# A Verified Capability-based Model for Information Flow Security with Dynamic Policies

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

February 2, 2018

## Contents



**theory** *Dynamic-model*
**imports** *Main*
**begin**

## 0.1 Security State Machine

**locale** *SM* =
  **fixes** *s0* :: *'s*
  **fixes** *step* :: *'s* ⇒ *'e* ⇒ *'s*
  **fixes** *domain* :: *'e* ⇒ (*'d option*)
  **fixes** *vpeq* :: *'s* ⇒ *'d* ⇒ *'s* ⇒ *bool* ((- ∼ - ∼ -))
  **fixes** *interferes* :: *'d* ⇒ *'s* ⇒ *'d* ⇒ *bool* ((- @ - ⇝-))
  **assumes**
    *vpeq-transitive-lemma* : ∀ *s t r d*. (*s* ∼ *d* ∼ *t*) ∧ (*t* ∼ *d* ∼ *r*) ⟶ (*s* ∼ *d* ∼ *r*) **and**
    *vpeq-symmetric-lemma* : ∀ *s t d*. (*s* ∼ *d* ∼ *t*) ⟶ (*t* ∼ *d* ∼ *s*) **and**
    *vpeq-reflexive-lemma* : ∀ *s d*. (*s* ∼ *d* ∼ *s*) **and**
    *interf-reflexive*: ∀ *d s*. (*d* @ *s* ⇝ *d*)
**begin**

  **definition** *non-interferes* :: *'d* ⇒ *'s* ⇒ *'d* ⇒ *bool* ((- @ - \⇝ -))
    **where** (*u* @ *s* \⇝ *v*) ≡ (*u* @ *s* ⇝ *v*)

  **definition** *ivpeq* :: *'s* ⇒ *'d set* ⇒ *'s* ⇒ *bool* ((- ≈ - ≈ -))
    **where** *ivpeq s D t* ≡ ∀ *d* ∈ *D*. (*s* ∼ *d* ∼ *t*)

  **primrec** *run* :: *'s* ⇒ *'e list* ⇒ *'s*
    **where** *run-Nil*: *run s* [] = *s* |
        *run-Cons*: *run s* (*a*#*as*) = *run* (*step s a*) *as*

  **definition** *reachable* :: *'s* ⇒ *'s* ⇒ *bool* ((- ↪ -) [*70*,*71*] *60*) **where**
    *reachable s1 s2* ≡ (∃ *as*. *run s1 as* = *s2* )

  **definition** *reachable0* :: *'s* ⇒ *bool* **where**
    *reachable0 s* ≡ *reachable s0 s*

  **declare** *non-interferes-def*[*cong*] **and** *ivpeq-def*[*cong*] **and** *reachable-def*[*cong*]
      **and** *reachable0-def*[*cong*] **and** *run.simps*(*1*)[*cong*] **and** *run.simps*(*2*)[*cong*]

  **lemma** *reachable-s0* : *reachable0 s0*
    **by** (*metis SM.reachable-def SM-axioms reachable0-def run.simps*(*1*))

  **lemma** *reachable-self* : *reachable s s*
    **using** *reachable-def run.simps*(*1*) **by** *fastforce*

  **lemma** *reachable-step* : *s'* = *step s a* ⟹ *reachable s s'*
    **proof**−
      **assume** *a0*: *s'* = *step s a*
      **then have** *s'* = *run s* [*a*] **by** *auto*
      **then show** *?thesis* **using** *reachable-def* **by** *blast*
    **qed**

  **lemma** *run-trans* : ∀ *C T V as bs*. *T* = *run C as* ∧ *V* = *run T bs* ⟶ *V* = *run C* (*as*@*bs*)
   **proof** −
    {
      **fix** *T V as bs*
      **have** ∀ *C*. *T* = *run C as* ∧ *V* = *run T bs* ⟶ *V* = *run C* (*as*@*bs*)
        **proof**(*induct as*)
          **case** *Nil* **show** *?case* **by** *simp*
        **next**
          **case** (*Cons c cs*)
          **assume** *a0*: ∀ *C*. *T* = *run C cs* ∧ *V* = *run T bs* ⟶ *V* = *run C* (*cs* @ *bs*)

```
      show ?case
        proof−
        {
          fix C
          have T = run C (c # cs) ∧ V = run T bs ⟶ V = run C ((c # cs) @ bs)
            proof
              assume b0: T = run C (c # cs) ∧ V = run T bs
              from b0 obtain C′ where b2: C′ = step C c ∧ T = run C′ cs by auto
              with a0 b0 have V = run C′ (cs@bs) by blast
              with b2 show V = run C ((c # cs) @ bs)
                using append-Cons run-Cons by auto
            qed
        }
        then show ?thesis by blast
        qed
    qed
  }
  then show ?thesis by auto
  qed

lemma reachable-trans : ⟦reachable C T; reachable T V⟧ ⟹ reachable C V
  proof−
    assume a0: reachable C T
    assume a1: reachable T V
    from a0 have C = T ∨ (∃ as. T = run C as) by auto
    then show ?thesis
      proof
        assume b0: C = T
        show ?thesis
          proof −
            from a1 have T = V ∨ (∃ as. V = run T as) by auto
            then show ?thesis
              proof
                assume c0: T=V
                with a0 show ?thesis by auto
              next
                assume c0: (∃ as. V = run T as)
                then show ?thesis using a1 b0 by auto
              qed
          qed
      next
        assume b0: ∃ as. T = run C as
        show ?thesis
          proof −
            from a1 have T = V ∨ (∃ as. V = run T as) by auto
            then show ?thesis
              proof
                assume c0: T=V
                then show ?thesis using a0 by auto
              next
                assume c0: (∃ as. V = run T as)
                from b0 obtain as where d0: T = run C as by auto
                from c0 obtain bs where d1: V = run T bs by auto
                then show ?thesis using d0  run-trans by fastforce
              qed
          qed
      qed
  qed
```

**lemma** *reachableStep* : $[\![reachable0\ C;\ C' = step\ C\ a]\!] \Longrightarrow reachable0\ C'$
**apply** (*simp add*: *reachable0-def*)
**using** *reachable-step reachable-trans* **by** *blast*

**lemma** *reachable0-reach* : $[\![reachable0\ C;\ reachable\ C\ C']\!] \Longrightarrow reachable0\ C'$
**using** *reachable-trans* **by** *fastforce*

**declare** *reachable-def*[*cong del*] **and** *reachable0-def*[*cong del*]
**end**

## 0.2 Information flow security properties

**locale** *SM-enabled* = *SM s0 step domain vpeq interferes*
**for** *s0* :: *'s* **and**
*step* :: *'s* ⇒ *'e* ⇒ *'s* **and**
*domain* :: *'e* ⇒ (*'d option*) **and**
*vpeq* :: *'s* ⇒ *'d* ⇒ *'s* ⇒ *bool* ((- ∼ - ∼ -)) **and**
*interferes* :: *'d* ⇒ *'s* ⇒ *'d* ⇒ *bool* ((- @ - ⇝-))
+
**assumes** *enabled0*: ∀ *s a*. *reachable0 s* ⟶ (∃ *s'*. *s'* = *step s a*)
**and** *policy-respect*: ∀ *v u s t*. (*s* ∼ *u* ∼ *t*)
⟶ (*interferes v s u* = *interferes v t u*)
**begin**

**lemma** *enabled* : *reachable0 s* ⟹ (∃ *s'*. *s'* = *step s a*)
**using** *enabled0* **by** *simp*

**primrec** *sources* :: *'e list* ⇒ *'d* ⇒ *'s* ⇒ *'d set* **where**
*sources-Nil*:*sources* [] *d s* = {*d*} |
*sources-Cons*:*sources* (*a* # *as*) *d s* = (⋃{*sources as d* (*step s a*)}) ∪
{*w* . *w* = *the* (*domain a*) ∧ (∃ *v* . *interferes w s v* ∧ *v*∈*sources as d* (*step s a*))}
**declare** *sources-Nil* [*simp del*]
**declare** *sources-Cons* [*simp del*]

**primrec** *ipurge* :: *'e list* ⇒ *'d* ⇒ *'s* ⇒ *'e list* **where**
*ipurge-Nil*: *ipurge* [] *u s* = [] |
*ipurge-Cons*: *ipurge* (*a*#*as*) *u s* = (*if* (*the* (*domain a*) ∈ (*sources* (*a*#*as*) *u s*))
*then*
*a* # *ipurge as u* (*step s a*)
*else*
*ipurge as u* (*step s a*)
)

**definition** *observ-equivalence* :: *'s* ⇒ *'e list* ⇒ *'s* ⇒
*'e list* ⇒ *'d* ⇒ *bool* ((- ◁ - ≅ - ◁ - @ -))
**where** *observ-equivalence s as t bs d* ≡
((*run s as*) ∼ *d* ∼ (*run t bs*))
**declare** *observ-equivalence-def*[*cong*]

**lemma** *observ-equiv-sym*:
(*s* ◁ *as* ≅ *t* ◁ *bs* @ *d*) ⟹ (*t* ◁ *bs* ≅ *s* ◁ *as* @ *d*)
**using** *observ-equivalence-def vpeq-symmetric-lemma* **by** *blast*

**lemma** *observ-equiv-trans*:
[*reachable0 t*; (*s* ◁ *as* ≅ *t* ◁ *bs* @ *d*); (*t* ◁ *bs* ≅ *x* ◁ *cs* @ *d*)] ⟹ (*s* ◁ *as* ≅ *x* ◁ *cs* @ *d*)

**using** *observ-equivalence-def vpeq-transitive-lemma* **by** *blast*

**definition** *noninterference-r* :: *bool*
**where** $noninterference\text{-}r \equiv \forall\, d\ as\ s.\ reachable0\ s \longrightarrow (s \lhd as \cong s \lhd (ipurge\ as\ d\ s)) @ d)$

**definition** *noninterference* :: *bool*
**where** $noninterference \equiv \forall\, d\ as.\ (s0 \lhd as \cong s0 \lhd (ipurge\ as\ d\ s0)) @ d)$

**definition** *weak-noninterference* :: *bool*
**where** $weak\text{-}noninterference \equiv \forall\, d\ as\ bs.\ ipurge\ as\ d\ s0 = ipurge\ bs\ d\ s0 \longrightarrow (s0 \lhd as \cong s0 \lhd bs @ d)$

**definition** *weak-noninterference-r* :: *bool*
**where** $weak\text{-}noninterference\text{-}r \equiv \forall\, d\ as\ bs\ s.\ reachable0\ s \wedge ipurge\ as\ d\ s = ipurge\ bs\ d\ s \longrightarrow (s \lhd as \cong s \lhd bs @ d)$

**definition** *noninfluence*::*bool*
**where** $noninfluence \equiv \forall\ d\ as\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \approx (sources\ as\ d\ s) \approx t) \longrightarrow (s \lhd as \cong t \lhd (ipurge\ as\ d\ t)) @ d)$

**definition** *weak-noninfluence* ::*bool*
**where** $weak\text{-}noninfluence \equiv \forall\ d\ as\ bs\ s\ t\ .\ reachable0\ s \wedge reachable0\ t \wedge (s \approx (sources\ as\ d\ s) \approx t) \wedge ipurge\ as\ d\ t = ipurge\ bs\ d\ t \longrightarrow (s \lhd as \cong t \lhd bs @ d)$

**definition** *weak-noninfluence2* ::*bool*
**where** $weak\text{-}noninfluence2 \equiv \forall\ d\ as\ bs\ s\ t\ .\ reachable0\ s \wedge reachable0\ t \wedge (s \approx (sources\ as\ d\ s) \approx t) \wedge ipurge\ as\ d\ s = ipurge\ bs\ d\ t \longrightarrow (s \lhd as \cong t \lhd bs @ d)$

**definition** *nonleakage* :: *bool*
**where** $nonleakage \equiv \forall\, d\ as\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \approx (sources\ as\ d\ s) \approx t) \longrightarrow (s \lhd as \cong t \lhd as @ d)$

**declare** *noninterference-r-def*[*cong*] **and** *noninterference-def*[*cong*] **and** *weak-noninterference-def*[*cong*] **and** *weak-noninterference-r-def*[*cong*] **and** *noninfluence-def*[*cong*] **and** *weak-noninfluence-def*[*cong*] **and** *weak-noninfluence2-def*[*cong*] **and** *nonleakage-def*[*cong*]

## 0.3 Unwinding conditions

**definition** *dynamic-step-consistent* :: *bool* **where**
$dynamic\text{-}step\text{-}consistent \equiv \forall\, a\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge (((the\ (domain\ a))) @ s \rightsquigarrow d) \longrightarrow (s \sim (the\ (domain\ a)) \sim t)) \longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

**definition** *dynamic-weakly-step-consistent* :: *bool* **where**
$dynamic\text{-}weakly\text{-}step\text{-}consistent \equiv \forall\, a\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge ((the\ (domain\ a))) @ s \rightsquigarrow d) \wedge (s \sim (the\ (domain\ a)) \sim t) \longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

**definition** *dynamic-weakly-step-consistent-e* :: *'e* ⇒ *bool* **where**
$dynamic\text{-}weakly\text{-}step\text{-}consistent\text{-}e\ a \equiv \forall\, d\ s\ t.\ reachable0\ s \wedge reachable0\ t \wedge (s \sim d \sim t) \wedge ((the\ (domain\ a))) @ s \rightsquigarrow d) \wedge (s \sim (the\ (domain\ a)) \sim t) \longrightarrow ((step\ s\ a) \sim d \sim (step\ t\ a))$

**lemma** *dynamic-weakly-step-consistent-all-evt* : $dynamic\text{-}weakly\text{-}step\text{-}consistent = (\forall\, a.\ dynamic\text{-}weakly\text{-}step\text{-}consistent\text{-}e\ a)$

**by** (*simp add*: *dynamic-weakly-step-consistent-def dynamic-weakly-step-consistent-e-def*)

**definition** *dynamic-local-respect* :: *bool* **where**
*dynamic-local-respect* ≡ ∀ *a d s*. *reachable0 s* ∧ ¬((*the* (*domain a*)) @ *s* ⇝ *d*) ⟶ (*s* ∼ *d* ∼ (*step s a*))

**definition** *dynamic-local-respect-e* :: *'e* ⇒ *bool* **where**
*dynamic-local-respect-e a* ≡ ∀ *d s*. *reachable0 s* ∧ ¬((*the* (*domain a*)) @ *s* ⇝ *d*) ⟶ (*s* ∼ *d* ∼ (*step s a*))

**lemma** *dynamic-local-respect-all-evt* : *dynamic-local-respect* = (∀ *a*. *dynamic-local-respect-e a*)
**by** (*simp add*: *dynamic-local-respect-def dynamic-local-respect-e-def*)

**declare** *dynamic-step-consistent-def* [*cong*] **and** *dynamic-weakly-step-consistent-def* [*cong*] **and**
*dynamic-local-respect-def* [*cong*]

**lemma** *step-cons-impl-weak* : *dynamic-step-consistent* ⟹ *dynamic-weakly-step-consistent*
**using** *dynamic-step-consistent-def dynamic-weakly-step-consistent-def* **by** *blast*

**definition** *lemma-local* :: *bool* **where**
*lemma-local* ≡ ∀ *s a as u*. *the* (*domain a*) ∉ *sources* (*a* # *as*) *u s* ⟶ (*s* ≈ (*sources* (*a* # *as*) *u s*) ≈ (*step s a*))

**lemma** *weak-with-step-cons*:
**assumes** *p1*: *dynamic-weakly-step-consistent*
**and** *p2*: *dynamic-local-respect*
**shows** *dynamic-step-consistent*
**proof** −
{
**fix** *a d s t*
**have** *reachable0 s* ∧ *reachable0 t* ∧ (*s* ∼ *d* ∼ *t*) ∧
(((*the* (*domain a*)) @ *s* ⇝ *d*) ⟶ (*s* ∼ (*the* (*domain a*)) ∼ *t*))
⟶ ((*step s a*) ∼ *d* ∼ (*step t a*))
**proof** −
{
**assume** *a0*: *reachable0 s*
**assume** *a1*: *reachable0 t*
**assume** *a2*: (*s* ∼ *d* ∼ *t*)
**assume** *a3*: (((*the* (*domain a*)) @ *s* ⇝ *d*) ⟶ (*s* ∼ (*the* (*domain a*)) ∼ *t*))
**have** ((*step s a*) ∼ *d* ∼ (*step t a*))
**proof** (*cases* ((*the* (*domain a*)) @ *s* ⇝ *d*))
**assume** *b0*: ((*the* (*domain a*)) @ *s* ⇝ *d*)
**have** *b1*: (*s* ∼ (*the* (*domain a*)) ∼ *t*)
**using** *b0 a3* **by** *auto*
**have** *b2*: ((*step s a*) ∼ *d* ∼ (*step t a*))
**using** *a0 a1 a2 b0 b1 p1 dynamic-weakly-step-consistent-def* **by** *blast*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬((*the* (*domain a*)) @ *s* ⇝ *d*)
**have** *b1*: ¬((*the* (*domain a*)) @ *t* ⇝ *d*)
**using** *a0 a1 a2 b0 policy-respect* **by** *auto*
**have** *b2*: *s* ∼ *d* ∼ (*step s a*)
**using** *b0 p2 a0* **by** *auto*
**have** *b3*: *t* ∼ *d* ∼ (*step t a*)
**using** *b1 p2 a1* **by** *auto*
**have** *b4*: ((*step s a*) ∼ *d* ∼ (*step t a*))
**using** *b2 b3 a2 vpeq-symmetric-lemma vpeq-transitive-lemma* **by** *blast*
**then show** *?thesis* **by** *auto*
**qed**
}
**then show** *?thesis* **by** *auto*

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

## 0.4 Lemmas for the inference framework

**lemma** *sources-refl*:*reachable0 s* $\Longrightarrow$ *u* $\in$ *sources as u s*
**apply**(*induct as arbitrary*: *s*)
**apply**(*simp add*: *sources-Nil*)
**apply**(*simp add*: *sources-Cons*)
**using** *enabled reachableStep*
**by** *metis*

**lemma** *lemma-1-sub-1* : $[\![$*reachable0 s* ;
*dynamic-local-respect*;
*the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;
(*s* $\approx$ (*sources* (*a* # *as*) *u s*) $\approx$ *t*)$]\!]$
$\Longrightarrow$ (*s* $\approx$ (*sources as u* (*step s a*)) $\approx$ (*step s a*))
**apply** (*simp add*:*dynamic-local-respect-def sources-Cons*)
**by** *blast*

**lemma** *lemma-1-sub-2* : $[\![$*reachable0 s* ;
*reachable0 t* ;
*dynamic-local-respect*;
*the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;
(*s* $\approx$ (*sources* (*a* # *as*) *u s*) $\approx$ *t*)$]\!]$
$\Longrightarrow$ (*t* $\approx$ (*sources as u* (*step s a*)) $\approx$ (*step t a*))
**proof** −
**assume** *a1*: *reachable0 s*
**assume** *a2*: *reachable0 t*
**assume** *a3*: *dynamic-local-respect*
**assume** *a6*: *the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*
**assume** *a7*: (*s* $\approx$ (*sources* (*a* # *as*) *u s*) $\approx$ *t*)
**have** *b1*: $\forall$ *v*. *v*$\in$*sources as u* (*step s a*) $\longrightarrow$ $\neg$*interferes* (*the* (*domain a*)) *s v*
**using** *a6 sources-Cons* **by** *auto*
**have** *b2*: *sources* (*a* # *as*) *u s* = *sources as u* (*step s a*)
**using** *a6 sources-Cons* **by** *auto*
**have** *b3*: $\forall$ *v*. *v*$\in$*sources as u* (*step s a*) $\longrightarrow$ (*s* $\sim$ *v* $\sim$ *t*)
**using** *a7 b2 ivpeq-def* **by** *blast*
**have** *b4*: $\forall$ *v*. *v*$\in$*sources as u* (*step s a*) $\longrightarrow$ $\neg$*interferes* (*the* (*domain a*)) *t v*
**using** *a1 a2 policy-respect b1 b3* **by** *blast*
**have** *b5*: $\forall$ *v*. *v*$\in$*sources as u* (*step s a*) $\longrightarrow$ (*t* $\sim$ *v* $\sim$ (*step t a*))
**using** *a2 a3 b4* **by** *auto*
**then show** *?thesis*
**using** *ivpeq-def* **by** *auto*
**qed**

**lemma** *lemma-1-sub-3* : $[\![$
*the* (*domain a*) $\notin$ *sources* (*a* # *as*) *u s*;
(*s* $\approx$ (*sources* (*a* # *as*) *u s*) $\approx$ *t*)$]\!]$
$\Longrightarrow$ (*s* $\approx$ (*sources as u* (*step s a*)) $\approx$ *t*)
**apply** (*simp add*:*sources-Cons*)
**apply** (*simp add*:*sources-Cons*)
**done**

**lemma** *lemma-1-sub-4* : $[\![$(*s* $\approx$ (*sources as u* (*step s a*)) $\approx$ *t*);

```
                (s ≈ (sources as u (step s a)) ≈ (step s a));
                (t ≈ (sources as u (step s a)) ≈ (step t a)) ⟧
            ⟹ ((step s a) ≈(sources as u (step s a)) ≈ (step t a))
  by (meson ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma)


lemma lemma-1 : ⟦reachable0 s;
            reachable0 t;
            dynamic-step-consistent;
            dynamic-local-respect;
            (s ≈ (sources (a # as) u s) ≈ t)⟧
            ⟹ ((step s a) ≈ (sources as u (step s a)) ≈ (step t a))
  apply (case-tac the (domain a)∈sources (a # as) u s)
  apply (simp add: dynamic-step-consistent-def)
  apply (simp add: sources-Cons)
   proof −
     assume a1: dynamic-local-respect
     assume a4: the (domain a) ∉ sources (a # as) u s
     assume a5: (s ≈ (sources (a # as) u s) ≈ t)
     assume b0: reachable0 s
     assume b1: reachable0 t

     have a6:(s ≈ (sources as u (step s a)) ≈ t)
      using a1 policy-respect a4 a5 lemma-1-sub-3 by auto
     then have a7: (s ≈ (sources as u (step s a)) ≈ (step s a))
      using b0 a1 policy-respect a4 a5 lemma-1-sub-1 by auto
     then have a8: (t ≈ (sources as u (step s a)) ≈ (step t a))
      using b1 b0 a1 policy-respect a4 a5 lemma-1-sub-2 by auto
     then show  ((step s a) ≈(sources as u (step s a)) ≈ (step t a))
      using a6 a7 lemma-1-sub-4 by blast
   qed

lemma lemma-2 : ⟦reachable0 s;
            dynamic-local-respect;
            the (domain a) ∉ sources (a # as) u s⟧
            ⟹ (s ≈ (sources as u (step s a)) ≈ (step s a))
  apply (simp add:dynamic-local-respect-def)
  apply (simp add:sources-Cons)
  by blast

lemma sources-eq1: ∀ s t as u. reachable0 s ∧
            reachable0 t ∧
            dynamic-step-consistent ∧
            dynamic-local-respect ∧
            (s ≈ (sources as u s) ≈ t)
            ⟶ (sources as u s) = (sources as u t)
  proof −
  {
   fix as
   have ∀ s t u. reachable0 s ∧
            reachable0 t ∧
            dynamic-step-consistent ∧
            dynamic-local-respect ∧
            (s ≈ (sources as u s) ≈ t)
            ⟶ (sources as u s) = (sources as u t)
    proof(induct as)
      case Nil then show ?case by (simp add: sources-Nil)
    next
```

```
    case (Cons b bs)
    assume p0: ∀ s t u.((reachable0 s)
                        ∧ (reachable0 t)
                        ∧ dynamic-step-consistent
                        ∧ dynamic-local-respect
                        ∧ (s ≈ (sources bs u s) ≈ t)) ⟶
                          (sources bs u s) = (sources bs u t)
    then show ?case
      proof −
      {
        fix s t u
        assume p1: reachable0 s
        assume p2: reachable0 t
        assume p3: dynamic-step-consistent
        assume p5: dynamic-local-respect
        assume p9: (s ≈ (sources (b # bs) u s) ≈ t)
        have a2: ((step s b) ≈ (sources bs u (step s b)) ≈ (step t b))
          using lemma-1 p1 p2 p3 policy-respect p5 p9 by blast
        have a3: sources (b # bs) u s = sources (b # bs) u t
          proof (cases the (domain b) ∈ (sources (b # bs) u s))
            assume b0: the (domain b) ∈ (sources (b # bs) u s)
            have b1: s ∼ (the(domain b)) ∼ t
              using b0 p9 by auto
            have b3: interferes (the (domain b)) s u = interferes (the (domain b)) t u
              using p1 p2 policy-respect p9 sources-refl by fastforce
            have b4: (sources bs u (step s b)) = (sources bs u (step t b))
              using a2 p0 p1 p2 p3 p5 reachableStep by blast
            have b5: ∀ v. v∈sources bs u (step s b)
                 ⟶ interferes (the (domain b)) s v = interferes (the (domain b)) t v
              using p1 p2 ivpeq-def policy-respect p9 sources-Cons by fastforce
            then show sources (b # bs) u s = sources (b # bs) u t
              using b4 b5 sources-Cons by auto
          next
            assume b0: the (domain b) ∉ (sources (b # bs) u s)
            have b1: sources (b # bs) u s = sources bs u (step s b)
              using b0 sources-Cons by auto
            have b2: (sources bs u (step s b)) = (sources bs u (step t b))
              using a2 p0 p1 p2 p3 p5 reachableStep by blast
            have b3: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the (domain b)) s v
              using b0 sources-Cons by auto
            have b4: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the (domain b)) t v
              using b1 b3 p1 p2 p9 policy-respect by fastforce
            have b5: ∀ v. v∈sources bs u (step t b)⟶¬ interferes (the (domain b)) t v
              by (simp add: b2 b4)
            have b6: the (domain b) ∉ (sources (b # bs) u t)
              using b0 b2 b5 sources.simps(2) by auto
            have b7: sources (b # bs) u t = sources bs u (step t b)
              using b6 sources-Cons by auto
            then show ?thesis
              by (simp add: b1 b2)
          qed
      }
      then show ?thesis by blast
      qed
  qed

}
then show ?thesis by blast
```

```
  qed

lemma ipurge-eq: ∀ s t as u. reachable0 s ∧
              reachable0 t ∧
              dynamic-step-consistent ∧
              dynamic-local-respect ∧
              (s ≈ (sources as u s) ≈ t)
              ⟶ (ipurge as u s) = (ipurge as u t)
  proof −
  {
   fix as
   have ∀ s t u. reachable0 s ∧
              reachable0 t ∧
              dynamic-step-consistent ∧
              dynamic-local-respect ∧
              (s ≈ (sources as u s) ≈ t)
              ⟶ (ipurge as u s) = (ipurge as u t)
     proof(induct as)
       case Nil then show ?case by (simp add: sources-Nil)
     next
       case (Cons b bs)
       assume p0: ∀ s t u.((reachable0 s)
                        ∧ (reachable0 t)
                        ∧ dynamic-step-consistent
                        ∧ dynamic-local-respect
                        ∧ (s ≈ (sources bs u s) ≈ t))
                        ⟶ (ipurge bs u s) = (ipurge bs u t)
       then show ?case
         proof −
         {
           fix s t u
           assume p1: reachable0 s
           assume p2: reachable0 t
           assume p3: dynamic-step-consistent
           assume p5: dynamic-local-respect
           assume p9: (s ≈ (sources (b # bs) u s) ≈ t)
           have a1: ((step s b) ≈ (sources bs u (step s b)) ≈ (step t b))
             using lemma-1 p1 p2 p3 p5 p9 by blast
           have a2: (ipurge bs u (step s b)) = (ipurge bs u (step t b))
             using a1 p0 p1 p2 p3 p5 p9 reachableStep by blast
           have a3: sources (b # bs) u s = sources (b # bs) u t
             using p1 p2 p3 p5 p9 sources-eq1 by blast
           have a4: ipurge (b # bs) u s = ipurge (b # bs) u t
             proof (cases the (domain b) ∈ (sources (b # bs) u s))
               assume b0: the (domain b) ∈ (sources (b # bs) u s)
               have b1: s ∼ (the(domain b)) ∼ t
                 using b0 p9 by auto
               have b3: the (domain b) ∈ (sources (b # bs) u t)
                 using a3 b0 by auto
               then show ?thesis
                 using a2 b0 ipurge-Cons by auto
             next
               assume b0: the (domain b) ∉ (sources (b # bs) u s)
               have b1: sources (b # bs) u s = sources bs u (step s b)
                 using b0 sources-Cons by auto
               have b3: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the (domain b)) s v
                 using b0 sources-Cons by auto
               have b4: ∀ v. v∈sources bs u (step s b)⟶¬ interferes (the (domain b)) t v
```

```
            using b1 b3 p1 p2 p9 policy-respect by fastforce
           have b5: the (domain b) ∉ (sources (b # bs) u t)
            using a3 b1 b4 interf-reflexive by auto
           have b6: ipurge (b # bs) u s = ipurge bs u (step s b)
            using b0 by auto
           have b7: ipurge (b # bs) u t = ipurge bs u (step t b)
            using b5 by auto
           then show ?thesis
            using b6 b7 a2 by auto
         qed
       }
       then show ?thesis by blast
       qed
    qed
  }
  then show ?thesis by blast
  qed

lemma non-influgence-lemma: ∀ s t as u. reachable0 s ∧
            reachable0 t ∧
            dynamic-step-consistent ∧
            dynamic-local-respect ∧
            (s ≈ (sources as u s) ≈ t)
            ⟶ ((s ◁ as ≅ t ◁ (ipurge as u t) @ u))
 proof −
 {
  fix as
  have  ∀ s t u. reachable0 s ∧
            reachable0 t ∧
            dynamic-step-consistent ∧
            dynamic-local-respect ∧
            (s ≈ (sources as u s) ≈ t)
            ⟶ ((s ◁ as ≅ t ◁ (ipurge as u t) @ u))
    proof (induct as)
     case Nil show ?case using sources-Nil by auto
    next
     case (Cons b bs)
     assume p0: ∀ s t u.((reachable0 s)
                    ∧ (reachable0 t)
                    ∧ dynamic-step-consistent
                    ∧ dynamic-local-respect
                    ∧ (s ≈ (sources bs u s) ≈ t)) ⟶
                      ((s ◁ bs ≅ t ◁ (ipurge bs u t) @ u))
     then show ?case
      proof −
      {
       fix s t u
       assume p1: reachable0 s
       assume p2: reachable0 t
       assume p3: dynamic-step-consistent
       assume p4: dynamic-local-respect
       assume p8: (s ≈ (sources (b # bs) u s) ≈ t)
       have a1: ((step s b) ≈ (sources bs u (step s b)) ≈ (step t b))
        using lemma-1 p1 p2 p3 p4 p8 by blast
       have s ◁ b # bs ≅ t ◁ ipurge (b # bs) u t @ u
        proof (cases the (domain b) ∈ sources (b # bs) u s)
         assume b0: the (domain b) ∈ sources (b # bs) u s
         have b1: interferes (the (domain b)) s u = interferes (the (domain b)) t u
```

