_{ES2} \restriction V_{\mathcal{V}}$
by (*simp add*: *projection-def*)
with $\alpha 2''$*-in-E2star* **show** *?thesis*
by (*simp add*: *list-subset-iff-projection-neutral*)
qed
moreover

note *α1''Cv1-empty α2''Cv2-empty generalized-zipping-lemma*
ultimately have $\exists \alpha'.\ (\beta\ @\ [c])\ @\ \alpha' \in Tr_{(ES1\ \|\ ES2)} \wedge \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$
by *blast*
}
thus *?thesis*
unfolding *BSIA-def*
by *auto*
qed

theorem *compositionality-FCD*:
$[\![$ *BSD* $\mathcal{V}1$ Tr_{ES1}; *BSD* $\mathcal{V}2$ Tr_{ES2};
$\nabla_\Gamma \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$; $\nabla_\Gamma \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$;
$\Upsilon_\Gamma \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$; $\Upsilon_\Gamma \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$;
$(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_\Gamma$;
$N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}$; $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$;
FCD $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}; *FCD* $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2} $]\!]$
$\Longrightarrow$ *FCD* Γ $\mathcal{V}$ $(Tr_{(ES1\ \|\ ES2)})$
proof −
assume *BSD1*: *BSD* $\mathcal{V}1$ Tr_{ES1}
and *BSD2*: *BSD* $\mathcal{V}2$ Tr_{ES2}
and *Nabla-inter-E1-subset-Nabla1*: $\nabla_\Gamma \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$
and *Nabla-inter-E2-subset-Nabla2*: $\nabla_\Gamma \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$
and *Upsilon-inter-E1-subset-Upsilon1*: $\Upsilon_\Gamma \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$
and *Upsilon-inter-E2-subset-Upsilon2*: $\Upsilon_\Gamma \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$
and *Delta1-N1-Delta2-N2-subset-Delta*: $(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_\Gamma$
and *N1-Delta1-E2-disjoint*: $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}$
and *N2-Delta2-E1-disjoint*: $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$
and *FCD1*: *FCD* $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}
and *FCD2*: *FCD* $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2}

{
fix α β c v'
assume *c-in-Cv-inter-Upsilon*: $c \in (C_{\mathcal{V}} \cap \Upsilon_\Gamma)$
and *v'-in-Vv-inter-Nabla*: $v' \in (V_{\mathcal{V}} \cap \nabla_\Gamma)$
and *βcv'α-in-Tr*: $(\beta\ @\ [c,v']\ @\ \alpha) \in Tr_{(ES1\ \|\ ES2)}$
and *αCv-empty*: $\alpha \restriction C_{\mathcal{V}} = []$

from *βcv'α-in-Tr*
have *βcv'α-E1-in-Tr1*: $(((\beta\ @\ [c,v'])\ @\ \alpha) \restriction E_{ES1}) \in Tr_{ES1}$
and *βcv'α-E2-in-Tr2*: $(((\beta\ @\ [c,v'])\ @\ \alpha) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *composeES-def*)+

interpret *CSES1*: *CompositionSupport ES1* $\mathcal{V}$ $\mathcal{V}1$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES1 validV1*)

interpret *CSES2*: *CompositionSupport ES2* $\mathcal{V}$ $\mathcal{V}2$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES2 validV2*)

from *CSES1.BSD-in-subsystem2*[*OF βcv'α-E1-in-Tr1 BSD1*] **obtain** *α1'*

where $\beta cv'E1\alpha1'$*-in-Tr1*: $(\beta \mathbin{@} [c,v']) \restriction E_{ES1} \mathbin{@} \alpha1' \in Tr_{ES1}$
and $\alpha1'$*Vv1-is-*α*Vv1*: $\alpha1' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and $\alpha1'$*Cv1-empty*: $\alpha1' \restriction C_{\mathcal{V}1} = []$
by *auto*

from *CSES2.BSD-in-subsystem2*[*OF* $\beta cv'\alpha$*-E2-in-Tr2 BSD2*] **obtain** $\alpha2'$
where $\beta cv'E2\alpha2'$*-in-Tr2*: $(\beta \mathbin{@} [c,v']) \restriction E_{ES2} \mathbin{@} \alpha2' \in Tr_{ES2}$
and $\alpha2'$*Vv2-is-*α*Vv2*: $\alpha2' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and $\alpha2'$*Cv2-empty*: $\alpha2' \restriction C_{\mathcal{V}2} = []$
by *auto*

from *c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV1*
have $c \notin E_{ES1} \vee (c \in E_{ES1} \wedge v' \notin E_{ES1}) \vee (c \in E_{ES1} \wedge v' \in E_{ES1})$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def)
moreover {
assume *c-notin-E1*: $c \notin E_{ES1}$

have *set* $[] \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
by *auto*
moreover
from $\beta cv'E1\alpha1'$*-in-Tr1 c-notin-E1* **have** $(\beta \restriction E_{ES1}) \mathbin{@} [] \mathbin{@} ([v'] \restriction E_{ES1}) \mathbin{@} \alpha1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
have $\alpha1' \restriction V_{\mathcal{V}1} = \alpha1' \restriction V_{\mathcal{V}1}$ **..**
moreover
note $\alpha1'$*Cv1-empty*
ultimately have $\exists\ \alpha1''\ \delta1''.$ *set* $\delta1'' \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
$\wedge\ (\beta \restriction E_{ES1}) \mathbin{@} \delta1'' \mathbin{@} ([v'] \restriction E_{ES1}) \mathbin{@} \alpha1'' \in Tr_{ES1}$
$\wedge\ \alpha1'' \restriction V_{\mathcal{V}1} = \alpha1' \restriction V_{\mathcal{V}1} \wedge \alpha1'' \restriction C_{\mathcal{V}1} = []$
by *blast*
}
moreover {
assume *c-in-E1*: $c \in E_{ES1}$
and *v'-notin-E1*: $v' \notin E_{ES1}$

from *c-in-E1 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E1-subset-Upsilon1
have *c-in-Cv1-Upsilon1*: $c \in (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
unfolding *properSeparationOfViews-def* **by** *auto*
hence *c-in-Cv1*: $c \in C_{\mathcal{V}1}$
by *auto*
moreover
from $\beta cv'E1\alpha1'$*-in-Tr1 c-in-E1 v'-notin-E1* **have** $(\beta \restriction E_{ES1}) \mathbin{@} [c] \mathbin{@} \alpha1' \in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
note $\alpha1'$*Cv1-empty BSD1*
ultimately obtain $\alpha1''$
where *first*: $(\beta \restriction E_{ES1}) \mathbin{@} \alpha1'' \in Tr_{ES1}$
and *second*: $\alpha1'' \restriction V_{\mathcal{V}1} = \alpha1' \restriction V_{\mathcal{V}1}$
and *third*: $\alpha1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSD-def*

by *blast*

have *set* $[] \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
by *auto*
moreover
from *first v′-notin-E1* **have** $(\beta \restriction E_{ES1}) @ [] @ ([v'] \restriction E_{ES1}) @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *second third*
ultimately
have $\exists\ \alpha 1''\ \delta 1''.\ set\ \delta 1'' \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
$\wedge\ (\beta \restriction E_{ES1}) @ \delta 1'' @ ([v'] \restriction E_{ES1}) @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
by *blast*
}
moreover {
assume *c-in-E1*: $c \in E_{ES1}$
and *v′-in-E1*: $v' \in E_{ES1}$

from *c-in-E1 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E1-subset-Upsilon1
have *c-in-Cv1-Upsilon1*: $c \in (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *v′-in-E1 v′-in-Vv-inter-Nabla propSepViews Nabla-inter-E1-subset-Nabla1*
have *v′-in-Vv1-inter-Nabla1*: $v' \in (V_{\mathcal{V}1} \cap \nabla_{\Gamma 1})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *βcv′E1α1′-in-Tr1 c-in-E1 v′-in-E1* **have** $(\beta \restriction E_{ES1}) @ [c,v'] @ \alpha 1' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *α1′Cv1-empty FCD1*
ultimately obtain *α1″ δ1″*
where *first*: $set\ \delta 1'' \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
and *second*: $(\beta \restriction E_{ES1}) @ \delta 1'' @ [v'] @ \alpha 1'' \in Tr_{ES1}$
and *third*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *fourth*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCD-def*
by *blast*

from *second v′-in-E1* **have** $(\beta \restriction E_{ES1}) @ \delta 1'' @ ([v'] \restriction E_{ES1}) @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
with *first third fourth*
have $\exists\ \alpha 1''\ \delta 1''.\ set\ \delta 1'' \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
$\wedge\ (\beta \restriction E_{ES1}) @ \delta 1'' @ ([v'] \restriction E_{ES1}) @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCD-def*
by *blast*
}
ultimately obtain *α1″ δ1″*
where *δ1″-in-Nv1-Delta1-star*: $set\ \delta 1'' \subseteq (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
and *βE1δ1″vE1α1″-in-Tr1*: $(\beta \restriction E_{ES1}) @ \delta 1'' @ ([v'] \restriction E_{ES1}) @ \alpha 1'' \in Tr_{ES1}$

and $\alpha 1''Vv1\text{-}is\text{-}\alpha 1'Vv1$: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and $\alpha 1''Cv1\text{-}empty$: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
by *blast*
with *validV1* **have** $\delta 1''\text{-}in\text{-}E1\text{-}star$: *set* $\delta 1'' \subseteq E_{ES1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

from *c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV2*
have $c \notin E_{ES2} \vee (c \in E_{ES2} \wedge v' \notin E_{ES2}) \vee (c \in E_{ES2} \wedge v' \in E_{ES2})$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def)
moreover {
assume *c-notin-E2*: $c \notin E_{ES2}$

have *set* $[] \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *auto*
moreover
from $\beta cv'E2\alpha 2'\text{-}in\text{-}Tr2$ *c-notin-E2* **have** $(\beta \restriction E_{ES2})$ @ [] @ $([v'] \restriction E_{ES2})$ @ $\alpha 2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
have $\alpha 2' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$ **..**
moreover
note $\alpha 2'Cv2\text{-}empty$
ultimately have $\exists\ \alpha 2''\ \delta 2''$. *set* $\delta 2'' \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
$\wedge\ (\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = []$
by *blast*
}
moreover {
assume *c-in-E2*: $c \in E_{ES2}$
and *v'-notin-E2*: $v' \notin E_{ES2}$

from *c-in-E2 c-in-Cv-inter-Upsilon propSepViews Upsilon-inter-E2-subset-Upsilon2*
have *c-in-Cv2-Upsilon2*: $c \in (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$
unfolding *properSeparationOfViews-def* **by** *auto*
hence *c-in-Cv2*: $c \in C_{\mathcal{V}2}$
by *auto*
moreover
from $\beta cv'E2\alpha 2'\text{-}in\text{-}Tr2$ *c-in-E2 v'-notin-E2* **have** $(\beta \restriction E_{ES2})$ @ $[c]$ @ $\alpha 2' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
moreover
note $\alpha 2'Cv2\text{-}empty$ *BSD2*
ultimately obtain $\alpha 2''$
where *first*: $(\beta \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
and *second*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *third*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSD-def*
by *blast*

have *set* $[] \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *auto*
moreover

from *first* v'*-notin-E2* **have** $(\beta \restriction E_{ES2})$ @ [] @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *second third*
ultimately
have $\exists\ \alpha 2''\ \delta 2''.\ set\ \delta 2'' \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
$\wedge\ (\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = []$
by *blast*
}
moreover {
assume *c-in-E2*: $c \in E_{ES2}$
and v'*-in-E2*: $v' \in E_{ES2}$

from *c-in-E2 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E2-subset-Upsilon2
have *c-in-Cv2-Upsilon2*: $c \in (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from v'*-in-E2* v'*-in-Vv-inter-Nabla propSepViews Nabla-inter-E2-subset-Nabla2*
have v'*-in-Vv2-inter-Nabla2*: $v' \in (V_{\mathcal{V}2} \cap \nabla_{\Gamma 2})$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from $\beta cv'E2\alpha 2'$*-in-Tr2 c-in-E2* v'*-in-E2* **have** $(\beta \restriction E_{ES2})$ @ $[c,v']$ @ $\alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note $\alpha 2'$*Cv2-empty FCD2*
ultimately obtain $\alpha 2''\ \delta 2''$
where *first*: $set\ \delta 2'' \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
and *second*: $(\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $[v']$ @ $\alpha 2'' \in Tr_{ES2}$
and *third*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *fourth*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *FCD-def*
by *blast*

from *second* v'*-in-E2* **have** $(\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
with *first third fourth*
have $\exists\ \alpha 2''\ \delta 2''.\ set\ \delta 2'' \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
$\wedge\ (\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *FCD-def*
by *blast*
}
ultimately obtain $\alpha 2''\ \delta 2''$
where $\delta 2''$*-in-Nv2-Delta2-star*: $set\ \delta 2'' \subseteq (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
and $\beta E2\delta 2''vE2\alpha 2''$*-in-Tr2*: $(\beta \restriction E_{ES2})$ @ $\delta 2''$ @ $([v'] \restriction E_{ES2})$ @ $\alpha 2'' \in Tr_{ES2}$
and $\alpha 2''Vv2$*-is-*$\alpha 2'Vv2$: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and $\alpha 2''$*Cv2-empty*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
by *blast*
with *validV2* **have** $\delta 2''$*-in-E2-star*: $set\ \delta 2'' \subseteq E_{ES2}$
by (*simp add*: *isViewOn-def V-valid-def*

VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

from *δ1''-in-Nv1-Delta1-star N1-Delta1-E2-disjoint*
have *δ1''E2-empty*: $\delta 1'' \restriction E_{ES2} = []$
proof −
from *δ1''-in-Nv1-Delta1-star* **have** $\delta 1'' = \delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
by (*simp only*: *list-subset-iff-projection-neutral*)
hence $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction E_{ES2}$
by *simp*
moreover
have $\delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction E_{ES2} = \delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2})$
by (*simp only*: *projection-def*, *auto*)
with *N1-Delta1-E2-disjoint* **have** $\delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}) \restriction E_{ES2} = []$
by (*simp add*: *projection-def*)
ultimately show *?thesis*
by *simp*
qed
moreover
from *δ2''-in-Nv2-Delta2-star N2-Delta2-E1-disjoint* **have** *δ2''E1-empty*: $\delta 2'' \restriction E_{ES1} = []$
proof −
from *δ2''-in-Nv2-Delta2-star* **have** $\delta 2'' = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by (*simp only*: *list-subset-iff-projection-neutral*)
hence $\delta 2'' \restriction E_{ES1} = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES1}$
by *simp*
moreover
have $\delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES1} = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1})$
by (*simp only*: *projection-def*, *auto*)
with *N2-Delta2-E1-disjoint* **have** $\delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES1} = []$
by (*simp add*: *projection-def*)
ultimately show *?thesis*
by *simp*
qed
moreover
note *βE1δ1''vE1α1''-in-Tr1 βE2δ2''vE2α2''-in-Tr2 δ1''-in-E1-star δ2''-in-E2-star*
ultimately have *βδ1''δ2''v'E1α1''-in-Tr1*: $(\beta \mathbin{@} \delta 1'' \mathbin{@} \delta 2'' \mathbin{@} [v']) \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
and *βδ1''δ2''v'E2α2''-in-Tr2*: $(\beta \mathbin{@} \delta 1'' \mathbin{@} \delta 2'' \mathbin{@} [v']) \restriction E_{ES2} \mathbin{@} \alpha 2'' \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute list-subset-iff-projection-neutral*, *auto*,
simp only: *projection-concatenation-commute list-subset-iff-projection-neutral*, *auto*)

have *set* $(\beta \mathbin{@} \delta 1'' \mathbin{@} \delta 2'' \mathbin{@} [v']) \subseteq E_{(ES1 \parallel ES2)}$
proof −
from *βcv'α-in-Tr* **have** *set* $\beta \subseteq E_{(ES1 \parallel ES2)}$
by (*simp add*: *composeES-def*)
moreover
note *δ1''-in-E1-star δ2''-in-E2-star*
moreover
from *v'-in-Vv-inter-Nabla VIsViewOnE*
have $v' \in E_{(ES1 \parallel ES2)}$
by (*simp add*:*isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately show *?thesis*
by (*simp add*: *composeES-def*, *auto*)

```
  qed
moreover
have set (α ↾ V_V) ⊆ V_V
  by (simp add: projection-def, auto)
moreover
from βE1δ1''vE1α1''-in-Tr1 validES1 have α1''-in-E1-star: set α1'' ⊆ E_ES1
  by (simp add: ES-valid-def traces-contain-events-def, auto)
moreover
from βE2δ2''vE2α2''-in-Tr2 validES2 have α2''-in-E2-star: set α2'' ⊆ E_ES2
  by (simp add: ES-valid-def traces-contain-events-def, auto)
moreover
note βδ1''δ2''v'E1α1''-in-Tr1 βδ1''δ2''v'E2α2''-in-Tr2
moreover
have (α ↾ V_V) ↾ E_ES1 = α1'' ↾ V_V
  proof −
    from α1''Vv1-is-α1'Vv1 α1'Vv1-is-αVv1 propSepViews
    have α ↾ (V_V ∩ E_ES1) = α1'' ↾ (E_ES1 ∩ V_V)
      unfolding properSeparationOfViews-def by (simp add: Int-commute)
    hence α ↾ V_V ↾ E_ES1 = α1'' ↾ E_ES1 ↾ V_V
      by (simp add: projection-def)
    with α1''-in-E1-star show ?thesis
      by (simp add: list-subset-iff-projection-neutral)
  qed
moreover
have (α ↾ V_V) ↾ E_ES2 = α2'' ↾ V_V
  proof −
    from α2''Vv2-is-α2'Vv2 α2'Vv2-is-αVv2 propSepViews
    have α ↾ (V_V ∩ E_ES2) = α2'' ↾ (E_ES2 ∩ V_V)
      unfolding properSeparationOfViews-def by (simp add: Int-commute)
    hence α ↾ V_V ↾ E_ES2 = α2'' ↾ E_ES2 ↾ V_V
      by (simp add: projection-def)
    with α2''-in-E2-star show ?thesis
      by (simp add: list-subset-iff-projection-neutral)
  qed
moreover
note α1''Cv1-empty α2''Cv2-empty generalized-zipping-lemma
ultimately obtain t
  where first: (β @ δ1'' @ δ2'' @ [v']) @ t ∈ Tr_(ES1 ∥ ES2)
  and second: t ↾ V_V = α ↾ V_V
  and third: t ↾ C_V = []
  by blast

from δ1''-in-Nv1-Delta1-star δ2''-in-Nv2-Delta2-star
have set (δ1'' @ δ2'') ⊆ (N_V ∩ Δ_Γ)
  proof −
    have set (δ1'' @ δ2'') ⊆ Δ_Γ
      proof −
        from δ1''-in-Nv1-Delta1-star δ2''-in-Nv2-Delta2-star
        have set (δ1'' @ δ2'') ⊆ Δ_Γ1 ∩ N_V1 ∪ Δ_Γ2 ∩ N_V2
          by auto
        with Delta1-N1-Delta2-N2-subset-Delta show ?thesis
          by auto
```

qed
moreover
have *set* $(\delta 1'' \mathbin{@} \delta 2'') \subseteq N_{\mathcal{V}}$
proof −
from *δ1''-in-Nv1-Delta1-star δ2''-in-Nv2-Delta2-star*
have *set* $(\delta 1'' \mathbin{@} \delta 2'') \subseteq (N_{\mathcal{V}1} \cup N_{\mathcal{V}2})$
by *auto*
with *Nv1-union-Nv2-subsetof-Nv* **show** *?thesis*
by *auto*
qed
ultimately show *?thesis*
by *auto*
qed
moreover
from *first* **have** $\beta \mathbin{@} (\delta 1'' \mathbin{@} \delta 2'') \mathbin{@} [v'] \mathbin{@} t \in Tr_{(ES1 \parallel ES2)}$
by *auto*
moreover
note *second third*
ultimately have $\exists \alpha'.\ \exists \gamma'.\ (set\ \gamma') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma})$
$\wedge\ ((\beta \mathbin{@} \gamma' \mathbin{@} [v'] \mathbin{@} \alpha') \in Tr_{(ES1 \parallel ES2)}$
$\wedge\ (\alpha' \restriction V_{\mathcal{V}}) = (\alpha \restriction V_{\mathcal{V}})$
$\wedge\ \alpha' \restriction C_{\mathcal{V}} = [])$
by *blast*
}
thus *?thesis*
unfolding *FCD-def*
by *auto*
qed

theorem *compositionality-FCI*:
$[\![$ *BSD* $\mathcal{V}1$ Tr_{ES1}; *BSD* $\mathcal{V}2$ Tr_{ES2}; *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}; *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2};
total ES1 $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$; *total ES2* $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$;
$\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$; $\nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$;
$\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$; $\Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$;
$(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_{\Gamma}$;
$(N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
$\vee\ (\ N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2})$;
FCI $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}; *FCI* $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2} $]\!]$
$\Longrightarrow$ *FCI* Γ $\mathcal{V}$ $(Tr_{(ES1 \parallel ES2)})$
proof −
assume *BSD1*: *BSD* $\mathcal{V}1$ Tr_{ES1}
and *BSD2*: *BSD* $\mathcal{V}2$ Tr_{ES2}
and *BSIA1*: *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}
and *BSIA2*: *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2}
and *total-ES1-C1-inter-Upsilon1*: *total ES1* $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1})$
and *total-ES2-C2-inter-Upsilon2*: *total ES2* $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2})$
and *Nabla-inter-E1-subset-Nabla1*: $\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$
and *Nabla-inter-E2-subset-Nabla2*: $\nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$
and *Upsilon-inter-E1-subset-Upsilon1*: $\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$
and *Upsilon-inter-E2-subset-Upsilon2*: $\Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$
and *Delta1-N1-Delta2-N2-subset-Delta*: $(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_{\Gamma}$

and *very-long-asm*: $(N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
$\vee$ $(N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2})$
and *FCI1*: *FCI* $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}
and *FCI2*: *FCI* $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2}