**using** *p1 p2 policy-respect p8 sources-refl* **by** *fastforce*
**have** *b2*: $\forall v.\ v \in sources\ bs\ u\ (step\ s\ b) \longrightarrow interferes\ (the\ (domain\ b))\ s\ v = interferes\ (the\ (domain\ b))\ t\ v$
**using** *p1 p2 ivpeq-def policy-respect p8 sources-Cons* **by** *fastforce*
**have** *b3*: $ipurge\ (b\ \#\ bs)\ u\ t = b\ \#\ (ipurge\ bs\ u\ (step\ t\ b))$
**by** (*metis b0 ipurge-Cons p1 p2 p3 p4 p8 sources-eq1*)
**have** *b4*: $(((step\ s\ b) \lhd bs) \cong ((step\ t\ b) \lhd (ipurge\ bs\ u\ (step\ t\ b)))\ @\ u))$
**using** *a1 p0 p1 p2 p3 p4 reachableStep* **by** *blast*
**show** *?thesis*
**using** *b3 b4* **by** *auto*
**next**
**assume** *b0*: $the\ (domain\ b) \notin sources\ (b\ \#\ bs)\ u\ s$
**have** *b1*: $ipurge\ (b\ \#\ bs)\ u\ t = (ipurge\ bs\ u\ (step\ t\ b))$
**by** (*metis a1 b0 ipurge-Cons ipurge-eq p1 p2 p3 p4 p8 reachableStep*)
**have** *b2*: $(s \approx (sources\ bs\ u\ (step\ s\ b)) \approx (step\ s\ b))$
**using** *b0 lemma-2 p1 p4* **by** *blast*
**have** *b3*:$(s \approx (sources\ bs\ u\ (step\ s\ b)) \approx t)$
**using** *b0 lemma-1-sub-3 p8* **by** *blast*
**have** *b4*: $((step\ s\ b) \approx (sources\ bs\ u\ (step\ s\ b)) \approx t)$
**by** (*meson b3 b2 ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma*)
**have** *b5*: $(((step\ s\ b) \lhd bs) \cong t \lhd (ipurge\ bs\ u\ t)\ @\ u))$
**using** *b4 p0 p1 p2 p3 p4 reachableStep* **by** *blast*
**have** *b6*: $(t \approx (sources\ bs\ u\ (step\ s\ b)) \approx (step\ t\ b))$
**using** *p1 p2 b0 lemma-1-sub-2 p4 p8* **by** *blast*
**have** *b7*: $ipurge\ bs\ u\ t = ipurge\ bs\ u\ (step\ t\ b)$
**by** (*metis a1 b4 ipurge-eq p1 p2 p3 p4 reachableStep*)
**have** *b8*: $(((step\ s\ b) \lhd bs) \cong t \lhd (ipurge\ bs\ u\ (step\ t\ b)))\ @\ u))$
**using** *b5 b7* **by** *auto*
**then show** *?thesis*
**using** *b1 observ-equivalence-def run-Cons* **by** *auto*
**qed**
**}**
**then show** *?thesis* **by** *blast*
**qed**
**qed**
**}**
**then show** *?thesis* **by** *blast*
**qed**

## 0.5 Interference framework of information flow security properties

**theorem** *nonintf-impl-weak*: $noninterference \Longrightarrow weak\text{-}noninterference$
**by** (*metis noninterference-def observ-equiv-sym observ-equiv-trans reachable-s0 weak-noninterference-def*)

**theorem** *wk-nonintf-r-impl-wk-nonintf*: $weak\text{-}noninterference\text{-}r \Longrightarrow weak\text{-}noninterference$
**using** *reachable-s0* **by** *auto*

**theorem** *nonintf-r-impl-noninterf*: $noninterference\text{-}r \Longrightarrow noninterference$
**using** *noninterference-def noninterference-r-def reachable-s0* **by** *auto*

**theorem** *nonintf-r-impl-wk-nonintf-r*: $noninterference\text{-}r \Longrightarrow weak\text{-}noninterference\text{-}r$
**by** (*metis noninterference-r-def observ-equiv-sym observ-equiv-trans weak-noninterference-r-def*)

**lemma** *noninf-impl-nonintf-r*: $noninfluence \Longrightarrow noninterference\text{-}r$
**using** *ivpeq-def noninfluence-def noninterference-r-def vpeq-reflexive-lemma* **by** *blast*

**lemma** *noninf-impl-nonlk*: $noninfluence \Longrightarrow nonleakage$
**using** *noninterference-r-def nonleakage-def observ-equiv-sym*

*observ-equiv-trans noninfluence-def noninf-impl-nonintf-r* **by** *blast*

**lemma** *wk-noninfl-impl-nonlk*: *weak-noninfluence* $\Longrightarrow$ *nonleakage*
**using** *weak-noninfluence-def nonleakage-def* **by** *blast*

**lemma** *wk-noninfl-impl-wk-nonintf-r*: *weak-noninfluence* $\Longrightarrow$ *weak-noninterference-r*
**using** *ivpeq-def weak-noninfluence-def vpeq-reflexive-lemma weak-noninterference-r-def* **by** *blast*

**lemma** *sources-step2*:
$[\![$*reachable0 s*; (*the* (*domain a*))@*s* $\rightsquigarrow$ *d*$]\!] \Longrightarrow$ *sources* [*a*] *d s* = {*the* (*domain a*),*d*}
**apply**(*auto simp*: *sources-Cons sources-Nil enabled dest*: *enabled*)
**done**

**lemma** *exec-equiv-both*:
$[\![$*reachable0 C1*; *reachable0 C2*;(*step C1 a*) $\lhd$ *as* $\cong$ (*step C2 b*) $\lhd$ *bs* @ *u*$]\!]$
$\Longrightarrow$ (*C1* $\lhd$ (*a* # *as*) $\cong$ *C2* $\lhd$ (*b* # *bs*) @ *u*)
**by** *auto*

**lemma** *sources-unwinding-step*:
$[\![$*reachable0 s*; *reachable0 t*;*s* $\approx$(*sources* (*a*#*as*) *d s*)$\approx$ *t*; *dynamic-step-consistent*$]\!]$
$\Longrightarrow$ ((*step s a*) $\approx$(*sources as d* (*step s a*))$\approx$ (*step t a*))
**apply**(*clarsimp simp*: *ivpeq-def sources-Cons*)
**using** *UnionI dynamic-step-consistent-def* **by** *blast*

**lemma** *nonlk-imp-sc*: *nonleakage* $\Longrightarrow$ *dynamic-step-consistent*
**proof** −
**assume** *p0*: *nonleakage*
**have** *p1*: $\forall$ *as d s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*) $\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *as* @ *d*)
**using** *p0 nonleakage-def* **by** *auto*
**have** *p2*: $\forall$ *a d s t*. *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\sim$ *d* $\sim$ *t*) $\wedge$
(((*the* (*domain a*)) @ *s* $\rightsquigarrow$ *d*) $\longrightarrow$ (*s* $\sim$ (*the* (*domain a*)) $\sim$ *t*))
$\longrightarrow$ ((*step s a*) $\sim$ *d* $\sim$ (*step t a*))
**proof** −
{
**fix** *a d s t*
**assume** *a0*: *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\sim$ *d* $\sim$ *t*) $\wedge$
(((*the* (*domain a*)) @ *s* $\rightsquigarrow$ *d*) $\longrightarrow$ (*s* $\sim$ (*the* (*domain a*)) $\sim$ *t*))
**have** *a4*: *s* $\approx$ (*sources* [] *d s*) $\approx$ *t*
**using** *a0 sources-Nil* **by** *auto*
**have** *a5*: (*s* $\lhd$ [] $\cong$ *t* $\lhd$ [] @ *d*)
**using** *a4 a0 p1* **by** *auto*
**have** *a6*: ((*step s a*) $\sim$ *d* $\sim$ (*step t a*))
**proof** (*cases* (*the* (*domain a*))@*s* $\rightsquigarrow$ *d*)
**assume** *b0*: (*the* (*domain a*))@*s* $\rightsquigarrow$ *d*
**have** *b1*: *sources* [*a*] *d s* = {*d*, (*the*(*domain a*))}
**using** *b0 sources-Cons sources-Nil* **by** *auto*
**have** *c0*: (*s* $\sim$ (*the* (*domain a*)) $\sim$ *t*)
**using** *b0 a0* **by** *auto*
**have** *b2*: *s* $\approx$ (*sources* [*a*] *d s*) $\approx$ *t*
**using** *b1 a0 c0* **by** *auto*
**have** *b3*: (*s* $\lhd$ [*a*] $\cong$ *t* $\lhd$ [*a*] @ *d*)
**using** *b2 a0 p1* **by** *auto*
**have** *b4*: ((*step s a*) $\sim$ *d* $\sim$ (*step t a*))
**using** *b3* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬((*the* (*domain a*))@*s* $\rightsquigarrow$ *d*)

```
        have b1: sources [a] d s = {d}
          using b0 sources-Cons sources-Nil by auto
        have b2: (s ≈ (sources [a] d s) ≈ t)
          using b1 a0 by auto
        have b3: (s ◁ [a] ≅ t ◁ [a] @ d)
          using b2 a0 p1 by auto
        have b4: ((step s a) ∼ d ∼ (step t a))
          using b3 by auto
        then show ?thesis by auto
      qed
      }
    then show ?thesis
      by auto
  qed
 then show ?thesis by auto
qed

lemma sc-imp-nonlk: dynamic-step-consistent ⟹ nonleakage
 proof −
  assume p0: dynamic-step-consistent
  have p1: ∀ a d s t. reachable0 s ∧ reachable0 t ∧ (s ∼ d ∼ t) ∧
        (s ∼ (the (domain a)) ∼ t) ⟶ ((step s a) ∼ d ∼ (step t a))
    using p0 dynamic-step-consistent-def by auto
  have p2: ∀ as d s t. reachable0 s ∧ reachable0 t
          ∧ (s ≈ (sources as d s) ≈ t) ⟶ (s ◁ as ≅ t ◁ as @ d)
    proof −
    {
      fix as
      have ∀ d s t. reachable0 s ∧ reachable0 t
          ∧ (s ≈ (sources as d s) ≈ t) ⟶ (s ◁ as ≅ t ◁ as @ d)
        proof (induct as)
          case Nil show ?case using sources-refl by auto
        next
          case (Cons b bs)
          assume a0: ∀ d s t. reachable0 s ∧ reachable0 t
            ∧ (s ≈ (sources bs d s) ≈ t) ⟶ (s ◁ bs ≅ t ◁ bs @ d)
          show ?case
            proof −
            {
              fix d s t
              assume b0: reachable0 s ∧ reachable0 t
              assume b1: (s ≈ (sources (b#bs) d s) ≈ t)
              have b2: ((step s b) ≈(sources bs d (step s b))≈ (step t b))
                using b0 b1 p0 sources-unwinding-step by auto
              have b3: (step s b) ◁ bs ≅ (step t b) ◁ bs @ d
                using Cons.hyps b0 b2 reachableStep by blast
              have b4: s ◁ b # bs ≅ t ◁ b # bs @ d
                using b3 by auto
              }
              then show ?thesis by auto
          qed
        qed
      }
    then show ?thesis by auto
  qed
 then show ?thesis by auto
qed
```

**theorem** *sc-eq-nonlk*: *dynamic-step-consistent = nonleakage*
**using** *nonlk-imp-sc sc-imp-nonlk* **by** *blast*

**lemma** *noninf-imp-dlr*: *noninfluence* $\Longrightarrow$ *dynamic-local-respect*
**proof** −
**assume** *p0*: *noninfluence*
**have** *p1*: $\forall$ *d as s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ (*ipurge as d t*) @ *d*)
**using** *p0 noninfluence-def* **by** *auto*
**have** $\forall$ *a d s*. *reachable0 s* $\wedge$ $\neg$((*the* (*domain a*)) @ *s* $\rightsquigarrow$ *d*)
$\longrightarrow$ (*s* $\sim$ *d* $\sim$ (*step s a*))
**proof** −
**{**
**fix** *a d s*
**assume** *a0*: *reachable0 s* $\wedge$ $\neg$((*the* (*domain a*)) @ *s* $\rightsquigarrow$ *d*)
**have** *a1*: *sources* [*a*] *d s* = {*d*}
**using** *a0 sources-Cons sources-Nil* **by** *auto*
**have** *a2*: (*ipurge* [*a*] *d s*) = []
**using** *a0 a1 interf-reflexive* **by** *auto*
**have** *a3*: *s* $\sim$ *d* $\sim$ *s*
**using** *vpeq-reflexive-lemma* **by** *auto*
**have** *a4*: (*s* $\approx$ (*sources* [*a*] *d s*) $\approx$ *s*)
**using** *a1 a3* **by** *auto*
**have** *a5*: (*s* $\lhd$ [*a*] $\cong$ *s* $\lhd$ (*ipurge* [*a*] *d s*) @ *d*)
**using** *a4 a0 p1* **by** *auto*
**have** *a6*: (*s* $\lhd$ [*a*] $\cong$ *s* $\lhd$ [] @ *d*)
**using** *a5 a2* **by** *auto*
**have** *a7*: (*s* $\sim$ *d* $\sim$ (*step s a*))
**using** *a6 vpeq-symmetric-lemma* **by** *auto*
**}**
**then show** *?thesis* **by** *auto*
**qed**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *noninf-imp-sc*: *noninfluence* $\Longrightarrow$ *dynamic-step-consistent*
**using** *nonlk-imp-sc noninf-impl-nonlk* **by** *blast*

**theorem** *UnwindingTheorem* : ⟦*dynamic-step-consistent*;
*dynamic-local-respect*⟧
$\Longrightarrow$ *noninfluence*
**proof** −
**assume** *p3*: *dynamic-step-consistent*
**assume** *p4*: *dynamic-local-respect*
**{**
**fix** *as d*
**have** $\forall$ *s t*. *reachable0 s* $\wedge$
*reachable0 t* $\wedge$
(*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ ((*s* $\lhd$ *as* $\cong$ *t* $\lhd$ (*ipurge as d t*) @ *d*))
**proof**(*induct as*)
**case** *Nil* **show** *?case* **using** *sources-Nil* **by** *auto*
**next**
**case** (*Cons b bs*)
**assume** *p0*: $\forall$ *s t*. *reachable0 s* $\wedge$
*reachable0 t* $\wedge$
(*s* $\approx$ (*sources bs d s*) $\approx$ *t*)

```
        ⟶ ((s ◁ bs ≅ t ◁ (ipurge bs d  t) @ d))
    then show ?case
     proof −
     {
      fix s t
      assume p1: reachable0 s
      assume p2: reachable0 t
      assume p8: (s ≈ (sources (b # bs) d s) ≈ t)
      have a1: ((step s b) ≈ (sources bs d (step s b)) ≈ (step t b))
       using lemma-1 p1 p2 p3 p4 p8 by blast
      have a2: s ◁ b # bs ≅ t ◁ ipurge (b # bs) d t @ d
       proof (cases the (domain b) ∈ sources (b # bs) d s)
        assume b0: the (domain b) ∈ sources (b # bs) d s
        have b1: interferes (the (domain b)) s d = interferes (the (domain b)) t d
         using p1 p2 policy-respect p8 sources-refl by fastforce
        have b2: ∀ v. v∈sources bs d (step s b)
             ⟶ interferes (the (domain b)) s v = interferes (the (domain b)) t v
         using p1 p2 ivpeq-def policy-respect p8 sources-Cons by fastforce
        have b3: ipurge (b # bs) d t = b # (ipurge bs d (step t b))
         by (metis b0 ipurge-Cons p1 p2 p3 p4 p8 sources-eq1)
        have b4: (((step s b) ◁ bs ≅ (step t b) ◁ (ipurge bs d (step t b)) @ d))
         using a1 p0 p1 p2 p3 p4 reachableStep by blast
        then show ?thesis
         using b3 b4 by auto
       next
        assume b0: the (domain b) ∉ sources (b # bs) d s
        have b1: ipurge (b # bs) d t = (ipurge bs d (step t b))
         by (metis a1 b0 ipurge-Cons ipurge-eq p1 p2 p3 p4 p8 reachableStep)
        have b2: (s ≈ (sources bs d (step s b)) ≈ (step s b))
         using b0 lemma-2 p1 p4 by blast
        have b3:(s ≈ (sources bs d (step s b)) ≈ t)
         using b0 lemma-1-sub-3 p8 by blast
        have b4: ((step s b) ≈ (sources bs d (step s b)) ≈ t)
         by (meson b3 b2 ivpeq-def vpeq-symmetric-lemma vpeq-transitive-lemma)
        have b5: (((step s b) ◁ bs ≅ t ◁ (ipurge bs d t) @ d))
         using b4 p0 p1 p2 p3 p4 reachableStep by blast
        have b6: (t ≈ (sources bs d (step s b)) ≈ (step t b))
         using p1 p2 b0 lemma-1-sub-2 p4 p8 by blast
        have b7: ipurge bs d t = ipurge bs d (step t b)
         by (metis a1 b4 ipurge-eq p1 p2 p3 p4 reachableStep)
        have b8: (((step s b) ◁ bs ≅ t ◁ (ipurge bs d (step t b)) @ d))
         using b5 b7 by auto
        then show ?thesis
         using b1 observ-equivalence-def run-Cons by auto
       qed
     }
     then show ?thesis by blast
     qed
   qed
   }
  then show ?thesis using noninfluence-def by blast
 qed

 theorem UnwindingTheorem1 : ⟦dynamic-weakly-step-consistent;
                    dynamic-local-respect⟧ ⟹ noninfluence
  using UnwindingTheorem weak-with-step-cons by blast

theorem uc-eq-noninf : (dynamic-step-consistent ∧ dynamic-local-respect) = noninfluence
```

**using** *UnwindingTheorem1 step-cons-impl-weak noninf-imp-dlr noninf-imp-sc* **by** *blast*

**theorem** *noninf-impl-weak*:*noninfluence* $\Longrightarrow$ *weak-noninfluence*
**proof** −
**assume** *p0*: *noninfluence*
**have** *p1*: $\forall$ *d as s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ (*ipurge as d t*) @ *d*)
**using** *p0 noninfluence-def* **by** *auto*
**have** *p2*: (*dynamic-step-consistent* $\wedge$ *dynamic-local-respect*)
**using** *p0 uc-eq-noninf* **by** *auto*
**have** $\forall$ *d as bs s t* . *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
$\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *bs* @ *d*)
**proof** −
**{**
**fix** *d as bs s t*
**assume** *a0*: *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
**have** *a4*: *noninterference-r*
**using** *noninf-impl-nonintf-r p0* **by** *auto*
**have** *a7*: *weak-noninterference-r*
**using** *a4 nonintf-r-impl-wk-nonintf-r* **by** *auto*
**have** *a6*: *ipurge as d s* = *ipurge as d t*
**using** *a0 p2 ipurge-eq* **by** *auto*
**have** *b1*: (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ (*ipurge as d t*) @ *d*)
**using** *a0 p1* **by** *auto*
**have** *b4*: (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *as* @ *d*)
**using** *a0 noninf-imp-sc nonleakage-def p0 sc-imp-nonlk* **by** *blast*
**have** *b5*: (*t* $\lhd$ *bs* $\cong$ *t* $\lhd$ (*ipurge bs d t*) @ *d*)
**using** *a0 a4* **by** *auto*
**have** *b6*: (*t* $\lhd$ *bs* $\cong$ *t* $\lhd$ (*ipurge as d t*) @ *d*)
**using** *b5 a0* **by** *auto*
**have** *b7*: (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *bs* @ *d*)
**using** *a0 b1 b6 observ-equiv-sym observ-equiv-trans* **by** *blast*
**}**
**then show** *?thesis* **by** *auto*
**qed**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *wk-nonintf-r-and-nonlk-impl-noninfl*: $[\![$*weak-noninterference-r*; *nonleakage*$]\!]$ $\Longrightarrow$ *weak-noninfluence*
**proof** −
**assume** *p0*: *weak-noninterference-r*
**assume** *p1*: *nonleakage*
**have** *p2*: $\forall$ *d as bs s*. *reachable0 s* $\wedge$ *ipurge as d s* = *ipurge bs d s*
$\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *s* $\lhd$ *bs* @ *d*)
**using** *weak-noninterference-r-def p0* **by** *auto*
**have** *p3*: $\forall$ *d as s t*. *reachable0 s* $\wedge$ *reachable0 t*
$\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*) $\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *as* @ *d*)
**using** *nonleakage-def p1* **by** *auto*
**have** $\forall$ *d as bs s t* . *reachable0 s* $\wedge$ *reachable0 t* $\wedge$ (*s* $\approx$ (*sources as d s*) $\approx$ *t*)
$\wedge$ *ipurge as d t* = *ipurge bs d t*
$\longrightarrow$ (*s* $\lhd$ *as* $\cong$ *t* $\lhd$ *bs* @ *d*)
**proof** −

```
      {
       fix d as bs s t
       assume a0: reachable0 s ∧ reachable0 t ∧ (s ≈ (sources as d s) ≈ t)
                    ∧ ipurge as d t = ipurge bs d t
       have a1: s ◁ as ≅ t ◁ as @ d
        using a0 p3 by blast
       have a2: t ◁ as ≅ t ◁ bs @ d
        using a0 p2 by auto
       have a3: (s ◁ as ≅ t ◁ bs @ d)
        using a0 a1 a2 observ-equiv-trans by blast
      }
      then show ?thesis by auto
     qed
    then show ?thesis by auto
   qed

   lemma nonintf-r-and-nonlk-impl-noninfl: ⟦noninterference-r; nonleakage⟧ ⟹ noninfluence
    proof −
     assume p0: noninterference-r
     assume p1: nonleakage
     have p2: ∀ d as s. reachable0 s ⟶ (s ◁ as ≅ s ◁ (ipurge as d s) @ d)
      using p0 noninterference-r-def by auto
     have p3: ∀ d as s t. reachable0 s ∧ reachable0 t
                    ∧ (s ≈ (sources as d s) ≈ t) ⟶ (s ◁ as ≅ t ◁ as @ d)
      using p1 nonleakage-def by auto
     have ∀ d as s t. reachable0 s ∧ reachable0 t
                    ∧ (s ≈ (sources as d s) ≈ t)
                    ⟶ (s ◁ as ≅ t ◁ (ipurge as d t) @ d)
     proof −
      {
       fix d as bs s t
       assume a0: reachable0 s ∧ reachable0 t
                    ∧ (s ≈ (sources as d s) ≈ t)
       have a1: s ◁ as ≅ t ◁ as @ d
        using p3 a0 by blast
       have a2: s ◁ as ≅ s ◁ (ipurge as d s) @ d
        using a0 p2 by fast
       have a3: t ◁ as ≅ t ◁ (ipurge as d t) @ d
        using a0 p2 by fast
       have s ◁ as ≅ t ◁ (ipurge as d t) @ d
        using a0 a1 a3 observ-equiv-trans by blast
      }
      then show ?thesis by auto
     qed
    then show ?thesis using noninfluence-def by blast
   qed

   theorem nonintf-r-and-nonlk-eq-strnoninfl: (noninterference-r ∧ nonleakage) = noninfluence
    using nonintf-r-and-nonlk-impl-noninfl noninf-impl-nonintf-r noninf-impl-nonlk by blast

 end
end
theory CapFlow
imports Dynamic-model
begin
```

## 0.6 Definitions

**type-synonym** *max-buffer-size* = *nat*
**type-synonym** *buffer-size* = *nat*
**typedecl** *Message*
**typedecl** *Endpoint*
**typedecl** *Domain*

**type-synonym** *domain-id* = *nat*
**type-synonym** *domain-name* = *string*

**type-synonym** *endpoint-name* = *string*
**type-synonym** *endpoint-id* = *nat*

**datatype**
*right* = *SEND*
|*TAKE*
|*GRANT*
|*REMOVE*

**record** *cap* = *target* :: *domain-id*
*rights* :: *right set*

**record** *Endpoint-Concig* =
*e-max-buf-size* :: *endpoint-id* ⇀ *max-buffer-size*
*e-name* :: *endpoint-id* ⇀ *endpoint-name*
*e-listener* :: *endpoint-id* ⇀ *domain-id*

**record** *Domain-Config* =
*d-name* :: *domain-id* ⇀ *domain-name*
*d-ep-set* :: *domain-id* ⇒ *endpoint-id set*

**record** *Sys-Config* =
*domconf* :: *Domain-Config*
*commconf* :: *Endpoint-Concig*

### 0.6.1 System state

**record** *State* =

*caps* :: *domain-id* ⇒ *cap set*
*e-msgs* :: *endpoint-id* ⇒ *Message set*
*e-buf-size* :: *endpoint-id* ⇀ *buffer-size*
*domain-endpoint* :: *endpoint-id* ⇀ *domain-id*

**datatype** *Event* = *Client-Lookup-Endpoint-Name domain-id endpoint-name*
| *Send-Queuing-Message domain-id endpoint-id Message*
| *Receive-Queuing-Message domain-id endpoint-id*
| *Get-My-Endpoints-Set domain-id*
| *Get-Caps domain-id*
| *Grant-Endpoint-Cap domain-id cap cap*
| *Remove-Cap-Right domain-id cap right*

### 0.6.2 Utility Functions used for Event Specification

**definition** *get-domain-name-from-domain-id* :: *Sys-Config* ⇒ *domain-id* ⇒ *domain-name option*
**where** *get-domain-name-from-domain-id sc did* ≡
*let*
*dm-conf* = (*domconf sc*)

```
        in
        if((d-name dm-conf) did ≠ None )
        then
          ((d-name dm-conf) did )
        else
          None

definition get-endpoints-from-domain-id :: Sys-Config ⇒ domain-id ⇒ endpoint-id set
  where get-endpoints-from-domain-id sc did ≡
        let
          dm-conf = (domconf sc)
        in
        if((d-name dm-conf) did ≠ None )
        then
          ((d-ep-set dm-conf) did )
        else
          {}

definition get-domain-id-from-domain-name :: Sys-Config ⇒ domain-name ⇒ domain-id option
  where get-domain-id-from-domain-name sc dname ≡
        let
          dm-conf = (domconf sc)
        in
        if(∃ did. the((d-name dm-conf) did) = dname )
        then
          Some (SOME did. the((d-name dm-conf) did) = dname)
        else
          None

definition get-listener-id-from-endpoint-id :: Sys-Config ⇒ endpoint-id ⇒ domain-id option
  where get-listener-id-from-endpoint-id sc eid ≡
        let
          ep-conf = (commconf sc)
        in
        if((e-listener ep-conf) eid ≠ None )
        then
          ((e-listener ep-conf) eid )
        else
          None

definition get-endpoint-name-from-endpoint-id :: Sys-Config ⇒ endpoint-id ⇒ endpoint-name option
  where get-endpoint-name-from-endpoint-id sc eid ≡
        let
          ep-conf = (commconf sc)
        in
        if((e-name ep-conf) eid ≠ None )
        then
          ((e-name ep-conf) eid )
        else
          None

definition get-endpoint-id-from-endpoint-name :: Sys-Config ⇒ endpoint-name ⇒ endpoint-id option
  where get-endpoint-id-from-endpoint-name sc ename ≡
        let
          ep-conf = (commconf sc)
        in
        if(∃ eid. the((e-name ep-conf) eid) = ename )
        then
```

```
    Some (SOME eid. the((e-name ep-conf) eid) = ename )
  else
    None

definition get-endpoint-max-bufsize-from-sc-by-id :: Sys-Config ⇒ endpoint-id ⇒ max-buffer-size option
  where get-endpoint-max-bufsize-from-sc-by-id sc eid ≡
  let
    ep-conf = (commconf sc)
  in
  if((e-max-buf-size ep-conf) eid ≠ None )
  then
    ((e-max-buf-size ep-conf) eid )
  else
    None

definition is-a-domain :: Sys-Config ⇒ domain-id ⇒ bool
  where is-a-domain sc did ≡
  let
    dm-conf = (domconf sc)
  in
  if((d-name dm-conf) did ≠ None )
  then
    True
  else
    False

definition is-an-endpoint :: Sys-Config ⇒ endpoint-id ⇒ bool
  where is-an-endpoint sc eid ≡
  let
    ep-conf = (commconf sc)
  in
  if((e-name ep-conf) eid ≠ None )
  then
    True
  else
    False

definition is-an-endpoint-of-domain :: Sys-Config ⇒ endpoint-id ⇒ domain-id ⇒ bool
  where is-an-endpoint-of-domain sc eid did ≡
  let
    dm-conf = (domconf sc)
  in
  if(eid ∈ ((d-ep-set dm-conf) did))
  then
    True
  else
    False

definition is-an-endpoint-of-listener :: Sys-Config ⇒ endpoint-id ⇒ domain-id ⇒ bool
  where is-an-endpoint-of-listener sc eid did ≡
  let
    ep-conf = (commconf sc)
  in
  if(the((e-listener ep-conf) eid) = did )
  then
    True
  else
    False
```

**definition** *get-domain-cap-set-from-domain-id* :: *State* ⇒ *domain-id* ⇒ *cap set*
**where** *get-domain-cap-set-from-domain-id s did* ≡
*let*
*cap-by-id* = *caps s*
*in*
*cap-by-id did*

**definition** *get-msg-set-from-endpoint-id* :: *State* ⇒ *endpoint-id* ⇒ *Message set*
**where** *get-msg-set-from-endpoint-id s eid* ≡
*let*
*msg-by-id* = *e-msgs s*
*in*
*msg-by-id eid*

**definition** *get-buf-size-from-endpoint-id* :: *State* ⇒ *endpoint-id* ⇒ *buffer-size option*
**where** *get-buf-size-from-endpoint-id s eid* ≡
*let*
*buf-size-by-id* = *e-buf-size s*
*in*
*buf-size-by-id eid*

**definition** *endpoint-is-full* :: *Sys-Config* ⇒ *State* ⇒ *endpoint-id* ⇒ *bool*
**where** *endpoint-is-full sc s eid* ≡
*get-buf-size-from-endpoint-id s eid*
= *get-endpoint-max-bufsize-from-sc-by-id sc eid*

**definition** *get-endpoints-of-domain* :: *State* ⇒ *domain-id* ⇒ *endpoint-id set*
**where** *get-endpoints-of-domain s did* ≡
*let*
*dom-ep* = *domain-endpoint s*
*in*
{*x*. *dom-ep x* = *Some did*}

**definition** *get-domain-id-from-endpoint* :: *State* ⇒ *endpoint-id* ⇒ *domain-id option*
**where** *get-domain-id-from-endpoint s eid* ≡
*let*
*dom-ep* = *domain-endpoint s*
*in*
*dom-ep eid*

**definition** *get-endpoint-msg-set-of-domain* :: *State* ⇒ *domain-id* ⇒ *Message set*
**where** *get-endpoint-msg-set-of-domain s did* ≡
*let*
*dom-ep* = *domain-endpoint s*;
*eid-set* = *get-endpoints-of-domain s did*;
*msg-of-ep* = *e-msgs s*
*in*
{*x*. ∃ *e*. *e*∈*eid-set* ∧ *x* ∈ *msg-of-ep e*}

**definition** *interferes* :: *domain-id* ⇒ *State* ⇒ *domain-id* ⇒ *bool*
**where** *interferes w s v* ≡

```
    if( w = v
        ∨ (∃ c. c∈(get-domain-cap-set-from-domain-id s w) ∧ target c = v))
    then
      True
    else
      False
```

### 0.6.3 Event specification

```
definition client-lookup-endpoint-name :: Sys-Config ⇒ State
          ⇒ domain-id ⇒ endpoint-name ⇒ (State × endpoint-id option)
  where client-lookup-endpoint-name sc s did ename ≡
          if(get-endpoint-id-from-endpoint-name sc ename ≠ None)
          then
            (s, get-endpoint-id-from-endpoint-name sc ename)
          else
            (s, None)

definition send-queuing-message :: State ⇒ domain-id ⇒ endpoint-id ⇒ Message ⇒ (State × bool)
  where send-queuing-message s did eid m ≡
          let
            dom-ep = domain-endpoint s;
            dst-dom = dom-ep eid;
            emsgs = e-msgs s;
            msg-set = get-msg-set-from-endpoint-id s eid;
            new-msg-set = insert m msg-set
          in
          if(get-domain-id-from-endpoint s eid ≠ None
             ∧ interferes did s (the (get-domain-id-from-endpoint s eid)))
          then
            (s(|
                e-msgs := emsgs(eid := new-msg-set)
              |), False)
          else
            (s, False)

definition receive-queuing-message :: State ⇒ domain-id ⇒ endpoint-id ⇒ (State × Message option)
  where receive-queuing-message s did eid ≡
          if(get-domain-id-from-endpoint s eid = Some did)
          then
            let
              emsgs = e-msgs s;
              msg-set = get-msg-set-from-endpoint-id s eid;
              m = SOME x. x∈msg-set;
              new-msg-set = msg-set − {m}
            in
              (s(|
                  e-msgs := emsgs(eid := new-msg-set)
                |), Some m)
          else
            (s, None)


definition get-my-endpoints-set :: State ⇒ domain-id ⇒ (State × endpoint-id set)
  where get-my-endpoints-set s did ≡
          if(get-endpoints-of-domain s did ≠ {})
          then
```

```
    (s, get-endpoints-of-domain s did)
  else
    (s, {})


definition get-caps :: State ⇒ domain-id ⇒ (State × cap set)
  where get-caps s did ≡
    (s, get-domain-cap-set-from-domain-id s did)

definition grant-endpoint-cap :: State ⇒ domain-id ⇒ cap ⇒ cap ⇒ (State × bool)
  where grant-endpoint-cap s did grant-cap dst-cap ≡
    if (grant-cap∈get-domain-cap-set-from-domain-id s did
      ∧ GRANT∈rights grant-cap
      ∧ target grant-cap ≠ target dst-cap
      ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
    then
      let
        did-dst = target grant-cap;
        caps0 = caps s;
        cs-dst = get-domain-cap-set-from-domain-id s did-dst
      in
      (s(|
        caps := caps0(did-dst := (insert dst-cap cs-dst))
      |), True)
    else
      (s, False)

definition get-takable-caps :: State ⇒ domain-id ⇒ cap ⇒ (State × cap set)
  where get-takable-caps s did take-cap ≡
    if (take-cap ∈ get-domain-cap-set-from-domain-id s did
      ∧ TAKE ∈ rights take-cap)
    then
      let
        did-dst = target take-cap
      in
      (s, get-domain-cap-set-from-domain-id s did-dst)
    else
      (s, {})

definition take-endpoint-cap :: State ⇒ domain-id ⇒ cap ⇒ cap ⇒ (State × bool)
  where take-endpoint-cap s did take-cap dst-cap ≡
    if (take-cap ∈  get-domain-cap-set-from-domain-id s did
      ∧ TAKE ∈ rights take-cap
      ∧ interferes did s (target dst-cap)
      ∧ dst-cap ∈ get-domain-cap-set-from-domain-id s (target take-cap))
    then
      let
        caps0 = caps s;
        cs-dst = get-domain-cap-set-from-domain-id s did
      in
      (s(|
        caps := caps0(did := (insert dst-cap cs-dst))
      |), True)
    else
      (s, False)

definition remove-cap-right :: State ⇒ domain-id ⇒ cap ⇒ right ⇒ (State × bool)
  where remove-cap-right s did rm-cap right-to-rm ≡
```

```
        let
          caps0 = caps s;
          cs-dst = get-domain-cap-set-from-domain-id s did;
          cs-rest = {c. c∈cs-dst ∧ c≠rm-cap}
        in
        if(rm-cap ∈  get-domain-cap-set-from-domain-id s did
          ∧ REMOVE ∈ rights rm-cap
          ∧ right-to-rm ∈ rights rm-cap
          ∧ REMOVE = right-to-rm
          ∧ {REMOVE} = rights rm-cap)
        then
          (s(|
             caps := caps0(did := (cs-rest))
            |), True)
        else if(
          rm-cap ∈  get-domain-cap-set-from-domain-id s did
          ∧ REMOVE ∈ rights rm-cap
          ∧ right-to-rm ∈ rights rm-cap
          )
        then
          let
            new-cap = (| target = target rm-cap,
                         rights = (rights rm-cap) − {right-to-rm}|)
          in
          (s(|
             caps := caps0(did := (insert new-cap cs-rest))
            |), True)
        else
          (s, False)

definition system-init :: Sys-Config ⇒ State
  where system-init sc ≡ (|
                          caps = (λ x. {}),
                          e-msgs = (λ x. {}),
                          e-buf-size = (λ x. None),
                          domain-endpoint = e-listener(commconf sc)
                         |)
```

## 0.7 Instantiation and Its Proofs of Security Model

```
consts sysconf :: Sys-Config
definition sys-config-witness :: Sys-Config
  where
    sys-config-witness ≡ (|
                          domconf  = (|
                                      d-name = (λ x. None),
                                      d-ep-set = (λ x. {})
                                     |) ,
                          commconf = (|
                                      e-max-buf-size = (λ x. None),
                                      e-name = (λ x. None),
                                      e-listener = (λ x. None)
                                     |)
                         |)


consts s0t :: State
definition s0t-witness :: State
```

**where** *s0t-witness* ≡ *system-init sysconf*
**specification** (*s0t*)
*s0t-init*: *s0t* = *system-init sysconf*
**by** *simp*

**definition** *exec-event* :: *State* ⇒ *Event* ⇒ *State*
**where** *exec-event s e* ≡
*case e of Client-Lookup-Endpoint-Name did ename* ⇒ *fst* (*client-lookup-endpoint-name sysconf s did ename*)
| *Send-Queuing-Message did eid m* ⇒ *fst* (*send-queuing-message s did eid m*)
| *Receive-Queuing-Message did eid* ⇒ *fst* (*receive-queuing-message s did eid*)
| *Get-My-Endpoints-Set did* ⇒ *fst* (*get-my-endpoints-set s did*)
| *Get-Caps did* ⇒*fst* (*get-caps s did*)
| *Grant-Endpoint-Cap did grant-cap dst-cap* ⇒ *fst* (*grant-endpoint-cap s did grant-cap dst-cap*)
| *Remove-Cap-Right did dst-cap right-to-rm* ⇒ *fst* (*remove-cap-right s did dst-cap right-to-rm*)

**definition** *domain-of-event* :: *Event* ⇒ *domain-id option*
**where** *domain-of-event e* ≡
*case e of Client-Lookup-Endpoint-Name did ename* ⇒ *Some did*
| *Send-Queuing-Message did eid m* ⇒ *Some did*
| *Receive-Queuing-Message did eid* ⇒ *Some did*
| *Get-My-Endpoints-Set did* ⇒ *Some did*
| *Get-Caps did* ⇒ *Some did*
| *Grant-Endpoint-Cap did grant-cap dst-cap* ⇒ *Some did*
| *Remove-Cap-Right did dst-cap right-to-rm* ⇒ *Some did*

**definition** *vpeq1* :: *State* ⇒ *domain-id* ⇒ *State* ⇒ *bool* ((- ∼ - ∼ -))
**where**
*vpeq1 s d t* ≡
*let*
*cs1* = *get-domain-cap-set-from-domain-id s d*;
*cs2* = *get-domain-cap-set-from-domain-id t d*;
*dom-eps1* = *get-endpoints-of-domain s d*;
*dom-eps2* = *get-endpoints-of-domain t d*
*in*
*if*(*cs1* = *cs2*
∧ (∀ *v*. *interferes v s d* ⟷ *interferes v t d*)
∧ *dom-eps1* = *dom-eps2*
∧ (∀ *ep*. *ep*∈*dom-eps1*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )
)
*then*
*True*
*else*
*False*

**declare** *vpeq1-def* [*cong*]

**lemma** *vpeq1-transitive-lemma* : ∀ *s t r d*. (*vpeq1 s d t*) ∧ (*vpeq1 t d r*) ⟶ (*vpeq1 s d r*)
**using** *vpeq1-def* **by** *auto*

**lemma** *vpeq1-symmetric-lemma* : ∀ *s t d*. (*vpeq1 s d t*) ⟶ (*vpeq1 t d s*)
**using** *vpeq1-def* **by** *auto*

```
lemma vpeq1-reflexive-lemma : ∀ s d. (vpeq1 s d s)
  using vpeq1-def by auto

lemma interf-reflexive-lemma : ∀ d s. interferes d s d
  using interferes-def by auto

lemma policy-respect-lemma : ∀ v u s t. (s ∼ u ∼ t)
                          ⟶ (interferes v s u = interferes v t u)
  using vpeq1-def by auto

lemma reachable-top: ∀ s a. (SM.reachable0 s0t exec-event) s ⟶ (∃ s'. s' = exec-event s a)
  proof −
  {
    fix s a
    assume p0: (SM.reachable0 s0t exec-event) s
    have (∃ s'. s' = exec-event s a)
      proof (induct a)
        case (Client-Lookup-Endpoint-Name x) show ?case
          apply (induct x)
          by (simp add: exec-event-def) +
        next
        case (Send-Queuing-Message x1a x2 x3a) show ?case
          apply (induct x1a)
          by (simp add: exec-event-def) +
        next
        case (Receive-Queuing-Message x) show ?case
          apply (induct x)
          by (simp add: exec-event-def) +
        next
        case (Get-My-Endpoints-Set x) show ?case
          apply (induct x)
          by (simp add: exec-event-def) +
        next
        case (Get-Caps x ) show ?case
          apply (induct x)
          by (simp add: exec-event-def) +
        case (Grant-Endpoint-Cap x1a x2 x3a ) show ?case
          apply (induct x1a)
          by (simp add: exec-event-def) +
        case (Remove-Cap-Right x1a x2 ) show ?case
          apply (induct x1a)
          by (simp add: exec-event-def) +
      qed
    }
  then show ?thesis by auto
  qed

declare Let-def [cong] and vpeq1-def[cong]

interpretation SM-enabled
    s0t exec-event domain-of-event vpeq1 interferes
    using vpeq1-transitive-lemma vpeq1-symmetric-lemma vpeq1-reflexive-lemma
          interf-reflexive-lemma policy-respect-lemma reachable-top
          SM.intro[of vpeq1 interferes]
          SM-enabled-axioms.intro[of s0t exec-event]
          SM-enabled.intro[of vpeq1 interferes s0t exec-event] by blast
```

## 0.8 Some lemmas of security proofs

## 0.9 Concrete unwinding condition of "local respect"

### 0.9.1 proving "client lookup endpoint name" satisfying the "local respect" property

**lemma** *client-lookup-endpoint-name-lcl-resp*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: $\neg$(*interferes did s d*)
  **and** *p2*: $s' =$ *fst* (*client-lookup-endpoint-name sysconf s did ename*)
**shows** $s \sim d \sim s'$
**proof** −
**{**
  **have** *a1*: $s = s'$
    **by** (*simp add*: *p2 client-lookup-endpoint-name-def p1*)
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *client-lookup-endpoint-name-lcl-resp-e*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: *a* = (*Client-Lookup-Endpoint-Name did ename*)
  **and** *p2*: $\neg$(*interferes* (*the* (*domain-of-event a*)) *s d*)
  **and** *p3*: $s' =$ *exec-event s a*
**shows** $s \sim d \sim s'$
**proof** −
**{**
  **have** *a0*: (*the* (*domain-of-event a*)) = *did*
    **using** *p1 domain-of-event-def* **by** *auto*
  **have** *a1*: $s' =$ *fst* (*client-lookup-endpoint-name sysconf s did ename*)
    **using** *p1 p3 exec-event-def* **by** *auto*
  **have** *a2*: $\neg$(*interferes did s d*)
    **using** *p2 a0* **by** *auto*
  **have** *a3*: $s \sim d \sim s'$
    **using** *a1 a2 p0 client-lookup-endpoint-name-lcl-resp* **by** *blast*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *client-lookup-endpoint-name-lcrsp-e*: *dynamic-local-respect-e* (*Client-Lookup-Endpoint-Name did ename*)
**proof** −
  **{**
    **have** $\forall$ *d s*. *reachable0 s*
        $\wedge$ $\neg$(*interferes* (*the* (*domain-of-event* (*Client-Lookup-Endpoint-Name did ename*))) *s d*)
        $\longrightarrow$ (*s* $\sim$ *d* $\sim$ (*exec-event s* (*Client-Lookup-Endpoint-Name did ename*)))
      **proof** −
      **{**
        **fix** *d s*
        **assume** *p1*: *reachable0 s*
        **assume** *p2*: $\neg$(*interferes* (*the* (*domain-of-event* (*Client-Lookup-Endpoint-Name did ename*))) *s d*)
        **have** (*s* $\sim$ *d* $\sim$ (*exec-event s* (*Client-Lookup-Endpoint-Name did ename*)))
          **using** *p1 p2 client-lookup-endpoint-name-lcl-resp-e* **by** *blast*
      **}**
      **then show** *?thesis* **by** *blast*
      **qed**
    **}**
  **then show** *?thesis*
    **using** *dynamic-local-respect-e-def* **by** *blast*
**qed**

### 0.9.2 proving "send queuing message" satisfying the "local respect" property

**lemma** *send-queuing-message-notchg-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: $s' = fst\ (send\text{-}queuing\text{-}message\ s\ did\ eid\ m)$
**shows** *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s' d*
**proof** (*cases* (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))))
**assume** *b0*: (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s' d*
**using** *b0 p2 send-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬ (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: $s = s'$
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b2*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s' d*
**using** *b1 get-domain-cap-set-from-domain-id-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *send-queuing-message-notchg-policy*:
**assumes** *p0*: *reachable0 s*
**and** *p2*: $s' = fst\ (send\text{-}queuing\text{-}message\ s\ did\ eid\ m)$
**shows** (∀ *v*. *interferes v s d* ⟷ *interferes v s' d*)
**proof** (*cases* (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))))
**assume** *b0*: (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s' d*
**using** *b0 p2 send-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b2*: ∀ *v*. *get-domain-cap-set-from-domain-id s v*
= *get-domain-cap-set-from-domain-id s' v*
**using** *b0 p2 send-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b3*: (∀ *v*. *interferes v s d* ⟷ *interferes v s' d*)
**using** *b1 b2 interferes-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬ (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: $s = s'$
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b2*: (∀ *v*. *interferes v s d* ⟷ *interferes v s' d*)
**using** *b1 interferes-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *send-queuing-message-notchg-dom-eps*:
**assumes** *p0*: *reachable0 s*
**and** *p2*: $s' = fst\ (send\text{-}queuing\text{-}message\ s\ did\ eid\ m)$
**shows** *get-endpoints-of-domain s d* = *get-endpoints-of-domain s' d*

**proof** (*cases* (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))))
**assume** *b0*: (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *b0 p2 send-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b2*: *domain-endpoint s* = *domain-endpoint s′*
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b3*: *get-endpoints-of-domain s d*
= *get-endpoints-of-domain s′ d*
**using** *b2 get-endpoints-of-domain-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬ (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: *s* = *s′*
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b2*: *get-endpoints-of-domain s d*
= *get-endpoints-of-domain s′ d*
**using** *b1 get-endpoints-of-domain-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *send-queuing-message-notchg-ep-msgs*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: *s′* = *fst* (*send-queuing-message s did eid m*)
**shows** (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id s′ ep* )
**proof** (*cases* (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))))
**assume** *b0*: (*get-domain-id-from-endpoint s eid* ≠ *None*
∧ *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*)))
**have** *b1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *b0 p2 send-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b2*: *domain-endpoint s* = *domain-endpoint s′*
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b3*: ∀ *e*. *e*≠*eid*
⟶ ((*e-msgs s*) *e*) = ((*e-msgs s′*) *e*)
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b4*: *domain-endpoint s* = *domain-endpoint s′*
**using** *b0 p2 send-queuing-message-def* **by** *auto*
**have** *b5*: *get-endpoints-of-domain s d*
= *get-endpoints-of-domain s′ d*
**using** *b2 get-endpoints-of-domain-def* **by** *auto*
**have** *b6*: (*the* (*get-domain-id-from-endpoint s eid*)) ≠ *d*
**using** *b0 p1* **by** *auto*
**have** *b7*: (*the* ((*domain-endpoint s*) *eid*)) ≠ *d*
**using** *b6 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b8*: *eid* ∉ *get-endpoints-of-domain s d*
**using** *b7 b6 get-endpoints-of-domain-def* **by** *auto*
**have** *b9*: ∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ ((*e-msgs s*) *ep*) = ((*e-msgs s′*) *ep*)
**using** *b8 b3* **by** *auto*
**have** *b10*: ∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id s′ ep*

```
    using b9 get-msg-set-from-endpoint-id-def by auto
  then show ?thesis by auto
next
  assume b0: ¬ (get-domain-id-from-endpoint s eid ≠ None
          ∧ interferes did s (the (get-domain-id-from-endpoint s eid)))
  have b1: s = s'
    using b0 p2 send-queuing-message-def by auto
  have b2: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
    using b1 by auto
  then show ?thesis by auto
qed

lemma send-queuing-message-lcl-resp:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (send-queuing-message s did eid m)
shows   s ∼ d ∼ s'
proof −
{
  have a0: did ≠ d
    using p1 interferes-def by auto
  have a1: get-domain-cap-set-from-domain-id s d
         = get-domain-cap-set-from-domain-id s' d
    using p0 p1 p2 send-queuing-message-notchg-domain-cap-set by auto
  have a2: (∀ v. interferes v s d ⟷ interferes v s' d)
    using p0 p1 p2 send-queuing-message-notchg-policy by auto
  have a3: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    using p0 p1 p2 send-queuing-message-notchg-dom-eps by auto
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
    using p0 p1 p2 send-queuing-message-notchg-ep-msgs by auto
  have a5: s ∼ d ∼ s'
    using a1 a2 a3 a4 by auto
  }
  then show ?thesis by auto
qed


lemma send-queuing-message-lcl-resp-e:
assumes p0: reachable0 s
  and   p1: a = Send-Queuing-Message did eid m
  and   p2: ¬(interferes (the (domain-of-event a)) s d)
  and   p3: s' = exec-event s a
shows   s ∼ d ∼ s'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p1 domain-of-event-def by auto
  have a1: s' = fst (send-queuing-message s did eid m)
    using p1 p3 exec-event-def by auto
  have a2: ¬(interferes did s d)
    using p2 a0 by auto
  have a3: s ∼ d ∼ s'
    using a1 a2 p0 send-queuing-message-lcl-resp by blast
}
then show ?thesis by auto
qed
```

**lemma** *send-queuing-message-lcl-lcrsp-e*: *dynamic-local-respect-e* (*Send-Queuing-Message did eid m*)
**proof** −
**{**
**have** ∀ *d s*. *reachable0 s*
∧ ¬(*interferes* (*the* (*domain-of-event* (*Send-Queuing-Message did eid m*))) *s d*)
⟶ (*s* ∼ *d* ∼ (*exec-event s* (*Send-Queuing-Message did eid m*)))
**proof** −
**{**
**fix** *d s*
**assume** *p1*: *reachable0 s*
**assume** *p2*: ¬(*interferes* (*the* (*domain-of-event* (*Send-Queuing-Message did eid m*))) *s d*)
**have** (*s* ∼ *d* ∼ (*exec-event s* (*Send-Queuing-Message did eid m*)))
**using** *p1 p2 send-queuing-message-lcl-resp-e* **by** *blast*
**}**
**then show** *?thesis* **by** *blast*
**qed**
**}**
**then show** *?thesis*
**using** *dynamic-local-respect-e-def* **by** *blast*
**qed**