{
fix α β c v'
assume *c-in-Cv-inter-Upsilon*: $c \in (C_{\mathcal{V}} \cap \Upsilon_{\Gamma})$
and *v'-in-Vv-inter-Nabla*: $v' \in (V_{\mathcal{V}} \cap \nabla_{\Gamma})$
and *βv'α-in-Tr*: $(\beta \mathbin{@} [v'] \mathbin{@} \alpha) \in Tr_{(ES1 \parallel ES2)}$
and *αCv-empty*: $\alpha \restriction C_{\mathcal{V}} = []$

from *βv'α-in-Tr*
have *βv'α-E1-in-Tr1*: $(((\beta \mathbin{@} [v']) \mathbin{@} \alpha) \restriction E_{ES1}) \in Tr_{ES1}$
and *βv'α-E2-in-Tr2*: $(((\beta \mathbin{@} [v']) \mathbin{@} \alpha) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *composeES-def*)+

interpret *CSES1*: *CompositionSupport ES1* $\mathcal{V}$ $\mathcal{V}1$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES1 validV1*)

interpret *CSES2*: *CompositionSupport ES2* $\mathcal{V}$ $\mathcal{V}2$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES2 validV2*)

from *CSES1.BSD-in-subsystem2*[*OF βv'α-E1-in-Tr1 BSD1*] **obtain** $\alpha 1'$
where *βv'E1α1'-in-Tr1*: $(\beta \mathbin{@} [v']) \restriction E_{ES1} \mathbin{@} \alpha 1' \in Tr_{ES1}$
and *α1'Vv1-is-αVv1*: $\alpha 1' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and *α1'Cv1-empty*: $\alpha 1' \restriction C_{\mathcal{V}1} = []$
by *auto*

from *CSES2.BSD-in-subsystem2*[*OF βv'α-E2-in-Tr2 BSD2*] **obtain** $\alpha 2'$
where *βv'E2α2'-in-Tr2*: $(\beta \mathbin{@} [v']) \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2}$
and *α2'Vv2-is-αVv2*: $\alpha 2' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and *α2'Cv2-empty*: $\alpha 2' \restriction C_{\mathcal{V}2} = []$
by *auto*

note *very-long-asm*
moreover {
assume *Nv1-inter-Delta1-inter-E2-empty*: $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}$
and *Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1*: $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$

let *?ALPHA2''-DELTA2''* $= \exists\ \alpha 2''\ \delta 2''.$ (
set $\alpha 2'' \subseteq E_{ES2} \wedge$ *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \delta 2'' \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = [])$

from *c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV2*
have $c \notin E_{ES2} \vee (c \in E_{ES2} \wedge v' \notin E_{ES2}) \vee (c \in E_{ES2} \wedge v' \in E_{ES2})$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def)
moreover {

```
  assume c-notin-E2: c ∉ E_ES2

  from validES2 βv'E2α2'-in-Tr2 have set α2' ⊆ E_ES2
    by (simp add: ES-valid-def traces-contain-events-def, auto)
  moreover
  have set [] ⊆ N_V2 ∩ Δ_Γ2
    by auto
  moreover
  from βv'E2α2'-in-Tr2 c-notin-E2
  have β ↾ E_ES2 @ [c] ↾ E_ES2 @ [] @ [v'] ↾ E_ES2 @ α2' ∈ Tr_ES2
    by (simp add: projection-def)
  moreover
  have α2' ↾ V_V2 = α2' ↾ V_V2 ..
  moreover
  note α2'Cv2-empty
  ultimately have ?ALPHA2''-DELTA2''
    by blast
}
moreover {
  assume c-in-E2: c ∈ E_ES2
    and  v'-notin-E2: v' ∉ E_ES2

  from c-in-E2 c-in-Cv-inter-Upsilon propSepViews
    Upsilon-inter-E2-subset-Upsilon2
  have c-in-Cv2-inter-Upsilon2: c ∈ C_V2 ∩ ϒ_Γ2
    unfolding properSeparationOfViews-def by auto
  hence c ∈ C_V2
    by auto
  moreover
  from βv'E2α2'-in-Tr2 v'-notin-E2 have β ↾ E_ES2 @ α2' ∈ Tr_ES2
    by (simp add: projection-def)
  moreover
  note α2'Cv2-empty
  moreover
  have (Adm V2 ϱ2 Tr_ES2 (β ↾ E_ES2) c)
    proof −
      from validES2 βv'E2α2'-in-Tr2 v'-notin-E2 have β ↾ E_ES2 ∈ Tr_ES2
        by (simp add: ES-valid-def traces-prefixclosed-def
          prefixclosed-def prefix-def projection-concatenation-commute)
      with total-ES2-C2-inter-Upsilon2 c-in-Cv2-inter-Upsilon2
      have β ↾ E_ES2 @ [c] ∈ Tr_ES2
        by (simp add: total-def)
      thus ?thesis
        unfolding Adm-def
        by blast
    qed
  moreover
  note BSIA2
  ultimately obtain α2''
    where one: β ↾ E_ES2 @ [c] @ α2'' ∈ Tr_ES2
    and two:   α2'' ↾ V_V2 = α2' ↾ V_V2
    and three: α2'' ↾ C_V2 = []
```

unfolding *BSIA-def*
by *blast*

from *one validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
from *one c-in-E2 v'-notin-E2*
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ [] @ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *two three*
ultimately have *?ALPHA2''-DELTA2''*
by *blast*
}
moreover {
assume *c-in-E2*: $c \in E_{ES2}$
and *v'-in-E2*: $v' \in E_{ES2}$

from *c-in-E2 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E2-subset-Upsilon2
have *c-in-Cv2-inter-Upsilon2*: $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *v'-in-E2 propSepViews v'-in-Vv-inter-Nabla Nabla-inter-E2-subset-Nabla2*
have $v' \in V_{\mathcal{V}2} \cap Nabla\ \Gamma 2$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *v'-in-E2* $\beta v' E2 \alpha 2'$*-in-Tr2* **have** $\beta \restriction E_{ES2} @ [v'] @ \alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note $\alpha 2' Cv2\text{-}empty\ FCI2$
ultimately obtain $\alpha 2''\ \delta 2''$
where *one*: *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *two*: $\beta \restriction E_{ES2} @ [c] @ \delta 2'' @ [v'] @ \alpha 2'' \in Tr_{ES2}$
and *three*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *four*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *FCI-def*
by *blast*

from *two validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
note *one*
moreover
from *two c-in-E2 v'-in-E2*
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta 2'' @ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *three four*

ultimately have *?ALPHA2''-DELTA2''*
by *blast*
}
ultimately obtain $\alpha 2''\ \delta 2''$
where $\alpha 2''$*-in-E2star*: *set* $\alpha 2'' \subseteq E_{ES2}$
and $\delta 2''$*-in-N2-inter-Delta2star*:*set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2*:
$\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta 2'' @ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
and $\alpha 2''$*Vv2-is-*$\alpha 2'$*Vv2*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and $\alpha 2''$*Cv2-empty*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
by *blast*

from *c-in-Cv-inter-Upsilon Upsilon-inter-E1-subset-Upsilon1*
propSepViews
have *cE1-in-Cv1-inter-Upsilon1*: *set* $([c] \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-def*, *auto*)

from $\delta 2''$*-in-N2-inter-Delta2star Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1*
propSepViews disjoint-Nv2-Vv1
have $\delta 2''$*E1-in-Cv1-inter-Upsilon1star*: *set* $(\delta 2'' \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
proof −
from $\delta 2''$*-in-N2-inter-Delta2star*
have *eq*: $\delta 2'' \restriction E_{ES1} = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1})$
by (*metis Int-commute Int-left-commute Int-lower1 Int-lower2*
projection-intersection-neutral subset-trans)

from *validV1 Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1 propSepViews*
disjoint-Nv2-Vv1
have $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
unfolding *properSeparationOfViews-def*
by (*simp add*:*isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*subst eq*, *simp only*: *projection-def*, *auto*)
qed

have *cδ2''E1-in-Cv1-inter-Upsilon1star*: *set* $((c \# \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
proof −
from *cE1-in-Cv1-inter-Upsilon1* $\delta 2''$*E1-in-Cv1-inter-Upsilon1star*
have *set* $(([c] @ \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by *auto*
qed

have $\exists\ \alpha 1''\ \delta 1''.$ *set* $\alpha 1'' \subseteq E_{ES1}$
$\wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$ $\wedge\ \beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
proof *cases*

assume *v′-in-E1*: $v' \in E_{ES1}$
with *Nabla-inter-E1-subset-Nabla1 propSepViews v′-in-Vv-inter-Nabla*
have *v′-in-Vv1-inter-Nabla1*: $v' \in V_{\mathcal{V}1} \cap Nabla\ \Gamma 1$
unfolding *properSeparationOfViews-def* **by** *auto*

have $[\![(\beta\ @\ [v']) \restriction E_{ES1}\ @\ \alpha 1' \in Tr_{ES1}$;
$\alpha 1' \restriction C_{\mathcal{V}1} = []$; *set* $((c\ \#\ \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$;
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$; *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$ $]\!]$
$\Longrightarrow \exists\ \alpha 1''\ \delta 1''.$ (*set* $\alpha 1'' \subseteq E_{ES1} \wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\cup\ C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES1}\ @\ [c] \restriction E_{ES1}\ @\ \delta 1'' @ [v'] \restriction E_{ES1}\ @\ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = \delta 2'' \restriction E_{ES1}$)
proof (*induct length* $((c\ \#\ \delta 2'') \restriction E_{ES1})$ *arbitrary*: $\beta\ \alpha 1'\ c\ \delta 2''$)
case *0*

from *0(2) validES1* **have** *set* $\alpha 1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1}\ @\ [c] \restriction E_{ES1}\ @\ []\ @\ [v'] \restriction E_{ES1}\ @\ \alpha 1' \in Tr_{ES1}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha 1' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$ ..
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = \delta 2'' \restriction E_{ES1}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)
ultimately show *?case*
by *blast*
next
case (*Suc n*)

from *projection-split-last*[*OF Suc(2)*] **obtain** $\mu\ c'\ \nu$
where *c′-in-E1*: $c' \in E_{ES1}$
and *cδ2′′-is-μc′ν*: $c\ \#\ \delta 2'' = \mu\ @\ [c']\ @\ \nu$
and *νE1-empty*: $\nu \restriction E_{ES1} = []$
and *n-is-length-μνE1*: $n = length\ ((\mu\ @\ \nu) \restriction E_{ES1})$
by *blast*

from *Suc(5) c′-in-E1 cδ2′′-is-μc′ν*
have *set* $(\mu \restriction E_{ES1}\ @\ [c']) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$

by (*simp only*: *cδ2''-is-μc'ν projection-concatenation-commute projection-def*, *auto*)
hence *c'-in-Cv1-inter-Upsilon1*: $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by *auto*
hence *c'-in-Cv1*: $c' \in C_{\mathcal{V}1}$ **and** *c'-in-Upsilon1*: $c' \in \Upsilon_{\Gamma 1}$
by *auto*
with *validV1* **have** *c'-in-E1*: $c' \in E_{ES1}$
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)

show *?case*
proof (*cases* μ)
case *Nil*
with *cδ2''-is-μc'ν* **have** *c-is-c'*: $c = c'$ **and** *δ2''-is-ν*: $\delta 2'' = \nu$
by *auto*
with *c'-in-Cv1-inter-Upsilon1* **have** $c \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by *simp*
moreover
note *v'-in-Vv1-inter-Nabla1*
moreover
from *v'-in-E1 Suc(3)* **have** $(\beta \restriction E_{ES1}) @ [v'] @ \alpha 1' \in Tr_{ES1}$
by (*simp add*: *projection-concatenation-commute projection-def*)
moreover
note *Suc(4) FCI1*
ultimately obtain $\alpha 1''\ \gamma$
where *one*: *set* $\gamma \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *two*: $\beta \restriction E_{ES1} @ [c] @ \gamma @ [v'] @ \alpha 1'' \in Tr_{ES1}$
and *three*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *four*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCI-def*
by *blast*

let *?DELTA1''* $= \nu \restriction E_{ES1} @ \gamma$

from *two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *one νE1-empty*
have *set ?DELTA1''* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @$ *?DELTA1''* $@ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
proof −
from *c-is-c' c'-in-E1* **have** $[c] = [c] \restriction E_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *v'-in-E1* **have** $[v'] = [v'] \restriction E_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *νE1-empty two*
ultimately show *?thesis*

```
      by auto
    qed
  moreover
  note three four
  moreover
  have ?DELTA1'' ⌈ (C_V1 ∩ ϒ_Γ1) = δ2'' ⌈ E_ES1
    proof −
      have γ ⌈ (C_V1 ∩ ϒ_Γ1) = []
        proof −
          from validV1 have N_V1 ∩ Δ_Γ1 ∩ (C_V1 ∩ ϒ_Γ1) = {}
            by (simp add: isViewOn-def V-valid-def
              VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
          with projection-intersection-neutral[OF one, of C_V1 ∩ ϒ_Γ1]
          show ?thesis
            by (simp add: projection-def)
        qed
      with δ2''-is-ν νE1-empty show ?thesis
        by (simp add: projection-concatenation-commute)
    qed
  ultimately show ?thesis
    by blast
next
  case (Cons x xs)
  with cδ2''-is-μc'ν have μ-is-c-xs: μ = [c] @ xs
    and δ2''-is-xs-c'-ν: δ2'' = xs @ [c'] @ ν
    by auto
  with n-is-length-μνE1 have n = length ((c # (xs @ ν)) ⌈ E_ES1)
    by auto
  moreover
  note Suc(3,4)
  moreover
  have set ((c # (xs @ ν)) ⌈ E_ES1) ⊆ C_V1 ∩ ϒ_Γ1
    proof −
      have res: c # (xs @ ν) = [c] @ (xs @ ν)
        by auto

      from Suc(5) cδ2''-is-μc'ν μ-is-c-xs νE1-empty
      show ?thesis
        by (subst res, simp only: cδ2''-is-μc'ν projection-concatenation-commute
          set-append, auto)
    qed
  moreover
  note Suc(6)
  moreover
  from Suc(7) δ2''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V2 ∩ Δ_Γ2
    by auto
  moreover note Suc(1)[of c xs @ ν β α1']
  ultimately obtain δ γ
    where one: set δ ⊆ E_ES1
    and two: set γ ⊆ N_V1 ∩ Δ_Γ1 ∪ C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
    and three: β ⌈ E_ES1 @ [c] ⌈ E_ES1 @ γ @ [v'] ⌈ E_ES1 @ δ ∈ Tr_ES1
    and four: δ ⌈ V_V1 = α1' ⌈ V_V1
```

and *five*: $\delta \restriction C_{\mathcal{V}1} = []$
and *six*: $\gamma \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = (xs \mathbin{@} \nu) \restriction E_{ES1}$
by *blast*

let *?BETA* $= \beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} \gamma$

note *c′-in-Cv1-inter-Upsilon1 v′-in-Vv1-inter-Nabla1*
moreover
from *three v′-in-E1* **have** *?BETA* $\mathbin{@} [v'] \mathbin{@} \delta \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *five FCI1*
ultimately obtain $\alpha 1'' \, \delta'$
where *fci-one*: *set* $\delta' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *fci-two*: *?BETA* $\mathbin{@} [c'] \mathbin{@} \delta' \mathbin{@} [v'] \mathbin{@} \alpha 1'' \in Tr_{ES1}$
and *fci-three*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \delta \restriction V_{\mathcal{V}1}$
and *fci-four*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCI-def*
by *blast*

let *?DELTA1″* $= \gamma \mathbin{@} [c'] \mathbin{@} \delta'$

from *fci-two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set ?DELTA1″* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
proof −
from *Suc*(7) *c′-in-Cv1-inter-Upsilon1 δ2″-is-xs-c′-ν*
have $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
with *two fci-one* **show** *?thesis*
by *auto*
qed
moreover
from *fci-two v′-in-E1*
have $\beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@}$ *?DELTA1″* $\mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *fci-three four* **have** $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
by *simp*
moreover
note *fci-four*
moreover
have *?DELTA1″* $\restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = \delta 2'' \restriction E_{ES1}$
proof −
have $\delta' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = []$
proof −
from *fci-one* **have** $\forall \ e \in$ *set* $\delta'. \ e \in N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
with *validV1* **have** $\forall \ e \in$ *set* $\delta'. \ e \notin C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by (*simp add*: *isViewOn-def V-valid-def*

VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*simp add*: *projection-def*)
qed
with *c′-in-E1 c′-in-Cv1-inter-Upsilon1 δ2′′-is-xs-c′-ν νE1-empty six*
show *?thesis*
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
qed
ultimately show *?thesis*
by *blast*
qed
qed
from *this*[*OF βv′E1α1′-in-Tr1 α1′Cv1-empty cδ2′′E1-in-Cv1-inter-Upsilon1star*
c-in-Cv-inter-Upsilon δ2′′-in-N2-inter-Delta2star]
obtain $\alpha 1''\ \delta 1''$
where *one*: *set* $\alpha 1'' \subseteq E_{ES1}$
and *two*: *set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *three*: $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
and *four*: $\delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = \delta 2'' \restriction E_{ES1}$
by *blast*

note *one two three*
moreover
have $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
proof −
from *projection-intersection-neutral*[*OF two*, *of* E_{ES2}]
Nv1-inter-Delta1-inter-E2-empty validV2
have $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES2})$
by (*simp only*: *Int-Un-distrib2*, *auto*)
moreover
from *validV2*
have $C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES2} = C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *simp*
hence $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by (*simp add*: *projection-def*)
with *four* **have** $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1} \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *simp*
hence $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES1}$
by (*simp only*: *projection-commute*)
with *δ2′′-in-N2-inter-Delta2star* **show** *?thesis*
by (*simp only*: *list-subset-iff-projection-neutral*)
qed
ultimately show *?thesis*
by *blast*
next
assume *v′-notin-E1*: $v' \notin E_{ES1}$

have $[\![(\beta @ [v']) \restriction E_{ES1} @ \alpha 1' \in Tr_{ES1}$;

$\alpha 1' \restriction C_{\mathcal{V}1} = []$; *set* $((c \# \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$;
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$; *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$ ⟧
$\Longrightarrow \exists\ \alpha 1''\ \delta 1''.$ (*set* $\alpha 1'' \subseteq E_{ES1} \wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1}$
$\cap\ \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$ $\wedge\ \beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$)
proof (*induct length* $((c \# \delta 2'') \restriction E_{ES1})$ *arbitrary*: $\beta\ \alpha 1'\ c\ \delta 2''$)
case *0*

from *0(2) validES1* **have** *set* $\alpha 1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ [] @ [v'] \restriction E_{ES1} @ \alpha 1' \in Tr_{ES1}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha 1' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$ ..
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)
ultimately show *?case*
by *blast*
next
case (*Suc n*)

from *projection-split-last*[*OF Suc(2)*] **obtain** $\mu\ c'\ \nu$
where *c′-in-E1*: $c' \in E_{ES1}$
and *cδ2″-is-μc′ν*: $c \# \delta 2'' = \mu @ [c'] @ \nu$
and *νE1-empty*: $\nu \restriction E_{ES1} = []$
and *n-is-length-μνE1*: $n = length\ ((\mu @ \nu) \restriction E_{ES1})$
by *blast*

from *Suc(5) c′-in-E1 cδ2″-is-μc′ν*
have *set* $(\mu \restriction E_{ES1} @ [c']) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by (*simp only*: *cδ2″-is-μc′ν projection-concatenation-commute*
projection-def, *auto*)
hence *c′-in-Cv1-inter-Upsilon1*: $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by *auto*
hence *c′-in-Cv1*: $c' \in C_{\mathcal{V}1}$ **and** *c′-in-Upsilon1*: $c' \in \Upsilon_{\Gamma 1}$
by *auto*

```
with validV1 have c'-in-E1: c' ∈ E_ES1
  by (simp add: isViewOn-def V-valid-def
    VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)

show ?case
  proof (cases μ)
    case Nil
    with cδ2''-is-μc'ν have c-is-c': c = c'
      and δ2''-is-ν: δ2'' = ν
      by auto
    with c'-in-Cv1-inter-Upsilon1 have c ∈ C_V1
      by simp
    moreover
    from v'-notin-E1 Suc(3) have (β ↾ E_ES1) @ α1' ∈ Tr_ES1
      by (simp add: projection-concatenation-commute projection-def)
    moreover
    note Suc(4)
    moreover
    have Adm V1 ϱ1 Tr_ES1 (β ↾ E_ES1) c
      proof −
        have β ↾ E_ES1 @ [c] ∈ Tr_ES1
          proof −
            from c-is-c' c'-in-Cv1-inter-Upsilon1
            have c ∈ C_V1 ∩ ϒ_Γ1
              by simp
            moreover
            from validES1 Suc(3)
            have (β ↾ E_ES1) ∈ Tr_ES1
              by (simp only: ES-valid-def traces-prefixclosed-def
                projection-concatenation-commute
                prefixclosed-def prefix-def, auto)
            moreover
            note total-ES1-C1-inter-Upsilon1
            ultimately show ?thesis
              unfolding total-def
              by blast
          qed
        thus ?thesis
          unfolding Adm-def
          by blast
      qed
    moreover
    note BSIA1
    ultimately obtain α1''
      where one: (β ↾ E_ES1) @ [c] @ α1'' ∈ Tr_ES1
      and two: α1'' ↾ V_V1 = α1' ↾ V_V1
      and three: α1'' ↾ C_V1 = []
      unfolding BSIA-def
      by blast

    let ?DELTA1'' = ν ↾ E_ES1
```

from *one validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from ν*E1-empty*
have *set ?DELTA1''* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *simp*
moreover
from *c-is-c' c'-in-E1 one v'-notin-E1* ν*E1-empty*
have $(\beta \restriction E_{ES1})$ @ $[c] \restriction E_{ES1}$ @ *?DELTA1''* @ $[v'] \restriction E_{ES1}$ @ $\alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *two three*
moreover
from ν*E1-empty* $\delta 2''$*-is-*ν **have** *?DELTA1''* $\restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
by (*simp add*: *projection-def*)
ultimately show *?thesis*
by *blast*
next
case (*Cons x xs*)
with $c\delta 2''$*-is-*$\mu c' \nu$
have μ*-is-c-xs*: $\mu = [c]$ @ *xs* **and** $\delta 2''$*-is-xs-c'-*ν: $\delta 2'' = xs$ @ $[c']$ @ ν
by *auto*
with *n-is-length-*$\mu\nu$*E1* **have** $n = length\ ((c \# (xs$ @ $\nu)) \restriction E_{ES1})$
by *auto*
moreover
note *Suc*(*3*,*4*)
moreover
have *set* $((c \# (xs$ @ $\nu)) \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
proof −
have *res*: $c \# (xs$ @ $\nu) = [c]$ @ $(xs$ @ $\nu)$
by *auto*

from *Suc*(*5*) $c\delta 2''$*-is-*$\mu c' \nu$ μ*-is-c-xs* ν*E1-empty*
show *?thesis*
by (*subst res*, *simp only*: $c\delta 2''$*-is-*$\mu c' \nu$ *projection-concatenation-commute set-append*, *auto*)
qed
moreover
note *Suc*(*6*)
moreover
from *Suc*(*7*) $\delta 2''$*-is-xs-c'-*ν **have** *set* $(xs$ @ $\nu) \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover note *Suc*(*1*)[*of c xs* @ ν β $\alpha 1'$]
ultimately obtain δ γ
where *one*: *set* $\delta \subseteq E_{ES1}$
and *two*: *set* $\gamma \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *three*: $\beta \restriction E_{ES1}$ @ $[c] \restriction E_{ES1}$ @ γ @ $[v'] \restriction E_{ES1}$ @ $\delta \in Tr_{ES1}$
and *four*: $\delta \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *five*: $\delta \restriction C_{\mathcal{V}1} = []$
and *six*: $\gamma \restriction E_{ES2} = (xs$ @ $\nu) \restriction E_{ES1}$
by *blast*

let *?BETA* = $\beta \restriction E_{ES1}$ @ $[c] \restriction E_{ES1}$ @ γ

from *c′-in-Cv1-inter-Upsilon1* **have** $c' \in C_{\mathcal{V}1}$
by *auto*
moreover
from *three v′-notin-E1* **have** *?BETA* @ $\delta \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *five*
moreover
have *Adm* $\mathcal{V}1$ $\varrho 1$ Tr_{ES1} *?BETA* c'
proof −
have *?BETA* @ $[c'] \in Tr_{ES1}$
proof −
from *validES1 three*
have *?BETA* $\in Tr_{ES1}$
by (*simp only*: *ES-valid-def traces-prefixclosed-def*
projection-concatenation-commute
prefixclosed-def prefix-def, *auto*)
moreover
note *c′-in-Cv1-inter-Upsilon1 total-ES1-C1-inter-Upsilon1*
ultimately show *?thesis*
unfolding *total-def*
by *blast*
qed
thus *?thesis*
unfolding *Adm-def*
by *blast*
qed
moreover
note *BSIA1*
ultimately obtain $\alpha 1''$
where *bsia-one*: *?BETA* @ $[c']$ @ $\alpha 1'' \in Tr_{ES1}$
and *bsia-two*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \delta \restriction V_{\mathcal{V}1}$
and *bsia-three*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSIA-def*
by *blast*

let *?DELTA1″* = γ @ $[c']$

from *bsia-one validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*:*isViewOn-def ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set ?DELTA1″* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
proof −
from *Suc*(γ) *c′-in-Cv1-inter-Upsilon1* *δ2″-is-xs-c′-ν*
have $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
with *two* **show** *?thesis*
by *auto*
qed

moreover
from *bsia-one* v'*-notin-E1*
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ ?DELTA1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *bsia-two four* **have** $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
by *simp*
moreover
note *bsia-three*
moreover
have $?DELTA1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
proof −
from *validV2 Suc*(7) $\delta 2''$*-is-xs-c'-*ν
have $c' \in E_{ES2}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
with c'*-in-E1* c'*-in-Cv1-inter-Upsilon1* $\delta 2''$*-is-xs-c'-*ν ν*E1-empty six*
show *?thesis*
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
qed
ultimately show *?thesis*
by *blast*
qed
qed
from *this*[*OF* $\beta v'E1 \alpha 1'$*-in-Tr1* $\alpha 1'$*Cv1-empty* $c\delta 2''$*E1-in-Cv1-inter-Upsilon1star*
c-in-Cv-inter-Upsilon $\delta 2''$*-in-N2-inter-Delta2star*]
show *?thesis*
by *blast*
qed
then obtain $\alpha 1'' \delta 1''$
where $\alpha 1''$*-in-E1star*: *set* $\alpha 1'' \subseteq E_{ES1}$
and $\delta 1''$*-in-N1-inter-Delta1star*:*set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1*:
$\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
and $\alpha 1''$*Vv1-is-*$\alpha 1'$*Vv1*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and $\alpha 1''$*Cv1-empty*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
and $\delta 1''$*E2-is-*$\delta 2''$*E1*: $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
by *blast*

from β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1* β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2*
validES1 validES2
have $\delta 1''$*-in-E1star*: *set* $\delta 1'' \subseteq E_{ES1}$ **and** $\delta 2''$*-in-E2star*: *set* $\delta 2'' \subseteq E_{ES2}$
by (*simp-all add*: *ES-valid-def traces-contain-events-def*, *auto*)
with $\delta 1''$*E2-is-*$\delta 2''$*E1 merge-property*[*of* $\delta 1'' E_{ES1} \delta 2'' E_{ES2}$] **obtain** δ'
where δ'*E1-is-*$\delta 1''$: $\delta' \restriction E_{ES1} = \delta 1''$
and δ'*E2-is-*$\delta 2''$: $\delta' \restriction E_{ES2} = \delta 2''$
and δ'*-contains-only-*$\delta 1''$*-*$\delta 2''$*-events*: *set* $\delta' \subseteq$ *set* $\delta 1'' \cup$ *set* $\delta 2''$
unfolding *Let-def*
by *auto*

let $?TAU = \beta @ [c] @ \delta' @ [v']$
let $?LAMBDA = \alpha \restriction V_{\mathcal{V}}$