### 0.9.3 proving "receive queuing message" satisfying the "local respect" property

**lemma** *receive-queuing-message-notchg-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: *s′* = *fst* (*receive-queuing-message s did eid*)
**shows** *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**proof** (*cases the*(*get-domain-id-from-endpoint s eid*) = *did*)
**assume** *a0*: *the*(*get-domain-id-from-endpoint s eid*) = *did*
**have** *a1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *a0*: *the*(*get-domain-id-from-endpoint s eid*) ≠ *did*
**have** *a1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *receive-queuing-message-notchg-policy*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: *s′* = *fst* (*receive-queuing-message s did eid*)
**shows** (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
**proof** (*cases the*(*get-domain-id-from-endpoint s eid*) = *did*)
**assume** *a0*: *the*(*get-domain-id-from-endpoint s eid*) = *did*
**have** *a1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *a2*: ∀ *v*. *get-domain-cap-set-from-domain-id s v*
= *get-domain-cap-set-from-domain-id s′ v*
**using** *a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *a3*: (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)

```
    using a1 a2 interferes-def by auto
  then show ?thesis by auto
next
  assume a0: the(get-domain-id-from-endpoint s eid) ≠ did
  have a1: s = s'
    using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
  have a2: (∀ v. interferes v s d ⟷ interferes v s' d)
    using a1 interferes-def by auto
  then show ?thesis by auto
qed

lemma receive-queuing-message-notchg-dom-eps:
  assumes p0: reachable0 s
    and   p2: s' = fst (receive-queuing-message s did eid)
  shows   get-endpoints-of-domain s d = get-endpoints-of-domain s' d
  proof (cases the(get-domain-id-from-endpoint s eid) = did)
    assume a0: the(get-domain-id-from-endpoint s eid) = did
    have a1: get-domain-cap-set-from-domain-id s d
          = get-domain-cap-set-from-domain-id s' d
      using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
    have a2: domain-endpoint s = domain-endpoint s'
      using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
    have a3: get-endpoints-of-domain s d
          = get-endpoints-of-domain s' d
      using a1 a2 get-endpoints-of-domain-def by auto
    then show ?thesis by auto
  next
    assume a0: the(get-domain-id-from-endpoint s eid) ≠ did
    have a1: s = s'
      using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
    have a2: get-endpoints-of-domain s d
          = get-endpoints-of-domain s' d
      using a1 get-endpoints-of-domain-def by auto
    then show ?thesis by auto
  qed

lemma receive-queuing-message-notchg-ep-msgs:
  assumes p0: reachable0 s
    and   p1: ¬(interferes did s d)
    and   p2: s' = fst (receive-queuing-message s did eid)
  shows   (∀ ep. ep∈get-endpoints-of-domain s d
          ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
  proof (cases the(get-domain-id-from-endpoint s eid) = did)
    assume a0: the(get-domain-id-from-endpoint s eid) = did
    have a1: get-domain-cap-set-from-domain-id s d
          = get-domain-cap-set-from-domain-id s' d
      using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
    have a2: domain-endpoint s = domain-endpoint s'
      using a0 p2 receive-queuing-message-def by auto
    have a3: get-endpoints-of-domain s d
          = get-endpoints-of-domain s' d
      using a1 a2 get-endpoints-of-domain-def by auto
    have a4: ∀ e. e≠eid
          ⟶ ((e-msgs s) e) = ((e-msgs s') e)
      using a0 p2 receive-queuing-message-def by auto
    have a5: d ≠ did
      using p1 interferes-def by auto
    have a6: the(get-domain-id-from-endpoint s eid) ≠ d
```

```
    using a5 a0 by auto
  have a7: (the ((domain-endpoint s) eid)) ≠ d
    using a6 get-domain-id-from-endpoint-def by auto
  have a8: eid ∉ get-endpoints-of-domain s d
    using a7 a6 get-endpoints-of-domain-def by auto
  have a9: ∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ ((e-msgs s) ep) = ((e-msgs s') ep)
    using a8 a4 by auto
  have a10: ∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep
    using a9 get-msg-set-from-endpoint-id-def by auto
  then show ?thesis by auto
next
  assume a0: the(get-domain-id-from-endpoint s eid) ≠ did
  have a1: s = s'
    using a0 p2 receive-queuing-message-def get-domain-cap-set-from-domain-id-def by auto
  have a2: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
    using a1 get-msg-set-from-endpoint-id-def by auto
  then show ?thesis by auto
qed

lemma receive-queuing-message-lcl-resp:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (receive-queuing-message s did eid)
shows   s ∼ d ∼ s'
proof −
{
  have a0: did ≠ d
    using p1 interferes-def by auto
  have a1: get-domain-cap-set-from-domain-id s d
         = get-domain-cap-set-from-domain-id s' d
    using p0 p1 p2 receive-queuing-message-notchg-domain-cap-set by auto
  have a2: (∀ v. interferes v s d ⟷ interferes v s' d)
    using p0 p1 p2 receive-queuing-message-notchg-policy by auto
  have a3: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    using p0 p1 p2 receive-queuing-message-notchg-dom-eps by auto
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
    using p0 p1 p2 receive-queuing-message-notchg-ep-msgs by auto
  have a5: s ∼ d ∼ s'
    using a1 a2 a3 a4 by auto
  }
  then show ?thesis by auto
qed

lemma receive-queuing-message-lcl-resp-e:
assumes p0: reachable0 s
  and   p1: a = (Receive-Queuing-Message did eid)
  and   p2: ¬(interferes (the (domain-of-event a)) s d)
  and   p3: s' = exec-event s a
shows   s ∼ d ∼ s'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p1 domain-of-event-def by auto
  have a1: s' = fst (receive-queuing-message s did eid)
```

```
    using p1 p3 exec-event-def by auto
  have a2: ¬(interferes did s d)
    using p2 a0 by auto
  have a3: s ∼ d ∼ s'
    using a1 a2 p0 receive-queuing-message-lcl-resp by blast
}
then show ?thesis by auto
qed

lemma receive-queuing-message-lcrsp-e: dynamic-local-respect-e (Receive-Queuing-Message did eid)
proof −
 {
  have ∀ d s. reachable0 s
       ∧ ¬(interferes (the (domain-of-event (Receive-Queuing-Message did eid))) s d)
       ⟶ (s ∼ d ∼ (exec-event s (Receive-Queuing-Message did eid)))
    proof −
    {
     fix d s
     assume p1: reachable0 s
     assume p2: ¬(interferes (the (domain-of-event (Receive-Queuing-Message did eid))) s d)
     have (s ∼ d ∼ (exec-event s (Receive-Queuing-Message did eid)))
       using p1 p2 receive-queuing-message-lcl-resp-e by blast
    }
    then show ?thesis by blast
    qed
  }
  then show ?thesis
    using dynamic-local-respect-e-def by blast
qed
```

### 0.9.4 proving "get my endpoints set" satisfying the "local respect" property

```
lemma get-my-endpoints-set-lcl-resp:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (get-my-endpoints-set s did)
shows   s ∼ d ∼ s'
proof −
{
  have a1: s = s'
    by (simp add: p2 get-my-endpoints-set-def p1)
}
then show ?thesis by auto
qed

lemma get-my-endpoints-set-lcl-resp-e:
assumes p0: reachable0 s
  and   p1: a = (Get-My-Endpoints-Set did)
  and   p2: ¬(interferes (the (domain-of-event a)) s d)
  and   p3: s' = exec-event s a
shows   s ∼ d ∼ s'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p1 domain-of-event-def by auto
  have a1: s' = fst (get-my-endpoints-set s did)
    using p1 p3 exec-event-def by auto
  have a2: ¬(interferes did s d)
```

```
    using p2 a0 by auto
  have a3: s ∼ d ∼ s'
    using a1 a2 p0 get-my-endpoints-set-lcl-resp by blast
}
then show ?thesis by auto
qed

lemma get-my-endpoints-set-lcrsp-e: dynamic-local-respect-e (Get-My-Endpoints-Set did)
proof −
  {
    have ∀ d s. reachable0 s
          ∧ ¬(interferes (the (domain-of-event (Get-My-Endpoints-Set did))) s d)
          ⟶ (s ∼ d ∼ (exec-event s (Get-My-Endpoints-Set did)))
      proof −
      {
        fix d s
        assume p1: reachable0 s
        assume p2: ¬(interferes (the (domain-of-event (Get-My-Endpoints-Set did))) s d)
        have (s ∼ d ∼ (exec-event s (Get-My-Endpoints-Set did)))
          using p1 p2 get-my-endpoints-set-lcl-resp-e by blast
      }
      then show ?thesis by blast
      qed
    }
  then show ?thesis
    using dynamic-local-respect-e-def by blast
qed
```

### 0.9.5 proving "get caps" satisfying the "local respect" property

```
lemma get-caps-lcl-resp:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (get-caps s did)
shows   s ∼ d ∼ s'
proof −
{
  have a1: s = s'
    by (simp add: p2 get-caps-def p1)
}
then show ?thesis by auto
qed

lemma get-caps-lcl-resp-e:
assumes p0: reachable0 s
  and   p1: a = (Get-Caps did)
  and   p2: ¬(interferes (the (domain-of-event a)) s d)
  and   p3: s' = exec-event s a
shows   s ∼ d ∼ s'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p1 domain-of-event-def by auto
  have a1: s' = fst (get-caps s did)
    using p1 p3 exec-event-def by auto
  have a2: ¬(interferes did s d)
    using p2 a0 by auto
  have a3: s ∼ d ∼ s'
```

**using** *a1 a2 p0 get-caps-lcl-resp* **by** *blast*
}
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *get-caps-lcrsp-e*: *dynamic-local-respect-e* (*Get-Caps did*)
**proof** −
{
**have** ∀ *d s*. *reachable0 s*
∧ ¬(*interferes* (*the* (*domain-of-event* (*Get-Caps did*))) *s d*)
⟶ (*s* ∼ *d* ∼ (*exec-event s* (*Get-Caps did*)))
**proof** −
{
**fix** *d s*
**assume** *p1*: *reachable0 s*
**assume** *p2*: ¬(*interferes* (*the* (*domain-of-event* (*Get-Caps did*))) *s d*)
**have** (*s* ∼ *d* ∼ (*exec-event s* (*Get-Caps did*)))
**using** *p1 p2 get-caps-lcl-resp-e* **by** *blast*
}
**then show** *?thesis* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-local-respect-e-def* **by** *blast*
**qed**

### 0.9.6 proving "grant endpoint cap" satisfying the "local respect" property

**lemma** *grant-endpoint-cap-notchg-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: *s′* = *fst* (*grant-endpoint-cap s did grant-cap dst-cap*)
**shows** *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**proof** (*cases grant-cap*∈*get-domain-cap-set-from-domain-id s did*
∧ *GRANT*∈*rights grant-cap*
∧ *target grant-cap* ≠ *target dst-cap*
∧ *dst-cap*∈*get-domain-cap-set-from-domain-id s did*)
**assume** *a0*: *grant-cap*∈*get-domain-cap-set-from-domain-id s did*
∧ *GRANT*∈*rights grant-cap*
∧ *target grant-cap* ≠ *target dst-cap*
∧ *dst-cap*∈*get-domain-cap-set-from-domain-id s did*
**let** *?did-dst* = *target grant-cap*
**have** *a1*: ∀ *v*. *v*≠*?did-dst*
⟶ (*caps s*) *v* = (*caps s′*) *v*
**using** *a0 p2 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *a2*: *d* ≠ *did*
**using** *p1 interferes-def* **by** *auto*
**have** *a3*: ¬(( *did* = *d*
∨ (∃ *c*. *c*∈(*get-domain-cap-set-from-domain-id s did*) ∧ *target c* = *d*)))
**using** *interferes-def p1* **by** *force*
**have** *a4*: ∀ *c*. *c*∈(*get-domain-cap-set-from-domain-id s did*)
⟶ *target c* ≠ *d*
**using** *a3* **by** *auto*
**have** *a5*: *?did-dst* ≠ *d*
**using** *a4 a0* **by** *auto*
**have** *a6*: (*caps s*) *d* = (*caps s′*) *d*
**using** *a1 a5* **by** *auto*

```
  have a7: get-domain-cap-set-from-domain-id s d
        = get-domain-cap-set-from-domain-id s' d
    using a6 get-domain-cap-set-from-domain-id-def by auto
  then show ?thesis by auto
next
  assume a0: ¬ (grant-cap ∈ get-domain-cap-set-from-domain-id s did
              ∧ GRANT ∈ rights grant-cap
              ∧ target grant-cap ≠ target dst-cap
              ∧ dst-cap ∈ get-domain-cap-set-from-domain-id s did)
  have a1: s = s'
    using a0 p2 grant-endpoint-cap-def by auto
  have a2: get-domain-cap-set-from-domain-id s d
        = get-domain-cap-set-from-domain-id s' d
    using a1 get-domain-cap-set-from-domain-id-def by auto
  then show ?thesis by auto
qed

lemma grant-endpoint-cap-notchg-policy:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
shows   (∀ v. interferes v s d ⟷ interferes v s' d)
proof (cases grant-cap∈get-domain-cap-set-from-domain-id s did
          ∧ GRANT∈rights grant-cap
          ∧ target grant-cap ≠ target dst-cap
          ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
  assume a0: grant-cap∈get-domain-cap-set-from-domain-id s did
          ∧ GRANT∈rights grant-cap
          ∧ target grant-cap ≠ target dst-cap
          ∧ dst-cap∈get-domain-cap-set-from-domain-id s did
  let ?did-dst = target grant-cap
  let ?did-granted = target dst-cap
  have a1: ∀ v. v≠?did-dst
          ⟶ (caps s) v = (caps s') v
    using a0 p2 grant-endpoint-cap-def by auto
  have a2: d ≠ did
    using p1 interferes-def by auto
  have a3: ¬(( did = d
          ∨ (∃ c. c∈(get-domain-cap-set-from-domain-id s did) ∧ target c = d)))
    using interferes-def p1 by force
  have a4: ∀ c. c∈(get-domain-cap-set-from-domain-id s did)
          ⟶ target c ≠ d
    using a3 by auto
  have a5: ?did-dst ≠ d
    using a4 a0 by auto
  have a6: (caps s) d = (caps s') d
    using a1 a5 by auto
  have a7: get-domain-cap-set-from-domain-id s d
        = get-domain-cap-set-from-domain-id s' d
    using a6 get-domain-cap-set-from-domain-id-def by auto
  have a8: ?did-granted ≠ d
    using a4 a0 by auto
  have a9: ∀ v. v≠?did-dst
          ⟶ get-domain-cap-set-from-domain-id s v
            = get-domain-cap-set-from-domain-id s' v
    using a1 get-domain-cap-set-from-domain-id-def by auto
  have a10: ∀ v. v≠?did-dst
          ⟶ interferes v s d ⟷ interferes v s' d
```

```
    using a9 a7 interferes-def by auto
  have a11: get-domain-cap-set-from-domain-id s' ?did-dst
          = {dst-cap} ∪ get-domain-cap-set-from-domain-id s ?did-dst
    using a0 p2 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have a12: interferes ?did-dst s d = interferes ?did-dst s' d
    using a11 a8 interferes-def by auto
  have a13: ∀ v. interferes v s d ⟷ interferes v s' d
    using a10 a12 by force
  then show ?thesis by auto
next
  assume a0: ¬ (grant-cap ∈ get-domain-cap-set-from-domain-id s did
              ∧ GRANT ∈ rights grant-cap
              ∧ target grant-cap ≠ target dst-cap
              ∧ dst-cap ∈ get-domain-cap-set-from-domain-id s did)
  have a1: s = s'
    using a0 p2 grant-endpoint-cap-def by auto
  have a2: (∀ v. interferes v s d ⟷ interferes v s' d)
    using a1 interferes-def by auto
  then show ?thesis by auto
qed

lemma grant-endpoint-cap-notchg-dom-eps:
assumes p0: reachable0 s
  and   p2: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
shows   get-endpoints-of-domain s d = get-endpoints-of-domain s' d
proof (cases grant-cap∈get-domain-cap-set-from-domain-id s did
          ∧ GRANT∈rights grant-cap
          ∧ target grant-cap ≠ target dst-cap
          ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
  assume a0: grant-cap∈get-domain-cap-set-from-domain-id s did
          ∧ GRANT∈rights grant-cap
          ∧ target grant-cap ≠ target dst-cap
          ∧ dst-cap∈get-domain-cap-set-from-domain-id s did
  have a2: domain-endpoint s = domain-endpoint s'
    using a0 p2 grant-endpoint-cap-def by auto
  have a3: get-endpoints-of-domain s d
        = get-endpoints-of-domain s' d
    using a2 get-endpoints-of-domain-def by auto
  then show ?thesis by auto
next
  assume a0: ¬(grant-cap∈get-domain-cap-set-from-domain-id s did
          ∧ GRANT∈rights grant-cap
          ∧ target grant-cap ≠ target dst-cap
          ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
  have a1: s = s'
    using a0 p2 grant-endpoint-cap-def by auto
  have a2: get-endpoints-of-domain s d
        = get-endpoints-of-domain s' d
    using a1 get-endpoints-of-domain-def by auto
  then show ?thesis by auto
qed

lemma grant-endpoint-cap-notchg-ep-msgs:
assumes p0: reachable0 s
  and   p1: ¬(interferes did s d)
  and   p2: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
shows   (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
```

**proof** (*cases grant-cap∈get-domain-cap-set-from-domain-id s did*
  ∧ *GRANT∈rights grant-cap*
  ∧ *target grant-cap ≠ target dst-cap*
  ∧ *dst-cap∈get-domain-cap-set-from-domain-id s did*)
 **assume** *a0*: *grant-cap∈get-domain-cap-set-from-domain-id s did*
  ∧ *GRANT∈rights grant-cap*
  ∧ *target grant-cap ≠ target dst-cap*
  ∧ *dst-cap∈get-domain-cap-set-from-domain-id s did*
 **have** *a2*: *domain-endpoint s = domain-endpoint s′*
  **using** *a0 p2 grant-endpoint-cap-def* **by** *auto*
 **have** *a3*: *get-endpoints-of-domain s d*
  = *get-endpoints-of-domain s′ d*
  **using** *a2 get-endpoints-of-domain-def* **by** *auto*
 **have** *a4*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
  ⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s′ ep* )
  **using** *a0 p2 grant-endpoint-cap-def get-msg-set-from-endpoint-id-def* **by** *auto*
 **then show** *?thesis* **by** *auto*
**next**
 **assume** *a0*: ¬(*grant-cap∈get-domain-cap-set-from-domain-id s did*
  ∧ *GRANT∈rights grant-cap*
  ∧ *target grant-cap ≠ target dst-cap*
  ∧ *dst-cap∈get-domain-cap-set-from-domain-id s did*)
 **have** *a1*: *s = s′*
  **using** *a0 p2 grant-endpoint-cap-def* **by** *auto*
 **have** *a2*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
  ⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s′ ep* )
  **using** *a1* **by** *auto*
 **then show** *?thesis* **by** *auto*
**qed**

**lemma** *grant-endpoint-cap-lcl-resp*:
**assumes** *p0*: *reachable0 s*
 **and** *p1*: ¬(*interferes did s d*)
 **and** *p2*: *s′* = *fst* (*grant-endpoint-cap s did eid m*)
**shows** *s* ∼ *d* ∼ *s′*
**proof** −
**{**
 **have** *a0*: *did ≠ d*
  **using** *p1 interferes-def* **by** *auto*
 **have** *a1*: *get-domain-cap-set-from-domain-id s d*
  = *get-domain-cap-set-from-domain-id s′ d*
  **using** *p0 p1 p2 grant-endpoint-cap-notchg-domain-cap-set* **by** *auto*
 **have** *a2*: (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
  **using** *p0 p1 p2 grant-endpoint-cap-notchg-policy* **by** *auto*
 **have** *a3*: *get-endpoints-of-domain s d = get-endpoints-of-domain s′ d*
  **using** *p0 p1 p2 grant-endpoint-cap-notchg-dom-eps* **by** *auto*
 **have** *a4*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
  ⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s′ ep* )
  **using** *p0 p1 p2 grant-endpoint-cap-notchg-ep-msgs* **by** *auto*
 **have** *a5*: *s* ∼ *d* ∼ *s′*
  **using** *a1 a2 a3 a4* **by** *auto*
 **}**
 **then show** *?thesis* **by** *auto*
**qed**

**lemma** *grant-endpoint-cap-lcl-resp-e*:
**assumes** *p0*: *reachable0 s*
 **and** *p1*: *a* = (*Grant-Endpoint-Cap did grant-cap dst-cap*)

**and** *p2*: $\neg$(*interferes* (*the* (*domain-of-event a*)) *s d*)
**and** *p3*: $s' = exec\text{-}event\ s\ a$
**shows** $s \sim d \sim s'$
**proof** −
{
**have** *a0*: (*the* (*domain-of-event a*)) = *did*
**using** *p1 domain-of-event-def* **by** *auto*
**have** *a1*: $s' = fst\ (grant\text{-}endpoint\text{-}cap\ s\ did\ grant\text{-}cap\ dst\text{-}cap)$
**using** *p1 p3 exec-event-def* **by** *auto*
**have** *a2*: $\neg$(*interferes did s d*)
**using** *p2 a0* **by** *auto*
**have** *a3*: $s \sim d \sim s'$
**using** *a1 a2 p0 grant-endpoint-cap-lcl-resp* **by** *blast*
}
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *grant-endpoint-cap-lcrsp-e*: *dynamic-local-respect-e* (*Grant-Endpoint-Cap did grant-cap dst-cap*)
**proof** −
{
**have** $\forall\ d\ s.\ reachable0\ s$
$\wedge\ \neg$(*interferes* (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) *s d*)
$\longrightarrow$ ($s \sim d \sim$ (*exec-event s* (*Grant-Endpoint-Cap did grant-cap dst-cap*)))
**proof** −
{
**fix** *d s*
**assume** *p1*: *reachable0 s*
**assume** *p2*: $\neg$(*interferes* (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) *s d*)
**have** ($s \sim d \sim$ (*exec-event s* (*Grant-Endpoint-Cap did grant-cap dst-cap*)))
**using** *p1 p2 grant-endpoint-cap-lcl-resp-e* **by** *blast*
}
**then show** *?thesis* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-local-respect-e-def* **by** *blast*
**qed**

### 0.9.7 proving "remove cap right" satisfying the "local respect" property

**lemma** *remove-cap-right-notchg-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: $\neg$(*interferes did s d*)
**and** *p2*: $s' = fst\ (remove\text{-}cap\text{-}right\ s\ did\ rm\text{-}cap\ right\text{-}to\text{-}rm)$
**shows** *get-domain-cap-set-from-domain-id s d*
$= get\text{-}domain\text{-}cap\text{-}set\text{-}from\text{-}domain\text{-}id\ s'\ d$
**proof** −
{
**have** *a0*: $d \neq did$
**using** *p1 interferes-def* **by** *auto*
**have** *get-domain-cap-set-from-domain-id s d*
$= get\text{-}domain\text{-}cap\text{-}set\text{-}from\text{-}domain\text{-}id\ s'\ d$
**proof** (*cases rm-cap* $\in$ *get-domain-cap-set-from-domain-id s did*
$\wedge$ *REMOVE* $\in$ *rights rm-cap*
$\wedge$ *right-to-rm* $\in$ *rights rm-cap*)
**assume** *b0*: *rm-cap* $\in$ *get-domain-cap-set-from-domain-id s did*
$\wedge$ *REMOVE* $\in$ *rights rm-cap*
$\wedge$ *right-to-rm* $\in$ *rights rm-cap*

```
    have get-domain-cap-set-from-domain-id s d
        = get-domain-cap-set-from-domain-id s' d
      proof (cases  REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap)
        assume c0: REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        have c1: ((caps s') did) = ?cs-rest
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: ∀ v. v≠did
                ⟶ ((caps s') v) = ((caps s) v)
          using b0 c0 p2 remove-cap-right-def by auto
        have c3: ∀ v. v≠did
                ⟶ get-domain-cap-set-from-domain-id s v
                  = get-domain-cap-set-from-domain-id s' v
          using c2 get-domain-cap-set-from-domain-id-def by auto
        have c4: get-domain-cap-set-from-domain-id s d
                  = get-domain-cap-set-from-domain-id s' d
          using c3 a0 by auto
        then show ?thesis by auto
      next
        assume c0: ¬(REMOVE = right-to-rm
                  ∧ {REMOVE} = rights rm-cap)
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        let ?new-cap = (| target = target rm-cap,
                          rights = (rights rm-cap) − {right-to-rm}|)
        have c1: ((caps s') did) = (insert ?new-cap ?cs-rest)
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: ∀ v. v≠did
                ⟶ ((caps s') v) = ((caps s) v)
          using b0 c0 p2 remove-cap-right-def by auto
        have c3: ∀ v. v≠did
                ⟶ get-domain-cap-set-from-domain-id s v
                  = get-domain-cap-set-from-domain-id s' v
          using c2 get-domain-cap-set-from-domain-id-def by auto
        have c4: get-domain-cap-set-from-domain-id s d
                  = get-domain-cap-set-from-domain-id s' d
          using c3 a0 by auto
        then show ?thesis by auto
      qed
    then show ?thesis by auto
  next
    assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
                ∧ REMOVE ∈ rights rm-cap
                ∧ right-to-rm ∈ rights rm-cap)
    have b1: s' = s
      using b0 p2 remove-cap-right-def by auto
    have b2: get-domain-cap-set-from-domain-id s d
            = get-domain-cap-set-from-domain-id s' d
      using b1 get-domain-cap-set-from-domain-id-def by auto
    then show ?thesis by auto
  qed
  }
then show ?thesis by auto
qed
```

**lemma** *remove-cap-right-notchg-policy*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: $\neg$(*interferes did s d*)
  **and** *p2*: *s′ = fst* (*remove-cap-right s did rm-cap right-to-rm*)
**shows** (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
**proof** −
**{**
  **have** *a0*: *d* ≠ *did*
    **using** *p1 interferes-def* **by** *auto*
  **have** (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
    **proof** (*cases rm-cap* ∈ *get-domain-cap-set-from-domain-id s did*
          ∧ *REMOVE* ∈ *rights rm-cap*
          ∧ *right-to-rm* ∈ *rights rm-cap*)
    **assume** *b0*: *rm-cap* ∈ *get-domain-cap-set-from-domain-id s did*
          ∧ *REMOVE* ∈ *rights rm-cap*
          ∧ *right-to-rm* ∈ *rights rm-cap*
    **have** *a1*: *d* ≠ *target rm-cap*
      **by** (*metis b0 interferes-def p1*)
    **have** (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
      **proof** (*cases REMOVE* = *right-to-rm*
            ∧ {*REMOVE*} = *rights rm-cap*)
      **assume** *c0*: *REMOVE* = *right-to-rm*
            ∧ {*REMOVE*} = *rights rm-cap*
      **let** *?cs-dst* = *get-domain-cap-set-from-domain-id s did*
      **let** *?cs-rest* = {*c*. *c*∈*?cs-dst* ∧ *c*≠*rm-cap*}
      **have** *c1*: ((*caps s′*) *did*) = *?cs-rest*
        **using** *b0 c0 p2 remove-cap-right-def* **by** *auto*
      **have** *c2*: ∀ *v*. *v*≠*did*
            ⟶ ((*caps s′*) *v*) = ((*caps s*) *v*)
        **using** *b0 c0 p2 remove-cap-right-def* **by** *auto*
      **have** *c3*: ∀ *v*. *v*≠*did*
            ⟶ *get-domain-cap-set-from-domain-id s v*
              = *get-domain-cap-set-from-domain-id s′ v*
        **using** *c2 get-domain-cap-set-from-domain-id-def* **by** *auto*
      **have** *c4*: *get-domain-cap-set-from-domain-id s d*
              = *get-domain-cap-set-from-domain-id s′ d*
        **using** *c3 a0* **by** *auto*
      **have** *c5*: *get-domain-cap-set-from-domain-id s d*
              = *get-domain-cap-set-from-domain-id s′ d*
        **using** *c3 a0* **by** *auto*
      **have** *c6*: ∀ *v*. *v*≠*did*
            ⟶ *interferes v s d* ⟷ *interferes v s′ d*
        **using** *c3 c5 interferes-def* **by** *auto*
      **have** *c7*: ((*caps s′*) *did*) = (*?cs-rest*)
        **using** *b0 c0 p2 remove-cap-right-def* **by** *auto*
      **have** *c8*: *get-domain-cap-set-from-domain-id s′ did*
          = *?cs-rest*
        **using** *c7 get-domain-cap-set-from-domain-id-def* **by** *auto*
      **have** *c9*: *get-domain-cap-set-from-domain-id s′ did*
            ⊆ *get-domain-cap-set-from-domain-id s did*
        **using** *c8* **by** *auto*
      **have** *c10*: ¬(∃ *c*. *c*∈(*get-domain-cap-set-from-domain-id s did*) ∧ *target c* = *d*)
        **by** (*metis interferes-def p1*)
      **have** *c11*: ¬(∃ *c*. *c*∈(*get-domain-cap-set-from-domain-id s′ did*) ∧ *target c* = *d*)
        **using** *c10 c9* **by** *auto*
      **have** *c12*: ¬(*interferes did s′ d*)
        **using** *a0 c11 interferes-def* **by** *auto*
      **have** *c13*: *interferes did s′ d* = *interferes did s d*

```
      using c12 p1 by auto
    have c14: (∀ v. interferes v s d ⟷ interferes v s' d)
      using c6 c13 by auto
    then show ?thesis by auto
  next
    assume c0: ¬(REMOVE = right-to-rm
          ∧ {REMOVE} = rights rm-cap)
    let ?cs-dst = get-domain-cap-set-from-domain-id s did
    let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
    let ?new-cap = (| target = target rm-cap,
                rights = (rights rm-cap) − {right-to-rm}|)
    have c1: ((caps s') did) = (insert ?new-cap ?cs-rest)
      using b0 c0 p2 remove-cap-right-def by auto
    have c2: ∀ v. v≠did
          ⟶ ((caps s') v) = ((caps s) v)
      using b0 c0 p2 remove-cap-right-def by auto
    have c3: ∀ v. v≠did
          ⟶ get-domain-cap-set-from-domain-id s v
            = get-domain-cap-set-from-domain-id s' v
      using c2 get-domain-cap-set-from-domain-id-def by auto
    have c4: get-domain-cap-set-from-domain-id s d
            = get-domain-cap-set-from-domain-id s' d
      using c3 a0 by auto
    have c5: get-domain-cap-set-from-domain-id s d
            = get-domain-cap-set-from-domain-id s' d
      using c3 a0 by auto
    have c6: ∀ v. v≠did
          ⟶ interferes v s d ⟷ interferes v s' d
      using c3 c5 interferes-def by auto
    have c7: get-domain-cap-set-from-domain-id s' did
            = (insert ?new-cap ?cs-rest)
      using c1 get-domain-cap-set-from-domain-id-def by auto
    have c8: ¬(∃ c. c∈(get-domain-cap-set-from-domain-id s did) ∧ target c = d)
      by (metis interferes-def p1)
    have c9: ¬(∃ c. c∈(get-domain-cap-set-from-domain-id s' did) ∧ target c = d)
      using c8 c7 a1 by auto
    have c10: ¬(interferes did s' d)
      using a0 c9 interferes-def by auto
    have c11: interferes did s' d = interferes did s d
      using c10 p1 by auto
    have c12: (∀ v. interferes v s d ⟷ interferes v s' d)
      using c6 c11 by auto
    then show ?thesis by auto
  qed
 then show ?thesis by auto
next
 assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
          ∧ REMOVE ∈ rights rm-cap
          ∧ right-to-rm ∈ rights rm-cap)
 have b1: s' = s
   using b0 p2 remove-cap-right-def by auto
 have b2: (∀ v. interferes v s d ⟷ interferes v s' d)
   using b1 interferes-def by auto
 then show ?thesis by auto
qed
}
then show ?thesis by auto
qed
```

```
lemma remove-cap-right-notchg-dom-eps:
assumes p0: reachable0 s
  and   p2: s' = fst (remove-cap-right s did rm-cap right-to-rm)
shows   get-endpoints-of-domain s d = get-endpoints-of-domain s' d
proof −
{
  have get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    proof (cases rm-cap ∈  get-domain-cap-set-from-domain-id s did
           ∧ REMOVE ∈ rights rm-cap
           ∧ right-to-rm ∈ rights rm-cap)
    assume b0: rm-cap ∈  get-domain-cap-set-from-domain-id s did
           ∧ REMOVE ∈ rights rm-cap
           ∧ right-to-rm ∈ rights rm-cap
    have get-endpoints-of-domain s d = get-endpoints-of-domain s' d
      proof (cases  REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap)
        assume c0: REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        have c1: ((caps s') did) = ?cs-rest
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
          using b0 c0 p2 remove-cap-right-def get-endpoints-of-domain-def by auto
        then show ?thesis by auto
      next
        assume c0: ¬(REMOVE = right-to-rm
                ∧ {REMOVE} = rights rm-cap)
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        let ?new-cap = (| target = target rm-cap,
                         rights = (rights rm-cap) − {right-to-rm}|)
        have c1: ((caps s') did) = (insert ?new-cap ?cs-rest)
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
          using b0 c0 p2 remove-cap-right-def get-endpoints-of-domain-def by auto
        then show ?thesis by auto
      qed
    then show ?thesis by auto
  next
    assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
                ∧ REMOVE ∈ rights rm-cap
                ∧ right-to-rm ∈ rights rm-cap)
    have b1: s' = s
      using b0 p2 remove-cap-right-def by auto
    have b2: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
      using b1 get-endpoints-of-domain-def by auto
    then show ?thesis by auto
  qed
  }
then show ?thesis by auto
qed

lemma remove-cap-right-notchg-ep-msgs:
assumes p0: reachable0 s
  and   p2: s' = fst (remove-cap-right s did rm-cap right-to-rm)
shows   (∀ ep. ep∈get-endpoints-of-domain s d
```

```
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
proof −
{
  have (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
    proof (cases rm-cap ∈  get-domain-cap-set-from-domain-id s did
           ∧ REMOVE ∈ rights rm-cap
           ∧ right-to-rm ∈ rights rm-cap)
    assume b0: rm-cap ∈  get-domain-cap-set-from-domain-id s did
           ∧ REMOVE ∈ rights rm-cap
           ∧ right-to-rm ∈ rights rm-cap
    have (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
      proof (cases  REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap)
        assume c0: REMOVE = right-to-rm
              ∧ {REMOVE} = rights rm-cap
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        have c1: ((caps s') did) = ?cs-rest
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: (∀ ep. ep∈get-endpoints-of-domain s d
          ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
          using b0 c0 p2 remove-cap-right-def get-endpoints-of-domain-def
              get-msg-set-from-endpoint-id-def by auto
        then show ?thesis by auto
      next
        assume c0: ¬(REMOVE = right-to-rm
                ∧ {REMOVE} = rights rm-cap)
        let ?cs-dst = get-domain-cap-set-from-domain-id s did
        let ?cs-rest = {c. c∈?cs-dst ∧ c≠rm-cap}
        let ?new-cap = (| target = target rm-cap,
                        rights = (rights rm-cap) − {right-to-rm}|)
        have c1: ((caps s') did) = (insert ?new-cap ?cs-rest)
          using b0 c0 p2 remove-cap-right-def by auto
        have c2: (∀ ep. ep∈get-endpoints-of-domain s d
          ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
          using b0 c0 p2 remove-cap-right-def get-endpoints-of-domain-def
              get-msg-set-from-endpoint-id-def by auto
        then show ?thesis by auto
      qed
    then show ?thesis by auto
  next
    assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
              ∧ REMOVE ∈ rights rm-cap
              ∧ right-to-rm ∈ rights rm-cap)
    have b1: s' = s
      using b0 p2 remove-cap-right-def by auto
    have b2: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep )
      using b1 get-endpoints-of-domain-def by auto
    then show ?thesis by auto
  qed
  }
then show ?thesis by auto
qed

lemma remove-cap-right-lcl-resp:
```

**assumes** *p0*: *reachable0 s*
**and** *p1*: ¬(*interferes did s d*)
**and** *p2*: *s′* = *fst* (*remove-cap-right s did rm-cap right-to-rm*)
**shows** *s* ∼ *d* ∼ *s′*
**proof** −
**{**
**have** *a0*: *did* ≠ *d*
**using** *p1 interferes-def* **by** *auto*
**have** *a1*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id s′ d*
**using** *p0 p1 p2 remove-cap-right-notchg-domain-cap-set* **by** *auto*
**have** *a2*: (∀ *v*. *interferes v s d* ⟷ *interferes v s′ d*)
**using** *p0 p1 p2 remove-cap-right-notchg-policy* **by** *auto*
**have** *a3*: *get-endpoints-of-domain s d* = *get-endpoints-of-domain s′ d*
**using** *p0 p1 p2 remove-cap-right-notchg-dom-eps* **by** *auto*
**have** *a4*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id s′ ep* )
**using** *p0 p1 p2 remove-cap-right-notchg-ep-msgs* **by** *auto*
**have** *a5*: *s* ∼ *d* ∼ *s′*
**using** *a1 a2 a3 a4* **by** *auto*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *remove-cap-right-lcl-resp-e*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *a* = (*Remove-Cap-Right did dst-cap right-to-rm*)
**and** *p2*: ¬(*interferes* (*the* (*domain-of-event a*)) *s d*)
**and** *p3*: *s′* = *exec-event s a*
**shows** *s* ∼ *d* ∼ *s′*
**proof** −
**{**
**have** *a0*: (*the* (*domain-of-event a*)) = *did*
**using** *p1 domain-of-event-def* **by** *auto*
**have** *a1*: *s′* = *fst* (*remove-cap-right s did dst-cap right-to-rm*)
**using** *p1 p3 exec-event-def* **by** *auto*
**have** *a2*: ¬(*interferes did s d*)
**using** *p2 a0* **by** *auto*
**have** *a3*: *s* ∼ *d* ∼ *s′*
**using** *a1 a2 p0 remove-cap-right-lcl-resp* **by** *blast*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *remove-cap-right-lcrsp-e*: *dynamic-local-respect-e* (*Remove-Cap-Right did dst-cap right-to-rm*)
**proof** −
**{**
**have** ∀ *d s*. *reachable0 s*
∧ ¬(*interferes* (*the* (*domain-of-event* (*Remove-Cap-Right did dst-cap right-to-rm*))) *s d*)
⟶ (*s* ∼ *d* ∼ (*exec-event s* (*Remove-Cap-Right did dst-cap right-to-rm*)))
**proof** −
**{**
**fix** *d s*
**assume** *p1*: *reachable0 s*
**assume** *p2*: ¬(*interferes* (*the* (*domain-of-event* (*Remove-Cap-Right did dst-cap right-to-rm*))) *s d*)
**have** (*s* ∼ *d* ∼ (*exec-event s* (*Remove-Cap-Right did dst-cap right-to-rm*)))
**using** *p1 p2 remove-cap-right-lcl-resp-e* **by** *blast*
**}**

**then show** *?thesis* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-local-respect-e-def* **by** *blast*
**qed**

### 0.9.8 proving the "dynamic local respect" property

**definition** *dynamic-local-respect-c* :: *bool* **where**
*dynamic-local-respect-c* ≡ ∀ *a d s*. *reachable0 s*
∧ ¬(*interferes* (*the* (*domain-of-event a*)) *s d*)
⟶ (*s* ∼ *d* ∼ (*exec-event s a*))

**theorem** *dynamic-local-respect*:*dynamic-local-respect*
**proof** −
{
**fix** *e*
**have** *dynamic-local-respect-e e*
**proof** (*induct e*)
**case** (*Client-Lookup-Endpoint-Name x x1*)
**show** *?case*
**using** *client-lookup-endpoint-name-lcrsp-e* **by** *blast*
**case** (*Send-Queuing-Message x1a x2 x3a*)
**show** *?case*
**using** *send-queuing-message-lcl-lcrsp-e* **by** *blast*
**case** (*Receive-Queuing-Message x*)
**show** *?case*
**using** *receive-queuing-message-lcrsp-e* **by** *blast*
**case** (*Get-My-Endpoints-Set x*)
**show** *?case*
**using** *get-my-endpoints-set-lcrsp-e* **by** *blast*
**case** (*Get-Caps x*)
**show** *?case*
**using** *get-caps-lcrsp-e* **by** *blast*
**case** (*Grant-Endpoint-Cap x1a x2 x3a*)
**show** *?case*
**using** *grant-endpoint-cap-lcrsp-e* **by** *blast*
**case** (*Remove-Cap-Right x1a x2 x3a*)
**show** *?case*
**using** *remove-cap-right-lcrsp-e* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-local-respect-all-evt* **by** *blast*
**qed**

## 0.10 Concrete unwinding condition of "weakly step consistent"

### 0.10.1 proving "client lookup endpoint name" satisfying the "weakly step consistent" property

**lemma** *client-lookup-endpoint-name-wsc*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: *s* ∼ *d* ∼ *t*
**and** *p3*: *interferes did s d*
**and** *p4*: *s* ∼ *did* ∼ *t*
**and** *p5*: *s'* = *fst* (*client-lookup-endpoint-name sysconf s did ename*)
**and** *p6*: *t'* = *fst* (*client-lookup-endpoint-name sysconf t did ename*)

```
 shows   s' ∼ d ∼ t'
 proof −
 {
  have a0: s = s'
    using p5 client-lookup-endpoint-name-def by auto
  have a1: t = t'
    using p6 client-lookup-endpoint-name-def by auto
  have a2: s' ∼ d ∼ t'
    using a0 a1 p2 by auto
 }
 then show ?thesis by auto
 qed

 lemma client-lookup-endpoint-name-wsc-e:
 assumes p0: reachable0 s
   and   p1: reachable0 t
   and   p2: a = (Client-Lookup-Endpoint-Name did ename)
   and   p3: s ∼ d ∼ t
   and   p4: interferes (the (domain-of-event a)) s d
   and   p5: s ∼ did ∼ t
   and   p6: s' = exec-event s a
   and   p7: t' = exec-event t a
 shows   s' ∼ d ∼ t'
 proof −
 {
  have a0: (the (domain-of-event a)) = did
    using p2 domain-of-event-def by auto
  have a1: s' = fst (client-lookup-endpoint-name sysconf s did ename)
    using p2 p6 exec-event-def by auto
  have a2: t' = fst (client-lookup-endpoint-name sysconf t did ename)
    using p2 p7 exec-event-def by auto
  have a3: (interferes did s d)
    using p4 a0 by auto
  have a4: s' ∼ d ∼ t'
    using a1 a2 a3 p0 p1 p3 p5 client-lookup-endpoint-name-wsc by blast
 }
 then show ?thesis by auto
 qed

lemma client-lookup-endpoint-name-dwsc-e: dynamic-weakly-step-consistent-e (Client-Lookup-Endpoint-Name did ename)
 proof −
   {
    have ∀ d s t. reachable0 s ∧ reachable0 t
          ∧ (s ∼ d ∼ t)
          ∧ (interferes (the (domain-of-event (Client-Lookup-Endpoint-Name did ename))) s d)
          ∧ (s ∼ (the (domain-of-event (Client-Lookup-Endpoint-Name did ename))) ∼ t)
       ⟶ ((exec-event s (Client-Lookup-Endpoint-Name did ename)) ∼ d ∼ (exec-event t (Client-Lookup-Endpoint-Name
did ename)))
      proof −
      {
       fix d s t
       assume p1: reachable0 s
       assume p2: reachable0 t
       assume p3: (s ∼ d ∼ t)
       assume p4: (interferes (the (domain-of-event (Client-Lookup-Endpoint-Name did ename))) s d)
       assume p5: (s ∼ (the (domain-of-event (Client-Lookup-Endpoint-Name did ename))) ∼ t)
     have ((exec-event s (Client-Lookup-Endpoint-Name did ename)) ∼ d ∼ (exec-event t (Client-Lookup-Endpoint-Name
did ename)))
```

**by** (*metis Event.simps*(*50*) *client-lookup-endpoint-name-wsc-e domain-of-event-def option.sel p1 p2 p3 p4 p5*)
}
**then show** *?thesis* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-weakly-step-consistent-e-def* **by** *blast*
**qed**

### 0.10.2 proving "send queuing message" satisfying the "weakly step consistent" property

**lemma** *send-queuing-message-wsc-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: $s \sim d \sim t$
**and** *p3*: *interferes did s d*
**and** *p4*: $s \sim did \sim t$
**and** *p5*: $s' = fst\ (send\text{-}queuing\text{-}message\ s\ did\ eid\ m)$
**and** *p6*: $t' = fst\ (send\text{-}queuing\text{-}message\ t\ did\ eid\ m)$
**shows** *get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d*
**proof** −
**{**
**have** *a0*: *get-endpoints-of-domain s d = get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a1*: $(\forall v.\ interferes\ v\ s\ d \longleftrightarrow interferes\ v\ t\ d)$
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a3*: *get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d*
**by** (*meson p2 vpeq1-def*)
**have** *get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d*
**proof** (*cases eid* ∈ *get-endpoints-of-domain s d*)
**assume** *b0*: *eid* ∈ *get-endpoints-of-domain s d*
**have** *b1*: *eid* ∈ {*x*. (*domain-endpoint s*) *x* = *Some d*}
**using** *b0 get-endpoints-of-domain-def* **by** *auto*
**have** *b2*: (*domain-endpoint s*) *eid* = *Some d*
**using** *b1* **by** *auto*
**have** *b3*: *get-domain-id-from-endpoint s eid* = *Some d*
**using** *b2 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b4*: *get-domain-id-from-endpoint s eid* ≠ *None*
**using** *b3* **by** *auto*
**have** *b5*: *eid* ∈ *get-endpoints-of-domain t d*
**using** *b0 a0* **by** *auto*
**have** *b6*: *eid* ∈ {*x*. (*domain-endpoint t*) *x* = *Some d*}
**using** *b5 get-endpoints-of-domain-def* **by** *auto*
**have** *b7*: (*domain-endpoint t*) *eid* = *Some d*
**using** *b6* **by** *auto*
**have** *b8*: *get-domain-id-from-endpoint t eid* = *Some d*
**using** *b7 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b9*: *get-domain-id-from-endpoint t eid* ≠ *None*
**using** *b8* **by** *auto*
**have** *b10*: (*the* (*get-domain-id-from-endpoint s eid*)) = *d*
**using** *b3* **by** *auto*
**have** *b11*: (*the* (*get-domain-id-from-endpoint t eid*)) = *d*
**using** *b8* **by** *auto*
**have** *b12*: *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))
**using** *p3 b10* **by** *auto*
**have** *b13*: *interferes did t* (*the* (*get-domain-id-from-endpoint t eid*))

```
    using a2 b11 by auto
  have b14: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s' d
    using b9 b12 p5 get-domain-cap-set-from-domain-id-def send-queuing-message-def by auto
  have b15: get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t' d
    using b10 b13 p6 get-domain-cap-set-from-domain-id-def send-queuing-message-def by auto
  have b16: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using b14 b15 a3 by auto
  then show ?thesis by auto
 next
  assume b0: eid ∉ get-endpoints-of-domain s d
  have b1: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s' d
    using p5 get-domain-cap-set-from-domain-id-def send-queuing-message-def by auto
  have b2: get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t' d
    using p6 get-domain-cap-set-from-domain-id-def send-queuing-message-def by auto
  have b16: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using b1 b2 a3 by auto
  then show ?thesis by auto
 qed
}
then show ?thesis by auto
qed

lemma send-queuing-message-wsc-policy:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (send-queuing-message s did eid m)
  and   p6: t' = fst (send-queuing-message t did eid m)
shows   (∀ v. interferes v s' d ⟷ interferes v t' d)
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ v. interferes v s d ⟷ interferes v s' d)
    using p0 p5 send-queuing-message-notchg-policy by auto
  have a5: (∀ v. interferes v t d ⟷ interferes v t' d)
    using p1 p6 send-queuing-message-notchg-policy by auto
  have a6: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using a1 a4 a5 by auto
  }
  then show ?thesis by auto
qed

lemma send-queuing-message-wsc-dom-eps:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (send-queuing-message s did eid m)
```

```
  and   p6: t' = fst (send-queuing-message t did eid m)
shows   get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    using p0 p5 send-queuing-message-notchg-dom-eps by auto
  have a5: get-endpoints-of-domain t d = get-endpoints-of-domain t' d
    using p1 p6 send-queuing-message-notchg-dom-eps by auto
  have a6: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using a0 a4 a5 by auto
  }
  then show ?thesis by auto
qed

lemma send-queuing-message-wsc-ep-msgs:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (send-queuing-message s did eid m)
  and   p6: t' = fst (send-queuing-message t did eid m)
shows   (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p2 vpeq1-def)
  have (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    proof (cases eid ∈ get-endpoints-of-domain s d)
      assume b0: eid ∈ get-endpoints-of-domain s d
      have b1: eid ∈ {x. (domain-endpoint s) x = Some d}
        using b0 get-endpoints-of-domain-def by auto
      have b2: (domain-endpoint s) eid = Some d
        using b1 by auto
      have b3: get-domain-id-from-endpoint s eid = Some d
        using b2 get-domain-id-from-endpoint-def by auto
      have b4: get-domain-id-from-endpoint s eid ≠ None
        using b3 by auto
      have b5: eid ∈ get-endpoints-of-domain t d
```

**using** *b0 a0* **by** *auto*
**have** *b6*: *eid* ∈ {*x*. (*domain-endpoint t*) *x* = *Some d*}
**using** *b5 get-endpoints-of-domain-def* **by** *auto*
**have** *b7*: (*domain-endpoint t*) *eid* = *Some d*
**using** *b6* **by** *auto*
**have** *b8*: *get-domain-id-from-endpoint t eid* = *Some d*
**using** *b7 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b9*: *get-domain-id-from-endpoint t eid* ≠ *None*
**using** *b8* **by** *auto*
**have** *b10*: (*the* (*get-domain-id-from-endpoint s eid*)) = *d*
**using** *b3* **by** *auto*
**have** *b11*: (*the* (*get-domain-id-from-endpoint t eid*)) = *d*
**using** *b8* **by** *auto*
**have** *b12*: *interferes did s* (*the* (*get-domain-id-from-endpoint s eid*))
**using** *p3 b10* **by** *auto*
**have** *b13*: *interferes did t* (*the* (*get-domain-id-from-endpoint t eid*))
**using** *a2 b11* **by** *auto*
**let** *?new-msg-set-s* = *insert m* (*get-msg-set-from-endpoint-id s eid*)
**let** *?new-msg-set-t* = *insert m* (*get-msg-set-from-endpoint-id t eid*)
**have** *b14*: (*e-msgs s′*) *eid* = *?new-msg-set-s*
**using** *b4 b12 p5 send-queuing-message-def* **by** *auto*
**have** *b15*: (*e-msgs t′*) *eid* = *?new-msg-set-t*
**using** *b9 b13 p6 send-queuing-message-def* **by** *auto*
**have** *b16*: ∀ *e*. *e*≠*eid*
⟶ ((*e-msgs s*) *e*) = ((*e-msgs s′*) *e*)
**using** *b4 b12 p5 send-queuing-message-def* **by** *auto*
**have** *b17*: ∀ *e*. *e*≠*eid*
⟶ ((*e-msgs t*) *e*) = ((*e-msgs t′*) *e*)
**using** *b9 b13 p6 send-queuing-message-def* **by** *auto*
**have** *b18*: ∀ *e*. *e*≠*eid*
⟶ *get-msg-set-from-endpoint-id s e* = *get-msg-set-from-endpoint-id s′ e*
**using** *b16 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *b19*: ∀ *e*. *e*≠*eid*
⟶ *get-msg-set-from-endpoint-id t e* = *get-msg-set-from-endpoint-id t′ e*
**using** *b17 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *b20*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d* ∧ *ep* ≠ *eid*
⟶ *get-msg-set-from-endpoint-id s′ ep* = *get-msg-set-from-endpoint-id t′ ep* )
**using** *a4 b19 b18* **by** *auto*
**have** *b21*: *get-msg-set-from-endpoint-id s eid* = *get-msg-set-from-endpoint-id t eid*
**using** *a4 b0 b5* **by** *auto*
**have** *b22*: (*e-msgs s′*) *eid* = (*e-msgs t′*) *eid*
**using** *b21 b14 b15* **by** *auto*
**have** *b23*: *get-msg-set-from-endpoint-id s′ eid* = *get-msg-set-from-endpoint-id t′ eid*
**using** *b22 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *b24*: *get-endpoints-of-domain s d* = *get-endpoints-of-domain s′ d*
**using** *p0 p5 send-queuing-message-notchg-dom-eps* **by** *auto*
**have** *b25*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s′ d*
⟶ *get-msg-set-from-endpoint-id s′ ep* = *get-msg-set-from-endpoint-id t′ ep* )
**using** *b23 b24 b20* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: *eid* ∉ *get-endpoints-of-domain s d*
**have** *b1*: ∀ *e*. *e*≠*eid*
⟶ ((*e-msgs s*) *e*) = ((*e-msgs s′*) *e*)
**using** *p5 send-queuing-message-def* **by** *auto*
**have** *b2*: ∀ *e*. *e*≠*eid*
⟶ ((*e-msgs t*) *e*) = ((*e-msgs t′*) *e*)
**using** *p6 send-queuing-message-def* **by** *auto*

```
    have b3: ∀ e. e≠eid
        ⟶ get-msg-set-from-endpoint-id s e = get-msg-set-from-endpoint-id s' e
      using b1 get-msg-set-from-endpoint-id-def by auto
    have b4: ∀ e. e≠eid
        ⟶ get-msg-set-from-endpoint-id t e = get-msg-set-from-endpoint-id t' e
      using b2 get-msg-set-from-endpoint-id-def by auto
    have b5: ∀ e. e∈get-endpoints-of-domain s d ∧ e≠eid
        ⟶ get-msg-set-from-endpoint-id s' e = get-msg-set-from-endpoint-id t' e
      using b3 b4 a4 by auto
    have b6: ∀ e. e∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s' e = get-msg-set-from-endpoint-id t' e
      using b5 b0 by auto
    have b7: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
      using p0 p5 send-queuing-message-notchg-dom-eps by auto
    have b8: ∀ e. e∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' e = get-msg-set-from-endpoint-id t' e
      using b7 b6 by auto
    then show ?thesis by auto
  qed
 }
 then show ?thesis by auto
qed

lemma send-queuing-message-wsc:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (send-queuing-message s did eid m)
  and   p6: t' = fst (send-queuing-message t did eid m)
shows   s' ∼ d ∼ t'
proof −
{
  have a0: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using p0 p1 p2 p3 p4 p5 p6 send-queuing-message-wsc-domain-cap-set by blast
  have a1: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using p0 p1 p2 p3 p4 p5 p6 send-queuing-message-wsc-policy by blast
  have a2: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using p0 p1 p2 p3 p4 p5 p6 send-queuing-message-wsc-dom-eps by blast
  have a3: (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    using p0 p1 p2 p3 p4 p5 p6 send-queuing-message-wsc-ep-msgs by blast
  have a4: s' ∼ d ∼ t'
    using a0 a1 a2 a3 vpeq1-def by auto
}
then show ?thesis by auto
qed


lemma send-queuing-message-wsc-e:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: a = (Send-Queuing-Message did eid m)
  and   p3: s ∼ d ∼ t
  and   p4: interferes (the (domain-of-event a)) s d
  and   p5: s ∼ did ∼ t
  and   p6: s' = exec-event s a
```