let *?T1* = $\alpha 1''$
let *?T2* = $\alpha 2''$

have *?TAU* $\in Tr_{(ES1 \parallel ES2)}$
proof −
from β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1* δ'*E1-is-*$\delta 1''$ *validES1*
have $\beta \restriction E_{ES1}$ @ $[c] \restriction E_{ES1}$ @ $\delta' \restriction E_{ES1}$ @ $[v'] \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence $(\beta$ @ $[c]$ @ δ' @ $[v']) \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *projection-def*, *auto*)
moreover
from β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2* δ'*E2-is-*$\delta 2''$ *validES2*
have $\beta \restriction E_{ES2}$ @ $[c] \restriction E_{ES2}$ @ $\delta' \restriction E_{ES2}$ @ $[v'] \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence $(\beta$ @ $[c]$ @ δ' @ $[v']) \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta v'\alpha$*-in-Tr c-in-Cv-inter-Upsilon VIsViewOnE* δ'*-contains-only-*$\delta 1''$*-*$\delta 2''$*-events* $\delta 1''$*-in-E1star* $\delta 2''$*-in-E2star*
have *set* $(\beta$ @ $[c]$ @ δ' @ $[v']) \subseteq E_{ES1} \cup E_{ES2}$
unfolding *composeES-def isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*
by *auto*
ultimately show *?thesis*
unfolding *composeES-def*
by *auto*
qed
hence *set ?TAU* $\subseteq E_{(ES1 \parallel ES2)}$
unfolding *composeES-def*
by *auto*
moreover
have *set ?LAMBDA* $\subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
note $\alpha 1''$*-in-E1star* $\alpha 2''$*-in-E2star*
moreover
from β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1* δ'*E1-is-*$\delta 1''$
have *?TAU* $\restriction E_{ES1}$ @ *?T1* $\in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
from β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2* δ'*E2-is-*$\delta 2''$
have *?TAU* $\restriction E_{ES2}$ @ *?T2* $\in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
have *?LAMBDA* $\restriction E_{ES1} =$ *?T1* $\restriction V_{\mathcal{V}}$
proof −
from *propSepViews* **have** *?LAMBDA* $\restriction E_{ES1} = \alpha \restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-sequence*)
moreover

from $\alpha 1''$*-in-E1star propSepViews*
have *?T1* $\restriction V_{\mathcal{V}} =$ *?T1* $\restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note $\alpha 1'$*Vv1-is-*α*Vv1* $\alpha 1''$*Vv1-is-*$\alpha 1'$*Vv1*
ultimately show *?thesis*
by *simp*
qed
moreover
have *?LAMBDA* $\restriction E_{ES2} =$ *?T2* $\restriction V_{\mathcal{V}}$
proof −
from *propSepViews*
have *?LAMBDA* $\restriction E_{ES2} = \alpha \restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-sequence*)
moreover
from $\alpha 2''$*-in-E2star propSepViews*
have *?T2* $\restriction V_{\mathcal{V}} =$ *?T2* $\restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note $\alpha 2'$*Vv2-is-*α*Vv2* $\alpha 2''$*Vv2-is-*$\alpha 2'$*Vv2*
ultimately show *?thesis*
by *simp*
qed
moreover
note $\alpha 1''$*Cv1-empty* $\alpha 2''$*Cv2-empty generalized-zipping-lemma*
ultimately obtain t
where *?TAU* @ $t \in Tr_{(ES1 \parallel ES2)}$
and $t \restriction V_{\mathcal{V}} =$ *?LAMBDA*
and $t \restriction C_{\mathcal{V}} = []$
by *blast*
moreover
have *set* $\delta' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma}$
proof −
from δ'*-contains-only-*$\delta 1''$*-*$\delta 2''$*-events*
$\delta 1''$*-in-N1-inter-Delta1star* $\delta 2''$*-in-N2-inter-Delta2star*
have *set* $\delta' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
with *Delta1-N1-Delta2-N2-subset-Delta Nv1-union-Nv2-subsetof-Nv*
show *?thesis*
by *auto*
qed
ultimately
have $\exists \alpha'\ \gamma'.$ (*set* $\gamma' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge \beta$ @ $[c]$ @ γ' @ $[v']$ @ $\alpha' \in Tr_{(ES1 \parallel ES2)}$
$\wedge\ \alpha' \restriction V_{\mathcal{V}} = \alpha \restriction V_{\mathcal{V}} \wedge \alpha' \restriction C_{\mathcal{V}} = []$)
by (*simp only*: *append-assoc*, *blast*)
}
moreover {
assume *Nv2-inter-Delta2-inter-E1-empty*: $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}$
and *Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2*: $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$

let *?ALPHA1''-DELTA1'' = ∃ α1'' δ1''. (*
set α1'' ⊆ E_ES1 ∧ set δ1'' ⊆ N_V1 ∩ Δ_Γ1
∧ β ↾ E_ES1 @ [c] ↾ E_ES1 @ δ1'' @ [v'] ↾ E_ES1 @ α1'' ∈ Tr_ES1
∧ α1'' ↾ V_V1 = α1' ↾ V_V1 ∧ α1'' ↾ C_V1 = [])

from *c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV1*
have *c ∉ E_ES1 ∨ (c ∈ E_ES1 ∧ v' ∉ E_ES1) ∨ (c ∈ E_ES1 ∧ v' ∈ E_ES1)*
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def)
moreover {
assume *c-notin-E1*: *c ∉ E_ES1*

from *validES1 βv'E1α1'-in-Tr1* **have** *set α1' ⊆ E_ES1*
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set [] ⊆ N_V1 ∩ Δ_Γ1*
by *auto*
moreover
from *βv'E1α1'-in-Tr1 c-notin-E1*
have *β ↾ E_ES1 @ [c] ↾ E_ES1 @ [] @ [v'] ↾ E_ES1 @ α1' ∈ Tr_ES1*
by (*simp add*: *projection-def*)
moreover
have *α1' ↾ V_V1 = α1' ↾ V_V1* **..**
moreover
note *α1'Cv1-empty*
ultimately have *?ALPHA1''-DELTA1''*
by *blast*
}
moreover {
assume *c-in-E1*: *c ∈ E_ES1*
and *v'-notin-E1*: *v' ∉ E_ES1*

from *c-in-E1 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E1-subset-Upsilon1
have *c-in-Cv1-inter-Upsilon1*: *c ∈ C_V1 ∩ ϒ_Γ1*
unfolding *properSeparationOfViews-def* **by** *auto*
hence *c ∈ C_V1*
by *auto*
moreover
from *βv'E1α1'-in-Tr1 v'-notin-E1* **have** *β ↾ E_ES1 @ α1' ∈ Tr_ES1*
by (*simp add*: *projection-def*)
moreover
note *α1'Cv1-empty*
moreover
have (*Adm V1 ϱ1 Tr_ES1 (β ↾ E_ES1) c*)
proof −
from *validES1 βv'E1α1'-in-Tr1 v'-notin-E1* **have** *β ↾ E_ES1 ∈ Tr_ES1*
by (*simp add*: *ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def projection-concatenation-commute)
with *total-ES1-C1-inter-Upsilon1 c-in-Cv1-inter-Upsilon1*
have *β ↾ E_ES1 @ [c] ∈ Tr_ES1*
by (*simp add*: *total-def*)

```
      thus ?thesis
        unfolding Adm-def
        by blast
    qed
  moreover
  note BSIA1
  ultimately obtain α1''
    where one: β ↑ E_ES1 @ [c] @ α1'' ∈ Tr_ES1
    and two:   α1'' ↑ V_V1 = α1' ↑ V_V1
    and three: α1'' ↑ C_V1 = []
    unfolding BSIA-def
    by blast

  from one validES1 have set α1'' ⊆ E_ES1
    by (simp add: ES-valid-def traces-contain-events-def, auto)
  moreover
  have set [] ⊆ N_V1 ∩ Δ_Γ1
    by auto
  moreover
  from one c-in-E1 v'-notin-E1
  have β ↑ E_ES1 @ [c] ↑ E_ES1 @ [] @ [v'] ↑ E_ES1 @ α1'' ∈ Tr_ES1
    by (simp add: projection-def)
  moreover
  note two three
  ultimately have ?ALPHA1''-DELTA1''
    by blast
}
moreover {
  assume c-in-E1: c ∈ E_ES1
    and  v'-in-E1: v' ∈ E_ES1

  from c-in-E1 c-in-Cv-inter-Upsilon propSepViews
    Upsilon-inter-E1-subset-Upsilon1
  have c-in-Cv1-inter-Upsilon1: c ∈ C_V1 ∩ ϒ_Γ1
    unfolding properSeparationOfViews-def by auto
  moreover
  from v'-in-E1 propSepViews v'-in-Vv-inter-Nabla Nabla-inter-E1-subset-Nabla1
  have v' ∈ V_V1 ∩ Nabla Γ1
   unfolding properSeparationOfViews-def by auto
  moreover
  from v'-in-E1  βv'E1α1'-in-Tr1 have β ↑ E_ES1 @ [v'] @ α1' ∈ Tr_ES1
    by (simp add: projection-def)
  moreover
  note α1'Cv1-empty FCI1
  ultimately obtain α1'' δ1''
    where one: set δ1'' ⊆ N_V1 ∩ Δ_Γ1
    and two: β ↑ E_ES1 @ [c] @ δ1'' @ [v'] @ α1'' ∈ Tr_ES1
    and three: α1'' ↑ V_V1 = α1' ↑ V_V1
    and four: α1'' ↑ C_V1 = []
    unfolding FCI-def
    by blast
```

from *two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
note *one*
moreover
from *two c-in-E1 v'-in-E1*
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *three four*
ultimately have *?ALPHA1''-DELTA1''*
by *blast*
}
ultimately obtain $\alpha 1'' \delta 1''$
where *α1''-in-E1star*: *set* $\alpha 1'' \subseteq E_{ES1}$
and *δ1''-in-N1-inter-Delta1star*:*set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *βE1-cE1-δ1''-v'E1-α1''-in-Tr1*:
$\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
and *α1''Vv1-is-α1'Vv1*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *α1''Cv1-empty*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
by *blast*

from *c-in-Cv-inter-Upsilon Upsilon-inter-E2-subset-Upsilon2 propSepViews*
have *cE2-in-Cv2-inter-Upsilon2*: *set* $([c] \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-def*, *auto*)

from *δ1''-in-N1-inter-Delta1star Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2*
propSepViews disjoint-Nv1-Vv2
have *δ1''E2-in-Cv2-inter-Upsilon2star*: *set* $(\delta 1'' \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
proof −
from *δ1''-in-N1-inter-Delta1star* **have** *eq*: $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2})$
by (*metis Int-commute Int-left-commute Int-lower2 Int-lower1*
projection-intersection-neutral subset-trans)

from *validV2 Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2*
propSepViews disjoint-Nv1-Vv2
have $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*subst eq*, *simp only*: *projection-def*, *auto*)
qed

have *cδ1''E2-in-Cv2-inter-Upsilon2star*: *set* $((c \# \delta 1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
proof −
from *cE2-in-Cv2-inter-Upsilon2 δ1''E2-in-Cv2-inter-Upsilon2star*
have *set* $(([c] @ \delta 1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by *auto*
qed

have $\exists\ \alpha2''\ \delta2''.\ set\ \alpha2'' \subseteq E_{ES2}$
$\wedge\ set\ \delta2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge\ \beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta2'' @ [v'] \restriction E_{ES2} @ \alpha2'' \in Tr_{ES2}$
$\wedge\ \alpha2'' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2} \wedge \alpha2'' \restriction C_{\mathcal{V}2} = []$
$\wedge\ \delta2'' \restriction E_{ES1} = \delta1'' \restriction E_{ES2}$
proof *cases*
assume *v'-in-E2*: $v' \in E_{ES2}$
with *Nabla-inter-E2-subset-Nabla2*
propSepViews v'-in-Vv-inter-Nabla
have *v'-in-Vv2-inter-Nabla2*: $v' \in V_{\mathcal{V}2} \cap Nabla\ \Gamma 2$
unfolding *properSeparationOfViews-def* **by** *auto*

have $[\![\ (\beta @ [v']) \restriction E_{ES2} @ \alpha2' \in Tr_{ES2}\ ;$
$\alpha2' \restriction C_{\mathcal{V}2} = []$; $set\ ((c \#\ \delta1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}\ ;$
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}\ ;\ set\ \delta1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}\ ]\!]$
$\Longrightarrow \exists\ \alpha2''\ \delta2''.\ (set\ \alpha2'' \subseteq E_{ES2} \wedge set\ \delta2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2}$
$\cap\ \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge\ \beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta2'' @ [v'] \restriction E_{ES2} @ \alpha2'' \in Tr_{ES2}$
$\wedge\ \alpha2'' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2} \wedge \alpha2'' \restriction C_{\mathcal{V}2} = []$
$\wedge\ \delta2'' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \delta1'' \restriction E_{ES2})$
proof (*induct length* $((c \#\ \delta1'') \restriction E_{ES2})$ *arbitrary*: $\beta\ \alpha2'\ c\ \delta1''$)
case *0*

from *0(2) validES2* **have** $set\ \alpha2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have $set\ [] \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ [] @ [v'] \restriction E_{ES2} @ \alpha2' \in Tr_{ES2}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES2}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha2' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2}$ **..**
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \delta1'' \restriction E_{ES2}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)
ultimately show *?case*
by *blast*
next
case (*Suc n*)

from *projection-split-last*[*OF Suc*(2)] **obtain** $\mu\ c'\ \nu$
  where *c'-in-E2*: $c' \in E_{ES2}$
  and *cδ1''-is-μc'ν*: $c\ \#\ \delta 1'' = \mu\ @\ [c']\ @\ \nu$
  and *νE2-empty*: $\nu \restriction E_{ES2} = []$
  and *n-is-length-μνE2*: $n = length\ ((\mu\ @\ \nu) \restriction E_{ES2})$
  by *blast*

from *Suc*(5) *c'-in-E2 cδ1''-is-μc'ν*
have *set* $(\mu \restriction E_{ES2}\ @\ [c']) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
  by (*simp only*: *cδ1''-is-μc'ν projection-concatenation-commute projection-def*, *auto*)
hence *c'-in-Cv2-inter-Upsilon2*: $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
  by *auto*
hence *c'-in-Cv2*: $c' \in C_{\mathcal{V}2}$ **and** *c'-in-Upsilon2*: $c' \in \Upsilon_{\Gamma 2}$
  by *auto*
with *validV2* **have** *c'-in-E2*: $c' \in E_{ES2}$
  by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)

show *?case*
  proof (*cases* μ)
    case *Nil*
    with *cδ1''-is-μc'ν* **have** *c-is-c'*: $c = c'$ **and** *δ1''-is-ν*: $\delta 1'' = \nu$
      by *auto*
    with *c'-in-Cv2-inter-Upsilon2* **have** $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
      by *simp*
    moreover
    note *v'-in-Vv2-inter-Nabla2*
    moreover
    from *v'-in-E2 Suc*(3) **have** $(\beta \restriction E_{ES2})\ @\ [v']\ @\ \alpha 2' \in Tr_{ES2}$
      by (*simp add*: *projection-concatenation-commute projection-def*)
    moreover
    note *Suc*(4) *FCI2*
    ultimately obtain $\alpha 2''\ \gamma$
      where *one*: *set* $\gamma \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
      and *two*: $\beta \restriction E_{ES2}\ @\ [c]\ @\ \gamma\ @\ [v']\ @\ \alpha 2'' \in Tr_{ES2}$
      and *three*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
      and *four*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
      unfolding *FCI-def*
      by *blast*

    let *?DELTA2''* $= \nu \restriction E_{ES2}\ @\ \gamma$

    from *two validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
      by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
    moreover
    from *one νE2-empty*
    have *set ?DELTA2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
      by *auto*
    moreover
    have $\beta \restriction E_{ES2}\ @\ [c] \restriction E_{ES2}\ @\ ?DELTA2''\ @\ [v'] \restriction E_{ES2}\ @\ \alpha 2'' \in Tr_{ES2}$

```
    proof −
      from c-is-c′ c′-in-E2 have [c] = [c] ↾ E_ES2
        by (simp add: projection-def)
      moreover
      from v′-in-E2 have [v′] = [v′] ↾ E_ES2
        by (simp add: projection-def)
      moreover
      note νE2-empty two
      ultimately show ?thesis
        by auto
    qed
  moreover
  note three four
  moreover
  have ?DELTA2'' ↾ (C_V2 ∩ ϒ_Γ2) = δ1'' ↾ E_ES2
    proof −
      have γ ↾ (C_V2 ∩ ϒ_Γ2) = []
        proof −
          from validV2 have N_V2 ∩ Δ_Γ2 ∩ (C_V2 ∩ ϒ_Γ2) = {}
            by (simp add: isViewOn-def V-valid-def
              VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
          with projection-intersection-neutral[OF one, of C_V2 ∩ ϒ_Γ2]
          show ?thesis
            by (simp add: projection-def)
        qed
      with δ1''-is-ν νE2-empty show ?thesis
        by (simp add: projection-concatenation-commute)
    qed
  ultimately show ?thesis
    by blast
next
  case (Cons x xs)
  with cδ1''-is-μc′ν have μ-is-c-xs: μ = [c] @ xs
    and δ1''-is-xs-c′-ν: δ1'' = xs @ [c′] @ ν
    by auto
  with n-is-length-μνE2 have n = length ((c # (xs @ ν)) ↾ E_ES2)
    by auto
  moreover
  note Suc(3,4)
  moreover
  have set ((c # (xs @ ν)) ↾ E_ES2) ⊆ C_V2 ∩ ϒ_Γ2
    proof −
      have res: c # (xs @ ν) = [c] @ (xs @ ν)
        by auto

      from Suc(5) cδ1''-is-μc′ν μ-is-c-xs νE2-empty
      show ?thesis
        by (subst res, simp only: cδ1''-is-μc′ν
          projection-concatenation-commute set-append, auto)
    qed
  moreover
  note Suc(6)
```

moreover
from *Suc*(7) *δ1''-is-xs-c'-ν* **have** *set* $(xs \mathbin{@} \nu) \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover note *Suc*(1)[*of c xs* @ *ν β α2'*]
ultimately obtain $\delta\ \gamma$
where *one*: *set* $\delta \subseteq E_{ES2}$
and *two*: *set* $\gamma \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *three*: $\beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \gamma \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \delta \in Tr_{ES2}$
and *four*: $\delta \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *five*: $\delta \restriction C_{\mathcal{V}2} = []$
and *six*: $\gamma \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = (xs \mathbin{@} \nu) \restriction E_{ES2}$
by *blast*

let *?BETA* $= \beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \gamma$

note *c'-in-Cv2-inter-Upsilon2 v'-in-Vv2-inter-Nabla2*
moreover
from *three v'-in-E2* **have** *?BETA* $@ [v'] @ \delta \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *five FCI2*
ultimately obtain $\alpha 2''\ \delta'$
where *fci-one*: *set* $\delta' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *fci-two*: *?BETA* $@ [c'] @ \delta' @ [v'] @ \alpha 2'' \in Tr_{ES2}$
and *fci-three*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \delta \restriction V_{\mathcal{V}2}$
and *fci-four*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *FCI-def*
by *blast*

let *?DELTA2''* $= \gamma @ [c'] @ \delta'$

from *fci-two validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set ?DELTA2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
proof −
from *Suc*(7) *c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν*
have $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
with *two fci-one* **show** *?thesis*
by *auto*
qed
moreover
from *fci-two v'-in-E2*
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @$ *?DELTA2''* $@ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
from *fci-three four* **have** $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
by *simp*
moreover
note *fci-four*

```
        moreover
        have ?DELTA2'' ⌊ (C_V2 ∩ Υ_Γ2) = δ1'' ⌊ E_ES2
          proof −
            have δ' ⌊ (C_V2 ∩ Υ_Γ2) = []
              proof −
                from fci-one have ∀ e ∈ set δ'. e ∈ N_V2 ∩ Δ_Γ2
                  by auto
                with validV2 have ∀ e ∈ set δ'. e ∉ C_V2 ∩ Υ_Γ2
                  by (simp add: isViewOn-def V-valid-def
                    VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
                thus ?thesis
                  by (simp add: projection-def)
              qed
            with c'-in-E2 c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν νE2-empty six
            show ?thesis
              by (simp only: projection-concatenation-commute projection-def, auto)
          qed
        ultimately show ?thesis
          by blast
      qed
qed
from this[OF βv'E2α2'-in-Tr2 α2'Cv2-empty cδ1''E2-in-Cv2-inter-Upsilon2star
  c-in-Cv-inter-Upsilon δ1''-in-N1-inter-Delta1star]
obtain α2'' δ2''
  where one: set α2'' ⊆ E_ES2
  and two: set δ2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
  and three: β ⌊ E_ES2 @ [c] ⌊ E_ES2 @ δ2'' @ [v'] ⌊ E_ES2 @ α2'' ∈ Tr_ES2
  ∧ α2'' ⌊ V_V2 = α2' ⌊ V_V2 ∧ α2'' ⌊ C_V2 = []
  and four: δ2'' ⌊ (C_V2 ∩ Υ_Γ2) = δ1'' ⌊ E_ES2
  by blast

note one two three
moreover
have δ2'' ⌊ E_ES1 = δ1'' ⌊ E_ES2
  proof −
    from projection-intersection-neutral[OF two, of E_ES1]
      Nv2-inter-Delta2-inter-E1-empty validV1
    have δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1 ∩ E_ES1)
      by (simp only: Int-Un-distrib2, auto)
    moreover
    from validV1
    have C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1 ∩ E_ES1 = C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def
        VN-disjoint-def NC-disjoint-def, auto)
    ultimately have δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1)
      by simp
    hence δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2) ⌊ (N_V1 ∩ Δ_Γ1)
      by (simp add: projection-def)
    with four have δ2'' ⌊ E_ES1 = δ1'' ⌊ E_ES2 ⌊ (N_V1 ∩ Δ_Γ1)
      by simp
    hence δ2'' ⌊ E_ES1 = δ1'' ⌊ (N_V1 ∩ Δ_Γ1) ⌊ E_ES2
      by (simp only: projection-commute)
```

with *δ1′′-in-N1-inter-Delta1star* **show** *?thesis*
by (*simp only*: *list-subset-iff-projection-neutral*)
qed
ultimately show *?thesis*
by *blast*
next
assume *v′-notin-E2*: $v' \notin E_{ES2}$

have
$[\![(\beta \mathbin{@} [v']) \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2} ; \alpha 2' \restriction C_{\mathcal{V}2} = [];$
set $((c \# \delta 1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$; $c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$;
set $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$ $]\!]$
$\Longrightarrow \exists\ \alpha 2''\ \delta 2''.$
$(set\ \alpha 2'' \subseteq E_{ES2} \wedge set\ \delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge\ \beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \delta 2'' \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = []$
$\wedge\ \delta 2'' \restriction E_{ES1} = \delta 1'' \restriction E_{ES2})$
proof (*induct length* $((c \# \delta 1'') \restriction E_{ES2})$ *arbitrary*: $\beta\ \alpha 2'\ c\ \delta 1''$)
case *0*

from *0(2) validES2* **have** *set* $\alpha 2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
have $\beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} [] \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES2}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha 2' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$ **..**
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction E_{ES1} = \delta 1'' \restriction E_{ES2}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)
ultimately show *?case*
by *blast*
next
case (*Suc n*)

from *projection-split-last*[*OF Suc(2)*] **obtain** $\mu\ c'\ \nu$
where *c′-in-E2*: $c' \in E_{ES2}$
and *cδ1′′-is-μc′ν*: $c \# \delta 1'' = \mu \mathbin{@} [c'] \mathbin{@} \nu$
and *νE2-empty*: $\nu \restriction E_{ES2} = []$
and *n-is-length-μνE2*: $n = length\ ((\mu \mathbin{@} \nu) \restriction E_{ES2})$

by *blast*

from *Suc(5)* *c′-in-E2* *cδ1′′-is-μc′ν* **have** *set* $(\mu \restriction E_{ES2} @ [c']) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by (*simp only*: *cδ1′′-is-μc′ν* *projection-concatenation-commute* *projection-def*, *auto*)
hence *c′-in-Cv2-inter-Upsilon2*: $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by *auto*
hence *c′-in-Cv2*: $c' \in C_{\mathcal{V}2}$ **and** *c′-in-Upsilon2*: $c' \in \Upsilon_{\Gamma 2}$
by *auto*
with *validV2* **have** *c′-in-E2*: $c' \in E_{ES2}$
by (*simp add*: *isViewOn-def* *V-valid-def* *VC-disjoint-def*
VN-disjoint-def *NC-disjoint-def*, *auto*)