**and** *p7*: $t' = exec\text{-}event\ t\ a$
**shows** $s' \sim d \sim t'$
**proof** −
{
**have** *a0*: $(the\ (domain\text{-}of\text{-}event\ a)) = did$
**using** *p2 domain-of-event-def* **by** *auto*
**have** *a1*: $s' = fst\ (send\text{-}queuing\text{-}message\ s\ did\ eid\ m)$
**using** *p2 p6 exec-event-def* **by** *auto*
**have** *a2*: $t' = fst\ (send\text{-}queuing\text{-}message\ t\ did\ eid\ m)$
**using** *p2 p7 exec-event-def* **by** *auto*
**have** *a3*: $(interferes\ did\ s\ d)$
**using** *p4 a0* **by** *auto*
**have** *a4*: $s' \sim d \sim t'$
**using** *a1 a2 a3 p0 p1 p3 p5 send-queuing-message-wsc* **by** *blast*
}
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *send-queuing-message-dwsc-e*: *dynamic-weakly-step-consistent-e* $(Send\text{-}Queuing\text{-}Message\ did\ eid\ m)$
**proof** −
{
**have** $\forall\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t$
$\wedge\ (s \sim d \sim t)$
$\wedge\ (interferes\ (the\ (domain\text{-}of\text{-}event\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)))\ s\ d)$
$\wedge\ (s \sim (the\ (domain\text{-}of\text{-}event\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m))) \sim t)$
$\longrightarrow ((exec\text{-}event\ s\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)) \sim d \sim (exec\text{-}event\ t\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)))$
**proof** −
{
**fix** *d s t*
**assume** *p1*: *reachable0 s*
**assume** *p2*: *reachable0 t*
**assume** *p3*: $(s \sim d \sim t)$
**assume** *p4*: $(interferes\ (the\ (domain\text{-}of\text{-}event\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)))\ s\ d)$
**assume** *p5*: $(s \sim (the\ (domain\text{-}of\text{-}event\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m))) \sim t)$
**have** $((exec\text{-}event\ s\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)) \sim d \sim (exec\text{-}event\ t\ (Send\text{-}Queuing\text{-}Message\ did\ eid\ m)))$
**by** $(metis\ Event.simps(51)\ domain\text{-}of\text{-}event\text{-}def\ option.sel\ p1\ p2\ p3\ p4\ p5\ send\text{-}queuing\text{-}message\text{-}wsc\text{-}e)$
}
**then show** *?thesis* **by** *blast*
**qed**
}
**then show** *?thesis*
**using** *dynamic-weakly-step-consistent-e-def* **by** *blast*
**qed**

### 0.10.3 proving "receive queuing message" satisfying the "weakly step consistent" property

**lemma** *receive-queuing-message-wsc-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: $s \sim d \sim t$
**and** *p3*: *interferes did s d*
**and** *p4*: $s \sim did \sim t$
**and** *p5*: $s' = fst\ (receive\text{-}queuing\text{-}message\ s\ did\ eid)$
**and** *p6*: $t' = fst\ (receive\text{-}queuing\text{-}message\ t\ did\ eid)$
**shows** $get\text{-}domain\text{-}cap\text{-}set\text{-}from\text{-}domain\text{-}id\ s'\ d = get\text{-}domain\text{-}cap\text{-}set\text{-}from\text{-}domain\text{-}id\ t'\ d$
**proof** −

{
**have** *a0*: *get-endpoints-of-domain s d = get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a1*: (∀ *v*. *interferes v s d* ⟷ *interferes v t d*)
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a3*: *get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d*
**by** (*meson p2 vpeq1-def*)
**have** *a4*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep* )
**by** (*meson p2 vpeq1-def*)
**have** *a5*: *get-endpoints-of-domain s did = get-endpoints-of-domain t did*
**by** (*meson p4 vpeq1-def*)
**have** *a6*: *get-domain-cap-set-from-domain-id s′ d = get-domain-cap-set-from-domain-id t′ d*
**proof** (*cases get-domain-id-from-endpoint s eid = Some did*)
**assume** *b0*: *get-domain-id-from-endpoint s eid = Some did*
**have** *b1*: ((*domain-endpoint s*) *eid*) = *Some did*
**using** *b0 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b2*: {*x*. (*domain-endpoint s*) *x* = *Some did*} = {*x*. (*domain-endpoint t*) *x* = *Some did*}
**using** *a5 b1 get-endpoints-of-domain-def* **by** *auto*
**have** *b3*: ((*domain-endpoint t*) *eid*) = *Some did*
**using** *b1 b2* **by** *auto*
**have** *b4*: *get-domain-id-from-endpoint t eid = Some did*
**using** *b3 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b5*: *get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s′ d*
**using** *b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def* **by** *auto*
**have** *b6*: *get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t′ d*
**using** *b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def* **by** *auto*
**have** *b7*: *get-domain-cap-set-from-domain-id s′ d = get-domain-cap-set-from-domain-id t′ d*
**using** *b5 b6 a3* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: *get-domain-id-from-endpoint s eid* ≠ *Some did*
**have** *b1*: ((*domain-endpoint s*) *eid*) ≠ *Some did*
**using** *b0 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b2*: {*x*. (*domain-endpoint s*) *x* = *Some did*} = {*x*. (*domain-endpoint t*) *x* = *Some did*}
**using** *a5 b1 get-endpoints-of-domain-def* **by** *auto*
**have** *b3*: ((*domain-endpoint t*) *eid*) ≠ *Some did*
**using** *b1 b2* **by** *auto*
**have** *b4*: *get-domain-id-from-endpoint t eid* ≠ *Some did*
**using** *b3 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b5*: *get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s′ d*
**using** *b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def* **by** *auto*
**have** *b6*: *get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t′ d*
**using** *b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def* **by** *auto*
**have** *b7*: *get-domain-cap-set-from-domain-id s′ d = get-domain-cap-set-from-domain-id t′ d*
**using** *b5 b6 a3* **by** *auto*
**then show** *?thesis* **by** *auto*
**qed**
}
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *receive-queuing-message-wsc-policy*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: *s* ∼ *d* ∼ *t*

```
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (receive-queuing-message s did eid)
  and   p6: t' = fst (receive-queuing-message t did eid)
shows   (∀ v. interferes v s' d ⟷ interferes v t' d)
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p2 vpeq1-def)
  have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
    by (meson p4 vpeq1-def)
  have a6: (∀ v. interferes v s' d ⟷ interferes v t' d)
    proof (cases get-domain-id-from-endpoint s eid = Some did)
      assume b0: get-domain-id-from-endpoint s eid = Some did
      have b1: ((domain-endpoint s) eid) = Some did
        using b0 get-domain-id-from-endpoint-def by auto
      have b2: {x. (domain-endpoint s) x = Some did} = {x. (domain-endpoint t) x = Some did}
        using a5 b1 get-endpoints-of-domain-def by auto
      have b3: ((domain-endpoint t) eid) = Some did
        using b1 b2 by auto
      have b4: get-domain-id-from-endpoint t eid = Some did
        using b3 get-domain-id-from-endpoint-def by auto
      have b5: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s' d
        using b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
      have b6: get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t' d
        using b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
      have b7: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
        using b5 b6 a3 by auto
      have b8: ∀ v. get-domain-cap-set-from-domain-id s v
        = get-domain-cap-set-from-domain-id s' v
        using b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
      have b9: (∀ v. interferes v s d ⟷ interferes v s' d)
        using b5 b8 interferes-def by auto
      have b10: ∀ v. get-domain-cap-set-from-domain-id t v
        = get-domain-cap-set-from-domain-id t' v
        using b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
      have b11: (∀ v. interferes v t d ⟷ interferes v t' d)
        using b6 b10 interferes-def by auto
      have b12: (∀ v. interferes v s' d ⟷ interferes v t' d)
        using b9 b11 a1 by auto
      then show ?thesis by auto
    next
      assume b0: get-domain-id-from-endpoint s eid ≠ Some did
      have b1: ((domain-endpoint s) eid) ≠ Some did
        using b0 get-domain-id-from-endpoint-def by auto
      have b2: {x. (domain-endpoint s) x = Some did} = {x. (domain-endpoint t) x = Some did}
        using a5 b1 get-endpoints-of-domain-def by auto
      have b3: ((domain-endpoint t) eid) ≠ Some did
        using b1 b2 by auto
```

```
    have b4: get-domain-id-from-endpoint t eid ≠ Some did
      using b3 get-domain-id-from-endpoint-def by auto
    have b5: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id s' d
      using b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
    have b6: get-domain-cap-set-from-domain-id t d = get-domain-cap-set-from-domain-id t' d
      using b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
    have b7: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
      using b5 b6 a3 by auto
    have b8: ∀ v. get-domain-cap-set-from-domain-id s v
      = get-domain-cap-set-from-domain-id s' v
      using b0 p5 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
    have b9: (∀ v. interferes v s d ⟷ interferes v s' d)
      using b5 b8 interferes-def by auto
    have b10: ∀ v. get-domain-cap-set-from-domain-id t v
      = get-domain-cap-set-from-domain-id t' v
      using b4 p6 get-domain-cap-set-from-domain-id-def receive-queuing-message-def by auto
    have b11: (∀ v. interferes v t d ⟷ interferes v t' d)
      using b6 b10 interferes-def by auto
    have b12: (∀ v. interferes v s' d ⟷ interferes v t' d)
      using b9 b11 a1 by auto
    then show ?thesis by auto
  qed
 }
 then show ?thesis by auto
qed

lemma receive-queuing-message-wsc-dom-eps:
assumes p0: reachable0 s
 and   p1: reachable0 t
 and   p2: s ∼ d ∼ t
 and   p3: interferes did s d
 and   p4: s ∼ did ∼ t
 and   p5: s' = fst (receive-queuing-message s did eid)
 and   p6: t' = fst (receive-queuing-message t did eid)
shows   get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
proof −
{
 have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
   by (meson p2 vpeq1-def)
 have a1: (∀ v. interferes v s d ⟷ interferes v t d)
   by (meson p2 vpeq1-def)
 have a2: interferes did t d
   using p3 a1 by auto
 have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
   by (meson p2 vpeq1-def)
 have a4: (∀ ep. ep∈get-endpoints-of-domain s d
       ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
   by (meson p2 vpeq1-def)
 have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
   by (meson p4 vpeq1-def)
 have a6: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
   proof (cases get-domain-id-from-endpoint s eid = Some did)
     assume b0: get-domain-id-from-endpoint s eid = Some did
     have b1: ((domain-endpoint s) eid) = Some did
       using b0 get-domain-id-from-endpoint-def by auto
     have b2: {x. (domain-endpoint s) x = Some did} = {x. (domain-endpoint t) x = Some did}
       using a5 b1 get-endpoints-of-domain-def by auto
     have b3: ((domain-endpoint t) eid) = Some did
```

```
    using b1 b2 by auto
  have b4: get-domain-id-from-endpoint t eid = Some did
    using b3 get-domain-id-from-endpoint-def by auto
  have b5: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    using b0 p5 get-endpoints-of-domain-def receive-queuing-message-def by auto
  have b6: get-endpoints-of-domain t d = get-endpoints-of-domain t' d
    using b4 p6 get-endpoints-of-domain-def receive-queuing-message-def by auto
  have b7: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using b5 b6 a0 by auto
  then show ?thesis by auto
next
  assume b0: get-domain-id-from-endpoint s eid ≠ Some did
  have b1: ((domain-endpoint s) eid) ≠ Some did
    using b0 get-domain-id-from-endpoint-def by auto
  have b2: {x. (domain-endpoint s) x = Some did} = {x. (domain-endpoint t) x = Some did}
    using a5 b1 get-endpoints-of-domain-def by auto
  have b3: ((domain-endpoint t) eid) ≠ Some did
    using b1 b2 by auto
  have b4: get-domain-id-from-endpoint t eid ≠ Some did
    using b3 get-domain-id-from-endpoint-def by auto
  have b5: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
    using b0 p5 get-endpoints-of-domain-def receive-queuing-message-def by auto
  have b6: get-endpoints-of-domain t d = get-endpoints-of-domain t' d
    using b4 p6 get-endpoints-of-domain-def receive-queuing-message-def by auto
  have b7: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using b5 b6 a0 by auto
  then show ?thesis by auto
qed
}
then show ?thesis by auto
qed

lemma receive-queuing-message-wsc-ep-msgs:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (receive-queuing-message s did eid)
  and   p6: t' = fst (receive-queuing-message t did eid)
shows   (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p2 vpeq1-def)
  have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
    by (meson p4 vpeq1-def)
  have a6: (∀ ep. ep∈get-endpoints-of-domain s did
```

$\longrightarrow$ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep* )
**by** (*meson p4 vpeq1-def*)
**have** ($\forall$ *ep*. *ep*$\in$*get-endpoints-of-domain s' d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep* )
**proof** (*cases get-domain-id-from-endpoint s eid = Some did*)
**assume** *b0*: *get-domain-id-from-endpoint s eid = Some did*
**have** *b1*: ((*domain-endpoint s*) *eid*) = *Some did*
**using** *b0 get-domain-id-from-endpoint-def* **by** *auto*
**have** *b2*: {*x*. (*domain-endpoint s*) *x* = *Some did*} = {*x*. (*domain-endpoint t*) *x* = *Some did*}
**using** *a5 b1 get-endpoints-of-domain-def* **by** *auto*
**have** *b3*: ((*domain-endpoint t*) *eid*) = *Some did*
**using** *b1 b2* **by** *auto*
**have** *b4*: *get-domain-id-from-endpoint t eid = Some did*
**using** *b3 get-domain-id-from-endpoint-def* **by** *auto*
**have** ($\forall$ *ep*. *ep*$\in$*get-endpoints-of-domain s' d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep* )
**proof** (*cases d* $\neq$ *did*)
**assume** *c0*: *d* $\neq$ *did*
**have** *c1*: $\forall$ *e*. *e*$\neq$*eid*
$\longrightarrow$ ((*e-msgs s*) *e*) = ((*e-msgs s'*) *e*)
**using** *b0 p5 receive-queuing-message-def* **by** *auto*
**have** *c2*: *get-domain-id-from-endpoint s eid* $\neq$ *Some d*
**using** *c0 b0* **by** *auto*
**have** *c3*: ((*domain-endpoint s*) *eid*) $\neq$ *Some d*
**using** *c2 get-domain-id-from-endpoint-def* **by** *auto*
**have** *c4*: *eid* $\notin$ {*x*. (*domain-endpoint s*) *x* = *Some d*}
**using** *c3* **by** *auto*
**have** *c5*: *eid* $\notin$ *get-endpoints-of-domain s d*
**using** *c4 get-endpoints-of-domain-def* **by** *auto*
**have** *c6*: $\forall$ *e*. *e*$\in$*get-endpoints-of-domain s d*
$\longrightarrow$ ((*e-msgs s*) *e*) = ((*e-msgs s'*) *e*)
**using** *c1 c5* **by** *auto*
**have** *c7*: $\forall$ *e*. *e*$\in$*get-endpoints-of-domain s d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s e*
= *get-msg-set-from-endpoint-id s' e*
**using** *c6 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *c8*: $\forall$ *e*. *e*$\neq$*eid*
$\longrightarrow$ ((*e-msgs t*) *e*) = ((*e-msgs t'*) *e*)
**using** *b4 p6 receive-queuing-message-def* **by** *auto*
**have** *c9*: *get-domain-id-from-endpoint t eid* $\neq$ *Some d*
**using** *c0 b4* **by** *auto*
**have** *c10*: ((*domain-endpoint t*) *eid*) $\neq$ *Some d*
**using** *c9 get-domain-id-from-endpoint-def* **by** *auto*
**have** *c11*: *eid* $\notin$ {*x*. (*domain-endpoint t*) *x* = *Some d*}
**using** *c10* **by** *auto*
**have** *c12*: *eid* $\notin$ *get-endpoints-of-domain t d*
**using** *c11 get-endpoints-of-domain-def* **by** *auto*
**have** *c13*: $\forall$ *e*. *e*$\in$*get-endpoints-of-domain t d*
$\longrightarrow$ ((*e-msgs t*) *e*) = ((*e-msgs t'*) *e*)
**using** *c8 c12* **by** *auto*
**have** *c14*: $\forall$ *e*. *e*$\in$*get-endpoints-of-domain t d*
$\longrightarrow$ *get-msg-set-from-endpoint-id t e*
= *get-msg-set-from-endpoint-id t' e*
**using** *c13 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *c15*: $\forall$ *e*. *e*$\in$*get-endpoints-of-domain s d*
$\longrightarrow$ *get-msg-set-from-endpoint-id t e*
= *get-msg-set-from-endpoint-id t' e*
**using** *c14 a0* **by** *auto*

**have** *c16*: $\forall e.\ e \in$ *get-endpoints-of-domain s d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s′ e*
$=$ *get-msg-set-from-endpoint-id t′ e*
**using** *c15 c7 a4* **by** *auto*
**have** *c17*: *get-endpoints-of-domain s d* $=$ *get-endpoints-of-domain s′ d*
**using** *p0 p5 receive-queuing-message-notchg-dom-eps* **by** *auto*
**have** *c18*: $\forall e.\ e \in$ *get-endpoints-of-domain s′ d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s′ e*
$=$ *get-msg-set-from-endpoint-id t′ e*
**using** *c16 c17* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *c0*: $\neg\ d \neq did$
**have** *c1*: $\forall e.\ e \neq eid$
$\longrightarrow$ $((e\text{-}msgs\ s)\ e) = ((e\text{-}msgs\ s')\ e)$
**using** *b0 p5 receive-queuing-message-def* **by** *auto*
**have** *c2*: $\forall e.\ e \neq eid$
$\longrightarrow$ $((e\text{-}msgs\ t)\ e) = ((e\text{-}msgs\ t')\ e)$
**using** *b4 p6 receive-queuing-message-def* **by** *auto*
**have** *c3*: $\forall e.\ e \neq eid \wedge e \in$ *get-endpoints-of-domain s d*
$\longrightarrow$ $((e\text{-}msgs\ s')\ e) = ((e\text{-}msgs\ t')\ e)$
**using** *c1 c2 a4 get-msg-set-from-endpoint-id-def* **by** *auto*
**let** *?msg-set-s* $=$ *get-msg-set-from-endpoint-id s eid*
**let** *?m-s* $=$ *SOME x*. $x \in$ *?msg-set-s*
**let** *?new-msg-set-s* $=$ *?msg-set-s* $-$ $\{$*?m-s*$\}$
**let** *?msg-set-t* $=$ *get-msg-set-from-endpoint-id t eid*
**let** *?m-t* $=$ *SOME x*. $x \in$ *?msg-set-t*
**let** *?new-msg-set-t* $=$ *?msg-set-t* $-$ $\{$*?m-t*$\}$
**have** *c4*: $((e\text{-}msgs\ s')\ eid) =$ *?new-msg-set-s*
**using** *b0 p5 receive-queuing-message-def* **by** *auto*
**have** *c5*: *eid* $\in$ *get-endpoints-of-domain s did*
**using** *b1 get-endpoints-of-domain-def* **by** *auto*
**have** *c6*: *get-msg-set-from-endpoint-id s eid*
$=$ *get-msg-set-from-endpoint-id t eid*
**using** *a6 c5* **by** *auto*
**have** *c7*: *?msg-set-s* $=$ *?msg-set-t*
**using** *c6* **by** *auto*
**have** *c8*: *?m-s* $=$ *?m-t*
**using** *c7* **by** *auto*
**have** *c9*: *?new-msg-set-s* $=$ *?new-msg-set-t*
**using** *c7 c8* **by** *auto*
**have** *c10*: $((e\text{-}msgs\ t')\ eid) =$ *?new-msg-set-t*
**using** *b4 p6 receive-queuing-message-def* **by** *auto*
**have** *c11*: $((e\text{-}msgs\ s')\ eid) = ((e\text{-}msgs\ t')\ eid)$
**using** *c4 c9 c10* **by** *auto*
**have** *c12*: *get-msg-set-from-endpoint-id s′ eid*
$=$ *get-msg-set-from-endpoint-id t′ eid*
**using** *c11 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *c13*: $\forall e.\ e \in$ *get-endpoints-of-domain s d*
$\longrightarrow$ $((e\text{-}msgs\ s')\ e) = ((e\text{-}msgs\ t')\ e)$
**using** *c3 c11* **by** *auto*
**have** *c14*: $\forall e.\ e \in$ *get-endpoints-of-domain s d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s′ e*
$=$ *get-msg-set-from-endpoint-id t′ e*
**using** *c13 get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *c15*: *get-endpoints-of-domain s d* $=$ *get-endpoints-of-domain s′ d*
**using** *p0 p5 receive-queuing-message-notchg-dom-eps* **by** *auto*
**have** *c16*: $\forall e.\ e \in$ *get-endpoints-of-domain s′ d*

```
                ⟶ get-msg-set-from-endpoint-id s' e
                = get-msg-set-from-endpoint-id t' e
        using c14 c15 by auto
       then show ?thesis by auto
     qed
     then show ?thesis by auto
   next
     assume b0: get-domain-id-from-endpoint s eid ≠ Some did
     have b1: ((domain-endpoint s) eid) ≠ Some did
       using b0 get-domain-id-from-endpoint-def by auto
     have b2: {x. (domain-endpoint s) x = Some did} = {x. (domain-endpoint t) x = Some did}
       using a5 b1 get-endpoints-of-domain-def by auto
     have b3: ((domain-endpoint t) eid) ≠ Some did
       using b1 b2 by auto
     have b4: get-domain-id-from-endpoint t eid ≠ Some did
       using b3 get-domain-id-from-endpoint-def by auto
     have b5: ∀ e. ((e-msgs s) e) = ((e-msgs s') e)
       using b0 p5 receive-queuing-message-def by auto
     have b6: ∀ e. ((e-msgs t) e) = ((e-msgs t') e)
       using b4 p6 receive-queuing-message-def by auto
     have b7: ∀ e. e∈get-endpoints-of-domain s d
             ⟶ get-msg-set-from-endpoint-id s' e
               = get-msg-set-from-endpoint-id t' e
       using b5 b6 a4 get-msg-set-from-endpoint-id-def by auto
     have b8: get-endpoints-of-domain s d = get-endpoints-of-domain s' d
       using p0 p5 receive-queuing-message-notchg-dom-eps by auto
     have b9: ∀ e. e∈get-endpoints-of-domain s' d
             ⟶ get-msg-set-from-endpoint-id s' e
               = get-msg-set-from-endpoint-id t' e
       using b7 b8 by auto
     then show ?thesis by auto
   qed
 }
 then show ?thesis by auto
qed

lemma receive-queuing-message-wsc:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (receive-queuing-message s did eid)
  and   p6: t' = fst (receive-queuing-message t did eid)
shows   s' ∼ d ∼ t'
proof −
{
  have a0: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using p0 p1 p2 p3 p4 p5 p6 receive-queuing-message-wsc-domain-cap-set by blast
  have a1: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using p0 p1 p2 p3 p4 p5 p6 receive-queuing-message-wsc-policy by blast
  have a2: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using p0 p1 p2 p3 p4 p5 p6 receive-queuing-message-wsc-dom-eps by blast
  have a3: (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    using p0 p1 p2 p3 p4 p5 p6 receive-queuing-message-wsc-ep-msgs by blast
  have a4: s' ∼ d ∼ t'
    using a0 a1 a2 a3 vpeq1-def by auto
```

```
}
then show ?thesis by auto
qed


lemma receive-queuing-message-wsc-e:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: a = (Receive-Queuing-Message did eid)
  and   p3: s ∼ d ∼ t
  and   p4: interferes (the (domain-of-event a)) s d
  and   p5: s ∼ did ∼ t
  and   p6: s' = exec-event s a
  and   p7: t' = exec-event t a
shows   s' ∼ d ∼ t'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p2 domain-of-event-def by auto
  have a1: s' = fst (receive-queuing-message s did eid)
    using p2 p6 exec-event-def by auto
  have a2: t' = fst (receive-queuing-message t did eid)
    using p2 p7 exec-event-def by auto
  have a3: (interferes did s d)
    using p4 a0 by auto
  have a4: s' ∼ d ∼ t'
    using a1 a2 a3 p0 p1 p3 p5 receive-queuing-message-wsc by blast
}
then show ?thesis by auto
qed

lemma receive-queuing-message-dwsc-e: dynamic-weakly-step-consistent-e (Receive-Queuing-Message did eid)
proof −
  {
    have ∀ d s t. reachable0 s ∧ reachable0 t
          ∧ (s ∼ d ∼ t)
          ∧ (interferes (the (domain-of-event (Receive-Queuing-Message did eid))) s d)
          ∧ (s ∼ (the (domain-of-event (Receive-Queuing-Message did eid))) ∼ t)
            ⟶ ((exec-event s (Receive-Queuing-Message did eid)) ∼ d ∼ (exec-event t (Receive-Queuing-Message did
eid)))
      proof −
      {
        fix d s t
        assume p1: reachable0 s
        assume p2: reachable0 t
        assume p3: (s ∼ d ∼ t)
        assume p4: (interferes (the (domain-of-event (Receive-Queuing-Message did eid))) s d)
        assume p5: (s ∼ (the (domain-of-event (Receive-Queuing-Message did eid))) ∼ t)
          have ((exec-event s (Receive-Queuing-Message did eid)) ∼ d ∼ (exec-event t (Receive-Queuing-Message did
eid)))
          by (metis Event.simps(52) domain-of-event-def option.sel p1 p2 p3 p4 p5 receive-queuing-message-wsc-e)
      }
      then show ?thesis by blast
      qed
    }
  then show ?thesis
    using dynamic-weakly-step-consistent-e-def by blast
qed
```

### 0.10.4 proving "get my endpoints set" satisfying the "weakly step consistent" property

**lemma** *get-my-endpoints-set-wsc*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: *reachable0 t*
  **and** *p2*: $s \sim d \sim t$
  **and** *p3*: *interferes did s d*
  **and** *p4*: $s \sim did \sim t$
  **and** *p5*: $s' = fst\ (get\text{-}my\text{-}endpoints\text{-}set\ s\ did)$
  **and** *p6*: $t' = fst\ (get\text{-}my\text{-}endpoints\text{-}set\ t\ did)$
**shows** $s' \sim d \sim t'$
**proof** −
**{**
  **have** *a0*: $s = s'$
    **using** *p5 get-my-endpoints-set-def* **by** *auto*
  **have** *a1*: $t = t'$
    **using** *p6 get-my-endpoints-set-def* **by** *auto*
  **have** *a2*: $s' \sim d \sim t'$
    **using** *a0 a1 p2* **by** *auto*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *get-my-endpoints-set-wsc-e*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: *reachable0 t*
  **and** *p2*: $a = (Get\text{-}My\text{-}Endpoints\text{-}Set\ did)$
  **and** *p3*: $s \sim d \sim t$
  **and** *p4*: *interferes* (*the* (*domain-of-event a*)) *s d*
  **and** *p5*: $s \sim did \sim t$
  **and** *p6*: $s' = exec\text{-}event\ s\ a$
  **and** *p7*: $t' = exec\text{-}event\ t\ a$
**shows** $s' \sim d \sim t'$
**proof** −
**{**
  **have** *a0*: (*the* (*domain-of-event a*)) = *did*
    **using** *p2 domain-of-event-def* **by** *auto*
  **have** *a1*: $s' = fst\ (get\text{-}my\text{-}endpoints\text{-}set\ s\ did)$
    **using** *p2 p6 exec-event-def* **by** *auto*
  **have** *a2*: $t' = fst\ (get\text{-}my\text{-}endpoints\text{-}set\ t\ did)$
    **using** *p2 p7 exec-event-def* **by** *auto*
  **have** *a3*: (*interferes did s d*)
    **using** *p4 a0* **by** *auto*
  **have** *a4*: $s' \sim d \sim t'$
    **using** *a1 a2 a3 p0 p1 p3 p5 get-my-endpoints-set-wsc* **by** *blast*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *get-my-endpoints-set-dwsc-e*: *dynamic-weakly-step-consistent-e* (*Get-My-Endpoints-Set did*)
**proof** −
  **{**
    **have** $\forall\ d\ s\ t.\ reachable0\ s \wedge reachable0\ t$
      $\wedge\ (s \sim d \sim t)$
      $\wedge\ (interferes\ (the\ (domain\text{-}of\text{-}event\ (Get\text{-}My\text{-}Endpoints\text{-}Set\ did)))\ s\ d)$
      $\wedge\ (s \sim (the\ (domain\text{-}of\text{-}event\ (Get\text{-}My\text{-}Endpoints\text{-}Set\ did))) \sim t)$
      $\longrightarrow ((exec\text{-}event\ s\ (Get\text{-}My\text{-}Endpoints\text{-}Set\ did)) \sim d \sim (exec\text{-}event\ t\ (Get\text{-}My\text{-}Endpoints\text{-}Set\ did)))$

```
    proof −
    {
      fix d s t
      assume p1: reachable0 s
      assume p2: reachable0 t
      assume p3: (s ∼ d ∼ t)
      assume p4: (interferes (the (domain-of-event (Get-My-Endpoints-Set did))) s d)
      assume p5: (s ∼ (the (domain-of-event (Get-My-Endpoints-Set did))) ∼ t)
      have ((exec-event s (Get-My-Endpoints-Set did)) ∼ d ∼ (exec-event t (Get-My-Endpoints-Set did)))
        by (metis Event.simps(53) domain-of-event-def option.sel p1 p2 p3 p4 p5 get-my-endpoints-set-wsc-e)
    }
    then show ?thesis by blast
    qed
  }
  then show ?thesis
    using dynamic-weakly-step-consistent-e-def by blast
qed
```

### 0.10.5 proving "get caps" satisfying the "weakly step consistent" property

```
lemma get-caps-wsc:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (get-caps s did)
  and   p6: t' = fst (get-caps t did)
shows   s' ∼ d ∼ t'
proof −
{
  have a0: s = s'
    using p5 get-caps-def by auto
  have a1: t = t'
    using p6 get-caps-def by auto
  have a2: s' ∼ d ∼ t'
    using a0 a1 p2 by auto
}
then show ?thesis by auto
qed


lemma get-caps-wsc-e:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: a = (Get-Caps did)
  and   p3: s ∼ d ∼ t
  and   p4: interferes (the (domain-of-event a)) s d
  and   p5: s ∼ did ∼ t
  and   p6: s' = exec-event s a
  and   p7: t' = exec-event t a
shows   s' ∼ d ∼ t'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p2 domain-of-event-def by auto
  have a1: s' = fst (get-caps s did)
    using p2 p6 exec-event-def by auto
```

```
  have a2: t' = fst (get-caps t did)
    using p2 p7 exec-event-def by auto
  have a3: (interferes did s d)
    using p4 a0 by auto
  have a4: s' ∼ d ∼ t'
    using a1 a2 a3 p0 p1 p3 p5 get-caps-wsc by blast
}
then show ?thesis by auto
qed

lemma get-caps-dwsc-e: dynamic-weakly-step-consistent-e (Get-Caps did)
proof −
 {
  have ∀ d s t. reachable0 s ∧ reachable0 t
       ∧ (s ∼ d ∼ t)
       ∧ (interferes (the (domain-of-event (Get-Caps did))) s d)
       ∧ (s ∼ (the (domain-of-event (Get-Caps did))) ∼ t)
       ⟶ ((exec-event s (Get-Caps did)) ∼ d ∼ (exec-event t (Get-Caps did)))
    proof −
    {
     fix d s t
     assume p1: reachable0 s
     assume p2: reachable0 t
     assume p3: (s ∼ d ∼ t)
     assume p4: (interferes (the (domain-of-event (Get-Caps did))) s d)
     assume p5: (s ∼ (the (domain-of-event (Get-Caps did))) ∼ t)
     have ((exec-event s (Get-Caps did)) ∼ d ∼ (exec-event t (Get-Caps did)))
       by (metis Event.simps(54) domain-of-event-def option.sel p1 p2 p3 p4 p5 get-caps-wsc-e)
    }
    then show ?thesis by blast
    qed
  }
 then show ?thesis
   using dynamic-weakly-step-consistent-e-def by blast
qed
```

### 0.10.6 proving "grant endpoint cap" satisfying the "weakly step consistent" property

```
lemma grant-endpoint-cap-wsc-domain-cap-set:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
  and   p6: t' = fst (grant-endpoint-cap t did grant-cap dst-cap)
shows   get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
```

```
      ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
  by (meson p2 vpeq1-def)
have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
  by (meson p4 vpeq1-def)
have a6: (∀ ep. ep∈get-endpoints-of-domain s did
      ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
  by (meson p4 vpeq1-def)
have a7: get-domain-cap-set-from-domain-id s did = get-domain-cap-set-from-domain-id t did
  by (meson p4 vpeq1-def)
have get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
  proof (cases grant-cap∈get-domain-cap-set-from-domain-id s did
            ∧ GRANT∈rights grant-cap
            ∧ target grant-cap ≠ target dst-cap
            ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
  assume b0: grant-cap∈get-domain-cap-set-from-domain-id s did
           ∧ GRANT∈rights grant-cap
           ∧ target grant-cap ≠ target dst-cap
           ∧ dst-cap∈get-domain-cap-set-from-domain-id s did
  have b1: grant-cap∈get-domain-cap-set-from-domain-id t did
           ∧ GRANT∈rights grant-cap
           ∧ target grant-cap ≠ target dst-cap
           ∧ dst-cap∈get-domain-cap-set-from-domain-id t did
    using a7 b0 by blast
  let ?did-dst = target grant-cap
  let ?cs-dst-s = get-domain-cap-set-from-domain-id s ?did-dst
  let ?cs-dst-t = get-domain-cap-set-from-domain-id t ?did-dst
  have b2: ∀ v. v≠?did-dst
        ⟶ (caps s) v = (caps s') v
    using b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b3: ∀ v. v≠?did-dst
        ⟶ (caps t) v = (caps t') v
    using b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b4: (caps s') ?did-dst = insert dst-cap ?cs-dst-s
    using b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b5: (caps t') ?did-dst = insert dst-cap ?cs-dst-t
    using b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    proof (cases d = ?did-dst)
      assume c0: d = ?did-dst
      have c1: ?cs-dst-s = ?cs-dst-t
        using a3 c0 by auto
      have c2: (caps s') ?did-dst = (caps t') ?did-dst
        using b4 b5 c1 by auto
      have c3: get-domain-cap-set-from-domain-id s' d
             = get-domain-cap-set-from-domain-id t' d
        using c2 c0 get-domain-cap-set-from-domain-id-def by auto
      then show ?thesis by auto
    next
      assume c0: d ≠ ?did-dst
      have c1: (caps s) d = (caps t) d
        using a3 get-domain-cap-set-from-domain-id-def by auto
      have c2: (caps s) d = (caps s') d
        using c0 b2 by auto
      have c3: (caps t) d = (caps t') d
        using c0 b3 by auto
      have c4: (caps s') d = (caps t') d
        using c1 c2 c3 by auto
      have c5: get-domain-cap-set-from-domain-id s' d
```

```
                = get-domain-cap-set-from-domain-id t' d
        using c4 get-domain-cap-set-from-domain-id-def by auto
       then show ?thesis by auto
      qed
    then show ?thesis by auto
  next
    assume b0: ¬(grant-cap∈get-domain-cap-set-from-domain-id s did
              ∧ GRANT∈rights grant-cap
              ∧ target grant-cap ≠ target dst-cap
              ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
    have b1: ¬(grant-cap∈get-domain-cap-set-from-domain-id t did
              ∧ GRANT∈rights grant-cap
              ∧ target grant-cap ≠ target dst-cap
              ∧ dst-cap∈get-domain-cap-set-from-domain-id t did)
      using a7 b0 by auto
    have b2: s = s'
      using b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
    have b3: t = t'
      using b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
    have b4: get-domain-cap-set-from-domain-id s' d
                = get-domain-cap-set-from-domain-id s d
      using b2 get-domain-cap-set-from-domain-id-def by auto
    have b5: get-domain-cap-set-from-domain-id t' d
                = get-domain-cap-set-from-domain-id t d
      using b3 get-domain-cap-set-from-domain-id-def by auto
    have b6: get-domain-cap-set-from-domain-id s' d
                = get-domain-cap-set-from-domain-id t' d
      using b4 b5 a3 by auto
    then show ?thesis by auto
  qed
  }
then show ?thesis by auto
qed

lemma grant-endpoint-cap-wsc-policy:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
  and   p6: t' = fst (grant-endpoint-cap t did grant-cap dst-cap)
shows   (∀ v. interferes v s' d ⟷ interferes v t' d)
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p2 vpeq1-def)
  have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
    by (meson p4 vpeq1-def)
```

**have** *a6*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s did*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )
**by** (*meson p4 vpeq1-def*)
**have** *a7*: *get-domain-cap-set-from-domain-id s did* = *get-domain-cap-set-from-domain-id t did*
**by** (*meson p4 vpeq1-def*)
**have** (∀ *v*. *interferes v s′ d* ⟷ *interferes v t′ d*)
**proof** (*cases grant-cap*∈*get-domain-cap-set-from-domain-id s did*
∧ *GRANT*∈*rights grant-cap*
∧ *target grant-cap* ≠ *target dst-cap*
∧ *dst-cap*∈*get-domain-cap-set-from-domain-id s did*)
**assume** *b0*: *grant-cap*∈*get-domain-cap-set-from-domain-id s did*
∧ *GRANT*∈*rights grant-cap*
∧ *target grant-cap* ≠ *target dst-cap*
∧ *dst-cap*∈*get-domain-cap-set-from-domain-id s did*
**have** *b1*: *grant-cap*∈*get-domain-cap-set-from-domain-id t did*
∧ *GRANT*∈*rights grant-cap*
∧ *target grant-cap* ≠ *target dst-cap*
∧ *dst-cap*∈*get-domain-cap-set-from-domain-id t did*
**using** *a7 b0* **by** *blast*
**let** *?did-dst* = *target grant-cap*
**let** *?cs-dst-s* = *get-domain-cap-set-from-domain-id s ?did-dst*
**let** *?cs-dst-t* = *get-domain-cap-set-from-domain-id t ?did-dst*
**have** *b2*: ∀ *v*. *v*≠*?did-dst*
⟶ (*caps s*) *v* = (*caps s′*) *v*
**using** *b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b3*: ∀ *v*. *v*≠*?did-dst*
⟶ (*caps t*) *v* = (*caps t′*) *v*
**using** *b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b4*: (*caps s′*) *?did-dst* = *insert dst-cap ?cs-dst-s*
**using** *b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b5*: (*caps t′*) *?did-dst* = *insert dst-cap ?cs-dst-t*
**using** *b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**let** *?dst-cap-d* = *target dst-cap*
**have** *b6*: *dst-cap* ∈ *get-domain-cap-set-from-domain-id s′ ?did-dst*
**using** *b4 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b7*: *interferes ?did-dst s′ ?dst-cap-d*
**using** *b6 interferes-def* **by** *auto*
**have** *b8*: *dst-cap* ∈ *get-domain-cap-set-from-domain-id t′ ?did-dst*
**using** *b5 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b9*: *interferes ?did-dst t′ ?dst-cap-d*
**using** *b8 interferes-def* **by** *auto*
**have** *b10*: *?did-dst* ≠ *?dst-cap-d*
**using** *b0* **by** *auto*
**have** *b11*: ∀ *v*. *v*≠*?did-dst*
⟶ *get-domain-cap-set-from-domain-id s v*
= *get-domain-cap-set-from-domain-id s′ v*
**using** *b2 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b12*: ∀ *v*. *v*≠*?did-dst*
⟶ *interferes v s d* = *interferes v s′ d*
**using** *b11 interferes-def* **by** *auto*
**have** *b13*: ∀ *v*. *v*≠*?did-dst*
⟶ *get-domain-cap-set-from-domain-id t v*
= *get-domain-cap-set-from-domain-id t′ v*
**using** *b3 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b14*: ∀ *v*. *v*≠*?did-dst*
⟶ *interferes v t d* = *interferes v t′ d*
**using** *b13 interferes-def* **by** *auto*
**have** *b15*: ∀ *v*. *v*≠*?did-dst*

```
        ⟶ interferes v s' d = interferes v t' d
    using b12 b14 a1 by auto
  have (∀ v. interferes v s' d ⟷ interferes v t' d)
    proof (cases d = ?dst-cap-d)
      assume c0: d = ?dst-cap-d
      have c1: interferes ?did-dst s' ?dst-cap-d = interferes ?did-dst t' ?dst-cap-d
        using b7 b9 by auto
      have c2: interferes ?did-dst s' d = interferes ?did-dst t' d
        using c1 c0 by auto
      have c3: ∀ v. v=?did-dst
          ⟶ interferes v s' d = interferes v t' d
        using c2 by auto
      have c4: (∀ v. interferes v s' d ⟷ interferes v t' d)
        using c3 b15 by blast
      then show ?thesis by auto
    next
      assume c0: d ≠ ?dst-cap-d
      have c1: get-domain-cap-set-from-domain-id s' ?did-dst
            = insert dst-cap (get-domain-cap-set-from-domain-id s ?did-dst)
        using b4 get-domain-cap-set-from-domain-id-def by auto
      have c2: get-domain-cap-set-from-domain-id t' ?did-dst
            = insert dst-cap (get-domain-cap-set-from-domain-id t ?did-dst)
        using b5 get-domain-cap-set-from-domain-id-def by auto
      have c3: interferes ?did-dst s' d = interferes ?did-dst s d
        using c1 c0 interferes-def by auto
      have c4: interferes ?did-dst t' d = interferes ?did-dst t d
        using c2 c0 interferes-def by auto
      have c5: interferes ?did-dst s' d = interferes ?did-dst t' d
        using c3 c4 a1 by auto
      have c6: ∀ v. v=?did-dst
          ⟶ interferes v s' d = interferes v t' d
        using c5 by auto
      have c7: (∀ v. interferes v s' d ⟷ interferes v t' d)
        using c6 b15 by blast
      then show ?thesis by auto
    qed
  then show ?thesis by auto
next
  assume b0: ¬(grant-cap∈get-domain-cap-set-from-domain-id s did
            ∧ GRANT∈rights grant-cap
            ∧ target grant-cap ≠ target dst-cap
            ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
  have b1: ¬(grant-cap∈get-domain-cap-set-from-domain-id t did
            ∧ GRANT∈rights grant-cap
            ∧ target grant-cap ≠ target dst-cap
            ∧ dst-cap∈get-domain-cap-set-from-domain-id t did)
    using a7 b0 by auto
  have b2: s = s'
    using b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b3: t = t'
    using b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b4: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using a1 b2 b3 by auto
  then show ?thesis by auto
qed
}
then show ?thesis by auto
qed
```

**lemma** *grant-endpoint-cap-wsc-dom-eps*:
**assumes** *p0*: *reachable0 s*
  **and** *p1*: *reachable0 t*
  **and** *p2*: $s \sim d \sim t$
  **and** *p3*: *interferes did s d*
  **and** *p4*: $s \sim did \sim t$
  **and** *p5*: $s' = fst\ (grant\text{-}endpoint\text{-}cap\ s\ did\ grant\text{-}cap\ dst\text{-}cap)$
  **and** *p6*: $t' = fst\ (grant\text{-}endpoint\text{-}cap\ t\ did\ grant\text{-}cap\ dst\text{-}cap)$
**shows** *get-endpoints-of-domain s' d = get-endpoints-of-domain t' d*
**proof** −
**{**
  **have** *a0*: *get-endpoints-of-domain s d = get-endpoints-of-domain t d*
    **by** (*meson p2 vpeq1-def*)
  **have** *a1*: $(\forall v.\ interferes\ v\ s\ d \longleftrightarrow interferes\ v\ t\ d)$
    **by** (*meson p2 vpeq1-def*)
  **have** *a2*: *interferes did t d*
    **using** *p3 a1* **by** *auto*
  **have** *a3*: *get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d*
    **by** (*meson p2 vpeq1-def*)
  **have** *a4*: $(\forall ep.\ ep \in get\text{-}endpoints\text{-}of\text{-}domain\ s\ d$
        $\longrightarrow get\text{-}msg\text{-}set\text{-}from\text{-}endpoint\text{-}id\ s\ ep = get\text{-}msg\text{-}set\text{-}from\text{-}endpoint\text{-}id\ t\ ep\ )$
    **by** (*meson p2 vpeq1-def*)
  **have** *a5*: *get-endpoints-of-domain s did = get-endpoints-of-domain t did*
    **by** (*meson p4 vpeq1-def*)
  **have** *a6*: $(\forall ep.\ ep \in get\text{-}endpoints\text{-}of\text{-}domain\ s\ did$
        $\longrightarrow get\text{-}msg\text{-}set\text{-}from\text{-}endpoint\text{-}id\ s\ ep = get\text{-}msg\text{-}set\text{-}from\text{-}endpoint\text{-}id\ t\ ep\ )$
    **by** (*meson p4 vpeq1-def*)
  **have** *a7*: *get-domain-cap-set-from-domain-id s did = get-domain-cap-set-from-domain-id t did*
    **by** (*meson p4 vpeq1-def*)
  **have** *get-endpoints-of-domain s' d = get-endpoints-of-domain t' d*
    **proof** (*cases grant-cap*$\in$*get-domain-cap-set-from-domain-id s did*
          $\wedge$ *GRANT*$\in$*rights grant-cap*
          $\wedge$ *target grant-cap* $\neq$ *target dst-cap*
          $\wedge$ *dst-cap*$\in$*get-domain-cap-set-from-domain-id s did*)
    **assume** *b0*: *grant-cap*$\in$*get-domain-cap-set-from-domain-id s did*
          $\wedge$ *GRANT*$\in$*rights grant-cap*
          $\wedge$ *target grant-cap* $\neq$ *target dst-cap*
          $\wedge$ *dst-cap*$\in$*get-domain-cap-set-from-domain-id s did*
    **have** *b1*: *grant-cap*$\in$*get-domain-cap-set-from-domain-id t did*
          $\wedge$ *GRANT*$\in$*rights grant-cap*
          $\wedge$ *target grant-cap* $\neq$ *target dst-cap*
          $\wedge$ *dst-cap*$\in$*get-domain-cap-set-from-domain-id t did*
      **using** *a7 b0* **by** *blast*
    **have** *b2*: *get-endpoints-of-domain s' d = get-endpoints-of-domain s d*
      **using** *b0 p5 grant-endpoint-cap-def get-endpoints-of-domain-def* **by** *auto*
    **have** *b3*: *get-endpoints-of-domain t' d = get-endpoints-of-domain t d*
      **using** *b1 p6 grant-endpoint-cap-def get-endpoints-of-domain-def* **by** *auto*
    **have** *b4*: *get-endpoints-of-domain s' d = get-endpoints-of-domain t' d*
      **using** *b2 b3 a0* **by** *auto*
    **then show** *?thesis* **by** *auto*
  **next**
    **assume** *b0*: ¬(*grant-cap*$\in$*get-domain-cap-set-from-domain-id s did*
          $\wedge$ *GRANT*$\in$*rights grant-cap*
          $\wedge$ *target grant-cap* $\neq$ *target dst-cap*
          $\wedge$ *dst-cap*$\in$*get-domain-cap-set-from-domain-id s did*)
    **have** *b1*: ¬(*grant-cap*$\in$*get-domain-cap-set-from-domain-id t did*
          $\wedge$ *GRANT*$\in$*rights grant-cap*

```
            ∧ target grant-cap ≠ target dst-cap
            ∧ dst-cap∈get-domain-cap-set-from-domain-id t did)
    using a7 b0 by auto
  have b2: s = s'
    using b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b3: t = t'
    using b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def by auto
  have b4: get-endpoints-of-domain s' d
            = get-endpoints-of-domain s d
    using b2 get-domain-cap-set-from-domain-id-def by auto
  have b5: get-endpoints-of-domain t' d
            = get-endpoints-of-domain t d
    using b3 get-domain-cap-set-from-domain-id-def by auto
  have b6: get-endpoints-of-domain s' d
            = get-endpoints-of-domain t' d
    using b4 b5 a0 by auto
  then show ?thesis by auto
 qed
 }
then show ?thesis by auto
qed

lemma grant-endpoint-cap-wsc-ep-msgs:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
  and   p6: t' = fst (grant-endpoint-cap t did grant-cap dst-cap)
shows  (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
proof −
{
  have a0: get-endpoints-of-domain s d = get-endpoints-of-domain t d
    by (meson p2 vpeq1-def)
  have a1: (∀ v. interferes v s d ⟷ interferes v t d)
    by (meson p2 vpeq1-def)
  have a2: interferes did t d
    using p3 a1 by auto
  have a3: get-domain-cap-set-from-domain-id s d = get-domain-cap-set-from-domain-id t d
    by (meson p2 vpeq1-def)
  have a4: (∀ ep. ep∈get-endpoints-of-domain s d
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p2 vpeq1-def)
  have a5: get-endpoints-of-domain s did = get-endpoints-of-domain t did
    by (meson p4 vpeq1-def)
  have a6: (∀ ep. ep∈get-endpoints-of-domain s did
        ⟶ get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep )
    by (meson p4 vpeq1-def)
  have a7: get-domain-cap-set-from-domain-id s did = get-domain-cap-set-from-domain-id t did
    by (meson p4 vpeq1-def)
  have (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    proof (cases grant-cap∈get-domain-cap-set-from-domain-id s did
            ∧ GRANT∈rights grant-cap
            ∧ target grant-cap ≠ target dst-cap
            ∧ dst-cap∈get-domain-cap-set-from-domain-id s did)
```

**assume** *b0*: *grant-cap∈get-domain-cap-set-from-domain-id s did*
∧ *GRANT∈rights grant-cap*
∧ *target grant-cap ≠ target dst-cap*
∧ *dst-cap∈get-domain-cap-set-from-domain-id s did*
**have** *b1*: *grant-cap∈get-domain-cap-set-from-domain-id t did*
∧ *GRANT∈rights grant-cap*
∧ *target grant-cap ≠ target dst-cap*
∧ *dst-cap∈get-domain-cap-set-from-domain-id t did*
**using** *a7 b0* **by** *blast*
**have** *b2*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id s ep* )
**using** *b0 p5 grant-endpoint-cap-def get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *b3*: (∀ *ep*. *ep∈get-endpoints-of-domain t d*
⟶ *get-msg-set-from-endpoint-id t′ ep = get-msg-set-from-endpoint-id t ep* )
**using** *b1 p6 grant-endpoint-cap-def get-msg-set-from-endpoint-id-def* **by** *auto*
**have** *b4*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id t′ ep = get-msg-set-from-endpoint-id t ep* )
**using** *b3 a0* **by** *auto*
**have** *b5*: *get-endpoints-of-domain s d = get-endpoints-of-domain s′ d*
**using** *p0 p5 grant-endpoint-cap-notchg-dom-eps* **by** *auto*
**have** *b6*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id t′ ep* )
**using** *b2 b4 a4* **by** *auto*
**have** *b7*: (∀ *ep*. *ep∈get-endpoints-of-domain s′ d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id t′ ep* )
**using** *b5 b6* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *b0*: ¬(*grant-cap∈get-domain-cap-set-from-domain-id s did*
∧ *GRANT∈rights grant-cap*
∧ *target grant-cap ≠ target dst-cap*
∧ *dst-cap∈get-domain-cap-set-from-domain-id s did*)
**have** *b1*: ¬(*grant-cap∈get-domain-cap-set-from-domain-id t did*
∧ *GRANT∈rights grant-cap*
∧ *target grant-cap ≠ target dst-cap*
∧ *dst-cap∈get-domain-cap-set-from-domain-id t did*)
**using** *a7 b0* **by** *auto*
**have** *b2*: *s = s′*
**using** *b0 p5 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b3*: *t = t′*
**using** *b1 p6 grant-endpoint-cap-def get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b4*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id s ep* )
**using** *b2 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b5*: (∀ *ep*. *ep∈get-endpoints-of-domain t d*
⟶ *get-msg-set-from-endpoint-id t′ ep = get-msg-set-from-endpoint-id t ep* )
**using** *b3 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *b6*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id t′ ep = get-msg-set-from-endpoint-id t ep* )
**using** *b5 a0* **by** *auto*
**have** *b7*: *get-endpoints-of-domain s d = get-endpoints-of-domain s′ d*
**using** *p0 p5 grant-endpoint-cap-notchg-dom-eps* **by** *auto*
**have** *b8*: (∀ *ep*. *ep∈get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id t′ ep* )
**using** *b4 b6 a4* **by** *auto*
**have** *b9*: (∀ *ep*. *ep∈get-endpoints-of-domain s′ d*
⟶ *get-msg-set-from-endpoint-id s′ ep = get-msg-set-from-endpoint-id t′ ep* )
**using** *b7 b8* **by** *auto*