show *?case*
proof (*cases* μ)
case *Nil*
with *cδ1′′-is-μc′ν* **have** *c-is-c′*: $c = c'$ **and** *δ1′′-is-ν*: $\delta 1'' = \nu$
by *auto*
with *c′-in-Cv2-inter-Upsilon2* **have** $c \in C_{\mathcal{V}2}$
by *simp*
moreover
from *v′-notin-E2* *Suc(3)* **have** $(\beta \restriction E_{ES2}) @ \alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-concatenation-commute* *projection-def*)
moreover
note *Suc(4)*
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $(\beta \restriction E_{ES2})$ c
proof −
have $\beta \restriction E_{ES2} @ [c] \in Tr_{ES2}$
proof −
from *c-is-c′* *c′-in-Cv2-inter-Upsilon2* **have** $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by *simp*
moreover
from *validES2* *Suc(3)* **have** $(\beta \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp only*: *ES-valid-def* *traces-prefixclosed-def*
projection-concatenation-commute
prefixclosed-def *prefix-def*, *auto*)
moreover
note *total-ES2-C2-inter-Upsilon2*
ultimately show *?thesis*
unfolding *total-def*
by *blast*
qed
thus *?thesis*
unfolding *Adm-def*
by *blast*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *one*: $(\beta \restriction E_{ES2}) @ [c] @ \alpha 2'' \in Tr_{ES2}$
and *two*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *three*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$

```
    unfolding BSIA-def
    by blast

  let ?DELTA2'' = ν ↑ E_ES2

  from one validES2 have set α2'' ⊆ E_ES2
    by (simp add: ES-valid-def traces-contain-events-def, auto)
  moreover
  from νE2-empty
  have set ?DELTA2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
    by simp
  moreover
  from c-is-c' c'-in-E2 one v'-notin-E2 νE2-empty
  have (β ↑ E_ES2) @ [c] ↑ E_ES2 @ ?DELTA2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
    by (simp add: projection-def)
  moreover
  note two three
  moreover
  from νE2-empty δ1''-is-ν have ?DELTA2'' ↑ E_ES1 = δ1'' ↑ E_ES2
    by (simp add: projection-def)
  ultimately show ?thesis
    by blast
next
  case (Cons x xs)
   with cδ1''-is-μc'ν have μ-is-c-xs: μ = [c] @ xs
    and δ1''-is-xs-c'-ν: δ1'' = xs @ [c'] @ ν
    by auto
  with n-is-length-μνE2 have n = length ((c # (xs @ ν)) ↑ E_ES2)
    by auto
  moreover
  note Suc(3,4)
  moreover
  have set ((c # (xs @ ν)) ↑ E_ES2) ⊆ C_V2 ∩ ϒ_Γ2
    proof −
      have res: c # (xs @ ν) = [c] @ (xs @ ν)
        by auto

      from Suc(5) cδ1''-is-μc'ν μ-is-c-xs νE2-empty
      show ?thesis
        by (subst res, simp only: cδ1''-is-μc'ν projection-concatenation-commute
          set-append, auto)
    qed
  moreover
  note Suc(6)
  moreover
  from Suc(7) δ1''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V1 ∩ Δ_Γ1
    by auto
  moreover note Suc(1)[of c xs @ ν β α2']
  ultimately obtain δ γ
    where one: set δ ⊆ E_ES2
    and two: set γ ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
    and three: β ↑ E_ES2 @ [c] ↑ E_ES2 @ γ @ [v'] ↑ E_ES2 @ δ ∈ Tr_ES2
```

and *four*: $\delta \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *five*: $\delta \restriction C_{\mathcal{V}2} = []$
and *six*: $\gamma \restriction E_{ES1} = (xs \mathbin{@} \nu) \restriction E_{ES2}$
by *blast*

let *?BETA* $= \beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \gamma$

from *c'-in-Cv2-inter-Upsilon2* **have** $c' \in C_{\mathcal{V}2}$
by *auto*
moreover
from *three v'-notin-E2* **have** *?BETA* $@ \delta \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *five*
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} *?BETA* c'
proof −
have *?BETA* $@ [c'] \in Tr_{ES2}$
proof −
from *validES2 three* **have** *?BETA* $\in Tr_{ES2}$
by (*simp only*: *ES-valid-def traces-prefixclosed-def*
projection-concatenation-commute prefixclosed-def prefix-def, *auto*)
moreover
note *c'-in-Cv2-inter-Upsilon2 total-ES2-C2-inter-Upsilon2*
ultimately show *?thesis*
unfolding *total-def*
by *blast*
qed
thus *?thesis*
unfolding *Adm-def*
by *blast*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *bsia-one*: *?BETA* $@ [c'] @ \alpha 2'' \in Tr_{ES2}$
and *bsia-two*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \delta \restriction V_{\mathcal{V}2}$
and *bsia-three*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSIA-def*
by *blast*

let *?DELTA2''* $= \gamma @ [c']$

from *bsia-one validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set ?DELTA2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
proof −
from *Suc*(7) *c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν*
have $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*

```
            with two show ?thesis
              by auto
          qed
        moreover
        from bsia-one v'-notin-E2
        have β ↑ E_ES2 @ [c] ↑ E_ES2 @ ?DELTA2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
          by (simp add: projection-def)
        moreover
        from bsia-two four have α2'' ↑ V_V2 = α2' ↑ V_V2
          by simp
        moreover
        note bsia-three
        moreover
        have ?DELTA2'' ↑ E_ES1 = δ1'' ↑ E_ES2
          proof −
            from validV1 Suc(7) δ1''-is-xs-c'-ν have c' ∈ E_ES1
              by (simp add: isViewOn-def V-valid-def
                VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
            with c'-in-E2 c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν νE2-empty six
            show ?thesis
              by (simp only: projection-concatenation-commute
                projection-def, auto)
          qed
        ultimately show ?thesis
          by blast
      qed
    qed
  from this[OF βv'E2α2'-in-Tr2 α2'Cv2-empty cδ1''E2-in-Cv2-inter-Upsilon2star
    c-in-Cv-inter-Upsilon δ1''-in-N1-inter-Delta1star]
  show ?thesis
    by blast
qed
then obtain α2'' δ2''
  where α2''-in-E2star: set α2'' ⊆ E_ES2
  and δ2''-in-N2-inter-Delta2star:set δ2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
  and βE2-cE2-δ2''-v'E2-α2''-in-Tr2:
  β ↑ E_ES2 @ [c] ↑ E_ES2 @ δ2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
  and α2''Vv2-is-α2'Vv2: α2'' ↑ V_V2 = α2' ↑ V_V2
  and α2''Cv2-empty: α2'' ↑ C_V2 = []
  and δ2''E1-is-δ1''E2: δ2'' ↑ E_ES1 = δ1'' ↑ E_ES2
  by blast

from βE2-cE2-δ2''-v'E2-α2''-in-Tr2 βE1-cE1-δ1''-v'E1-α1''-in-Tr1
  validES2 validES1
have δ2''-in-E2star: set δ2'' ⊆ E_ES2 and δ1''-in-E1star: set δ1'' ⊆ E_ES1
  by (simp-all add: ES-valid-def traces-contain-events-def, auto)
with δ2''E1-is-δ1''E2 merge-property[of δ2'' E_ES2 δ1'' E_ES1] obtain δ'
  where δ'E2-is-δ2'': δ' ↑ E_ES2 = δ2''
  and δ'E1-is-δ1'': δ' ↑ E_ES1 = δ1''
  and δ'-contains-only-δ2''-δ1''-events: set δ' ⊆ set δ2'' ∪ set δ1''
  unfolding Let-def
  by auto
```

let *?TAU* = β @ [c] @ δ' @ [v']
let *?LAMBDA* = $\alpha \restriction V_{\mathcal{V}}$
let *?T2* = $\alpha 2''$
let *?T1* = $\alpha 1''$

have *?TAU* $\in Tr_{(ES1 \parallel ES2)}$
proof −
from β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2* δ'*E2-is-*$\delta 2''$ *validES2*
have $\beta \restriction E_{ES2}$ @ [c] $\restriction E_{ES2}$ @ $\delta' \restriction E_{ES2}$ @ [v'] $\restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence (β @ [c] @ δ' @ [v']) $\restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *projection-def*, *auto*)
moreover
from β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1* δ'*E1-is-*$\delta 1''$ *validES1*
have $\beta \restriction E_{ES1}$ @ [c] $\restriction E_{ES1}$ @ $\delta' \restriction E_{ES1}$ @ [v'] $\restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence (β @ [c] @ δ' @ [v']) $\restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta v'\alpha$*-in-Tr c-in-Cv-inter-Upsilon VIsViewOnE* δ'*-contains-only-*$\delta 2''$*-*$\delta 1''$*-events* $\delta 2''$*-in-E2star* $\delta 1''$*-in-E1star*
have *set* (β @ [c] @ δ' @ [v']) $\subseteq E_{ES2} \cup E_{ES1}$
unfolding *composeES-def isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*
by *auto*
ultimately show *?thesis*
unfolding *composeES-def*
by *auto*
qed
hence *set ?TAU* $\subseteq E_{(ES1 \parallel ES2)}$
unfolding *composeES-def*
by *auto*
moreover
have *set ?LAMBDA* $\subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
note $\alpha 2''$*-in-E2star* $\alpha 1''$*-in-E1star*
moreover
from β*E2-cE2-*$\delta 2''$*-v'E2-*$\alpha 2''$*-in-Tr2* δ'*E2-is-*$\delta 2''$
have *?TAU* $\restriction E_{ES2}$ @ *?T2* $\in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
from β*E1-cE1-*$\delta 1''$*-v'E1-*$\alpha 1''$*-in-Tr1* δ'*E1-is-*$\delta 1''$
have *?TAU* $\restriction E_{ES1}$ @ *?T1* $\in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
have *?LAMBDA* $\restriction E_{ES2}$ = *?T2* $\restriction V_{\mathcal{V}}$
proof −

from *propSepViews*
have *?LAMBDA* ↾ $E_{ES2} = \alpha$ ↾ $V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** (*simp only*: *projection-sequence*)
moreover
from *α2''-in-E2star propSepViews*
have *?T2* ↾ $V_{\mathcal{V}} =$ *?T2* ↾ $V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note *α2'Vv2-is-αVv2 α2''Vv2-is-α2'Vv2*
ultimately show *?thesis*
by *simp*
qed
moreover
have *?LAMBDA* ↾ $E_{ES1} =$ *?T1* ↾ $V_{\mathcal{V}}$
proof −
from *propSepViews*
have *?LAMBDA* ↾ $E_{ES1} = \alpha$ ↾ $V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-sequence*)
moreover
from *α1''-in-E1star propSepViews*
have *?T1* ↾ $V_{\mathcal{V}} =$ *?T1* ↾ $V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note *α1'Vv1-is-αVv1 α1''Vv1-is-α1'Vv1*
ultimately show *?thesis*
by *simp*
qed
moreover
note *α2''Cv2-empty α1''Cv1-empty generalized-zipping-lemma*
ultimately obtain *t*
where *?TAU* @ $t \in Tr_{(ES1 \parallel ES2)}$
and t ↾ $V_{\mathcal{V}} =$ *?LAMBDA*
and t ↾ $C_{\mathcal{V}} = []$
by *blast*
moreover
have *set* $\delta' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma}$
proof −
from *δ'-contains-only-δ2''-δ1''-events δ2''-in-N2-inter-Delta2star*
δ1''-in-N1-inter-Delta1star
have *set* $\delta' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
with *Delta1-N1-Delta2-N2-subset-Delta Nv1-union-Nv2-subsetof-Nv* **show** *?thesis*
by *auto*
qed
ultimately have $\exists \alpha'\, \gamma'.$ (*set* $\gamma' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge \beta$ @ $[c]$ @ γ' @ $[v']$ @ $\alpha' \in Tr_{(ES1 \parallel ES2)}$
$\wedge\ \alpha'$ ↾ $V_{\mathcal{V}} = \alpha$ ↾ $V_{\mathcal{V}} \wedge \alpha'$ ↾ $C_{\mathcal{V}} = []$)
by (*simp only*: *append-assoc*, *blast*)
}
ultimately have $\exists \alpha'\, \gamma'.$ (*set* $\gamma' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge \beta$ @ $[c]$ @ γ' @ $[v']$ @ $\alpha' \in Tr_{(ES1 \parallel ES2)}$
$\wedge\ \alpha'$ ↾ $V_{\mathcal{V}} = \alpha$ ↾ $V_{\mathcal{V}} \wedge \alpha'$ ↾ $C_{\mathcal{V}} = []$)

by *blast*
}
thus *?thesis*
unfolding *FCI-def*
by *blast*
qed

theorem *compositionality-FCIA*:
⟦ *BSD* $\mathcal{V}1$ Tr_{ES1}; *BSD* $\mathcal{V}2$ Tr_{ES2}; *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}; *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2};
$(\varrho 1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1}$; $(\varrho 2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}$;
total ES1 $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$; *total ES2* $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$;
$\nabla_\Gamma \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$; $\nabla_\Gamma \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$;
$\Upsilon_\Gamma \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$; $\Upsilon_\Gamma \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$;
$(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_\Gamma$;
$(N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
$\vee\ (\ N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2})$;
FCIA $\varrho 1$ $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}; *FCIA* $\varrho 2$ $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2} ⟧
$\Longrightarrow$ *FCIA* ϱ Γ $\mathcal{V}$ $(Tr_{(ES1\ \parallel\ ES2)})$
proof −
assume *BSD1*: *BSD* $\mathcal{V}1$ Tr_{ES1}
and *BSD2*: *BSD* $\mathcal{V}2$ Tr_{ES2}
and *BSIA1*: *BSIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}
and *BSIA2*: *BSIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2}
and *ϱ1v1-subset-ϱv-inter-E1*: $(\varrho 1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1}$
and *ϱ2v2-subset-ϱv-inter-E2*: $(\varrho 2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}$
and *total-ES1-C1-inter-Upsilon1-inter-N2-inter-Delta2*:
total ES1 $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
and *total-ES2-C2-inter-Upsilon2-inter-N1-inter-Delta1*:
total ES2 $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1})$
and *Nabla-inter-E1-subset-Nabla1*: $\nabla_\Gamma \cap E_{ES1} \subseteq \nabla_{\Gamma 1}$
and *Nabla-inter-E2-subset-Nabla2*: $\nabla_\Gamma \cap E_{ES2} \subseteq \nabla_{\Gamma 2}$
and *Upsilon-inter-E1-subset-Upsilon1*: $\Upsilon_\Gamma \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$
and *Upsilon-inter-E2-subset-Upsilon2*: $\Upsilon_\Gamma \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}$
and *Delta1-N1-Delta2-N2-subset-Delta*: $(\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_\Gamma$
and *very-long-asm*: $(N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
$\vee\ (\ N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2})$
and *FCIA1*: *FCIA* $\varrho 1$ $\Gamma 1$ $\mathcal{V}1$ Tr_{ES1}
and *FCIA2*: *FCIA* $\varrho 2$ $\Gamma 2$ $\mathcal{V}2$ Tr_{ES2}

{
fix α β c v'
assume *c-in-Cv-inter-Upsilon*: $c \in (C_{\mathcal{V}} \cap \Upsilon_\Gamma)$
and *v'-in-Vv-inter-Nabla*: $v' \in (V_{\mathcal{V}} \cap \nabla_\Gamma)$
and *βv'α-in-Tr*: $(\beta\ @\ [v']\ @\ \alpha) \in Tr_{(ES1\ \parallel\ ES2)}$
and *αCv-empty*: $\alpha \upharpoonright C_{\mathcal{V}} = []$
and *Adm*: *Adm* $\mathcal{V}$ ϱ $(Tr_{(ES1\ \parallel\ ES2)})$ β c

interpret *CSES1*: *CompositionSupport ES1* $\mathcal{V}$ $\mathcal{V}1$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES1 validV1*)

interpret *CSES2*: *CompositionSupport ES2* $\mathcal{V}$ $\mathcal{V}2$
using *propSepViews* **unfolding** *properSeparationOfViews-def*
by (*simp add*: *CompositionSupport-def validES2 validV2*)

from $\beta v'\alpha$-*in-Tr*
have $\beta v'\alpha$-*E1-in-Tr1*: $(((\beta \mathbin{@} [v']) \mathbin{@} \alpha) \restriction E_{ES1}) \in Tr_{ES1}$
and $\beta v'\alpha$-*E2-in-Tr2*: $(((\beta \mathbin{@} [v']) \mathbin{@} \alpha) \restriction E_{ES2}) \in Tr_{ES2}$
by (*simp add*: *composeES-def*)+

from *CSES1.BSD-in-subsystem2*[*OF* $\beta v'\alpha$-*E1-in-Tr1 BSD1*] **obtain** $\alpha 1'$
where $\beta v' E1 \alpha 1'$-*in-Tr1*: $(\beta \mathbin{@} [v']) \restriction E_{ES1} \mathbin{@} \alpha 1' \in Tr_{ES1}$
and $\alpha 1' Vv1$-*is-*α*Vv1*: $\alpha 1' \restriction V_{\mathcal{V}1} = \alpha \restriction V_{\mathcal{V}1}$
and $\alpha 1' Cv1$-*empty*: $\alpha 1' \restriction C_{\mathcal{V}1} = []$
by *auto*

from *CSES2.BSD-in-subsystem2*[*OF* $\beta v'\alpha$-*E2-in-Tr2 BSD2*] **obtain** $\alpha 2'$
where $\beta v' E2 \alpha 2'$-*in-Tr2*: $(\beta \mathbin{@} [v']) \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2}$
and $\alpha 2' Vv2$-*is-*α*Vv2*: $\alpha 2' \restriction V_{\mathcal{V}2} = \alpha \restriction V_{\mathcal{V}2}$
and $\alpha 2' Cv2$-*empty*: $\alpha 2' \restriction C_{\mathcal{V}2} = []$
by *auto*

note *very-long-asm*
moreover {
assume *Nv1-inter-Delta1-inter-E2-empty*: $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}$
and *Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1*: $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}$

let *?ALPHA2''-DELTA2''* $= \exists\ \alpha 2''\ \delta 2''.$ (
set $\alpha 2'' \subseteq E_{ES2} \wedge$ *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} \delta 2'' \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \alpha 2'' \in Tr_{ES2}$
$\wedge\ \alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2} \wedge \alpha 2'' \restriction C_{\mathcal{V}2} = []$)

from *c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV2*
have $c \notin E_{ES2} \vee (c \in E_{ES2} \wedge v' \notin E_{ES2}) \vee (c \in E_{ES2} \wedge v' \in E_{ES2})$
by (*simp add*: *V-valid-def isViewOn-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def)
moreover {
assume *c-notin-E2*: $c \notin E_{ES2}$

from *validES2* $\beta v' E2 \alpha 2'$-*in-Tr2* **have** *set* $\alpha 2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
from $\beta v' E2 \alpha 2'$-*in-Tr2 c-notin-E2*
have $\beta \restriction E_{ES2} \mathbin{@} [c] \restriction E_{ES2} \mathbin{@} [] \mathbin{@} [v'] \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
have $\alpha 2' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$ ..
moreover
note $\alpha 2' Cv2$-*empty*

ultimately have *?ALPHA2''-DELTA2''*
by *blast*
}
moreover {
assume *c-in-E2*: $c \in E_{ES2}$
and *v'-notin-E2*: $v' \notin E_{ES2}$

from *c-in-E2 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E2-subset-Upsilon2
have *c-in-Cv2-inter-Upsilon2*: $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def* **by** *auto*
hence $c \in C_{\mathcal{V}2}$
by *auto*
moreover
from *βv'E2α2'-in-Tr2 v'-notin-E2* **have** $\beta \restriction E_{ES2} \mathbin{@} \alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *α2'Cv2-empty*
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $(\beta \restriction E_{ES2})$ c
proof −
from *Adm* **obtain** γ
where *γϱv-is-βϱv*: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and *γc-in-Tr*: $(\gamma \mathbin{@} [c]) \in Tr_{(ES1 \parallel ES2)}$
unfolding *Adm-def*
by *auto*

from *c-in-E2 γc-in-Tr* **have** $(\gamma \restriction E_{ES2}) \mathbin{@} [c] \in Tr_{ES2}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2)$
proof −
from *γϱv-is-βϱv* **have** $\gamma \restriction E_{ES2} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES2} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with *ϱ2v2-subset-ϱv-inter-E2* **have** $\gamma \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction (\varrho 2\ \mathcal{V}2)$
by (*metis Int-subset-iff γϱv-is-βϱv projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *one*: $\beta \restriction E_{ES2} \mathbin{@} [c] \mathbin{@} \alpha 2'' \in Tr_{ES2}$
and *two*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *three*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSIA-def*
by *blast*

from *one validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$

by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* [] ⊆ $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
from *one c-in-E2 v′-notin-E2*
have $\beta \restriction E_{ES2}$ @ [*c*] $\restriction E_{ES2}$ @ [] @ [v'] $\restriction E_{ES2}$ @ $\alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *two three*
ultimately have *?ALPHA2″-DELTA2″*
by *blast*
}
moreover {
assume *c-in-E2*: $c \in E_{ES2}$
and *v′-in-E2*: $v' \in E_{ES2}$

from *c-in-E2 c-in-Cv-inter-Upsilon propSepViews*
Upsilon-inter-E2-subset-Upsilon2
have *c-in-Cv2-inter-Upsilon2*: $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *v′-in-E2 propSepViews v′-in-Vv-inter-Nabla Nabla-inter-E2-subset-Nabla2*
have $v' \in V_{\mathcal{V}2} \cap$ *Nabla* $\Gamma 2$
unfolding *properSeparationOfViews-def* **by** *auto*
moreover
from *v′-in-E2* $\beta v'E2\alpha 2'$*-in-Tr2* **have** $\beta \restriction E_{ES2}$ @ [v'] @ $\alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note $\alpha 2'Cv2\text{-}empty$
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} ($\beta \restriction E_{ES2}$) *c*
proof −
from *Adm* **obtain** γ
where $\gamma\varrho v\text{-}is\text{-}\beta\varrho v$: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and $\gamma c\text{-}in\text{-}Tr$: $(\gamma$ @ [*c*]$) \in Tr_{(ES1 \parallel ES2)}$
unfolding *Adm-def*
by *auto*

from *c-in-E2* $\gamma c\text{-}in\text{-}Tr$ **have** ($\gamma \restriction E_{ES2}$) @ [*c*] $\in Tr_{ES2}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2)$
proof −
from $\gamma\varrho v\text{-}is\text{-}\beta\varrho v$ **have** $\gamma \restriction E_{ES2} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES2} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with $\varrho 2v2\text{-}subset\text{-}\varrho v\text{-}inter\text{-}E2$ **have** $\gamma \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction (\varrho 2\ \mathcal{V}2)$
by (*metis Int-subset-iff* $\gamma\varrho v\text{-}is\text{-}\beta\varrho v$ *projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*

```
    by auto
  qed
  moreover
  note FCIA2
  ultimately obtain α2'' δ2''
    where one: set δ2'' ⊆ N_V2 ∩ Δ_Γ2
    and two: β ↑ E_ES2 @ [c] @ δ2'' @ [v'] @ α2'' ∈ Tr_ES2
    and three: α2'' ↑ V_V2 = α2' ↑ V_V2
    and four: α2'' ↑ C_V2 = []
    unfolding FCIA-def
    by blast

  from two validES2 have set α2'' ⊆ E_ES2
    by (simp add: ES-valid-def traces-contain-events-def, auto)
  moreover
  note one
  moreover
  from two c-in-E2 v'-in-E2
  have β ↑ E_ES2 @ [c] ↑ E_ES2 @ δ2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
    by (simp add: projection-def)
  moreover
  note three four
  ultimately have ?ALPHA2''-DELTA2''
    by blast
}
ultimately obtain α2'' δ2''
  where α2''-in-E2star: set α2'' ⊆ E_ES2
  and δ2''-in-N2-inter-Delta2star:set δ2'' ⊆ N_V2 ∩ Δ_Γ2
  and βE2-cE2-δ2''-v'E2-α2''-in-Tr2:
        β ↑ E_ES2 @ [c] ↑ E_ES2 @ δ2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
  and α2''Vv2-is-α2'Vv2: α2'' ↑ V_V2 = α2' ↑ V_V2
  and α2''Cv2-empty: α2'' ↑ C_V2 = []
  by blast

from c-in-Cv-inter-Upsilon Upsilon-inter-E1-subset-Upsilon1 propSepViews
have cE1-in-Cv1-inter-Upsilon1: set ([c] ↑ E_ES1) ⊆ C_V1 ∩ ϒ_Γ1
  unfolding properSeparationOfViews-def by (simp add: projection-def, auto)

from δ2''-in-N2-inter-Delta2star Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1
 propSepViews disjoint-Nv2-Vv1
have δ2''E1-in-Cv1-inter-Upsilon1star: set (δ2'' ↑ E_ES1) ⊆ C_V1 ∩ ϒ_Γ1
  proof −
    from δ2''-in-N2-inter-Delta2star
    have eq: δ2'' ↑ E_ES1 = δ2'' ↑ (N_V2 ∩ Δ_Γ2 ∩ E_ES1)
      by (metis Int-commute Int-left-commute Int-lower1 Int-lower2
        projection-intersection-neutral subset-trans)

    from validV1 Nv2-inter-Delta2-inter-E1-subsetof-Upsilon1
      propSepViews disjoint-Nv2-Vv1
    have N_V2 ∩ Δ_Γ2 ∩ E_ES1 ⊆ C_V1 ∩ ϒ_Γ1
      unfolding properSeparationOfViews-def
      by (simp add: isViewOn-def V-valid-def
```

VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*subst eq*, *simp only*: *projection-def*, *auto*)
qed

have *cδ2''E1-in-Cv1-inter-Upsilon1star*: *set* $((c \# \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
proof −
from *cE1-in-Cv1-inter-Upsilon1* *δ2''E1-in-Cv1-inter-Upsilon1star*
have *set* $(([c] \mathbin{@} \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*
by *auto*
qed

have
$\exists\ \alpha 1''\ \delta 1''.$ *set* $\alpha 1'' \subseteq E_{ES1} \wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} \delta 1'' \mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
proof *cases*
assume *v'-in-E1*: $v' \in E_{ES1}$
with *Nabla-inter-E1-subset-Nabla1 propSepViews v'-in-Vv-inter-Nabla*
have *v'-in-Vv1-inter-Nabla1*: $v' \in V_{\mathcal{V}1} \cap Nabla\ \Gamma 1$
unfolding *properSeparationOfViews-def* **by** *auto*

have $[\![(\beta \mathbin{@} [v']) \restriction E_{ES1} \mathbin{@} \alpha 1' \in Tr_{ES1}$;
$\alpha 1' \restriction C_{\mathcal{V}1} = []$; *set* $((c \# \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$;
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$; *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$;
Adm $\mathcal{V}\ \varrho\ (Tr_{(ES1 \parallel ES2)})\ \beta\ c\ ]\!]$
$\Longrightarrow \exists\ \alpha 1''\ \delta 1''.$
(*set* $\alpha 1'' \subseteq E_{ES1} \wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} \delta 1'' \mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) = \delta 2'' \restriction E_{ES1}$)
proof (*induct length* $((c \# \delta 2'') \restriction E_{ES1})$ *arbitrary*: $\beta\ \alpha 1'\ c\ \delta 2''$)
case *0*

from *0(2)* *validES1* **have** *set* $\alpha 1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} [] \mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1' \in Tr_{ES1}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)

```
    qed
  moreover
  have α1′ ↾ V_V1 = α1′ ↾ V_V1 ..
  moreover
  note 0(3)
  moreover
  from 0(1) have [] ↾ (C_V1 ∩ Υ_Γ1) = δ2′′ ↾ E_ES1
    by (simp add: projection-def, split if-split-asm, auto)
  ultimately show ?case
    by blast
next
  case (Suc n)

  from projection-split-last[OF Suc(2)] obtain μ c′ ν
    where c′-in-E1: c′ ∈ E_ES1
    and cδ2′′-is-μc′ν: c # δ2′′ = μ @ [c′] @ ν
    and νE1-empty: ν ↾ E_ES1 = []
    and n-is-length-μνE1: n = length ((μ @ ν) ↾ E_ES1)
    by blast

  from Suc(5) c′-in-E1 cδ2′′-is-μc′ν have set (μ ↾ E_ES1 @ [c′]) ⊆ C_V1 ∩ Υ_Γ1
    by (simp only: cδ2′′-is-μc′ν projection-concatenation-commute
      projection-def, auto)
  hence c′-in-Cv1-inter-Upsilon1: c′ ∈ C_V1 ∩ Υ_Γ1
    by auto
  hence c′-in-Cv1: c′ ∈ C_V1 and c′-in-Upsilon1: c′ ∈ Υ_Γ1
    by auto
  with validV1 have c′-in-E1: c′ ∈ E_ES1
    by (simp add: isViewOn-def V-valid-def VC-disjoint-def
      VN-disjoint-def NC-disjoint-def, auto)

  show ?case
    proof (cases μ)
      case Nil
      with cδ2′′-is-μc′ν have c-is-c′: c = c′ and δ2′′-is-ν: δ2′′ = ν
        by auto
      with c′-in-Cv1-inter-Upsilon1 have c ∈ C_V1 ∩ Υ_Γ1
        by simp
      moreover
      note v′-in-Vv1-inter-Nabla1
      moreover
      from v′-in-E1 Suc(3) have (β ↾ E_ES1) @ [v′] @ α1′ ∈ Tr_ES1
        by (simp add: projection-concatenation-commute projection-def)
      moreover
      note Suc(4)
      moreover
      have Adm V1 ϱ1 Tr_ES1 (β ↾ E_ES1) c
        proof −
          from Suc(8) obtain γ
            where γϱv-is-βϱv: γ ↾ (ϱ V) = β ↾ (ϱ V)
            and γc-in-Tr: (γ @ [c]) ∈ Tr_(ES1 ∥ ES2)
            unfolding Adm-def
```

by *auto*

from *c-is-c′ c′-in-E1* γ*c-in-Tr* **have** $(\gamma \restriction E_{ES1}) \mathbin{@} [c] \in Tr_{ES1}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES1} \restriction (\varrho 1\ \mathcal{V}1) = \beta \restriction E_{ES1} \restriction (\varrho 1\ \mathcal{V}1)$
proof −
from $\gamma\varrho v$*-is-*$\beta\varrho v$ **have** $\gamma \restriction E_{ES1} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES1} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with $\varrho 1v1$*-subset-*ϱv*-inter-E1* **have** $\gamma \restriction (\varrho 1\ \mathcal{V}1) = \beta \restriction (\varrho 1\ \mathcal{V}1)$
by (*metis Int-subset-iff* $\gamma\varrho v$*-is-*$\beta\varrho v$ *projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *FCIA1*
ultimately obtain $\alpha 1''\ \gamma$
where *one*: *set* $\gamma \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *two*: $\beta \restriction E_{ES1} \mathbin{@} [c] \mathbin{@} \gamma \mathbin{@} [v'] \mathbin{@} \alpha 1'' \in Tr_{ES1}$
and *three*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *four*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCIA-def*
by *blast*

let *?DELTA1″* $= \nu \restriction E_{ES1} \mathbin{@} \gamma$

from *two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *one* ν*E1-empty*
have *set ?DELTA1″* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@}$ *?DELTA1″* $\mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
proof −
from *c-is-c′ c′-in-E1* **have** $[c] = [c] \restriction E_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *v′-in-E1* **have** $[v'] = [v'] \restriction E_{ES1}$
by (*simp add*: *projection-def*)
moreover
note ν*E1-empty two*
ultimately show *?thesis*
by *auto*
qed
moreover
note *three four*
moreover

```
  have ?DELTA1'' ↾ (C_V1 ∩ Υ_Γ1) = δ2'' ↾ E_ES1
    proof −
      have γ ↾ (C_V1 ∩ Υ_Γ1) = []
        proof −
          from validV1 have N_V1 ∩ Δ_Γ1 ∩ (C_V1 ∩ Υ_Γ1) = {}
            by (simp add: isViewOn-def V-valid-def
              VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
          with projection-intersection-neutral[OF one, of C_V1 ∩ Υ_Γ1]
          show ?thesis
            by (simp add: projection-def)
        qed
      with δ2''-is-ν νE1-empty show ?thesis
        by (simp add: projection-concatenation-commute)
    qed
  ultimately show ?thesis
    by blast
next
  case (Cons x xs)
  with cδ2''-is-μc'ν
  have μ-is-c-xs: μ = [c] @ xs and δ2''-is-xs-c'-ν: δ2'' = xs @ [c'] @ ν
    by auto
  with n-is-length-μνE1 have n = length ((c # (xs @ ν)) ↾ E_ES1)
    by auto
  moreover
  note Suc(3,4)
  moreover
  have set ((c # (xs @ ν)) ↾ E_ES1) ⊆ C_V1 ∩ Υ_Γ1
    proof −
      have res: c # (xs @ ν) = [c] @ (xs @ ν)
        by auto

      from Suc(5) cδ2''-is-μc'ν μ-is-c-xs νE1-empty
      show ?thesis
        by (subst res, simp only: cδ2''-is-μc'ν
          projection-concatenation-commute set-append, auto)
    qed
  moreover
  note Suc(6)
  moreover
  from Suc(7) δ2''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V2 ∩ Δ_Γ2
    by auto
  moreover note Suc(8) Suc(1)[of c xs @ ν β α1']
  ultimately obtain δ γ
    where one: set δ ⊆ E_ES1
    and two: set γ ⊆ N_V1 ∩ Δ_Γ1 ∪ C_V1 ∩ Υ_Γ1 ∩ N_V2 ∩ Δ_Γ2
    and three: β ↾ E_ES1 @ [c] ↾ E_ES1 @ γ @ [v'] ↾ E_ES1 @ δ ∈ Tr_ES1
    and four: δ ↾ V_V1 = α1' ↾ V_V1
    and five: δ ↾ C_V1 = []
    and six: γ ↾ (C_V1 ∩ Υ_Γ1) = (xs @ ν) ↾ E_ES1
    by blast
```

let *?BETA* = $\beta \restriction E_{ES1}$ @ $[c] \restriction E_{ES1}$ @ γ

note *c′-in-Cv1-inter-Upsilon1 v′-in-Vv1-inter-Nabla1*
moreover
from *three v′-in-E1* **have** *?BETA* @ $[v']$ @ $\delta \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *five*
moreover
have *Adm* $\mathcal{V}1$ $\varrho 1$ Tr_{ES1} *?BETA* c'
proof −
have *?BETA* @ $[c'] \in Tr_{ES1}$
proof −
from *Suc*(7) *c′-in-Cv1-inter-Upsilon1* $\delta 2''$*-is-xs-c′-*ν
have $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
from *validES1 three* **have** *?BETA* $\in Tr_{ES1}$
by (*unfold ES-valid-def traces-prefixclosed-def*
prefixclosed-def prefix-def, *auto*)
moreover
note *total-ES1-C1-inter-Upsilon1-inter-N2-inter-Delta2*
ultimately show *?thesis*
unfolding *total-def*
by *blast*
qed
thus *?thesis*
unfolding *Adm-def*
by *blast*
qed
moreover
note *FCIA1*
ultimately obtain $\alpha 1''$ δ'
where *fcia-one*: *set* $\delta' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *fcia-two*: *?BETA* @ $[c']$ @ δ' @ $[v']$ @ $\alpha 1'' \in Tr_{ES1}$
and *fcia-three*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \delta \restriction V_{\mathcal{V}1}$
and *fcia-four*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *FCIA-def*
by *blast*

let *?DELTA1″* = γ @ $[c']$ @ δ'

from *fcia-two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set ?DELTA1″* $\subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
proof −
from *Suc*(7) *c′-in-Cv1-inter-Upsilon1* $\delta 2''$*-is-xs-c′-*ν
have $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
with *two fcia-one* **show** *?thesis*
by *auto*

```
      qed
    moreover
    from fcia-two v'-in-E1
    have β ↾ E_ES1 @ [c] ↾ E_ES1 @ ?DELTA1'' @ [v'] ↾ E_ES1 @ α1'' ∈ Tr_ES1
      by (simp add: projection-def)
    moreover
    from fcia-three four have α1'' ↾ V_V1 = α1' ↾ V_V1
      by simp
    moreover
    note fcia-four
    moreover
    have ?DELTA1'' ↾ (C_V1 ∩ ϒ_Γ1) = δ2'' ↾ E_ES1
      proof −
        have δ' ↾ (C_V1 ∩ ϒ_Γ1) = []
          proof −
            from fcia-one have ∀ e ∈ set δ'. e ∈ N_V1 ∩ Δ_Γ1
              by auto
            with validV1 have ∀ e ∈ set δ'. e ∉ C_V1 ∩ ϒ_Γ1
              by (simp add: isViewOn-def V-valid-def
                VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
            thus ?thesis
              by (simp add: projection-def)
          qed
        with c'-in-E1 c'-in-Cv1-inter-Upsilon1 δ2''-is-xs-c'-ν νE1-empty six
        show ?thesis
          by (simp only: projection-concatenation-commute projection-def, auto)
      qed
    ultimately show ?thesis
      by blast
  qed
qed
from this[OF βv'E1α1'-in-Tr1 α1'Cv1-empty cδ2''E1-in-Cv1-inter-Upsilon1star
  c-in-Cv-inter-Upsilon δ2''-in-N2-inter-Delta2star Adm]
obtain α1'' δ1''
  where one: set α1'' ⊆ E_ES1
  and two: set δ1'' ⊆ N_V1 ∩ Δ_Γ1 ∪ C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
  and three: β ↾ E_ES1 @ [c] ↾ E_ES1 @ δ1'' @ [v'] ↾ E_ES1 @ α1'' ∈ Tr_ES1
  ∧ α1'' ↾ V_V1 = α1' ↾ V_V1 ∧ α1'' ↾ C_V1 = []
  and four: δ1'' ↾ (C_V1 ∩ ϒ_Γ1) = δ2'' ↾ E_ES1
  by blast

note one two three
moreover
have δ1'' ↾ E_ES2 = δ2'' ↾ E_ES1
  proof −
    from projection-intersection-neutral[OF two, of E_ES2]
      Nv1-inter-Delta1-inter-E2-empty validV2
    have δ1'' ↾ E_ES2 = δ1'' ↾ (C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2 ∩ E_ES2)
      by (simp only: Int-Un-distrib2, auto)
    moreover
    from validV2
    have C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2 ∩ E_ES2 = C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
```

by (*simp add*:*isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *simp*
hence $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}) \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by (*simp add*: *projection-def*)
with *four* **have** $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1} \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2})$
by *simp*
hence $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction (N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}) \restriction E_{ES1}$
by (*simp only*: *projection-commute*)
with *δ2''-in-N2-inter-Delta2star* **show** *?thesis*
by (*simp only*: *list-subset-iff-projection-neutral*)
qed
ultimately show *?thesis*
by *blast*
next
assume *v'-notin-E1*: $v' \notin E_{ES1}$

have $[\![(\beta @ [v']) \restriction E_{ES1} @ \alpha 1' \in Tr_{ES1}$;
$\alpha 1' \restriction C_{\mathcal{V}1} = []$; *set* $((c \# \delta 2'') \restriction E_{ES1}) \subseteq C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1}$;
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$; *set* $\delta 2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$;
Adm $\mathcal{V}$ ϱ $(Tr_{(ES1 \parallel ES2)})$ β c $]\!]$
$\Longrightarrow \exists\ \alpha 1''\ \delta 1''.$ (*set* $\alpha 1'' \subseteq E_{ES1} \wedge$ *set* $\delta 1'' \subseteq N_{\mathcal{V}1}$
$\cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
$\wedge\ \beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
$\wedge\ \alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1} \wedge \alpha 1'' \restriction C_{\mathcal{V}1} = []$
$\wedge\ \delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$)
proof (*induct length* $((c \# \delta 2'') \restriction E_{ES1})$ *arbitrary*: β $\alpha 1'$ c $\delta 2''$)
case *0*

from *0(2)* *validES1* **have** *set* $\alpha 1' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
moreover
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ [] @ [v'] \restriction E_{ES1} @ \alpha 1' \in Tr_{ES1}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES1}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha 1' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$ ..
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)

```
  ultimately show ?case
    by blast
next
  case (Suc n)

  from projection-split-last[OF Suc(2)] obtain μ c' ν
    where c'-in-E1: c' ∈ E_ES1
    and cδ2''-is-μc'ν: c # δ2'' = μ @ [c'] @ ν
    and νE1-empty: ν ↑ E_ES1 = []
    and n-is-length-μνE1: n = length ((μ @ ν) ↑ E_ES1)
    by blast

  from Suc(5) c'-in-E1 cδ2''-is-μc'ν have set (μ ↑ E_ES1 @ [c']) ⊆ C_V1 ∩ Υ_Γ1
    by (simp only: cδ2''-is-μc'ν projection-concatenation-commute projection-def, auto)
  hence c'-in-Cv1-inter-Upsilon1: c' ∈ C_V1 ∩ Υ_Γ1
    by auto
  hence c'-in-Cv1: c' ∈ C_V1 and c'-in-Upsilon1: c' ∈ Υ_Γ1
    by auto
  with validV1 have c'-in-E1: c' ∈ E_ES1
    by (simp add:isViewOn-def V-valid-def VC-disjoint-def
      VN-disjoint-def NC-disjoint-def, auto)

  show ?case
    proof (cases μ)
      case Nil
      with cδ2''-is-μc'ν have c-is-c': c = c' and δ2''-is-ν: δ2'' = ν
        by auto
      with c'-in-Cv1-inter-Upsilon1 have c ∈ C_V1
        by simp
      moreover
      from v'-notin-E1 Suc(3) have (β ↑ E_ES1) @ α1' ∈ Tr_ES1
        by (simp add: projection-concatenation-commute projection-def)
      moreover
      note Suc(4)
      moreover
      have Adm V1 ϱ1 Tr_ES1 (β ↑ E_ES1) c
        proof −
          from Suc(8) obtain γ
            where γϱv-is-βϱv: γ ↑ (ϱ V) = β ↑ (ϱ V)
            and γc-in-Tr: (γ @ [c]) ∈ Tr_(ES1 ∥ ES2)
            unfolding Adm-def
            by auto

          from c-is-c' c'-in-E1 γc-in-Tr have (γ ↑ E_ES1) @ [c] ∈ Tr_ES1
            by (simp add: projection-def composeES-def)
          moreover
          have γ ↑ E_ES1 ↑ (ϱ1 V1) = β ↑ E_ES1 ↑ (ϱ1 V1)
          proof −
            from γϱv-is-βϱv have γ ↑ E_ES1 ↑ (ϱ V) = β ↑ E_ES1 ↑ (ϱ V)
              by (metis projection-commute)
            with ϱ1v1-subset-ϱv-inter-E1 have γ ↑ (ϱ1 V1) = β ↑ (ϱ1 V1)
              by (metis Int-subset-iff γϱv-is-βϱv projection-subset-elim)
```

```
      thus ?thesis
        by (metis projection-commute)
    qed
    ultimately show ?thesis unfolding Adm-def
      by auto
  qed
 moreover
 note BSIA1
 ultimately obtain α1''
  where one: (β ↑ E_ES1) @ [c] @ α1'' ∈ Tr_ES1
  and two: α1'' ↑ V_V1 = α1' ↑ V_V1
  and three: α1'' ↑ C_V1 = []
  unfolding BSIA-def
  by blast

 let ?DELTA1'' = ν ↑ E_ES1

 from one validES1 have set α1'' ⊆ E_ES1
  by (simp add: ES-valid-def traces-contain-events-def, auto)
 moreover
 from νE1-empty
 have set ?DELTA1'' ⊆ N_V1 ∩ Δ_Γ1 ∪ C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
  by simp
 moreover
 from c-is-c' c'-in-E1 one v'-notin-E1 νE1-empty
 have (β ↑ E_ES1) @ [c] ↑ E_ES1 @ ?DELTA1'' @ [v'] ↑ E_ES1 @ α1'' ∈ Tr_ES1
  by (simp add: projection-def)
 moreover
 note two three
 moreover
 from νE1-empty δ2''-is-ν have ?DELTA1'' ↑ E_ES2 = δ2'' ↑ E_ES1
  by (simp add: projection-def)
 ultimately show ?thesis
  by blast
next
 case (Cons x xs)
 with cδ2''-is-μc'ν
 have μ-is-c-xs: μ = [c] @ xs and δ2''-is-xs-c'-ν: δ2'' = xs @ [c'] @ ν
  by auto
 with n-is-length-μνE1 have n = length ((c # (xs @ ν)) ↑ E_ES1)
  by auto
 moreover
 note Suc(3,4)
 moreover
 have set ((c # (xs @ ν)) ↑ E_ES1) ⊆ C_V1 ∩ ϒ_Γ1
  proof −
   have res: c # (xs @ ν) = [c] @ (xs @ ν)
    by auto

   from Suc(5) cδ2''-is-μc'ν μ-is-c-xs νE1-empty
   show ?thesis
    by (subst res, simp only: cδ2''-is-μc'ν projection-concatenation-commute
```

```
      set-append, auto)
  qed
moreover
note Suc(6)
moreover
from Suc(7) δ2''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V2 ∩ Δ_Γ2
  by auto
moreover note Suc(8) Suc(1)[of c xs @ ν β α1']
ultimately obtain δ γ
  where one: set δ ⊆ E_ES1
  and two: set γ ⊆ N_V1 ∩ Δ_Γ1 ∪ C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
  and three: β ↑ E_ES1 @ [c] ↑ E_ES1 @ γ @ [v'] ↑ E_ES1 @ δ ∈ Tr_ES1
  and four: δ ↑ V_V1 = α1' ↑ V_V1
  and five: δ ↑ C_V1 = []
  and six: γ ↑ E_ES2 = (xs @ ν) ↑ E_ES1
  by blast


let ?BETA = β ↑ E_ES1 @ [c] ↑ E_ES1 @ γ

from c'-in-Cv1-inter-Upsilon1 have c' ∈ C_V1
  by auto
moreover
from three v'-notin-E1 have ?BETA @ δ ∈ Tr_ES1
  by (simp add: projection-def)
moreover
note five
moreover
have Adm V1 ϱ1 Tr_ES1 ?BETA c'
  proof −
    have ?BETA @ [c'] ∈ Tr_ES1
      proof −
        from Suc(7) c'-in-Cv1-inter-Upsilon1 δ2''-is-xs-c'-ν
        have c' ∈ C_V1 ∩ ϒ_Γ1 ∩ N_V2 ∩ Δ_Γ2
          by auto
        moreover
        from validES1 three have ?BETA ∈ Tr_ES1
          by (unfold ES-valid-def traces-prefixclosed-def
            prefixclosed-def prefix-def, auto)
        moreover
        note total-ES1-C1-inter-Upsilon1-inter-N2-inter-Delta2
        ultimately show ?thesis
          unfolding total-def
          by blast
      qed
    thus ?thesis
      unfolding Adm-def
      by blast
  qed
moreover
note BSIA1
ultimately obtain α1''
```

where *bsia-one*: $?BETA \mathbin{@} [c'] \mathbin{@} \alpha 1'' \in Tr_{ES1}$
and *bsia-two*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \delta \restriction V_{\mathcal{V}1}$
and *bsia-three*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSIA-def*
by *blast*

let $?DELTA1'' = \gamma \mathbin{@} [c']$

from *bsia-one validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $?DELTA1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
proof −
from *Suc*(γ) *c'-in-Cv1-inter-Upsilon1* *δ2''-is-xs-c'-ν*
have $c' \in C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
by *auto*
with *two* **show** *?thesis*
by *auto*
qed
moreover
from *bsia-one v'-notin-E1*
have $\beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} ?DELTA1'' \mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
from *bsia-two four* **have** $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
by *simp*
moreover
note *bsia-three*
moreover
have $?DELTA1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
proof −
from *validV2 Suc*(γ) *δ2''-is-xs-c'-ν* **have** $c' \in E_{ES2}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
with *c'-in-E1* *c'-in-Cv1-inter-Upsilon1* *δ2''-is-xs-c'-ν* *νE1-empty six*
show *?thesis*
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
qed
ultimately show *?thesis*
by *blast*
qed
qed
from *this*[*OF* *βv'E1α1'-in-Tr1* *α1'Cv1-empty* *cδ2''E1-in-Cv1-inter-Upsilon1star*
c-in-Cv-inter-Upsilon *δ2''-in-N2-inter-Delta2star Adm*]
show *?thesis*
by *blast*
qed
then obtain $\alpha 1'' \delta 1''$
where *α1''-in-E1star*: *set* $\alpha 1'' \subseteq E_{ES1}$
and *δ1''-in-N1-inter-Delta1star*:*set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cup C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *βE1-cE1-δ1''-v'E1-α1''-in-Tr1*:
$\beta \restriction E_{ES1} \mathbin{@} [c] \restriction E_{ES1} \mathbin{@} \delta 1'' \mathbin{@} [v'] \restriction E_{ES1} \mathbin{@} \alpha 1'' \in Tr_{ES1}$

and $\alpha 1''Vv1\text{-}is\text{-}\alpha 1'Vv1$: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and $\alpha 1''Cv1\text{-}empty$: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
and $\delta 1''E2\text{-}is\text{-}\delta 2''E1$: $\delta 1'' \restriction E_{ES2} = \delta 2'' \restriction E_{ES1}$
by *blast*

from $\beta E1\text{-}cE1\text{-}\delta 1''\text{-}v'E1\text{-}\alpha 1''\text{-}in\text{-}Tr1$ $\beta E2\text{-}cE2\text{-}\delta 2''\text{-}v'E2\text{-}\alpha 2''\text{-}in\text{-}Tr2$ *validES1* *validES2*
have $\delta 1''\text{-}in\text{-}E1star$: *set* $\delta 1'' \subseteq E_{ES1}$ **and** $\delta 2''\text{-}in\text{-}E2star$: *set* $\delta 2'' \subseteq E_{ES2}$
by (*simp-all add*: *ES-valid-def traces-contain-events-def*, *auto*)
with $\delta 1''E2\text{-}is\text{-}\delta 2''E1$ *merge-property*[*of* $\delta 1''$ E_{ES1} $\delta 2''$ E_{ES2}] **obtain** δ'
where $\delta'E1\text{-}is\text{-}\delta 1''$: $\delta' \restriction E_{ES1} = \delta 1''$
and $\delta'E2\text{-}is\text{-}\delta 2''$: $\delta' \restriction E_{ES2} = \delta 2''$
and $\delta'\text{-}contains\text{-}only\text{-}\delta 1''\text{-}\delta 2''\text{-}events$: *set* $\delta' \subseteq$ *set* $\delta 1'' \cup$ *set* $\delta 2''$
unfolding *Let-def*
by *auto*

let *?TAU* $= \beta$ @ $[c]$ @ δ' @ $[v']$
let *?LAMBDA* $= \alpha \restriction V_{\mathcal{V}}$
let *?T1* $= \alpha 1''$
let *?T2* $= \alpha 2''$

have *?TAU* $\in Tr_{(ES1 \parallel ES2)}$
proof −
from $\beta E1\text{-}cE1\text{-}\delta 1''\text{-}v'E1\text{-}\alpha 1''\text{-}in\text{-}Tr1$ $\delta'E1\text{-}is\text{-}\delta 1''$ *validES1*
have $\beta \restriction E_{ES1}$ @ $[c] \restriction E_{ES1}$ @ $\delta' \restriction E_{ES1}$ @ $[v'] \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def* *prefixclosed-def prefix-def*)
hence $(\beta$ @ $[c]$ @ δ' @ $[v']) \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta E2\text{-}cE2\text{-}\delta 2''\text{-}v'E2\text{-}\alpha 2''\text{-}in\text{-}Tr2$ $\delta'E2\text{-}is\text{-}\delta 2''$ *validES2*
have $\beta \restriction E_{ES2}$ @ $[c] \restriction E_{ES2}$ @ $\delta' \restriction E_{ES2}$ @ $[v'] \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def* *prefixclosed-def prefix-def*)
hence $(\beta$ @ $[c]$ @ δ' @ $[v']) \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta v'\alpha\text{-}in\text{-}Tr$ *c-in-Cv-inter-Upsilon VIsViewOnE* $\delta'\text{-}contains\text{-}only\text{-}\delta 1''\text{-}\delta 2''\text{-}events$ $\delta 1''\text{-}in\text{-}E1star$ $\delta 2''\text{-}in\text{-}E2star$
have *set* $(\beta$ @ $[c]$ @ δ' @ $[v']) \subseteq E_{ES1} \cup E_{ES2}$
unfolding *composeES-def isViewOn-def V-valid-def* *VC-disjoint-def VN-disjoint-def NC-disjoint-def*
by *auto*
ultimately show *?thesis*
unfolding *composeES-def*
by *auto*
qed
hence *set ?TAU* $\subseteq E_{(ES1 \parallel ES2)}$
unfolding *composeES-def*
by *auto*
moreover