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

lemma grant-endpoint-cap-wsc:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
  and   p6: t' = fst (grant-endpoint-cap t did grant-cap dst-cap)
shows   s' ∼ d ∼ t'
proof −
{
  have a0: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using p0 p1 p2 p3 p4 p5 p6 grant-endpoint-cap-wsc-domain-cap-set by blast
  have a1: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using p0 p1 p2 p3 p4 p5 p6 grant-endpoint-cap-wsc-policy by blast
  have a2: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using p0 p1 p2 p3 p4 p5 p6 grant-endpoint-cap-wsc-dom-eps by blast
  have a3: (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    using p0 p1 p2 p3 p4 p5 p6 grant-endpoint-cap-wsc-ep-msgs by blast
  have a4: s' ∼ d ∼ t'
    using a0 a1 a2 a3 vpeq1-def by auto
}
then show ?thesis by auto
qed

lemma grant-endpoint-cap-wsc-e:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: a = (Grant-Endpoint-Cap did grant-cap dst-cap)
  and   p3: s ∼ d ∼ t
  and   p4: interferes (the (domain-of-event a)) s d
  and   p5: s ∼ did ∼ t
  and   p6: s' = exec-event s a
  and   p7: t' = exec-event t a
shows   s' ∼ d ∼ t'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p2 domain-of-event-def by auto
  have a1: s' = fst (grant-endpoint-cap s did grant-cap dst-cap)
    using p2 p6 exec-event-def by auto
  have a2: t' = fst (grant-endpoint-cap t did grant-cap dst-cap)
    using p2 p7 exec-event-def by auto
  have a3: (interferes did s d)
    using p4 a0 by auto
  have a4: s' ∼ d ∼ t'
    using a1 a2 a3 p0 p1 p3 p5 grant-endpoint-cap-wsc by blast
}
then show ?thesis by auto
qed
```

**lemma** *grant-endpoint-cap-dwsc-e*: *dynamic-weakly-step-consistent-e* (*Grant-Endpoint-Cap did grant-cap dst-cap*)
**proof** −
**{**
**have** ∀ *d s t*. *reachable0 s* ∧ *reachable0 t*
∧ (*s* ∼ *d* ∼ *t*)
∧ (*interferes* (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) *s d*)
∧ (*s* ∼ (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) ∼ *t*)
⟶ ((*exec-event s* (*Grant-Endpoint-Cap did grant-cap dst-cap*)) ∼ *d* ∼ (*exec-event t* (*Grant-Endpoint-Cap did grant-cap dst-cap*)))
**proof** −
**{**
**fix** *d s t*
**assume** *p1*: *reachable0 s*
**assume** *p2*: *reachable0 t*
**assume** *p3*: (*s* ∼ *d* ∼ *t*)
**assume** *p4*: (*interferes* (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) *s d*)
**assume** *p5*: (*s* ∼ (*the* (*domain-of-event* (*Grant-Endpoint-Cap did grant-cap dst-cap*))) ∼ *t*)
**have** ((*exec-event s* (*Grant-Endpoint-Cap did grant-cap dst-cap*)) ∼ *d* ∼ (*exec-event t* (*Grant-Endpoint-Cap did grant-cap dst-cap*)))
**by** (*metis Event.simps*(55) *domain-of-event-def option.sel p1 p2 p3 p4 p5 grant-endpoint-cap-wsc-e*)
**}**
**then show** *?thesis* **by** *blast*
**qed**
**}**
**then show** *?thesis*
**using** *dynamic-weakly-step-consistent-e-def* **by** *blast*
**qed**

### 0.10.7 proving "remove cap right" satisfying the "weakly step consistent" property

**lemma** *remove-cap-right-wsc-domain-cap-set*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: *s* ∼ *d* ∼ *t*
**and** *p3*: *interferes did s d*
**and** *p4*: *s* ∼ *did* ∼ *t*
**and** *p5*: *s'* = *fst* (*remove-cap-right s did rm-cap right-to-rm*)
**and** *p6*: *t'* = *fst* (*remove-cap-right t did rm-cap right-to-rm*)
**shows** *get-domain-cap-set-from-domain-id s' d*
= *get-domain-cap-set-from-domain-id t' d*
**proof** −
**{**
**have** *a0*: *get-endpoints-of-domain s d* = *get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a1*: (∀ *v*. *interferes v s d* ⟷ *interferes v t d*)
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a3*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id t d*
**by** (*meson p2 vpeq1-def*)
**have** *a4*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )
**by** (*meson p2 vpeq1-def*)
**have** *a5*: *get-endpoints-of-domain s did* = *get-endpoints-of-domain t did*
**by** (*meson p4 vpeq1-def*)
**have** *a6*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s did*
⟶ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )

**by** (*meson p4 vpeq1-def*)
**have** *a7*: *get-domain-cap-set-from-domain-id s did*
= *get-domain-cap-set-from-domain-id t did*
**by** (*meson p4 vpeq1-def*)
**have** *get-domain-cap-set-from-domain-id s' d*
= *get-domain-cap-set-from-domain-id t' d*
**proof** (*cases rm-cap* ∈ *get-domain-cap-set-from-domain-id s did*
∧ *REMOVE* ∈ *rights rm-cap*
∧ *right-to-rm* ∈ *rights rm-cap*)
**assume** *b0*: *rm-cap* ∈ *get-domain-cap-set-from-domain-id s did*
∧ *REMOVE* ∈ *rights rm-cap*
∧ *right-to-rm* ∈ *rights rm-cap*
**have** *b1*: *rm-cap* ∈ *get-domain-cap-set-from-domain-id t did*
∧ *REMOVE* ∈ *rights rm-cap*
∧ *right-to-rm* ∈ *rights rm-cap*
**using** *a7 b0* **by** *auto*
**have** *get-domain-cap-set-from-domain-id s' d*
= *get-domain-cap-set-from-domain-id t' d*
**proof** (*cases* *REMOVE* = *right-to-rm*
∧ {*REMOVE*} = *rights rm-cap*)
**assume** *c0*: *REMOVE* = *right-to-rm*
∧ {*REMOVE*} = *rights rm-cap*
**let** *?cs-dst-s* = *get-domain-cap-set-from-domain-id s did*
**let** *?cs-rest-s* = {*c*. *c*∈*?cs-dst-s* ∧ *c*≠*rm-cap*}
**let** *?cs-dst-t* = *get-domain-cap-set-from-domain-id t did*
**let** *?cs-rest-t* = {*c*. *c*∈*?cs-dst-t* ∧ *c*≠*rm-cap*}
**have** *c1*: ((*caps s'*) *did*) = *?cs-rest-s*
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c2*: ((*caps t'*) *did*) = *?cs-rest-t*
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c3*: *?cs-rest-s* = *?cs-rest-t*
**using** *a7* **by** *auto*
**have** *c4*: ((*caps s'*) *did*) = ((*caps t'*) *did*)
**using** *c1 c2 c3* **by** *auto*
**have** *c5*: ∀ *v*. *v*≠*did*
⟶ ((*caps s'*) *v*) = ((*caps s*) *v*)
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c6*: ∀ *v*. *v*≠*did*
⟶ ((*caps t'*) *v*) = ((*caps t*) *v*)
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c7*: ∀ *v*. *v*≠*did* ∧ *v*=*d*
⟶ ((*caps s'*) *v*) = ((*caps t'*) *v*)
**using** *c5 c6 a3 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *c8*: *d*≠*did*
⟶ ((*caps s'*) *d*) = ((*caps t'*) *d*)
**using** *c7* **by** *auto*
**have** *c9*: ((*caps s'*) *d*) = ((*caps t'*) *d*)
**using** *c4 c8* **by** *auto*
**have** *c10*: *get-domain-cap-set-from-domain-id s' d*
= *get-domain-cap-set-from-domain-id t' d*
**using** *c9 get-domain-cap-set-from-domain-id-def* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *c0*: ¬(*REMOVE* = *right-to-rm*
∧ {*REMOVE*} = *rights rm-cap*)
**let** *?cs-dst-s* = *get-domain-cap-set-from-domain-id s did*
**let** *?cs-rest-s* = {*c*. *c*∈*?cs-dst-s* ∧ *c*≠*rm-cap*}
**let** *?cs-dst-t* = *get-domain-cap-set-from-domain-id t did*

```
      let ?cs-rest-t = {c. c∈?cs-dst-t ∧ c≠rm-cap}
      let ?new-cap = (| target = target rm-cap,
                        rights = (rights rm-cap) − {right-to-rm}|)
      have c1: ((caps s') did) = (insert ?new-cap ?cs-rest-s)
        using b0 c0 p5 remove-cap-right-def by auto
      have c2: ((caps t') did) = (insert ?new-cap ?cs-rest-t)
        using b1 c0 p6 remove-cap-right-def by auto
      have c3: ?cs-rest-s = ?cs-rest-t
        using a7 by auto
      have c4: ((caps s') did) = ((caps t') did)
        using c1 c2 c3 by auto
      have c5: ∀ v. v≠did
             ⟶ ((caps s') v) = ((caps s) v)
        using b0 c0 p5 remove-cap-right-def by auto
      have c6: ∀ v. v≠did
             ⟶ ((caps t') v) = ((caps t) v)
        using b1 c0 p6 remove-cap-right-def by auto
      have c7: ∀ v. v≠did ∧ v=d
             ⟶ ((caps s') v) = ((caps t') v)
        using c5 c6 a3 get-domain-cap-set-from-domain-id-def by auto
      have c8: d≠did
             ⟶ ((caps s') d) = ((caps t') d)
        using c7 by auto
      have c9: ((caps s') d) = ((caps t') d)
        using c4 c8 by auto
      have c10: get-domain-cap-set-from-domain-id s' d
             = get-domain-cap-set-from-domain-id t' d
        using c9 get-domain-cap-set-from-domain-id-def by auto
      then show ?thesis by auto
    qed
  then show ?thesis by auto
 next
  assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
             ∧ REMOVE ∈ rights rm-cap
             ∧ right-to-rm ∈ rights rm-cap)
  have b1: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id t did
             ∧ REMOVE ∈ rights rm-cap
             ∧ right-to-rm ∈ rights rm-cap)
    using b0 a7 by auto
  have b2: s' = s
    using b0 p5 remove-cap-right-def by auto
  have b3: get-domain-cap-set-from-domain-id s d
         = get-domain-cap-set-from-domain-id s' d
    using b2 get-domain-cap-set-from-domain-id-def by auto
  have b4: t' = t
    using b1 p6 remove-cap-right-def by auto
  have b5: get-domain-cap-set-from-domain-id t d
         = get-domain-cap-set-from-domain-id t' d
    using b4 get-domain-cap-set-from-domain-id-def by auto
  have b6: get-domain-cap-set-from-domain-id s' d
         = get-domain-cap-set-from-domain-id t' d
    using a3 b3 b5 by auto
  then show ?thesis by auto
 qed
 }
then show ?thesis by auto
qed
```

**lemma** *remove-cap-right-wsc-policy*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: $s \sim d \sim t$
**and** *p3*: *interferes did s d*
**and** *p4*: $s \sim did \sim t$
**and** *p5*: $s' = fst\ (remove\text{-}cap\text{-}right\ s\ did\ rm\text{-}cap\ right\text{-}to\text{-}rm)$
**and** *p6*: $t' = fst\ (remove\text{-}cap\text{-}right\ t\ did\ rm\text{-}cap\ right\text{-}to\text{-}rm)$
**shows** $(\forall v.\ interferes\ v\ s'\ d \longleftrightarrow interferes\ v\ t'\ d)$
**proof** −
**{**
**have** *a0*: *get-endpoints-of-domain s d* = *get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a1*: $(\forall v.\ interferes\ v\ s\ d \longleftrightarrow interferes\ v\ t\ d)$
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a3*: *get-domain-cap-set-from-domain-id s d*
= *get-domain-cap-set-from-domain-id t d*
**by** (*meson p2 vpeq1-def*)
**have** *a4*: $(\forall ep.\ ep \in$ *get-endpoints-of-domain s d*
$\longrightarrow$ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )
**by** (*meson p2 vpeq1-def*)
**have** *a5*: *get-endpoints-of-domain s did* = *get-endpoints-of-domain t did*
**by** (*meson p4 vpeq1-def*)
**have** *a6*: $(\forall ep.\ ep \in$ *get-endpoints-of-domain s did*
$\longrightarrow$ *get-msg-set-from-endpoint-id s ep* = *get-msg-set-from-endpoint-id t ep* )
**by** (*meson p4 vpeq1-def*)
**have** *a7*: *get-domain-cap-set-from-domain-id s did*
= *get-domain-cap-set-from-domain-id t did*
**by** (*meson p4 vpeq1-def*)
**have** $(\forall v.\ interferes\ v\ s'\ d \longleftrightarrow interferes\ v\ t'\ d)$
**proof** (*cases rm-cap* $\in$ *get-domain-cap-set-from-domain-id s did*
$\wedge$ *REMOVE* $\in$ *rights rm-cap*
$\wedge$ *right-to-rm* $\in$ *rights rm-cap*)
**assume** *b0*: *rm-cap* $\in$ *get-domain-cap-set-from-domain-id s did*
$\wedge$ *REMOVE* $\in$ *rights rm-cap*
$\wedge$ *right-to-rm* $\in$ *rights rm-cap*
**have** *b1*: *rm-cap* $\in$ *get-domain-cap-set-from-domain-id t did*
$\wedge$ *REMOVE* $\in$ *rights rm-cap*
$\wedge$ *right-to-rm* $\in$ *rights rm-cap*
**using** *a7 b0* **by** *auto*
**have** $(\forall v.\ interferes\ v\ s'\ d \longleftrightarrow interferes\ v\ t'\ d)$
**proof** (*cases REMOVE* = *right-to-rm*
$\wedge$ $\{REMOVE\}$ = *rights rm-cap*)
**assume** *c0*: *REMOVE* = *right-to-rm*
$\wedge$ $\{REMOVE\}$ = *rights rm-cap*
**let** *?cs-dst-s* = *get-domain-cap-set-from-domain-id s did*
**let** *?cs-rest-s* = $\{c.\ c \in$ *?cs-dst-s* $\wedge\ c \neq$ *rm-cap*$\}$
**let** *?cs-dst-t* = *get-domain-cap-set-from-domain-id t did*
**let** *?cs-rest-t* = $\{c.\ c \in$ *?cs-dst-t* $\wedge\ c \neq$ *rm-cap*$\}$
**have** *c1*: $((caps\ s')\ did)$ = *?cs-rest-s*
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c2*: $((caps\ t')\ did)$ = *?cs-rest-t*
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c3*: *?cs-rest-s* = *?cs-rest-t*
**using** *a7* **by** *auto*
**have** *c4*: $((caps\ s')\ did) = ((caps\ t')\ did)$

**using** *c1 c2 c3* **by** *auto*
**have** *c5*: $\forall\, v.\ v{\neq}did$
$\longrightarrow ((caps\ s')\ v) = ((caps\ s)\ v)$
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c6*: $\forall\, v.\ v{\neq}did$
$\longrightarrow ((caps\ t')\ v) = ((caps\ t)\ v)$
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c7*: $\forall\, v.\ v{\neq}did \wedge v{=}d$
$\longrightarrow ((caps\ s')\ v) = ((caps\ t')\ v)$
**using** *c5 c6 a3 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *c8*: $d{\neq}did$
$\longrightarrow ((caps\ s')\ d) = ((caps\ t')\ d)$
**using** *c7* **by** *auto*
**have** *c9*: $((caps\ s')\ d) = ((caps\ t')\ d)$
**using** *c4 c8* **by** *auto*
**have** *c10*: *get-domain-cap-set-from-domain-id s' d*
$=$ *get-domain-cap-set-from-domain-id t' d*
**using** *c9 get-domain-cap-set-from-domain-id-def* **by** *auto*
**have** *c11*: $\forall\, v.\ v{\neq}did$
$\longrightarrow$ *interferes v s d* $\longleftrightarrow$ *interferes v s' d*
**using** *c5 get-domain-cap-set-from-domain-id-def interferes-def* **by** *auto*
**have** *c12*: $\forall\, v.\ v{\neq}did$
$\longrightarrow$ *interferes v t d* $\longleftrightarrow$ *interferes v t' d*
**using** *c6 get-domain-cap-set-from-domain-id-def interferes-def* **by** *auto*
**have** *c13*: $\forall\, v.\ v{\neq}did$
$\longrightarrow$ *interferes v s' d* $\longleftrightarrow$ *interferes v t' d*
**using** *c11 c12 a1* **by** *auto*
**have** *c14*: *interferes did s' d* $\longleftrightarrow$ *interferes did t' d*
**using** *c4 get-domain-cap-set-from-domain-id-def interferes-def* **by** *auto*
**have** *c15*: $(\forall\, v.$ *interferes v s' d* $\longleftrightarrow$ *interferes v t' d*$)$
**using** *c13 c14* **by** *auto*
**then show** *?thesis* **by** *auto*
**next**
**assume** *c0*: $\neg(REMOVE = right\text{-}to\text{-}rm$
$\wedge\ \{REMOVE\} = rights\ rm\text{-}cap)$
**let** *?cs-dst-s* $=$ *get-domain-cap-set-from-domain-id s did*
**let** *?cs-rest-s* $= \{c.\ c{\in}?cs\text{-}dst\text{-}s \wedge c{\neq}rm\text{-}cap\}$
**let** *?cs-dst-t* $=$ *get-domain-cap-set-from-domain-id t did*
**let** *?cs-rest-t* $= \{c.\ c{\in}?cs\text{-}dst\text{-}t \wedge c{\neq}rm\text{-}cap\}$
**let** *?new-cap* $= (\!|\ target = target\ rm\text{-}cap,$
$rights = (rights\ rm\text{-}cap) - \{right\text{-}to\text{-}rm\}\ |\!)$
**have** *c1*: $((caps\ s')\ did) = (insert\ ?new\text{-}cap\ ?cs\text{-}rest\text{-}s)$
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c2*: $((caps\ t')\ did) = (insert\ ?new\text{-}cap\ ?cs\text{-}rest\text{-}t)$
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c3*: *?cs-rest-s* $=$ *?cs-rest-t*
**using** *a7* **by** *auto*
**have** *c4*: $((caps\ s')\ did) = ((caps\ t')\ did)$
**using** *c1 c2 c3* **by** *auto*
**have** *c5*: $\forall\, v.\ v{\neq}did$
$\longrightarrow ((caps\ s')\ v) = ((caps\ s)\ v)$
**using** *b0 c0 p5 remove-cap-right-def* **by** *auto*
**have** *c6*: $\forall\, v.\ v{\neq}did$
$\longrightarrow ((caps\ t')\ v) = ((caps\ t)\ v)$
**using** *b1 c0 p6 remove-cap-right-def* **by** *auto*
**have** *c7*: $\forall\, v.\ v{\neq}did \wedge v{=}d$
$\longrightarrow ((caps\ s')\ v) = ((caps\ t')\ v)$
**using** *c5 c6 a3 get-domain-cap-set-from-domain-id-def* **by** *auto*

```
      have c8: d≠did
            ⟶ ((caps s') d) = ((caps t') d)
        using c7 by auto
      have c9: ((caps s') d) = ((caps t') d)
        using c4 c8 by auto
      have c10: get-domain-cap-set-from-domain-id s' d
            = get-domain-cap-set-from-domain-id t' d
        using c9 get-domain-cap-set-from-domain-id-def by auto
      have c11: ∀ v. v≠did
            ⟶ interferes v s d ⟷ interferes v s' d
        using c5 get-domain-cap-set-from-domain-id-def interferes-def by auto
      have c12: ∀ v. v≠did
            ⟶ interferes v t d ⟷ interferes v t' d
        using c6 get-domain-cap-set-from-domain-id-def interferes-def by auto
      have c13: ∀ v. v≠did
            ⟶ interferes v s' d ⟷ interferes v t' d
        using c11 c12 a1 by auto
      have c14: interferes did s' d ⟷ interferes did t' d
        using c4 c10 get-domain-cap-set-from-domain-id-def interferes-def by auto
      have c15: (∀ v. interferes v s' d ⟷ interferes v t' d)
        using c13 c14 by auto
      then show ?thesis by auto
    qed
  then show ?thesis by auto
 next
  assume b0: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id s did
            ∧ REMOVE ∈ rights rm-cap
            ∧ right-to-rm ∈ rights rm-cap)
  have b1: ¬(rm-cap ∈  get-domain-cap-set-from-domain-id t did
            ∧ REMOVE ∈ rights rm-cap
            ∧ right-to-rm ∈ rights rm-cap)
    using b0 a7 by auto
  have b2: s' = s
    using b0 p5 remove-cap-right-def by auto
  have b3: (∀ v. interferes v s d ⟷ interferes v s' d)
    using b2 get-domain-cap-set-from-domain-id-def by auto
  have b4: t' = t
    using b1 p6 remove-cap-right-def by auto
  have b5: (∀ v. interferes v t d ⟷ interferes v t' d)
    using b4 get-domain-cap-set-from-domain-id-def by auto
  have b6: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using a1 b3 b5 by auto
  then show ?thesis by auto
 qed
 }
then show ?thesis by auto
qed

lemma remove-cap-right-wsc-dom-eps:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (remove-cap-right s did rm-cap right-to-rm)
  and   p6: t' = fst (remove-cap-right t did rm-cap right-to-rm)
shows   get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
proof −
```

**{**
**have** *a0*: *get-endpoints-of-domain s d = get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a1*: (∀ *v*. *interferes v s d* ⟷ *interferes v t d*)
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a4*: *get-endpoints-of-domain s d = get-endpoints-of-domain s' d*
**using** *p0 p5 remove-cap-right-notchg-dom-eps* **by** *auto*
**have** *a5*: *get-endpoints-of-domain t d = get-endpoints-of-domain t' d*
**using** *p1 p6 remove-cap-right-notchg-dom-eps* **by** *auto*
**have** *a6*: *get-endpoints-of-domain s' d = get-endpoints-of-domain t' d*
**using** *a0 a4 a5* **by** *auto*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *remove-cap-right-wsc-ep-msgs*:
**assumes** *p0*: *reachable0 s*
**and** *p1*: *reachable0 t*
**and** *p2*: *s ∼ d ∼ t*
**and** *p3*: *interferes did s d*
**and** *p4*: *s ∼ did ∼ t*
**and** *p5*: *s' = fst* (*remove-cap-right s did rm-cap right-to-rm*)
**and** *p6*: *t' = fst* (*remove-cap-right t did rm-cap right-to-rm*)
**shows** (∀ *ep*. *ep*∈*get-endpoints-of-domain s' d*
⟶ *get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep* )
**proof** −
**{**
**have** *a0*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id t ep* )
**by** (*meson p2 vpeq1-def*)
**have** *a1*: (∀ *v*. *interferes v s d* ⟷ *interferes v t d*)
**by** (*meson p2 vpeq1-def*)
**have** *a2*: *interferes did t d*
**using** *p3 a1* **by** *auto*
**have** *a4*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s ep = get-msg-set-from-endpoint-id s' ep* )
**using** *p0 p5 remove-cap-right-notchg-ep-msgs* **by** *auto*
**have** *a5*: (∀ *ep*. *ep*∈*get-endpoints-of-domain t d*
⟶ *get-msg-set-from-endpoint-id t ep = get-msg-set-from-endpoint-id t' ep* )
**using** *p1 p6 remove-cap-right-notchg-ep-msgs* **by** *auto*
**have** *a6*: *get-endpoints-of-domain s d = get-endpoints-of-domain t d*
**by** (*meson p2 vpeq1-def*)
**have** *a7*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s d*
⟶ *get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep* )
**using** *a0 a4 a5 a6* **by** *auto*
**have** *a8*: *get-endpoints-of-domain s d = get-endpoints-of-domain s' d*
**using** *p0 p5 remove-cap-right-notchg-dom-eps* **by** *auto*
**have** *a9*: (∀ *ep*. *ep*∈*get-endpoints-of-domain s' d*
⟶ *get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep* )
**using** *a7 a8* **by** *auto*
**}**
**then show** *?thesis* **by** *auto*
**qed**

**lemma** *remove-cap-right-wsc*:
**assumes** *p0*: *reachable0 s*

```
  and   p1: reachable0 t
  and   p2: s ∼ d ∼ t
  and   p3: interferes did s d
  and   p4: s ∼ did ∼ t
  and   p5: s' = fst (remove-cap-right s did rm-cap right-to-rm)
  and   p6: t' = fst (remove-cap-right t did rm-cap right-to-rm)
shows   s' ∼ d ∼ t'
proof −
{
  have a0: get-domain-cap-set-from-domain-id s' d = get-domain-cap-set-from-domain-id t' d
    using p0 p1 p2 p3 p4 p5 p6 remove-cap-right-wsc-domain-cap-set by blast
  have a1: (∀ v. interferes v s' d ⟷ interferes v t' d)
    using p0 p1 p2 p3 p4 p5 p6 remove-cap-right-wsc-policy by blast
  have a2: get-endpoints-of-domain s' d = get-endpoints-of-domain t' d
    using p0 p1 p2 p3 p4 p5 p6 remove-cap-right-wsc-dom-eps by blast
  have a3: (∀ ep. ep∈get-endpoints-of-domain s' d
        ⟶ get-msg-set-from-endpoint-id s' ep = get-msg-set-from-endpoint-id t' ep )
    using p0 p1 p2 p3 p4 p5 p6 remove-cap-right-wsc-ep-msgs by blast
  have a4: s' ∼ d ∼ t'
    using a0 a1 a2 a3 vpeq1-def by auto
}
then show ?thesis by auto
qed

lemma remove-cap-right-wsc-e:
assumes p0: reachable0 s
  and   p1: reachable0 t
  and   p2: a = (Remove-Cap-Right did dst-cap right-to-rm)
  and   p3: s ∼ d ∼ t
  and   p4: interferes (the (domain-of-event a)) s d
  and   p5: s ∼ did ∼ t
  and   p6: s' = exec-event s a
  and   p7: t' = exec-event t a
shows   s' ∼ d ∼ t'
proof −
{
  have a0: (the (domain-of-event a)) = did
    using p2 domain-of-event-def by auto
  have a1: s' = fst (remove-cap-right s did dst-cap right-to-rm)
    using p2 p6 exec-event-def by auto
  have a2: t' = fst (remove-cap-right t did dst-cap right-to-rm)
    using p2 p7 exec-event-def by auto
  have a3: (interferes did s d)
    using p4 a0 by auto
  have a4: s' ∼ d ∼ t'
    using a1 a2 a3 p0 p1 p3 p5 remove-cap-right-wsc by blast
}
then show ?thesis by auto
qed

lemma remove-cap-right-dwsc-e: dynamic-weakly-step-consistent-e (Remove-Cap-Right did dst-cap right-to-rm)
proof −
  {
    have ∀ d s t. reachable0 s ∧ reachable0 t
        ∧ (s ∼ d ∼ t)
        ∧ (interferes (the (domain-of-event (Remove-Cap-Right did dst-cap right-to-rm))) s d)
        ∧ (s ∼ (the (domain-of-event (Remove-Cap-Right did dst-cap right-to-rm))) ∼ t)
        ⟶ ((exec-event s (Remove-Cap-Right did dst-cap right-to-rm)) ∼ d ∼ (exec-event t (Remove-Cap-Right did
```

*dst-cap right-to-rm*)))
**proof** −
**{**
**fix** *d s t*
**assume** *p1*: *reachable0 s*
**assume** *p2*: *reachable0 t*
**assume** *p3*: (*s* ∼ *d* ∼ *t*)
**assume** *p4*: (*interferes* (*the* (*domain-of-event* (*Remove-Cap-Right did dst-cap right-to-rm*))) *s d*)
**assume** *p5*: (*s* ∼ (*the* (*domain-of-event* (*Remove-Cap-Right did dst-cap right-to-rm*))) ∼ *t*)
**have** ((*exec-event s* (*Remove-Cap-Right did dst-cap right-to-rm*)) ∼ *d* ∼ (*exec-event t* (*Remove-Cap-Right did dst-cap right-to-rm*)))
**by** (*metis Event.simps*(*56*) *domain-of-event-def option.sel p1 p2 p3 p4 p5 remove-cap-right-wsc-e*)
**}**
**then show** *?thesis* **by** *blast*
**qed**
**}**
**then show** *?thesis*
**using** *dynamic-weakly-step-consistent-e-def* **by** *blast*
**qed**

### 0.10.8 proving the "dynamic step consistent" property

**theorem** *dynamic-weakly-step-consistent*:*dynamic-weakly-step-consistent*
**proof** −
**{**
**fix** *e*
**have** *dynamic-weakly-step-consistent-e e*
**proof** (*induct e*)
**case** (*Client-Lookup-Endpoint-Name x x1*)
**show** *?case*
**using** *client-lookup-endpoint-name-dwsc-e* **by** *blast*
**case** (*Send-Queuing-Message x1a x2 x3a*)
**show** *?case*
**using** *send-queuing-message-dwsc-e* **by** *blast*
**case** (*Receive-Queuing-Message x*)
**show** *?case*
**using** *receive-queuing-message-dwsc-e* **by** *blast*
**case** (*Get-My-Endpoints-Set x*)
**show** *?case*
**using** *get-my-endpoints-set-dwsc-e* **by** *blast*
**case** (*Get-Caps x*)
**show** *?case*
**using** *get-caps-dwsc-e* **by** *blast*
**case** (*Grant-Endpoint-Cap x1a x2 x3a*)
**show** *?case*
**using** *grant-endpoint-cap-dwsc-e* **by** *blast*
**case** (*Remove-Cap-Right x1a x2 x3a*)
**show** *?case*
**using** *remove-cap-right-dwsc-e* **by** *blast*
**qed**
**}**
**then show** *?thesis*
**using** *dynamic-weakly-step-consistent-all-evt* **by** *blast*
**qed**

**theorem** *noninfluence-sat*: *noninfluence*
**using** *dynamic-local-respect uc-eq-noninf dynamic-weakly-step-consistent weak-with-step-cons* **by** *blast*

**theorem** *weak-noninfluence-sat*: *weak-noninfluence* **using** *noninf-impl-weak noninfluence-sat* **by** *blast*

**theorem** *nonleakage-sat*: *nonleakage*
**using** *noninf-impl-nonlk noninfluence-sat* **by** *blast*

**theorem** *noninterference-r-sat*: *noninterference-r*
**using** *noninf-impl-nonintf-r noninfluence-sat* **by** *blast*

**theorem** *noninterference-sat*: *noninterference*
**using** *noninterference-r-sat nonintf-r-impl-noninterf* **by** *blast*

**theorem** *weak-noninterference-r-sat*: *weak-noninterference-r*
**using** *noninterference-r-sat nonintf-r-impl-wk-nonintf-r* **by** *blast*

**theorem** *weak-noninterference-sat*: *weak-noninterference*
**using** *noninterference-sat nonintf-impl-weak* **by** *blast*
**end**