have *set ?LAMBDA* $\subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
note *α1''-in-E1star α2''-in-E2star*
moreover
from *βE1-cE1-δ1''-v'E1-α1''-in-Tr1 δ'E1-is-δ1''*
have *?TAU* $\restriction E_{ES1}$ @ *?T1* $\in Tr_{ES1}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
from *βE2-cE2-δ2''-v'E2-α2''-in-Tr2 δ'E2-is-δ2''*
have *?TAU* $\restriction E_{ES2}$ @ *?T2* $\in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
have *?LAMBDA* $\restriction E_{ES1} =$ *?T1* $\restriction V_{\mathcal{V}}$
proof −
from *propSepViews* **have** *?LAMBDA* $\restriction E_{ES1} = \alpha \restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** (*simp only*: *projection-sequence*)
moreover
from *α1''-in-E1star propSepViews*
have *?T1* $\restriction V_{\mathcal{V}} =$ *?T1* $\restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note *α1'Vv1-is-αVv1 α1''Vv1-is-α1'Vv1*
ultimately show *?thesis*
by *simp*
qed
moreover
have *?LAMBDA* $\restriction E_{ES2} =$ *?T2* $\restriction V_{\mathcal{V}}$
proof −
from *propSepViews* **have** *?LAMBDA* $\restriction E_{ES2} = \alpha \restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** (*simp only*: *projection-sequence*)
moreover
from *α2''-in-E2star propSepViews* **have** *?T2* $\restriction V_{\mathcal{V}} =$ *?T2* $\restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note *α2'Vv2-is-αVv2 α2''Vv2-is-α2'Vv2*
ultimately show *?thesis*
by *simp*
qed
moreover
note *α1''Cv1-empty α2''Cv2-empty generalized-zipping-lemma*
ultimately obtain *t*
where *?TAU* @ $t \in Tr_{(ES1 \parallel ES2)}$
and $t \restriction V_{\mathcal{V}} =$ *?LAMBDA*
and $t \restriction C_{\mathcal{V}} = []$
by *blast*
moreover
have *set* $\delta' \subseteq N_{\mathcal{V}} \cap \Delta_{\Gamma}$
proof −
from *δ'-contains-only-δ1''-δ2''-events δ1''-in-N1-inter-Delta1star*

```
      δ2''-in-N2-inter-Delta2star
    have set δ' ⊆ N_V1 ∩ Δ_Γ1 ∪ N_V2 ∩ Δ_Γ2
      by auto
    with Delta1-N1-Delta2-N2-subset-Delta Nv1-union-Nv2-subsetof-Nv
    show ?thesis
      by auto
  qed
  ultimately have ∃α' γ'. (set γ' ⊆ N_V ∩ Δ_Γ ∧ β @ [c] @ γ' @ [v'] @ α' ∈ Tr_(ES1 ∥ ES2)
    ∧ α' ↑ V_V = α ↑ V_V ∧ α' ↑ C_V = [])
    by (simp only: append-assoc, blast)
}
moreover {
  assume Nv2-inter-Delta2-inter-E1-empty: N_V2 ∩ Δ_Γ2 ∩ E_ES1 = {}
    and  Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2: N_V1 ∩ Δ_Γ1 ∩ E_ES2 ⊆ ϒ_Γ2

  let ?ALPHA1''-DELTA1'' = ∃ α1'' δ1''. (
    set α1'' ⊆ E_ES1 ∧ set δ1'' ⊆ N_V1 ∩ Δ_Γ1
    ∧ β ↑ E_ES1 @ [c] ↑ E_ES1 @ δ1'' @ [v'] ↑ E_ES1 @ α1'' ∈ Tr_ES1
    ∧ α1'' ↑ V_V1 = α1' ↑ V_V1 ∧ α1'' ↑ C_V1 = [])

  from c-in-Cv-inter-Upsilon v'-in-Vv-inter-Nabla validV1
  have c ∉ E_ES1 ∨ (c ∈ E_ES1 ∧ v' ∉ E_ES1) ∨ (c ∈ E_ES1 ∧ v' ∈ E_ES1)
    by (simp add: isViewOn-def V-valid-def VC-disjoint-def
      VN-disjoint-def NC-disjoint-def)
  moreover {
    assume c-notin-E1: c ∉ E_ES1

    from validES1 βv'E1α1'-in-Tr1 have set α1' ⊆ E_ES1
      by (simp add: ES-valid-def traces-contain-events-def, auto)
    moreover
    have set [] ⊆ N_V1 ∩ Δ_Γ1
      by auto
    moreover
    from βv'E1α1'-in-Tr1 c-notin-E1
    have β ↑ E_ES1 @ [c] ↑ E_ES1 @ [] @ [v'] ↑ E_ES1 @ α1' ∈ Tr_ES1
      by (simp add: projection-def)
    moreover
    have α1' ↑ V_V1 = α1' ↑ V_V1 ..
    moreover
    note α1'Cv1-empty
    ultimately have ?ALPHA1''-DELTA1''
      by blast
  }
  moreover {
    assume c-in-E1: c ∈ E_ES1
      and  v'-notin-E1: v' ∉ E_ES1

    from c-in-E1 c-in-Cv-inter-Upsilon propSepViews
      Upsilon-inter-E1-subset-Upsilon1
    have c-in-Cv1-inter-Upsilon1: c ∈ C_V1 ∩ ϒ_Γ1
      unfolding properSeparationOfViews-def by auto
    hence c ∈ C_V1
```

by *auto*
moreover
from $\beta v' E1 \alpha 1'$*-in-Tr1* v'*-notin-E1* **have** $\beta \restriction E_{ES1} @ \alpha 1' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note $\alpha 1' Cv1$*-empty*
moreover
have *Adm* $\mathcal{V}1$ $\varrho 1$ Tr_{ES1} $(\beta \restriction E_{ES1})$ c
proof −
from *Adm* **obtain** γ
where $\gamma \varrho v$*-is-*$\beta \varrho v$: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and γc*-in-Tr*: $(\gamma @ [c]) \in Tr_{(ES1 \parallel ES2)}$
unfolding *Adm-def*
by *auto*

from *c-in-E1* γc*-in-Tr* **have** $(\gamma \restriction E_{ES1}) @ [c] \in Tr_{ES1}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES1} \restriction (\varrho 1\ \mathcal{V}1) = \beta \restriction E_{ES1} \restriction (\varrho 1\ \mathcal{V}1)$
proof −
from $\gamma \varrho v$*-is-*$\beta \varrho v$ **have** $\gamma \restriction E_{ES1} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES1} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with $\varrho 1 v1$*-subset-*ϱv*-inter-E1* **have** $\gamma \restriction (\varrho 1\ \mathcal{V}1) = \beta \restriction (\varrho 1\ \mathcal{V}1)$
by (*metis Int-subset-iff* $\gamma \varrho v$*-is-*$\beta \varrho v$ *projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *BSIA1*
ultimately obtain $\alpha 1''$
where *one*: $\beta \restriction E_{ES1} @ [c] @ \alpha 1'' \in Tr_{ES1}$
and *two*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *three*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
unfolding *BSIA-def*
by *blast*

from *one validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have *set* $[] \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
from *one c-in-E1* v'*-notin-E1*
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ [] @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *two three*
ultimately have *?ALPHA1''-DELTA1''*
by *blast*

```
}
moreover {
  assume c-in-E1: c ∈ E_ES1
    and  v'-in-E1: v' ∈ E_ES1

  from c-in-E1 c-in-Cv-inter-Upsilon propSepViews
    Upsilon-inter-E1-subset-Upsilon1
  have c-in-Cv1-inter-Upsilon1: c ∈ C_V1 ∩ ϒ_Γ1
    unfolding properSeparationOfViews-def by auto
  moreover
  from v'-in-E1 propSepViews v'-in-Vv-inter-Nabla
    Nabla-inter-E1-subset-Nabla1
  have v' ∈ V_V1 ∩ Nabla Γ1
    unfolding properSeparationOfViews-def by auto
  moreover
  from v'-in-E1  βv'E1α1'-in-Tr1 have β ↾ E_ES1 @ [v'] @ α1' ∈ Tr_ES1
    by (simp add: projection-def)
  moreover
  note α1'Cv1-empty
  moreover
  have Adm V1 ϱ1 Tr_ES1 (β ↾ E_ES1) c
  proof −
    from Adm obtain γ
      where γϱv-is-βϱv: γ ↾ (ϱ V) = β ↾ (ϱ V)
      and γc-in-Tr: (γ @ [c]) ∈ Tr_(ES1 ∥ ES2)
      unfolding Adm-def
      by auto

    from c-in-E1 γc-in-Tr have (γ ↾ E_ES1) @ [c] ∈ Tr_ES1
      by (simp add: projection-def composeES-def)
    moreover
    have γ ↾ E_ES1 ↾ (ϱ1 V1) = β ↾ E_ES1 ↾ (ϱ1 V1)
    proof −
      from γϱv-is-βϱv have γ ↾ E_ES1 ↾ (ϱ V) = β ↾ E_ES1 ↾ (ϱ V)
        by (metis projection-commute)
      with ϱ1v1-subset-ϱv-inter-E1 have γ ↾ (ϱ1 V1) = β ↾ (ϱ1 V1)
        by (metis Int-subset-iff γϱv-is-βϱv projection-subset-elim)
      thus ?thesis
        by (metis projection-commute)
    qed
    ultimately show ?thesis unfolding Adm-def
      by auto
  qed
  moreover
  note FCIA1
  ultimately obtain α1'' δ1''
    where one: set δ1'' ⊆ N_V1 ∩ Δ_Γ1
    and two: β ↾ E_ES1 @ [c] @ δ1'' @ [v'] @ α1'' ∈ Tr_ES1
    and three: α1'' ↾ V_V1 = α1' ↾ V_V1
    and four: α1'' ↾ C_V1 = []
    unfolding FCIA-def
    by blast
```

from *two validES1* **have** *set* $\alpha 1'' \subseteq E_{ES1}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
note *one*
moreover
from *two c-in-E1 v'-in-E1*
have $\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
by (*simp add*: *projection-def*)
moreover
note *three four*
ultimately have *?ALPHA1''-DELTA1''*
by *blast*
}
ultimately obtain $\alpha 1'' \delta 1''$
where *α1''-in-E1star*: *set* $\alpha 1'' \subseteq E_{ES1}$
and *δ1''-in-N1-inter-Delta1star*:*set* $\delta 1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *βE1-cE1-δ1''-v'E1-α1''-in-Tr1*:
$\beta \restriction E_{ES1} @ [c] \restriction E_{ES1} @ \delta 1'' @ [v'] \restriction E_{ES1} @ \alpha 1'' \in Tr_{ES1}$
and *α1''Vv1-is-α1'Vv1*: $\alpha 1'' \restriction V_{\mathcal{V}1} = \alpha 1' \restriction V_{\mathcal{V}1}$
and *α1''Cv1-empty*: $\alpha 1'' \restriction C_{\mathcal{V}1} = []$
by *blast*

from *c-in-Cv-inter-Upsilon Upsilon-inter-E2-subset-Upsilon2 propSepViews*
have *cE2-in-Cv2-inter-Upsilon2*: *set* $([c] \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def* **by** (*simp add*: *projection-def*, *auto*)

from *δ1''-in-N1-inter-Delta1star Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2*
propSepViews disjoint-Nv1-Vv2
have *δ1''E2-in-Cv2-inter-Upsilon2star*: *set* $(\delta 1'' \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
proof −
from *δ1''-in-N1-inter-Delta1star*
have *eq*: $\delta 1'' \restriction E_{ES2} = \delta 1'' \restriction (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2})$
by (*metis Int-commute Int-left-commute Int-lower2 Int-lower1*
projection-intersection-neutral subset-trans)

from *validV2 Nv1-inter-Delta1-inter-E2-subsetof-Upsilon2*
propSepViews disjoint-Nv1-Vv2
have $N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
thus *?thesis*
by (*subst eq*, *simp only*: *projection-def*, *auto*)
qed

have *cδ1''E2-in-Cv2-inter-Upsilon2star*: *set* $((c \# \delta 1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
proof −
from *cE2-in-Cv2-inter-Upsilon2 δ1''E2-in-Cv2-inter-Upsilon2star*
have *set* $(([c] @ \delta 1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
thus *?thesis*

by *auto*
qed

have $\exists\ \alpha2''\ \delta2''.\ set\ \alpha2'' \subseteq E_{ES2}$
$\wedge\ set\ \delta2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge\ \beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta2'' @ [v'] \restriction E_{ES2} @ \alpha2'' \in Tr_{ES2}$
$\wedge\ \alpha2'' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2} \wedge \alpha2'' \restriction C_{\mathcal{V}2} = []$
$\wedge\ \delta2'' \restriction E_{ES1} = \delta1'' \restriction E_{ES2}$
proof *cases*
assume *v'-in-E2*: $v' \in E_{ES2}$
with *Nabla-inter-E2-subset-Nabla2 propSepViews v'-in-Vv-inter-Nabla*
have *v'-in-Vv2-inter-Nabla2*: $v' \in V_{\mathcal{V}2} \cap Nabla\ \Gamma 2$
unfolding *properSeparationOfViews-def* **by** *auto*

have $[\![(\beta @ [v']) \restriction E_{ES2} @ \alpha2' \in Tr_{ES2}$;
$\alpha2' \restriction C_{\mathcal{V}2} = []$; $set\ ((c \#\ \delta1'') \restriction E_{ES2}) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$;
$c \in C_{\mathcal{V}} \cap \Upsilon_{\Gamma}$; $set\ \delta1'' \subseteq N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$;
$Adm\ \mathcal{V}\ \varrho\ (Tr_{(ES1\ \|\ ES2)})\ \beta\ c\ ]\!]$
$\Longrightarrow \exists\ \alpha2''\ \delta2''.$
$(set\ \alpha2'' \subseteq E_{ES2} \wedge set\ \delta2'' \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
$\wedge\ \beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta2'' @ [v'] \restriction E_{ES2} @ \alpha2'' \in Tr_{ES2}$
$\wedge\ \alpha2'' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2} \wedge \alpha2'' \restriction C_{\mathcal{V}2} = []$
$\wedge\ \delta2'' \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \delta1'' \restriction E_{ES2})$
proof (*induct length* $((c \#\ \delta1'') \restriction E_{ES2})$ *arbitrary*: $\beta\ \alpha2'\ c\ \delta1''$)
case *0*

from *0(2) validES2* **have** $set\ \alpha2' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
have $set\ [] \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ [] @ [v'] \restriction E_{ES2} @ \alpha2' \in Tr_{ES2}$
proof −
note *0(2)*
moreover
from *0(1)* **have** $c \notin E_{ES2}$
by (*simp add*: *projection-def*, *auto*)
ultimately show *?thesis*
by (*simp add*: *projection-concatenation-commute projection-def*)
qed
moreover
have $\alpha2' \restriction V_{\mathcal{V}2} = \alpha2' \restriction V_{\mathcal{V}2}$ ..
moreover
note *0(3)*
moreover
from *0(1)* **have** $[] \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \delta1'' \restriction E_{ES2}$
by (*simp add*: *projection-def*, *split if-split-asm*, *auto*)
ultimately show *?case*
by *blast*
next

case (*Suc n*)

from *projection-split-last*[*OF Suc*(*2*)] **obtain** $\mu\ c'\ \nu$
  where *c′-in-E2*: $c' \in E_{ES2}$
  and *cδ1′′-is-μc′ν*: $c\ \#\ \delta 1'' = \mu\ @\ [c']\ @\ \nu$
  and *νE2-empty*: $\nu \restriction E_{ES2} = []$
  and *n-is-length-μνE2*: $n = length\ ((\mu\ @\ \nu) \restriction E_{ES2})$
  by *blast*

from *Suc*(*5*) *c′-in-E2 cδ1′′-is-μc′ν* **have** *set* $(\mu \restriction E_{ES2}\ @\ [c']) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
  by (*simp only*: *cδ1′′-is-μc′ν projection-concatenation-commute projection-def*, *auto*)
hence *c′-in-Cv2-inter-Upsilon2*: $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
  by *auto*
hence *c′-in-Cv2*: $c' \in C_{\mathcal{V}2}$ **and** *c′-in-Upsilon2*: $c' \in \Upsilon_{\Gamma 2}$
  by *auto*
with *validV2* **have** *c′-in-E2*: $c' \in E_{ES2}$
  by (*simp add*: *isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*, *auto*)

show *?case*
  proof (*cases* μ)
    case *Nil*
    with *cδ1′′-is-μc′ν* **have** *c-is-c′*: $c = c'$ **and** *δ1′′-is-ν*: $\delta 1'' = \nu$
      by *auto*
    with *c′-in-Cv2-inter-Upsilon2* **have** $c \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
      by *simp*
    moreover
    note *v′-in-Vv2-inter-Nabla2*
    moreover
    from *v′-in-E2 Suc*(*3*) **have** $(\beta \restriction E_{ES2})\ @\ [v']\ @\ \alpha 2' \in Tr_{ES2}$
      by (*simp add*: *projection-concatenation-commute projection-def*)
    moreover
    note *Suc*(*4*)
    moreover
    have *Adm* $\mathcal{V}2\ \varrho 2\ Tr_{ES2}\ (\beta \restriction E_{ES2})\ c$
      proof −
        from *Suc*(*8*) **obtain** γ
          where *γϱv-is-βϱv*: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
          and *γc-in-Tr*: $(\gamma\ @\ [c]) \in Tr_{(ES1\ \|\ ES2)}$
          unfolding *Adm-def*
          by *auto*

        from *c-is-c′ c′-in-E2 γc-in-Tr* **have** $(\gamma \restriction E_{ES2})\ @\ [c] \in Tr_{ES2}$
          by (*simp add*: *projection-def composeES-def*)
        moreover
        have $\gamma \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2)$
        proof −
          from *γϱv-is-βϱv* **have** $\gamma \restriction E_{ES2} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES2} \restriction (\varrho\ \mathcal{V})$
            by (*metis projection-commute*)
          with *ϱ2v2-subset-ϱv-inter-E2* **have** $\gamma \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction (\varrho 2\ \mathcal{V}2)$
            by (*metis Int-subset-iff γϱv-is-βϱv projection-subset-elim*)

thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *FCIA2*
ultimately obtain $\alpha 2''\ \gamma$
where *one*: *set* $\gamma \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2}$
and *two*: $\beta \restriction E_{ES2} @ [c] @ \gamma @ [v'] @ \alpha 2'' \in Tr_{ES2}$
and *three*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *four*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *FCIA-def*
by *blast*

let *?DELTA2''* $= \nu \restriction E_{ES2} @ \gamma$

from *two validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)
moreover
from *one* ν*E2-empty*
have *set ?DELTA2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @$ *?DELTA2''* $@ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
proof −
from *c-is-c' c'-in-E2* **have** $[c] = [c] \restriction E_{ES2}$
by (*simp add*: *projection-def*)
moreover
from *v'-in-E2* **have** $[v'] = [v'] \restriction E_{ES2}$
by (*simp add*: *projection-def*)
moreover
note ν*E2-empty two*
ultimately show *?thesis*
by *auto*
qed
moreover
note *three four*
moreover
have *?DELTA2''* $\restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \delta 1'' \restriction E_{ES2}$
proof −
have $\gamma \restriction (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = []$
proof −
from *validV2* **have** $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}) = \{\}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
with *projection-intersection-neutral*[*OF one*, *of* $C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$]
show *?thesis*
by (*simp add*: *projection-def*)
qed

```
    with δ1''-is-ν νE2-empty show ?thesis
      by (simp add: projection-concatenation-commute)
  qed
 ultimately show ?thesis
  by blast
next
 case (Cons x xs)
 with cδ1''-is-μc'ν
 have μ-is-c-xs: μ = [c] @ xs and δ1''-is-xs-c'-ν: δ1'' = xs @ [c'] @ ν
  by auto
 with n-is-length-μνE2 have n = length ((c # (xs @ ν)) ↾ E_ES2)
  by auto
 moreover
 note Suc(3,4)
 moreover
 have set ((c # (xs @ ν)) ↾ E_ES2) ⊆ C_V2 ∩ ϒ_Γ2
  proof −
   have res: c # (xs @ ν) = [c] @ (xs @ ν)
    by auto

   from Suc(5) cδ1''-is-μc'ν μ-is-c-xs νE2-empty
   show ?thesis
    by (subst res, simp only: cδ1''-is-μc'ν
      projection-concatenation-commute set-append, auto)
  qed
 moreover
 note Suc(6)
 moreover
 from Suc(7) δ1''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V1 ∩ Δ_Γ1
  by auto
 moreover note Suc(8) Suc(1)[of c xs @ ν β α2']
 ultimately obtain δ γ
  where one: set δ ⊆ E_ES2
  and two: set γ ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
  and three: β ↾ E_ES2 @ [c] ↾ E_ES2 @ γ @ [v'] ↾ E_ES2 @ δ ∈ Tr_ES2
  and four: δ ↾ V_V2 = α2' ↾ V_V2
  and five: δ ↾ C_V2 = []
  and six: γ ↾ (C_V2 ∩ ϒ_Γ2) = (xs @ ν) ↾ E_ES2
  by blast


 let ?BETA = β ↾ E_ES2 @ [c] ↾ E_ES2 @ γ

 note c'-in-Cv2-inter-Upsilon2 v'-in-Vv2-inter-Nabla2
 moreover
 from three v'-in-E2 have ?BETA @ [v'] @ δ ∈ Tr_ES2
  by (simp add: projection-def)
 moreover
 note five
 moreover
 have Adm V2 ϱ2 Tr_ES2 ?BETA c'
  proof −
```

```
    have ?BETA @ [c'] ∈ Tr_ES2
      proof −
        from Suc(7) c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν
        have c' ∈ C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
          by auto
        moreover
        from validES2 three have ?BETA ∈ Tr_ES2
          by (unfold ES-valid-def traces-prefixclosed-def
            prefixclosed-def prefix-def, auto)
        moreover
        note total-ES2-C2-inter-Upsilon2-inter-N1-inter-Delta1
        ultimately show ?thesis
          unfolding total-def
          by blast
      qed
    thus ?thesis
      unfolding Adm-def
      by blast
  qed
moreover
note FCIA2
ultimately obtain α2'' δ'
  where fcia-one: set δ' ⊆ N_V2 ∩ Δ_Γ2
  and fcia-two: ?BETA @ [c'] @ δ' @ [v'] @ α2'' ∈ Tr_ES2
  and fcia-three: α2'' ↑ V_V2 = δ ↑ V_V2
  and fcia-four: α2'' ↑ C_V2 = []
  unfolding FCIA-def
  by blast

let ?DELTA2'' = γ @ [c'] @ δ'

from fcia-two validES2 have set α2'' ⊆ E_ES2
  by (simp add: ES-valid-def traces-contain-events-def, auto)
moreover
have set ?DELTA2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
  proof −
    from Suc(7) c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν
    have c' ∈ C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
      by auto
    with two fcia-one show ?thesis
      by auto
  qed
moreover
from fcia-two v'-in-E2
have β ↑ E_ES2 @ [c] ↑ E_ES2 @ ?DELTA2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
  by (simp add: projection-def)
moreover
from fcia-three four have α2'' ↑ V_V2 = α2' ↑ V_V2
  by simp
moreover
note fcia-four
moreover
```

```
      have ?DELTA2'' ⌊ (C_V2 ∩ Υ_Γ2) = δ1'' ⌊ E_ES2
        proof −
          have δ' ⌊ (C_V2 ∩ Υ_Γ2) = []
            proof −
              from fcia-one have ∀ e ∈ set δ'. e ∈ N_V2 ∩ Δ_Γ2
                by auto
              with validV2 have ∀ e ∈ set δ'. e ∉ C_V2 ∩ Υ_Γ2
                by (simp add:isViewOn-def V-valid-def
                  VC-disjoint-def VN-disjoint-def NC-disjoint-def, auto)
              thus ?thesis
                by (simp add: projection-def)
            qed
          with c'-in-E2 c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν νE2-empty six
          show ?thesis
            by (simp only: projection-concatenation-commute projection-def, auto)
        qed
        ultimately show ?thesis
          by blast
      qed
qed
from this[OF βv'E2α2'-in-Tr2 α2'Cv2-empty
  cδ1''E2-in-Cv2-inter-Upsilon2star c-in-Cv-inter-Upsilon δ1''-in-N1-inter-Delta1star Adm]
obtain α2'' δ2''
  where one: set α2'' ⊆ E_ES2
  and two: set δ2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
  and three: β ⌊ E_ES2 @ [c] ⌊ E_ES2 @ δ2'' @ [v'] ⌊ E_ES2 @ α2'' ∈ Tr_ES2
  ∧ α2'' ⌊ V_V2 = α2' ⌊ V_V2 ∧ α2'' ⌊ C_V2 = []
  and four: δ2'' ⌊ (C_V2 ∩ Υ_Γ2) = δ1'' ⌊ E_ES2
  by blast

note one two three
moreover
have δ2'' ⌊ E_ES1 = δ1'' ⌊ E_ES2
  proof −
    from projection-intersection-neutral[OF two, of E_ES1]
      Nv2-inter-Delta2-inter-E1-empty validV1
    have δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1 ∩ E_ES1)
      by (simp only: Int-Un-distrib2, auto)
    moreover
    from validV1
    have C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1 ∩ E_ES1 = C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1
      by (simp add: isViewOn-def V-valid-def VC-disjoint-def
        VN-disjoint-def NC-disjoint-def, auto)
    ultimately have δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2 ∩ N_V1 ∩ Δ_Γ1)
      by simp
    hence δ2'' ⌊ E_ES1 = δ2'' ⌊ (C_V2 ∩ Υ_Γ2) ⌊ (N_V1 ∩ Δ_Γ1)
      by (simp add: projection-def)
    with four have δ2'' ⌊ E_ES1 = δ1'' ⌊ E_ES2 ⌊ (N_V1 ∩ Δ_Γ1)
      by simp
    hence δ2'' ⌊ E_ES1 = δ1'' ⌊ (N_V1 ∩ Δ_Γ1) ⌊ E_ES2
      by (simp only: projection-commute)
    with δ1''-in-N1-inter-Delta1star show ?thesis
```

```
      by (simp only: list-subset-iff-projection-neutral)
    qed
  ultimately show ?thesis
    by blast
next
  assume v'-notin-E2: v' ∉ E_ES2

  have ⟦ (β @ [v']) ↑ E_ES2 @ α2' ∈ Tr_ES2 ;
   α2' ↑ C_V2 = []; set ((c # δ1'') ↑ E_ES2) ⊆ C_V2 ∩ ϒ_Γ2 ;
    c ∈ C_V ∩ ϒ_Γ ; set δ1'' ⊆ N_V1 ∩ Δ_Γ1;
   Adm V ϱ (Tr_(ES1 ∥ ES2)) β c ⟧
   ⟹ ∃ α2'' δ2''.
    (set α2'' ⊆ E_ES2 ∧ set δ2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
    ∧ β ↑ E_ES2 @ [c] ↑ E_ES2 @ δ2'' @ [v'] ↑ E_ES2 @ α2'' ∈ Tr_ES2
    ∧ α2'' ↑ V_V2 = α2' ↑ V_V2 ∧ α2'' ↑ C_V2 = []
   ∧ δ2'' ↑ E_ES1 = δ1'' ↑ E_ES2)
   proof (induct length ((c # δ1'') ↑ E_ES2) arbitrary: β α2' c δ1'')
     case 0

     from 0(2) validES2 have set α2' ⊆ E_ES2
       by (simp add: ES-valid-def traces-contain-events-def, auto)
     moreover
     have set [] ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
       by auto
     moreover
     have β ↑ E_ES2 @ [c] ↑ E_ES2 @ [] @ [v'] ↑ E_ES2 @ α2' ∈ Tr_ES2
       proof −
         note 0(2)
         moreover
         from 0(1) have c ∉ E_ES2
           by (simp add: projection-def, auto)
         ultimately show ?thesis
           by (simp add: projection-concatenation-commute projection-def)
       qed
     moreover
     have α2' ↑ V_V2 = α2' ↑ V_V2 ..
     moreover
     note 0(3)
     moreover
     from 0(1) have [] ↑ E_ES1 = δ1'' ↑ E_ES2
       by (simp add: projection-def, split if-split-asm, auto)
     ultimately show ?case
       by blast
   next
     case (Suc n)

     from projection-split-last[OF Suc(2)] obtain μ c' ν
       where c'-in-E2: c' ∈ E_ES2
       and cδ1''-is-μc'ν: c # δ1'' = μ @ [c'] @ ν
       and νE2-empty: ν ↑ E_ES2 = []
       and n-is-length-μνE2: n = length ((μ @ ν) ↑ E_ES2)
       by blast
```

from *Suc(5) c′-in-E2 cδ1′′-is-μc′ν* **have** *set* $(\mu \restriction E_{ES2} @ [c']) \subseteq C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by (*simp only*: *cδ1′′-is-μc′ν projection-concatenation-commute projection-def*, *auto*)
hence *c′-in-Cv2-inter-Upsilon2*: $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2}$
by *auto*
hence *c′-in-Cv2*: $c' \in C_{\mathcal{V}2}$ **and** *c′-in-Upsilon2*: $c' \in \Upsilon_{\Gamma 2}$
by *auto*
with *validV2* **have** *c′-in-E2*: $c' \in E_{ES2}$
by (*simp add*:*isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)

show *?case*
proof (*cases* μ)
case *Nil*
with *cδ1′′-is-μc′ν* **have** *c-is-c′*: $c = c'$ **and** *δ1′′-is-ν*: $\delta 1'' = \nu$
by *auto*
with *c′-in-Cv2-inter-Upsilon2* **have** $c \in C_{\mathcal{V}2}$
by *simp*
moreover
from *v′-notin-E2 Suc(3)* **have** $(\beta \restriction E_{ES2}) @ \alpha 2' \in Tr_{ES2}$
by (*simp add*: *projection-concatenation-commute projection-def*)
moreover
note *Suc(4)*
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} $(\beta \restriction E_{ES2})$ c
proof −
from *Suc(8)* **obtain** γ
where *γϱv-is-βϱv*: $\gamma \restriction (\varrho\ \mathcal{V}) = \beta \restriction (\varrho\ \mathcal{V})$
and *γc-in-Tr*: $(\gamma @ [c]) \in Tr_{(ES1 \parallel ES2)}$
unfolding *Adm-def*
by *auto*

from *c-is-c′ c′-in-E2 γc-in-Tr* **have** $(\gamma \restriction E_{ES2}) @ [c] \in Tr_{ES2}$
by (*simp add*: *projection-def composeES-def*)
moreover
have $\gamma \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction E_{ES2} \restriction (\varrho 2\ \mathcal{V}2)$
proof −
from *γϱv-is-βϱv* **have** $\gamma \restriction E_{ES2} \restriction (\varrho\ \mathcal{V}) = \beta \restriction E_{ES2} \restriction (\varrho\ \mathcal{V})$
by (*metis projection-commute*)
with *ϱ2v2-subset-ϱv-inter-E2*
have $\gamma \restriction (\varrho 2\ \mathcal{V}2) = \beta \restriction (\varrho 2\ \mathcal{V}2)$
by (*metis Int-subset-iff γϱv-is-βϱv projection-subset-elim*)
thus *?thesis*
by (*metis projection-commute*)
qed
ultimately show *?thesis* **unfolding** *Adm-def*
by *auto*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *one*: $(\beta \restriction E_{ES2}) @ [c] @ \alpha 2'' \in Tr_{ES2}$

```
    and two: α2'' ↓ V_V2 = α2' ↓ V_V2
    and three: α2'' ↓ C_V2 = []
    unfolding BSIA-def
    by blast

  let ?DELTA2'' = ν ↓ E_ES2

  from one validES2 have set α2'' ⊆ E_ES2
    by (simp add: ES-valid-def traces-contain-events-def, auto)
  moreover
  from νE2-empty
  have set ?DELTA2'' ⊆ N_V2 ∩ Δ_Γ2 ∪ C_V2 ∩ ϒ_Γ2 ∩ N_V1 ∩ Δ_Γ1
    by simp
  moreover
  from c-is-c' c'-in-E2 one v'-notin-E2 νE2-empty
  have (β ↓ E_ES2) @ [c] ↓ E_ES2 @ ?DELTA2'' @ [v'] ↓ E_ES2 @ α2'' ∈ Tr_ES2
    by (simp add: projection-def)
  moreover
  note two three
  moreover
  from νE2-empty δ1''-is-ν have ?DELTA2'' ↓ E_ES1 = δ1'' ↓ E_ES2
    by (simp add: projection-def)
  ultimately show ?thesis
    by blast
next
  case (Cons x xs)
   with cδ1''-is-μc'ν have μ-is-c-xs: μ = [c] @ xs
    and δ1''-is-xs-c'-ν: δ1'' = xs @ [c'] @ ν
    by auto
  with n-is-length-μνE2 have n = length ((c # (xs @ ν)) ↓ E_ES2)
    by auto
  moreover
  note Suc(3,4)
  moreover
  have set ((c # (xs @ ν)) ↓ E_ES2) ⊆ C_V2 ∩ ϒ_Γ2
    proof −
      have res: c # (xs @ ν) = [c] @ (xs @ ν)
        by auto

      from Suc(5) cδ1''-is-μc'ν μ-is-c-xs νE2-empty
      show ?thesis
        by (subst res, simp only: cδ1''-is-μc'ν
          projection-concatenation-commute set-append, auto)
    qed
  moreover
  note Suc(6)
  moreover
  from Suc(7) δ1''-is-xs-c'-ν have set (xs @ ν) ⊆ N_V1 ∩ Δ_Γ1
    by auto
  moreover note Suc(8) Suc(1)[of c xs @ ν β α2']
  ultimately obtain δ γ
    where one: set δ ⊆ E_ES2
```

and *two*: *set* $\gamma \subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *three*: $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \gamma @ [v'] \restriction E_{ES2} @ \delta \in Tr_{ES2}$
and *four*: $\delta \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *five*: $\delta \restriction C_{\mathcal{V}2} = []$
and *six*: $\gamma \restriction E_{ES1} = (xs @ \nu) \restriction E_{ES2}$
by *blast*

let *?BETA* $= \beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \gamma$

from *c'-in-Cv2-inter-Upsilon2* **have** $c' \in C_{\mathcal{V}2}$
by *auto*
moreover
from *three v'-notin-E2* **have** *?BETA* $@ \delta \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
note *five*
moreover
have *Adm* $\mathcal{V}2$ $\varrho 2$ Tr_{ES2} *?BETA* c'
proof −
have *?BETA* $@ [c'] \in Tr_{ES2}$
proof −
from *Suc*(7) *c'-in-Cv2-inter-Upsilon2* *δ1''-is-xs-c'-ν*
have $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
moreover
from *validES2 three* **have** *?BETA* $\in Tr_{ES2}$
by (*unfold ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*, *auto*)
moreover
note *total-ES2-C2-inter-Upsilon2-inter-N1-inter-Delta1*
ultimately show *?thesis*
unfolding *total-def*
by *blast*
qed
thus *?thesis*
unfolding *Adm-def*
by *blast*
qed
moreover
note *BSIA2*
ultimately obtain $\alpha 2''$
where *bsia-one*: *?BETA* $@ [c'] @ \alpha 2'' \in Tr_{ES2}$
and *bsia-two*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \delta \restriction V_{\mathcal{V}2}$
and *bsia-three*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
unfolding *BSIA-def*
by *blast*

let *?DELTA2''* $= \gamma @ [c']$

from *bsia-one validES2* **have** *set* $\alpha 2'' \subseteq E_{ES2}$
by (*simp add*: *ES-valid-def traces-contain-events-def*, *auto*)

moreover
have *set ?DELTA2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
proof −
from *Suc*(7) *c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν*
have $c' \in C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
with *two* **show** *?thesis*
by *auto*
qed
moreover
from *bsia-one v'-notin-E2*
have $\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @$ *?DELTA2''* $@ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
by (*simp add*: *projection-def*)
moreover
from *bsia-two four* **have** $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
by *simp*
moreover
note *bsia-three*
moreover
have *?DELTA2''* $\restriction E_{ES1} = \delta 1'' \restriction E_{ES2}$
proof −
from *validV1 Suc*(7) *δ1''-is-xs-c'-ν* **have** $c' \in E_{ES1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
with *c'-in-E2 c'-in-Cv2-inter-Upsilon2 δ1''-is-xs-c'-ν νE2-empty six*
show *?thesis*
by (*simp only*: *projection-concatenation-commute projection-def*, *auto*)
qed
ultimately show *?thesis*
by *blast*
qed
qed
from *this*[*OF βv'E2α2'-in-Tr2 α2'Cv2-empty cδ1''E2-in-Cv2-inter-Upsilon2star*
c-in-Cv-inter-Upsilon δ1''-in-N1-inter-Delta1star Adm]
show *?thesis*
by *blast*
qed
then obtain *α2'' δ2''*
where *α2''-in-E2star*: *set α2''* $\subseteq E_{ES2}$
and *δ2''-in-N2-inter-Delta2star*:*set δ2''* $\subseteq N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
and *βE2-cE2-δ2''-v'E2-α2''-in-Tr2*:
$\beta \restriction E_{ES2} @ [c] \restriction E_{ES2} @ \delta 2'' @ [v'] \restriction E_{ES2} @ \alpha 2'' \in Tr_{ES2}$
and *α2''Vv2-is-α2'Vv2*: $\alpha 2'' \restriction V_{\mathcal{V}2} = \alpha 2' \restriction V_{\mathcal{V}2}$
and *α2''Cv2-empty*: $\alpha 2'' \restriction C_{\mathcal{V}2} = []$
and *δ2''E1-is-δ1''E2*: $\delta 2'' \restriction E_{ES1} = \delta 1'' \restriction E_{ES2}$
by *blast*

from *βE2-cE2-δ2''-v'E2-α2''-in-Tr2 βE1-cE1-δ1''-v'E1-α1''-in-Tr1*
validES2 validES1
have *δ2''-in-E2star*: *set δ2''* $\subseteq E_{ES2}$ **and** *δ1''-in-E1star*: *set δ1''* $\subseteq E_{ES1}$
by (*simp-all add*: *ES-valid-def traces-contain-events-def*, *auto*)
with *δ2''E1-is-δ1''E2 merge-property*[*of δ2'' E*$_{ES2}$ *δ1'' E*$_{ES1}$] **obtain** *δ'*

where $\delta'E2\text{-}is\text{-}\delta2''$: $\delta' \restriction E_{ES2} = \delta2''$
and $\delta'E1\text{-}is\text{-}\delta1''$: $\delta' \restriction E_{ES1} = \delta1''$
and $\delta'\text{-}contains\text{-}only\text{-}\delta2''\text{-}\delta1''\text{-}events$: $set\ \delta' \subseteq set\ \delta2'' \cup set\ \delta1''$
unfolding *Let-def*
by *auto*

let $?TAU = \beta\ @\ [c]\ @\ \delta'\ @\ [v']$
let $?LAMBDA = \alpha \restriction V_{\mathcal{V}}$
let $?T2 = \alpha2''$
let $?T1 = \alpha1''$

have $?TAU \in Tr_{(ES1 \parallel ES2)}$
proof −
from $\beta E2\text{-}cE2\text{-}\delta2''\text{-}v'E2\text{-}\alpha2''\text{-}in\text{-}Tr2\ \delta'E2\text{-}is\text{-}\delta2''\ validES2$
have $\beta \restriction E_{ES2}\ @\ [c] \restriction E_{ES2}\ @\ \delta' \restriction E_{ES2}\ @\ [v'] \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence $(\beta\ @\ [c]\ @\ \delta'\ @\ [v']) \restriction E_{ES2} \in Tr_{ES2}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta E1\text{-}cE1\text{-}\delta1''\text{-}v'E1\text{-}\alpha1''\text{-}in\text{-}Tr1\ \delta'E1\text{-}is\text{-}\delta1''\ validES1$
have $\beta \restriction E_{ES1}\ @\ [c] \restriction E_{ES1}\ @\ \delta' \restriction E_{ES1}\ @\ [v'] \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *ES-valid-def traces-prefixclosed-def prefixclosed-def prefix-def*)
hence $(\beta\ @\ [c]\ @\ \delta'\ @\ [v']) \restriction E_{ES1} \in Tr_{ES1}$
by (*simp add*: *projection-def*, *auto*)
moreover
from $\beta v'\alpha\text{-}in\text{-}Tr\ c\text{-}in\text{-}Cv\text{-}inter\text{-}Upsilon\ VIsViewOnE\ \delta'\text{-}contains\text{-}only\text{-}\delta2''\text{-}\delta1''\text{-}events\ \delta2''\text{-}in\text{-}E2star\ \delta1''\text{-}in\text{-}E1star$
have $set\ (\beta\ @\ [c]\ @\ \delta'\ @\ [v']) \subseteq E_{ES2} \cup E_{ES1}$
unfolding *composeES-def isViewOn-def V-valid-def VC-disjoint-def VN-disjoint-def NC-disjoint-def*
by *auto*
ultimately show *?thesis*
unfolding *composeES-def*
by *auto*
qed
hence $set\ ?TAU \subseteq E_{(ES1 \parallel ES2)}$
unfolding *composeES-def*
by *auto*
moreover
have $set\ ?LAMBDA \subseteq V_{\mathcal{V}}$
by (*simp add*: *projection-def*, *auto*)
moreover
note $\alpha2''\text{-}in\text{-}E2star\ \alpha1''\text{-}in\text{-}E1star$
moreover
from $\beta E2\text{-}cE2\text{-}\delta2''\text{-}v'E2\text{-}\alpha2''\text{-}in\text{-}Tr2\ \delta'E2\text{-}is\text{-}\delta2''$
have $?TAU \restriction E_{ES2}\ @\ ?T2 \in Tr_{ES2}$
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
from $\beta E1\text{-}cE1\text{-}\delta1''\text{-}v'E1\text{-}\alpha1''\text{-}in\text{-}Tr1\ \delta'E1\text{-}is\text{-}\delta1''$

have *?TAU* ↾ E_{ES1} @ *?T1* ∈ Tr_{ES1}
by (*simp only*: *projection-concatenation-commute*, *auto*)
moreover
have *?LAMBDA* ↾ E_{ES2} = *?T2* ↾ $V_{\mathcal{V}}$
proof −
from *propSepViews* **have** *?LAMBDA* ↾ E_{ES2} = α ↾ $V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def* **by** (*simp only*: *projection-sequence*)
moreover
from $\alpha 2''$*-in-E2star propSepViews* **have** *?T2* ↾ $V_{\mathcal{V}}$ = *?T2* ↾ $V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note $\alpha 2'$*Vv2-is-*α*Vv2* $\alpha 2''$*Vv2-is-*$\alpha 2'$*Vv2*
ultimately show *?thesis*
by *simp*
qed
moreover
have *?LAMBDA* ↾ E_{ES1} = *?T1* ↾ $V_{\mathcal{V}}$
proof −
from *propSepViews* **have** *?LAMBDA* ↾ E_{ES1} = α ↾ $V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def* **by** (*simp only*: *projection-sequence*)
moreover
from $\alpha 1''$*-in-E1star propSepViews* **have** *?T1* ↾ $V_{\mathcal{V}}$ = *?T1* ↾ $V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*metis Int-commute projection-intersection-neutral*)
moreover
note $\alpha 1'$*Vv1-is-*α*Vv1* $\alpha 1''$*Vv1-is-*$\alpha 1'$*Vv1*
ultimately show *?thesis*
by *simp*
qed
moreover
note $\alpha 2''$*Cv2-empty* $\alpha 1''$*Cv1-empty generalized-zipping-lemma*
ultimately obtain *t*
where *?TAU* @ *t* ∈ $Tr_{(ES1 \parallel ES2)}$
and *t* ↾ $V_{\mathcal{V}}$ = *?LAMBDA*
and *t* ↾ $C_{\mathcal{V}}$ = []
by *blast*
moreover
have *set* δ' ⊆ $N_{\mathcal{V}} \cap \Delta_{\Gamma}$
proof −
from δ'*-contains-only-*$\delta 2''$*-*$\delta 1''$*-events*
$\delta 2''$*-in-N2-inter-Delta2star* $\delta 1''$*-in-N1-inter-Delta1star*
have *set* δ' ⊆ $N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cup N_{\mathcal{V}1} \cap \Delta_{\Gamma 1}$
by *auto*
with *Delta1-N1-Delta2-N2-subset-Delta Nv1-union-Nv2-subsetof-Nv* **show** *?thesis*
by *auto*
qed
ultimately have ∃ α' γ'. (*set* γ' ⊆ $N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge \beta$ @ [*c*] @ γ' @ [v'] @ α' ∈ $Tr_{(ES1 \parallel ES2)}$
∧ α' ↾ $V_{\mathcal{V}}$ = α ↾ $V_{\mathcal{V}} \wedge \alpha'$ ↾ $C_{\mathcal{V}}$ = [])
by (*simp only*: *append-assoc*, *blast*)
}
ultimately have ∃ α' γ'. (*set* γ' ⊆ $N_{\mathcal{V}} \cap \Delta_{\Gamma} \wedge \beta$ @ [*c*] @ γ' @ [v'] @ α' ∈ $Tr_{(ES1 \parallel ES2)}$

```
    ∧ α' ↓ V_V = α ↓ V_V ∧ α' ↓ C_V = [])
    by blast
  }
  thus ?thesis
    unfolding FCIA-def
    by blast
qed


theorem compositionality-R:
⟦ R V1 Tr_ES1; R V2 Tr_ES2 ⟧ ⟹ R V (Tr_(ES1 ∥ ES2))
  proof −
    assume R1: R V1 Tr_ES1
    and R2: R V2 Tr_ES2

    {
      fix τ'
      assume τ'-in-Tr: τ' ∈ Tr_(ES1 ∥ ES2)
      hence τ'E1-in-Tr1: τ' ↓ E_ES1 ∈ Tr_ES1
        and τ'E2-in-Tr2: τ' ↓ E_ES2 ∈ Tr_ES2
        unfolding composeES-def
        by auto
      with R1 R2 obtain τ1' τ2'
        where τ1'-in-Tr1: τ1' ∈ Tr_ES1
        and τ1'Cv1-empty: τ1' ↓ C_V1 = []
        and τ1'Vv1-is-τ'-E1-Vv1: τ1' ↓ V_V1 = τ' ↓ E_ES1 ↓ V_V1
        and τ2'-in-Tr2: τ2' ∈ Tr_ES2
        and τ2'Cv2-empty: τ2' ↓ C_V2 = []
        and τ2'Vv2-is-τ'-E2-Vv2: τ2' ↓ V_V2 = τ' ↓ E_ES2 ↓ V_V2
        unfolding R-def
        by blast

      have set [] ⊆ E_(ES1 ∥ ES2)
        by auto
      moreover
      have set (τ' ↓ V_V) ⊆ V_V
        by (simp add: projection-def, auto)
      moreover
      from validES1 τ1'-in-Tr1 have τ1'-in-E1: set τ1' ⊆ E_ES1
        by (simp add: ES-valid-def traces-contain-events-def, auto)
      moreover
      from validES2 τ2'-in-Tr2 have τ2'-in-E2: set τ2' ⊆ E_ES2
        by (simp add: ES-valid-def traces-contain-events-def, auto)
      moreover
      from τ1'-in-Tr1 have [] ↓ E_ES1 @ τ1' ∈ Tr_ES1
        by (simp add: projection-def)
      moreover
      from τ2'-in-Tr2 have [] ↓ E_ES2 @ τ2' ∈ Tr_ES2
        by (simp add: projection-def)
      moreover
      have τ' ↓ V_V ↓ E_ES1 = τ1' ↓ V_V
        proof −
```

from *projection-intersection-neutral*[*OF* $\tau 1'$*-in-E1*, *of* $V_{\mathcal{V}}$] *propSepViews*
have $\tau 1' \restriction V_{\mathcal{V}} = \tau 1' \restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *Int-commute*)
moreover
from *propSepViews* **have** $\tau' \restriction V_{\mathcal{V}} \restriction E_{ES1} = \tau' \restriction V_{\mathcal{V}1}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *projection-sequence*)
moreover {
have $\tau' \restriction E_{ES1} \restriction V_{\mathcal{V}1} = \tau' \restriction (E_{ES1} \cap V_{\mathcal{V}1})$
by (*simp add*: *projection-def*)
moreover
from *validV1* **have** $E_{ES1} \cap V_{\mathcal{V}1} = V_{\mathcal{V}1}$
by (*simp add*: *isViewOn-def V-valid-def*
VC-disjoint-def VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\tau' \restriction E_{ES1} \restriction V_{\mathcal{V}1} = \tau' \restriction V_{\mathcal{V}1}$
by *simp*
}
moreover
note $\tau 1' Vv1\text{-}is\text{-}\tau'\text{-}E1\text{-}Vv1$
ultimately show *?thesis*
by *simp*
qed
moreover
have $\tau' \restriction V_{\mathcal{V}} \restriction E_{ES2} = \tau 2' \restriction V_{\mathcal{V}}$
proof −
from *projection-intersection-neutral*[*OF* $\tau 2'$*-in-E2*, *of* $V_{\mathcal{V}}$] *propSepViews*
have $\tau 2' \restriction V_{\mathcal{V}} = \tau 2' \restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *Int-commute*)
moreover
from *propSepViews* **have** $\tau' \restriction V_{\mathcal{V}} \restriction E_{ES2} = \tau' \restriction V_{\mathcal{V}2}$
unfolding *properSeparationOfViews-def*
by (*simp add*: *projection-sequence*)
moreover {
have $\tau' \restriction E_{ES2} \restriction V_{\mathcal{V}2} = \tau' \restriction (E_{ES2} \cap V_{\mathcal{V}2})$
by (*simp add*: *projection-def*)
moreover
from *validV2* **have** $E_{ES2} \cap V_{\mathcal{V}2} = V_{\mathcal{V}2}$
by (*simp add:isViewOn-def V-valid-def VC-disjoint-def*
VN-disjoint-def NC-disjoint-def, *auto*)
ultimately have $\tau' \restriction E_{ES2} \restriction V_{\mathcal{V}2} = \tau' \restriction V_{\mathcal{V}2}$
by *simp*
}
moreover
note $\tau 2' Vv2\text{-}is\text{-}\tau'\text{-}E2\text{-}Vv2$
ultimately show *?thesis*
by *simp*
qed
moreover
note $\tau 1' Cv1\text{-}empty\ \tau 2' Cv2\text{-}empty\ generalized\text{-}zipping\text{-}lemma$
ultimately have $\exists t.\ [] \ @\ t \in Tr_{(ES1 \parallel ES2)} \wedge t \restriction V_{\mathcal{V}} = \tau' \restriction V_{\mathcal{V}} \wedge t \restriction C_{\mathcal{V}} = []$

by *blast*
}
thus *?thesis*
unfolding *R-def*
by *auto*
qed

end

locale *CompositionalityStrictBSPs* = *Compositionality* +

assumes *NV-inter-E1-is-NV1*: $N_{\mathcal{V}} \cap E_{ES1} = N_{\mathcal{V}1}$
and *NV-inter-E2-is-NV2*: $N_{\mathcal{V}} \cap E_{ES2} = N_{\mathcal{V}2}$

sublocale *CompositionalityStrictBSPs* $\subseteq$ *Compositionality*
by (*unfold-locales*)

context *CompositionalityStrictBSPs*
begin

theorem *compositionality-SR*:
$[\![SR\ \mathcal{V}1\ Tr_{ES1};\ SR\ \mathcal{V}2\ Tr_{ES2}]\!] \Longrightarrow SR\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})$
proof −
assume *SR* $\mathcal{V}1$ Tr_{ES1}
and *SR* $\mathcal{V}2$ Tr_{ES2}
{
let $?\mathcal{V}_1{}' = (\!| V = V_{\mathcal{V}1} \cup N_{\mathcal{V}1},\ N = \{\},\ C = C_{\mathcal{V}1} |\!)$
let $?\mathcal{V}_2{}' = (\!| V = V_{\mathcal{V}2} \cup N_{\mathcal{V}2},\ N = \{\},\ C = C_{\mathcal{V}2} |\!)$
let $?\mathcal{V}' = (\!| V = V_{\mathcal{V}} \cup N_{\mathcal{V}},\ N = \{\},\ C = C_{\mathcal{V}} |\!)$

from *validV1* **have** $\mathcal{V}_1{}'IsViewOnE_1$: *isViewOn* $?\mathcal{V}_1{}'\ E_{ES1}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *validV2* **have** $\mathcal{V}_2{}'IsViewOnE_2$: *isViewOn* $?\mathcal{V}_2{}'\ E_{ES2}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *VIsViewOnE* **have** $\mathcal{V}'IsViewOnE$: *isViewOn* $?\mathcal{V}'\ E_{(ES1\|ES2)}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*

from *propSepViews NV-inter-E1-is-NV1*
have $V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1{}'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews NV-inter-E2-is-NV2*
have $V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2{}'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1{}'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2{}'}$
unfolding *properSeparationOfViews-def* **by** *auto*
have $N_{?\mathcal{V}_1{}'} \cap N_{?\mathcal{V}_2{}'} = \{\}$

by *auto*

note *properSeparation-*$\mathcal{V}_1\mathcal{V}_2$=$\langle V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1'}\rangle$ $\langle V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2'}\rangle$
$\langle C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1'}\rangle$ $\langle C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2'}\rangle$ $\langle N_{?\mathcal{V}_1'} \cap N_{?\mathcal{V}_2'} = \{\}\rangle$

have *wbc1*: $N_{?\mathcal{V}_1'} \cap E_{ES1}=\{\} \wedge N_{?\mathcal{V}_2'} \cap E_{ES2}=\{\}$
by *auto*

from $\langle$*SR* $\mathcal{V}1$ $Tr_{ES1}\rangle$ **have** *R* $?\mathcal{V}_1'$ Tr_{ES1}
using *validES1 validV1 BSPTaxonomyDifferentCorrections.SR-implies-R-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
from $\langle$*SR* $\mathcal{V}2$ $Tr_{ES2}\rangle$ **have** *R* $?\mathcal{V}_2'$ Tr_{ES2}
using *validES2 validV2 BSPTaxonomyDifferentCorrections.SR-implies-R-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*

from *validES1 validES2 composableES1ES2* $\mathcal{V}'$*IsViewOnE* $\mathcal{V}_1'$*IsViewOnE*$_1$ $\mathcal{V}_2'$*IsViewOnE*$_2$
properSeparation-$\mathcal{V}_1\mathcal{V}_2$ *wbc1*
have *Compositionality ES1 ES2* $?\mathcal{V}'$ $?\mathcal{V}_1'$ $?\mathcal{V}_2'$ **unfolding** *Compositionality-def*
by (*simp add*: *properSeparationOfViews-def wellBehavedComposition-def*)
with $\langle R\ ?\mathcal{V}_1'\ Tr_{ES1}\rangle$ $\langle R\ ?\mathcal{V}_2'\ Tr_{ES2}\rangle$ **have** *R* $?\mathcal{V}'$ $Tr_{(ES1\|ES2)}$
using *Compositionality.compositionality-R* **by** *blast*

from *validES1 validES2 composeES-yields-ES validVC*
have *BSPTaxonomyDifferentCorrections* (*ES1*‖*ES2*) $\mathcal{V}$
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
with $\langle R\ ?\mathcal{V}'\ Tr_{(ES1\|ES2)}\rangle$ **have** *SR* $\mathcal{V}$ $Tr_{(ES1\|ES2)}$
using *BSPTaxonomyDifferentCorrections.R-implies-SR-for-modified-view* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

theorem *compositionality-SD*:
$[\![$ *SD* $\mathcal{V}1$ Tr_{ES1}; *SD* $\mathcal{V}2$ Tr_{ES2} $]\!] \Longrightarrow$ *SD* $\mathcal{V}$ $(Tr_{(ES1\ \|\ ES2)})$
proof −
assume *SD* $\mathcal{V}1$ Tr_{ES1}
and *SD* $\mathcal{V}2$ Tr_{ES2}
{
let $?\mathcal{V}_1'$=$(\!|$ $V = V_{\mathcal{V}1} \cup N_{\mathcal{V}1}$, $N = \{\}$, $C = C_{\mathcal{V}1}$ $|\!)$
let $?\mathcal{V}_2'$=$(\!|$ $V = V_{\mathcal{V}2} \cup N_{\mathcal{V}2}$, $N = \{\}$, $C = C_{\mathcal{V}2}$ $|\!)$
let $?\mathcal{V}'$ =$(\!|$ V=$V_{\mathcal{V}} \cup N_{\mathcal{V}}$, N=$\{\}$, C=$C_{\mathcal{V}}$ $|\!)$

from *validV1* **have** $\mathcal{V}_1'$*IsViewOnE*$_1$: *isViewOn* $?\mathcal{V}_1'$ E_{ES1}
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *validV2* **have** $\mathcal{V}_2'$*IsViewOnE*$_2$: *isViewOn* $?\mathcal{V}_2'$ E_{ES2}
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *VIsViewOnE* **have** $\mathcal{V}'$*IsViewOnE*: *isViewOn* $?\mathcal{V}'$ $E_{(ES1\|ES2)}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*

from *propSepViews* $N\mathcal{V}$*-inter-E1-is-*$N\mathcal{V}1$
have $V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews* $N\mathcal{V}$*-inter-E2-is-*$N\mathcal{V}2$
have $V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2'}$
unfolding *properSeparationOfViews-def* **by** *auto*
have $N_{?\mathcal{V}_1'} \cap N_{?\mathcal{V}_2'} = \{\}$
by *auto*

note *properSeparation-*$\mathcal{V}_1\mathcal{V}_2$=‹$V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1'}$› ‹$V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2'}$›
‹$C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1'}$› ‹$C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2'}$› ‹$N_{?\mathcal{V}_1'} \cap N_{?\mathcal{V}_2'} = \{\}$›

have *wbc1*: $N_{?\mathcal{V}_1'} \cap E_{ES1} = \{\} \wedge N_{?\mathcal{V}_2'} \cap E_{ES2} = \{\}$
by *auto*

from ‹*SD* $\mathcal{V}1$ Tr_{ES1}› **have** *BSD* $?\mathcal{V}_1'$ Tr_{ES1}
using *validES1 validV1 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
from ‹*SD* $\mathcal{V}2$ Tr_{ES2}› **have** *BSD* $?\mathcal{V}_2'$ Tr_{ES2}
using *validES2 validV2 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*

from *validES1 validES2 composableES1ES2* $\mathcal{V}'$*IsViewOnE* $\mathcal{V}_1'$*IsViewOnE*$_1$ $\mathcal{V}_2'$*IsViewOnE*$_2$
properSeparation-$\mathcal{V}_1\mathcal{V}_2$ *wbc1*
have *Compositionality ES1 ES2* $?\mathcal{V}'$ $?\mathcal{V}_1'$ $?\mathcal{V}_2'$
unfolding *Compositionality-def*
by (*simp add*: *properSeparationOfViews-def wellBehavedComposition-def*)
with ‹*BSD* $?\mathcal{V}_1'$ Tr_{ES1}› ‹*BSD* $?\mathcal{V}_2'$ Tr_{ES2}› **have** *BSD* $?\mathcal{V}'$ $Tr_{(ES1 \| ES2)}$
using *Compositionality.compositionality-BSD* **by** *blast*

from *validES1 validES2 composeES-yields-ES validVC*
have *BSPTaxonomyDifferentCorrections* (*ES1*‖*ES2*) $\mathcal{V}$
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
with ‹*BSD* $?\mathcal{V}'$ $Tr_{(ES1 \| ES2)}$› **have** *SD* $\mathcal{V}$ $Tr_{(ES1 \| ES2)}$
using *BSPTaxonomyDifferentCorrections.BSD-implies-SD-for-modified-view* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

theorem *compositionality-SI*:
⟦*SD* $\mathcal{V}1$ Tr_{ES1}; *SD* $\mathcal{V}2$ Tr_{ES2}; *SI* $\mathcal{V}1$ Tr_{ES1}; *SI* $\mathcal{V}2$ Tr_{ES2} ⟧
⟹ *SI* $\mathcal{V}$ $(Tr_{(ES1 \parallel ES2)})$
proof −

assume *SD* $\mathcal{V}1$ Tr_{ES1}
  and *SD* $\mathcal{V}2$ Tr_{ES2}
  and *SI* $\mathcal{V}1$ Tr_{ES1}
  and *SI* $\mathcal{V}2$ Tr_{ES2}
{
 let $?\mathcal{V}_1{}' = (\!| V = V_{\mathcal{V}1} \cup N_{\mathcal{V}1},\ N = \{\},\ C = C_{\mathcal{V}1} |\!)$
 let $?\mathcal{V}_2{}' = (\!| V = V_{\mathcal{V}2} \cup N_{\mathcal{V}2},\ N = \{\},\ C = C_{\mathcal{V}2} |\!)$
 let $?\mathcal{V}' = (\!| V = V_{\mathcal{V}} \cup N_{\mathcal{V}},\ N = \{\},\ C = C_{\mathcal{V}} |\!)$

 from *validV1* **have** $\mathcal{V}_1{}'$*IsViewOnE*$_1$: *isViewOn* $?\mathcal{V}_1{}'$ E_{ES1}
  unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
 from *validV2* **have** $\mathcal{V}_2{}'$*IsViewOnE*$_2$: *isViewOn* $?\mathcal{V}_2{}'$ E_{ES2}
  unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
 from *VIsViewOnE* **have** $\mathcal{V}'$*IsViewOnE*: *isViewOn* $?\mathcal{V}'$ $E_{(ES1 \| ES2)}$
  unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*

 from *propSepViews* $N\mathcal{V}$*-inter-E1-is-N*$\mathcal{V}1$
 have $V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1{}'}$
  unfolding *properSeparationOfViews-def* **by** *auto*
 from *propSepViews* $N\mathcal{V}$*-inter-E2-is-N*$\mathcal{V}2$
 have $V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2{}'}$
  unfolding *properSeparationOfViews-def* **by** *auto*
 from *propSepViews*
 have $C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1{}'}$
  unfolding *properSeparationOfViews-def* **by** *auto*
 from *propSepViews*
 have $C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2{}'}$
  unfolding *properSeparationOfViews-def* **by** *auto*
 have $N_{?\mathcal{V}_1{}'} \cap N_{?\mathcal{V}_2{}'} = \{\}$
  by *auto*

 note *properSeparation-*$\mathcal{V}_1\mathcal{V}_2$=‹$V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1{}'}$› ‹$V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2{}'}$›
  ‹$C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1{}'}$› ‹$C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2{}'}$› ‹$N_{?\mathcal{V}_1{}'} \cap N_{?\mathcal{V}_2{}'} = \{\}$›

 have *wbc1*: $N_{?\mathcal{V}_1{}'} \cap E_{ES1} = \{\} \wedge N_{?\mathcal{V}_2{}'} \cap E_{ES2} = \{\}$
  by *auto*

 from ‹*SD* $\mathcal{V}1$ Tr_{ES1}› **have** *BSD* $?\mathcal{V}_1{}'$ Tr_{ES1}
  using *validES1 validV1 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
  unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
 from ‹*SD* $\mathcal{V}2$ Tr_{ES2}› **have** *BSD* $?\mathcal{V}_2{}'$ Tr_{ES2}
  using *validES2 validV2 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
  unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
 from ‹*SI* $\mathcal{V}1$ Tr_{ES1}› **have** *BSI* $?\mathcal{V}_1{}'$ Tr_{ES1}
  using *validES1 validV1 BSPTaxonomyDifferentCorrections.SI-implies-BSI-for-modified-view*
  unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
 from ‹*SI* $\mathcal{V}2$ Tr_{ES2}› **have** *BSI* $?\mathcal{V}_2{}'$ Tr_{ES2}
  using *validES2 validV2 BSPTaxonomyDifferentCorrections.SI-implies-BSI-for-modified-view*
  unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*

from *validES1 validES2 composableES1ES2* $\mathcal{V}'$*IsViewOnE* $\mathcal{V}_1$*'IsViewOnE*$_1$ $\mathcal{V}_2$*'IsViewOnE*$_2$
properSeparation-$\mathcal{V}_1\mathcal{V}_2$ *wbc1*
have *Compositionality ES1 ES2* $?\mathcal{V}'$ $?\mathcal{V}_1'$ $?\mathcal{V}_2'$ **unfolding** *Compositionality-def*
by (*simp add*: *properSeparationOfViews-def wellBehavedComposition-def*)
with ‹*BSD* $?\mathcal{V}_1'$ Tr_{ES1}› ‹*BSD* $?\mathcal{V}_2'$ Tr_{ES2}› ‹*BSI* $?\mathcal{V}_1'$ Tr_{ES1}› ‹*BSI* $?\mathcal{V}_2'$ Tr_{ES2}›
have *BSI* $?\mathcal{V}'$ $Tr_{(ES1\|ES2)}$
using *Compositionality.compositionality-BSI* **by** *blast*

from *validES1 validES2 composeES-yields-ES validVC*
have *BSPTaxonomyDifferentCorrections* (*ES1*‖*ES2*) $\mathcal{V}$
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
with ‹*BSI* $?\mathcal{V}'$ $Tr_{(ES1\|ES2)}$› **have** *SI* $\mathcal{V}$ $Tr_{(ES1\|ES2)}$
using *BSPTaxonomyDifferentCorrections.BSI-implies-SI-for-modified-view* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

theorem *compositionality-SIA*:
⟦*SD* $\mathcal{V}1$ Tr_{ES1}; *SD* $\mathcal{V}2$ Tr_{ES2}; *SIA* $\varrho1$ $\mathcal{V}1$ Tr_{ES1}; *SIA* $\varrho2$ $\mathcal{V}2$ Tr_{ES2};
($\varrho1$ $\mathcal{V}1$) ⊆ (ϱ $\mathcal{V}$) ∩ E_{ES1}; ($\varrho2$ $\mathcal{V}2$) ⊆ (ϱ $\mathcal{V}$) ∩ E_{ES2} ⟧
⟹ *SIA* ϱ $\mathcal{V}$ ($Tr_{(ES1\ \|\ ES2)}$)
proof −
assume *SD* $\mathcal{V}1$ Tr_{ES1}
and *SD* $\mathcal{V}2$ Tr_{ES2}
and *SIA* $\varrho1$ $\mathcal{V}1$ Tr_{ES1}
and *SIA* $\varrho2$ $\mathcal{V}2$ Tr_{ES2}
and ($\varrho1$ $\mathcal{V}1$) ⊆ (ϱ $\mathcal{V}$) ∩ E_{ES1}
and ($\varrho2$ $\mathcal{V}2$) ⊆ (ϱ $\mathcal{V}$) ∩ E_{ES2}
{
let $?\mathcal{V}_1'$ =(⦇ V = $V_{\mathcal{V}1}$ ∪ $N_{\mathcal{V}1}$, N = {}, C = $C_{\mathcal{V}1}$⦈)
let $?\mathcal{V}_2'$=(⦇ V = $V_{\mathcal{V}2}$ ∪ $N_{\mathcal{V}2}$, N = {}, C = $C_{\mathcal{V}2}$ ⦈)
let $?\mathcal{V}'$ =(⦇ V=$V_{\mathcal{V}}$ ∪ $N_{\mathcal{V}}$, N={}, C=$C_{\mathcal{V}}$ ⦈)

let $?\varrho1'$::'a *Rho* =λ$\mathcal{V}$. *if* $\mathcal{V}$=$?\mathcal{V}_1'$ *then* $\varrho1$ $\mathcal{V}1$ *else* {}
let $?\varrho2'$::'a *Rho* =λ$\mathcal{V}$. *if* $\mathcal{V}$=$?\mathcal{V}_2'$ *then* $\varrho2$ $\mathcal{V}2$ *else* {}
let $?\varrho'$::'a *Rho* =λ$\mathcal{V}'$. *if* $\mathcal{V}'$=$?\mathcal{V}'$ *then* ϱ $\mathcal{V}$ *else* {}

have ($?\varrho1'$ $?\mathcal{V}_1'$) = ($\varrho1$ $\mathcal{V}1$) **by** *simp*
have ($?\varrho2'$ $?\mathcal{V}_2'$) = ($\varrho2$ $\mathcal{V}2$) **by** *simp*
have ($?\varrho'$ $?\mathcal{V}'$) = (ϱ $\mathcal{V}$) **by** *simp*

from *validV1* **have** $\mathcal{V}_1$*'IsViewOnE*$_1$: *isViewOn* $?\mathcal{V}_1'$ E_{ES1}
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *validV2* **have** $\mathcal{V}_2$*'IsViewOnE*$_2$: *isViewOn* $?\mathcal{V}_2'$ E_{ES2}
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*
from *VIsViewOnE* **have** $\mathcal{V}'$*IsViewOnE*: *isViewOn* $?\mathcal{V}'$ $E_{(ES1\|ES2)}$
unfolding *isViewOn-def V-valid-def VN-disjoint-def NC-disjoint-def VC-disjoint-def* **by** *auto*

from *propSepViews* $N\mathcal{V}$*-inter-E1-is-*$N\mathcal{V}1$
have $V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews* $N\mathcal{V}$*-inter-E2-is-*$N\mathcal{V}2$
have $V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1'}$
unfolding *properSeparationOfViews-def* **by** *auto*
from *propSepViews*
have $C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2'}$
unfolding *properSeparationOfViews-def* **by** *auto*
have $N_{?\mathcal{V}_1'} \cap N_{?\mathcal{V}_2'} = \{\}$
by *auto*

note *properSeparation-*$\mathcal{V}_1\mathcal{V}_2$=‹$V_{?\mathcal{V}'} \cap E_{ES1} = V_{?\mathcal{V}_1'}$› ‹$V_{?\mathcal{V}'} \cap E_{ES2} = V_{?\mathcal{V}_2'}$›
‹$C_{?\mathcal{V}'} \cap E_{ES1} \subseteq C_{?\mathcal{V}_1'}$› ‹$C_{?\mathcal{V}'} \cap E_{ES2} \subseteq C_{?\mathcal{V}_2'}$› ‹$N_{?\mathcal{V}_1'} \cap N_{?\mathcal{V}_2'} = \{\}$›

have *wbc1*: $N_{?\mathcal{V}_1'} \cap E_{ES1} = \{\} \wedge N_{?\mathcal{V}_2'} \cap E_{ES2} = \{\}$
by *auto*

from ‹*SD* $\mathcal{V}1$ Tr_{ES1}› **have** *BSD* $?\mathcal{V}_1'$ Tr_{ES1}
using *validES1 validV1 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*
from ‹*SD* $\mathcal{V}2$ Tr_{ES2}› **have** *BSD* $?\mathcal{V}_2'$ Tr_{ES2}
using *validES2 validV2 BSPTaxonomyDifferentCorrections.SD-implies-BSD-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *auto*

from ‹*SIA* $\varrho 1$ $\mathcal{V}1$ Tr_{ES1}› ‹$(?\varrho 1'\ ?\mathcal{V}_1') = (\varrho 1\ \mathcal{V}1)$› **have** *BSIA* $?\varrho 1'$ $?\mathcal{V}_1'$ Tr_{ES1}
using *validES1 validV1 BSPTaxonomyDifferentCorrections.SIA-implies-BSIA-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *fastforce*
from ‹*SIA* $\varrho 2$ $\mathcal{V}2$ Tr_{ES2}› ‹$(?\varrho 2'\ ?\mathcal{V}_2') = (\varrho 2\ \mathcal{V}2)$› **have** *BSIA* $?\varrho 2'$ $?\mathcal{V}_2'$ Tr_{ES2}
using *validES2 validV2 BSPTaxonomyDifferentCorrections.SIA-implies-BSIA-for-modified-view*
unfolding *BSPTaxonomyDifferentCorrections-def* **by** *fastforce*

from *validES1 validES2 composableES1ES2* $\mathcal{V}'$*IsViewOnE* $\mathcal{V}_1'$*IsViewOnE*$_1$ $\mathcal{V}_2'$*IsViewOnE*$_2$
properSeparation-$\mathcal{V}_1\mathcal{V}_2$ *wbc1*
have *Compositionality ES1 ES2* $?\mathcal{V}'$ $?\mathcal{V}_1'$ $?\mathcal{V}_2'$
unfolding *Compositionality-def*
by (*simp add*: *properSeparationOfViews-def wellBehavedComposition-def*)
from ‹$(\varrho 1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1}$› ‹$(?\varrho 1'\ ?\mathcal{V}_1') = (\varrho 1\ \mathcal{V}1)$› ‹$(?\varrho'\ ?\mathcal{V}') = (\varrho\ \mathcal{V})$›
have $?\varrho 1'\ ?\mathcal{V}_1' \subseteq ?\varrho'\ ?\mathcal{V}' \cap E_{ES1}$
by *auto*
from ‹$(\varrho 2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}$› ‹$(?\varrho 2'\ ?\mathcal{V}_2') = (\varrho 2\ \mathcal{V}2)$› ‹$(?\varrho'\ ?\mathcal{V}') = (\varrho\ \mathcal{V})$›
have $?\varrho 2'\ ?\mathcal{V}_2' \subseteq ?\varrho'\ ?\mathcal{V}' \cap E_{ES2}$
by *auto*

from ‹*Compositionality ES1 ES2* $?\mathcal{V}'$ $?\mathcal{V}_1'$ $?\mathcal{V}_2'$› ‹*BSD* $?\mathcal{V}_1'$ Tr_{ES1}› ‹*BSD* $?\mathcal{V}_2'$ Tr_{ES2}›
‹*BSIA* $?\varrho 1'$ $?\mathcal{V}_1'$ Tr_{ES1}› ‹*BSIA* $?\varrho 2'$ $?\mathcal{V}_2'$ Tr_{ES2}›

```
      ‹?ϱ1' ?𝒱₁' ⊆ ?ϱ' ?𝒱' ∩ E_ES1› ‹?ϱ2' ?𝒱₂' ⊆ ?ϱ' ?𝒱' ∩ E_ES2›
    have BSIA ?ϱ' ?𝒱' Tr_(ES1‖ES2)
      using Compositionality.compositionality-BSIA by fastforce

    from validES1 validES2 composeES-yields-ES validVC
    have BSPTaxonomyDifferentCorrections (ES1‖ES2) 𝒱
      unfolding BSPTaxonomyDifferentCorrections-def by auto
    with ‹BSIA ?ϱ' ?𝒱' Tr_(ES1‖ES2)› ‹(?ϱ' ?𝒱') = (ϱ 𝒱)› have SIA ϱ 𝒱 Tr_(ES1‖ES2)
      using BSPTaxonomyDifferentCorrections.BSIA-implies-SIA-for-modified-view by fastforce
  }
  thus ?thesis
    by auto
qed
end

end
```

Acknowledgments

This work was partially funded by the DFG (German Research Foundation) under the projects FM-SecEng (MA 3326/1-2, MA 3326/1-3) and RSCP (MA 3326/4-3).

References

[1] S. Grewe, H. Mantel, M. Tasch, R. Gay, and H. Sudbrock. I-MAKS – A Framework for Information-Flow Security in Isabelle/HOL. Technical Report TUD-CS-2018-0056, TU Darmstadt, 2018.

[2] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 185–199, 2000.

[3] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security.* PhD thesis, Saarland University, Saarbrücken, Germany, 2003.